Layer 2 VPN Architectures

By Wei Luo, - CCIE No. 13,291, Carlos Pignataro, - CCIE No. 4619, Dmitry Bokotey, - CCIE No. 4460, Anthony Chan, - CCIE No. 10,266

- Table of Contents
- Index

Publisher : Cisco Press
Pub Date : March 10, 2005
ISBN : 1-58705-168-0
Pages : 648

Master the world of Layer 2 VPNs to provide enhanced services and enjoy productivity gains

- Learn about Layer 2 Virtual Private Networks (VPNs)
- Reduce costs and extend the reach of your services by unifying your network architecture
- Gain from the first book to address Layer 2 VPN application utilizing both ATOM and L2TP protocols
- Review strategies that allow large enterprise customers to enhance their service offerings while maintaining routing control

For a majority of Service Providers, a significant portion of their revenues are still derived from data and voice services based on legacy transport technologies. Although Layer 3 MPLS VPNs fulfill the market need for some customers, they have some drawbacks. Ideally, carriers with existing legacy Layer 2 and Layer 3 networks would like to move toward a single backbone while new carriers would like to sell the lucrative Layer 2 services over their existing Layer 3 cores. The solution in these cases is a technology that would allow Layer 2 transport over a Layer 3 infrastructure.

Layer 2 VPN Architectures introduces readers to Layer 2 Virtual Private Network (VPN) concepts, and describes Layer 2 VPN techniques via introductory case studies and comprehensive design scenarios. This book assists readers looking to meet those requirements by explaining the history and implementation details of the two technologies available from the Cisco Unified

VPN suite: Any Transport over MPLS (ATOM) for MPLS-based cores and Layer 2 Tunneling Protocol version 3 (L2TPv3) for native IP cores. The structure of this book is focused on first introducing the reader to Layer 2 VPN benefits and implementation requirements and comparing them to those of Layer 3 based VPNs, such as MPLS, then progressively covering each currently available solution in greater detail.

Layer 2 VPN Architectures

By Wei Luo, - CCIE No. 13,291, Carlos Pignataro, - CCIE No. 4619, Dmitry Bokotey, - CCIE No. 4460, Anthony Chan, - CCIE No. 10,266

Publisher : Cisco Press
Pub Date : March 10, 2005
ISBN : 1-58705-168-0
Pages : 648

- Table of Contents
- Index

Copyright

Layer 2 VPN Architectures

Wei Luo

Carlos Pignataro

Dmitry Bokotey

Anthony Chan

Published by:
Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

Printed in the United States of America 1 2 3 4 5 6 7 8 9 0

First Printing February 2005

Library of Congress Cataloging-in-Publication Number: 2003109688

ISBN: 1-58705-168-0

Warning and Disclaimer

This book is designed to provide information about **Layer 2 VPN architectures**. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an "as is" basis. The author, Cisco Press, and Cisco Systems, Inc., shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the author and are not necessarily those of Cisco Systems, Inc.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc., cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers' feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through e-mail at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Corporate and Government Sales

Cisco Press offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information please contact: U.S. Corporate and Government Sales 1-800-382-3419 corpsales@pearsontechgroup.com

For sales outside the U.S. please contact: International Sales international@pearsoned.com

Publisher	John Wait
Editor-in-Chief	John Kane
Executive Editor	Brett Bartow
Cisco Representative	Anthony Wolfenden
Cisco Press Program Manager	Jeff Brady
Production Manager	Patrick Kanouse
Development Editor	Dayna Isley
Copy Editor	Karen Gill
Technical Editors	Archana Sharma, Aamer Akhter, John Chang, Wen Zhang

Team Coordinator	Tammi Barnett
Book and Cover Designer	Louisa Adair
Composition	Mark Shirar
Indexer	Tim Wright

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

European Headquarters
Cisco Systems International BV
Haarlerbergpark
Haarlerbergweg 13-19
1101 CH Amsterdam
The Netherlands
www-europe.cisco.com
Tel: 31 0 20 357 1000
Fax: 31 0 20 357 1100

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-7660
Fax: 408 527-0883

Asia Pacific Headquarters
Cisco Systems, Inc.

Capital Tower
168 Robinson Road
#22-01 to #29-01
Singapore 068912
www.cisco.com
Tel: +65 6317 7777
Fax: +65 6317 7799

Cisco Systems has more than 200 offices in the following countries and regions. Addresses, phone numbers, and fax numbers are listed on the **Cisco.com Web site at www.cisco.com/go/offices**.

Argentina • Australia • Austria • Belgium • Brazil • Bulgaria • Canada • Chile • China PRC • Colombia • Costa Rica • Croatia • Czech Republic • Denmark • Dubai, UAE • Finland • France • Germany • Greece • Hong Kong SAR • Hungary • India • Indonesia • Ireland • Israel • Italy • Japan • Korea • Luxembourg • Malaysia • Mexico • The Netherlands • New Zealand • Norway • Peru • Philippines • Poland • Portugal • Puerto Rico • Romania • Russia • Saudi Arabia • Scotland • Singapore • Slovakia • Slovenia • South Africa • Spain • Sweden • Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • United States • Venezuela • Vietnam • Zimbabwe

Printed in the USA

Dedications

Wei Luo*:* This book is dedicated to my mother, Ximen, in loving memory for her everlasting selfless devotion and belief in me. This book is also dedicated to my father, Jiyang Luo, my sister and brother-in-law, Michelle and Tong Ge, and my

lovely nephew and niece, Jesse and Lauren, for all their sacrifices and support over the years and their enduring love.

Carlos Pignataro*:* I dedicate this book to my son, Luca, and my wife, Veronica, for filling my life with joy. I also dedicate this book to my mother, Elena Renee Goenaga, in loving memory, and to my father, Juan Carlos, with heartfelt gratitude and deepest love.

Dmitry Bokotey*:* To dear friends Tom Kladek, Kevin Taylor, and Carlos Pignataro who opened all the doors.

Anthony Chan*:* Dedicated to my wonderful parents, Foon and Sin Ying Chan, my brother and sister-in-law, Johnny Chan and Diana Chu, and to the memory of my grandmother, Fung Yu Choy, for their guidance and wisdom. I also dedicate this book to my niece, Alicia. May your future be bright and filled with opportunities.

About the Authors

Wei Luo, CCIE No. 13,291, is a technical leader at Cisco Systems, Inc. Since joining Cisco in 1998, Wei has led many product design and development initiatives in remote-access networks, WANs, and MPLS technologies. He is the principle designer and developer for Cisco Pseudowire Emulation and Layer 2 VPN products, such as AToM and VPLS. He actively participates in IETF standardization processes, contributing to and authoring various RFCs and Internet drafts in the IETF working groups. Wei has B.S. and M.S. degrees in computer science.

Carlos Pignataro, CCIE No. 4619, is a senior engineer in the Escalation Team for Cisco Systems, Inc. In this role he is responsible for handling difficult and complex escalations, working on critical or stalled software defects, and participating in the new product and development process. Carlos has a B.S. in electrical engineering and an M.S. in telecommunications and networking. Carlos has contributed to IETF Internet drafts, is an active speaker at Net-workers conventions, and has authored *Cisco Multiservice Switching Networks* also by Cisco Press.

Dmitry Bokotey, CCIE No. 4460, holds a quadruple CCIE title in the fields of Routing and Switching, ISP Dial, Security, and Service Provider. He is a network consulting engineer with the Central Engineering and Metro Ethernet team of Cisco Systems. For the past twelve years, he has designed and implemented diverse networking environments for various large enterprise and service provider customers. Over the course of his career, he has presented seminars on numerous advanced networking subjects. He is coauthor on two other books published by Cisco Press: *CCIE Practical Studies: Security* and *CCNP Practical Studies: Remote Access*.

Anthony Chan, Service Provider CCIE No. 10,266, is a network consulting engineer for Cisco Systems' Advanced Services Central Engineering organization. Anthony participates in MPLS and routing technology teams, which provide focused design and proactive support to service provider and enterprise customers. He holds a bachelor's degree in electrical engineering from Northwestern University and has previously worked at Ford Motor Company and International Network Services.

About the Technical Reviewers

Archana Sharma, CCIE No. 3080, has over five years of troubleshooting experience with the campus switched networks and has been working on the Catalyst switches since they were first released by Cisco. Archana has been troubleshooting and resolving customer issues since 1995 when she joined Cisco as a support engineer in the RTP TAC team supporting the Cisco campus switching product line. She has extensive troubleshooting experience with both Layer 2 switching and Layer 3 switching and provides escalation support for these in her role as team lead and as a Cisco diagnostic engineer.

Aamer Akhter, CCIE No. 4543, joined Cisco Systems in 1998 after graduating from Georiga Tech with a B.S. in electrical engineering to work in the Cisco Technical Assistance Center. He then went on to support Cisco's larger enterprise customers in the NSA unit where he helped design and deploy several large Layer 2 networks. Aamer later moved to Networked Solutions Integration Test Engineering (NSITE), where after a brief stint with IPSec VPNs, he moved into a new group for testing MPLS-VPNs. Four years later, MPLS-VPNS had matured much but testing of MPLS related technologies still continues. Aamer is currently leading a team for testing Layer 3 VPNs and related technologies in a cross-Cisco effort.

John Chang, CCIE No. 2736, is a solution test lead engineer at Cisco Systems, Inc. In this role, John provides technical leadership to verify the function, scalability, and performance of remote access and IP VPN solutions. With over 20 years of network engineering experience, he has designed, developed, and supported a variety of LAN and WANs. John has B.S. and M.S. degrees in electrical engineering.

Wen Zhang, CCIE No. 4302, is a member of the TAC escalation team at Cisco Systems. He currently focuses on VPN and security technologies. With Cisco since 1997, Wen was a regular contributor to the Cisco Open Forum. He earned his B.S. and M.S. degrees in electrical engineering from Clemson University.

Acknowledgments

Wei Luo: I would like to thank all the people from Cisco Press who helped make this book possible, and special thanks go to our Executive Editor Brett Bartow for putting up with us.

I want to thank Greg Burns, Gary Green, and other colleagues at Cisco Systems for the unreserved support, wisdom, and insight. Without all of your talent and excellence in making the Layer 2 VPN technology a reality, this book would not have been possible.

This book is a true work of collaboration. I want to thank my coauthors for all their hard work and devotion. I cannot thank my coauthors without giving my special gratefulness to Dmitry Bokotey for opening the door for me and always being there.

Carlos Pignataro: I would like to thank my coauthors for their teamwork and talent that made this book possible. Thanks to our reviewers for all their helpful comments. I also want to thank all the people at the Cisco Systems family that in one way or another make these technologies a reality.

Special thanks to W. Mark Townsley for his most valued insight, openness, and for giving me a great opportunity to learn, and to my colleague Wen Zhang for always helping me find answers to my questions with his expertise and experience.

I would like to thank as well our Executive Editor Brett Bartow and our Development Editor Dayna Isley for their professionalism and patience, along with all the Cisco Press team.

Finally, I want to thank all the design engineers, network architects, and NOC engineers that keep Layer 2 VPN networks running smoothly.

Dmitry Bokotey: This book is a product of collective effort. I would like to thank my coauthors, Anthony Chan, Wei Luo, and Carlos Pignataro, for their enormous talent and expertise.

As always, I'm grateful to my wife, Alina, for her help with writing and editing my chapters.

I would also like to thank Brett Bartow and others at Cisco Press for another fruitful collaboration. Your patience is truly appreciated.

Big thanks to my Cisco Systems family for the opportunity to explore and implement the technology that became the basis for this book.

Finally, I want to thank my mom and dad for being there for me and my little Alyssa for letting me be there for her.

Anthony Chan: Thanks to my coauthors for their continued perseverance during this entire process.

Thanks to Brett Bartow and Dayna Isley and others from Cisco Press for their patience despite my numerous requests for extensions along the way.

I would also like to thank all Cisco engineering and architecture team who developed and supported L2TPv3 and UTI for creating an equivalent IP pseudowire solution.

This Book Is Safari Enabled

The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

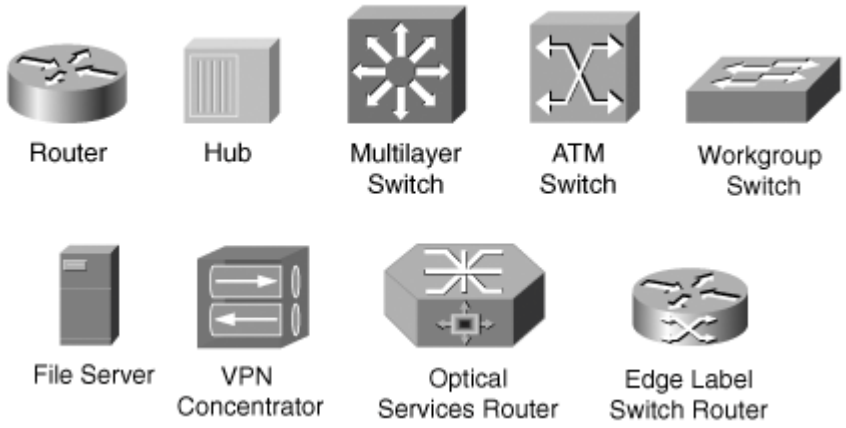
To gain 45-day Safari Enabled access to this book:

- Go to http://www.ciscopress.com/safarienabled
- Complete the brief registration form
- Enter the coupon code P7OE-29WP-IPO0-DHPJ-G8BD

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please e-mail customer-service@safaribooksonline.com.

Icons Used in This Book

Cisco Systems uses the following standard icons to represent different networking devices. You will encounter several of these icons within this book.



Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in the IOS Command Reference. The Command Reference describes these conventions as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italics* indicate arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets [ ] indicate optional elements.
- Braces { } indicate a required choice.
- Braces within brackets [{ }] indicate a required choice within an optional element.

Introduction

Until recently, the VPN landscape has been quite complex as service providers have struggled with how best to accommodate traditional access technologies (such as, dial, Frame Relay, and ATM) along with new ones (like, Ethernet and wireless) and Layer 3 VPNs over a common network infrastructure. A new solution, enabling service providers to converge Layer 2 and Layer 3 services and provide legacy data services over an IP or MPLS backbone, promises to simplify matters, benefiting both service providers and enterprises.

The historical disconnect between legacy Layer 2 and Layer 3 VPN solutions has forced service providers to build, operate, and maintain separate infrastructures to accommodate various VPN access technologies. However, this costly proposition is no longer necessary. As part of its new Unified VPN Suite, Cisco Systems now offers next-generation Layer 2 VPN services like Layer 2 Tunneling Protocol version 3 (L2TPv3) and Any Transport over MPLS (AToM) that enable service providers to offer Frame Relay, ATM, Ethernet, and leased line services over a common IP/MPLS core network. By unifying multiple network layers and providing an integrated set of software services and management tools over this infrastructure, the Cisco Layer 2 VPN solution enables established carriers, IP-oriented ISP/CLECs, and large-enterprise customers (LECs) to reach a broader set of potential VPN customers and offer truly global VPNs.

Although Layer 3 MPLS VPNs fulfill the market need for some customers, they have some drawbacks. Namely, Layer 3 MPLS VPNs only handle IP traffic, and they require the customer to change their usual CPE/subscriber model from a Layer 2 peering model to interfacing with the service provider at Layer 3. Ideally, carriers with existing legacy Layer 2 and Layer 3 networks would like to move towards a single backbone while new carriers would like to sell the lucrative Layer 2 services over their existing Layer 3 cores.

The solution in these cases is a technology that allows Layer 2 transport over a Layer 3 infrastructure. This book assists readers looking to meet those requirements by explaining the history and implementation details of the two technologies available from the Cisco Unified VPN suite: AToM for MPLS-based cores and L2TPv3 for native IP cores.

Goals and Methods

The goal of this book is to introduce you to the technologies and practices related to Layer 2 VPN architectures and Pseudowire Emulation, including the following:

- Addresses Layer 2 VPN applications utilizing both AToM and L2TPv3 protocols providing extensive conceptual background.
- Compares Layer 3 versus Layer 2 provider provisioned VPNs.
- Discusses IETF standardization activities for pseudowire emulation edge-to-edge and Layer 2 VPNs.
- Describes mechanisms used to decode control plane signaling and data plane pseudowire packets.
- Specifies and exemplifies how to maintain quality of service (QoS) in AToM and L2TPv3 pseudowire environments.
- Provides advanced AToM topics such as load-sharing, inter-AS scenarios, and VCCV.
- Describes Path MTU Discovery mechanisms and issues in L2TPv3 networks.
- Introduces VPLS concepts, design, and configurations.
- Details how to achieve security and authentication.

In addition to describing the concepts related to Layer 2 VPNs, this book provides an extensive collection of case studies that show you how the technologies and architectures work. The case studies include both AToM and L2TPv3 and reveal real world service provider and enterprise design problems and solutions with hands-on configuration examples and implementation details. The case studies include all Layer 2 technologies transported using AToM and L2TPv3 pseudowires including Ethernet, Ethernet VLAN, HDLC, PPP, Frame Relay, ATM AAL5 and ATM cells, and advanced cases.

After reading this book, you should be able to understand, describe, and explain Layer 2 architectures and design, configure and troubleshoot complex network scenarios that use AToM and L2TPv3.

How This Book Is Organized

Although this book could be read cover-to-cover, it is designed to be flexible and allow you to easily move between chapters and sections of chapters to cover just the material that you need more work with. Accordingly, the book follows a modular design. *Layer 2 VPN Architectures* is divided into the following main parts and chapters:

- **Part I: Foundation** The book begins by explaining the existing market drivers for Layer 2 VPNs and explores where each of the various types of VPNs exist. It introduces the architectural framework and choices for Layer 2 VPNs and delves into pseudowire emulation realizations and details. This part also describes the architectural reference model and standardarization process of Layer 2 VPNs and pseudowire technologies, and introduces you to AToM and L2TPv3.

  Chapter 1, "Understanding Layer 2 VPNs": This chapter introduces L2VPNs and its motivations. It also compares Layer 2 versus Layer 3 VPNs.

  Chapter 2, "Pseudowire Emulation Framework and Standards" This chapter presents the pseudowire emulation reference model and architectural components, defines key terminology, and explains the history and standardization of pseudowire emulation in the IETF.

  Chapter 3, "Layer 2 VPN Architectures" This chapter introduces AToM and L2TPv3 and presents business and technical factors to be considered when choosing a Layer 2 VPN technology.

- **Part II: Layer 2 Protocol Primer** This part provides a complete overview of Layer 2 LAN and WAN technologies.

  Chapter 4, "LAN Protocols" This chapter includes and overview of LAN protocols, such as Ethernet II and 802.3, Ethernet dot1Q, Ethernet QinQ, spanning tree, and related technologies.

  Chapter 5, "WAN Data-Link Protocols" This chapter outlines different WAN protocols including HDLC, PPP, Frame Relay, and ATM.

- **Part III: Any Transport over MPLS** The chapters in this part cover the theoretical and operational details of MPLS and LDP as they pertain to AToM, analyze the control plane (pseudowire signaling) and data plane (data encapsulation), describe the design and implementation of AToM technologies, and provide LAN and WAN protocols over MPLS and advanced AToM case studies.

Chapter 6, "Understanding Any Transport over MPLS" This chapter details AToM and LDP operations for pseudowire signaling and describes AToM pseudowire encapsulation.

Chapter 7, "LAN Protocols over MPLS Case Studies" This chapter presents the underlying theory and case studies for LAN protocols over MPLS including port-to-port and dot1Q modes.

Chapter 8, "WAN Protocols over MPLS Case Studies" This chapter presents the underlying theory and case studies for all WAN protocols over MPLS and their various modes of operation.

Chapter 9, "Advanced AToM Case Studies" This chapter concludes the AToM section with advanced case studies such as load sharing, preferred path selection, AToM with traffic engineering (TE), AToM over GRE, inter-AS AToM, VCCV and QoS.

- **Part IV: Layer 2 Tunneling Protocol Version 3** This part discusses the theory on Layer 2 protocols over Layer 2 Tunneling Protocol version 3 (L2TPv3) in IP networks, analyzes the control plane L2TPv3 protocol interactions and data plane encapsulation details, and provides LAN and WAN protocols and advanced case studies.

  Chapter 10, "Understanding L2TPv3" This chapter starts with Universal Transport Interface (UTI) history and evolvement into L2TPv3; it then details L2TPv3 control plane including tunnels, sessions, cookies, AVPs, control plane messages and message formats, as well as the L2TPv3 data plane including the data packet formats.

  Chapter 11, "LAN Protocols over L2TPv3 Case Studies" This chapter presents the underlying theory and case studies for LAN protocols over L2TPv3 including static sessions, static sessions with keepalives, and dynamic sessions for Ethernet port-to-port and VLAN modes with and without VLAN rewrite.

  Chapter 12, "WAN Protocols over L2TPv3 Case Studies" This chapter presents the fundamental theory and case studies for all WAN protocols over L2TPv3 including HDLC, PPP, Frame Relay (DLCI and port modes), and ATM (AAL5 and the various Cell Relay modes).

  Chapter 13, "Advanced L2TPv3 Case Studies" This chapter details advanced case studies for L2TPv3 networks including Path MTU Discovery, ATM OAM Emulation and cell packing, and QoS.

- **Part V: Additional Layer 2 VPN Architectures** This part presents Any-to-Any Layer 2 VPN interworking, local switching, and Virtual Private LAN Service (VPLS). The part includes both architectural and theoretical frameworks, and configuration and design case studies.

Chapter 14, "Layer 2 Interworking and Local Switching" This chapter introduces the related Layer 2 VPN architectures of Layer 2 IP and Ethernet interworking (that is, routed and bridged interworking, respectively), Layer 2 local switching, and the combinations of interworking with local switching. This chapter includes details and case studies for both AToM and L2TPv3.

Chapter 15, "Virtual Private LAN Service" This chapter introduces the VPLS application with theory, configuration, and multiple case studies.

The book concludes with an appendix that summarizes the Cisco and IETF L2TPv3 AVP attribute types.

Part I: Foundation

Chapter 1. Understanding Layer 2 VPNs

This chapter covers the following topics:

- Understanding traditional VPNs
- Introducing enhanced Layer 2 VPNs

A virtual private network (VPN) is a data network that utilizes a portion of a shared public network to extend a customer's private network. This provides private communications between end users, such as remote offices and telecommuters. VPNs can be broadly categorized as either Layer 2 VPNs or Layer 3 VPNs.

This chapter begins with an overview of traditional VPNs, both Layer 2 and Layer 3, followed by a more in-depth look at enhanced Layer 2 VPN solutions over IP/Multiprotocol Label Switching (MPLS) and the factors that motivated their evolution. This chapter also covers the different types of Layer 2 VPNs available today.

Understanding Traditional VPNs

This section offers examples of traditional, older forms of VPNs (as opposed to the newer, enhanced Layer 2 VPNs that are the topic of this book)specifically Layer 3 VPNs and legacy Layer 2 VPNs.

Legacy Layer 2 VPNs

Originally, VPNs were built using leased lines to provide connectivity between various customer locations. A customer bought the leased line as a service from the provider. The leased line was installed between the customer's sites that required interconnectivity. The line was dedicated to that customer, and others did not share it.

Since its introduction in the 1990s, Frame Relay has dominated the field of early VPN technologies. Frame Relay has enabled service providers to offer the same basic connectivity to their customers as with the leased lines, except instead of provisioning a dedicated line for each customer, they have been able to use a shared line and allocate a virtual circuit for each customer to keep each customer's traffic separate. The virtual circuits are referred to as *permanent virtual circuits* (PVC). By configuring PVCs, the data-link connection identifiers (DLCI) associated with various devices are established. This builds a tunnel for customer traffic to follow a dedicated path through the service provider's shared network.

A service provider merely supplies the Layer 2 connectivity and is not involved in the Layer 3 aspects of the customer's traffic (hence the name, Layer 2 VPNs). The advantage of Layer 2 VPNs is the independence that customers have in terms of controlling their Layer 3 network design for routing, addressing, and so on.

Frame Relay's independence from all Layer 3 protocols has made it a popular choice for LAN-to-LAN connections and intranet communications. Service providers also offer ATM-based VPNs as a higher-speed alternative to Frame Relay. Currently, most service providers offer Layer 2 VPNs using Frame Relay, ATM, or combinations of the two.

Layer 3 VPNs

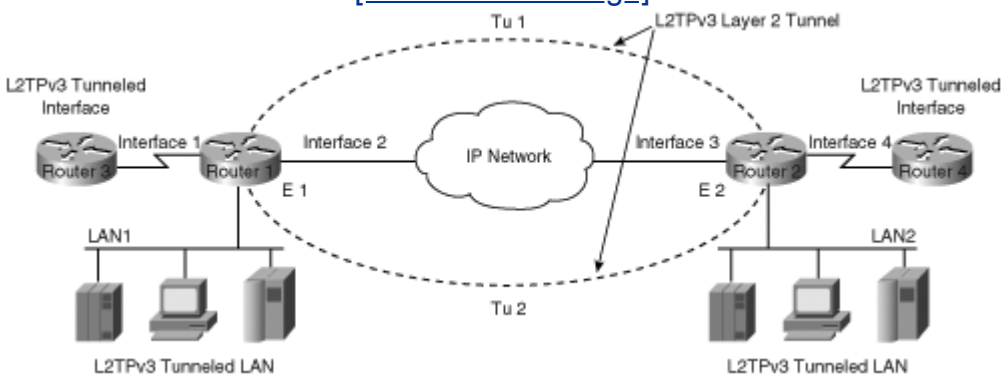
Currently, the most widely used Layer 3-based VPN technologies are IP Security (IPsec) and MPLS Border Gateway Protocol (BGP) VPNs. These technologies can service intranet, extranet, and Internet access applications for securely interconnecting customer's remote sites.

In Layer 3 VPNs, the service provider offers a leased line or PVC connection between a customer and the nearest point of presence (POP) on the service provider's network.

Figure 1-1 shows an example of a basic MPLS VPN model.

Figure 1-1. *Private BGP Network with Private IP Addresses*

[View full size image]

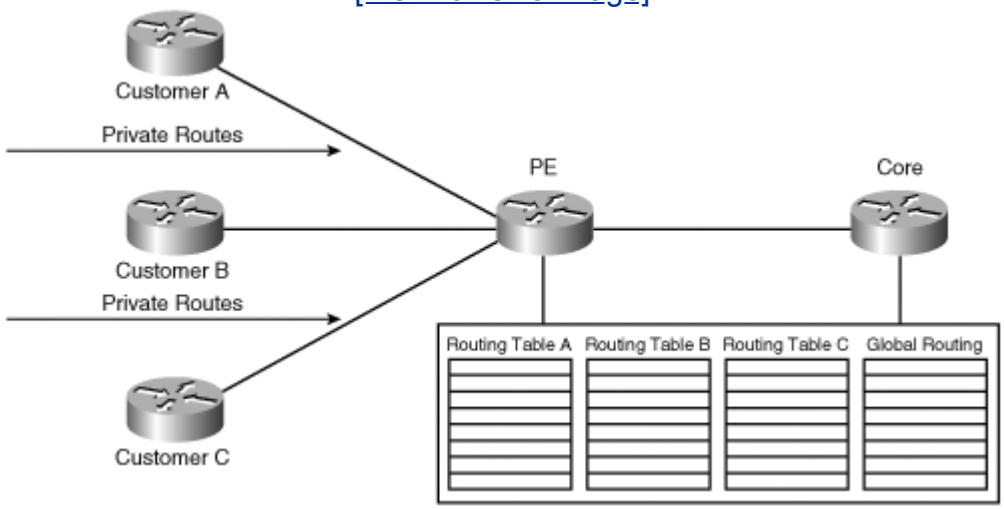


In an MPLS VPN, the customer edge (CE) router peers up with the PE router at Layer 3 instead of the other CE routers (as is the case with enhanced Layer 2 VPNs), providing the PE router with routing and forwarding information for the private network. The PE router then collects one private routing table for each customer and stores the tables along with the public Internet routing information.

In Figure 1-2, not all of the customer's private networks are passed on to the global routing table.

Figure 1-2. *PE/CE Relationship in an MPLS VPN*

[View full size image]



Through Layer 3 VPNs, customers rely on Internet service provider (ISP) IP/MPLS-based backbones for private and secure any-site-to-any-site communication.

Challenges of Traditional VPNs

Layer 3 VPNs also have several limitations. For instance, IP is the only protocol that is supported over the MPLS Layer 3 VPN network. The customer gives up control of its routing to the service provider, which might not be desirable for both parties. Also, over-utilization of the PE routers is possible. To become truly scalable, the MPLS VPN implementation requires a wide deployment of high-end, more powerful and, thus, more expensive routers.

As mentioned, legacy Layer 2 connection services provide the point-to-point connectivity upon which private networks are built. To support a customer's Layer 3 traffic, a separate Layer 3 network has to be built. This results in service providers having to maintain separate networks for Layer 2 and Layer 3 traffic, which is difficult and costly.

Another challenge that traditional Layer 2 service providers face is that if they have to expand their networks, the highest speeds they can go to with ATM in the core is OC48. They cannot grow to higher speeds or make use of more cost-effective technologies, such as Ethernet. Therefore, service providers have been searching for ways to maximize the efficiency and cost of their infrastructures and simplify management.

These goals can be achieved in an environment in which multiple Layer 2 services can be transported across a common IP/MPLS backbone. Newly developed IP-based services allow customers to minimize their network expenses while improving their productivity and competitiveness. For service providers, these new developments mean an opportunity to offer savings to their customers, which, in turn, can prompt an increase in customer base and service revenue.

The following types of service providers would benefit from such a solution:

- Carriers that currently offer only circuit-based Layer 2 infrastructures and would like to expand Layer 3 infrastructure to sell more services
- Service providers that currently offer only Layer 3 infrastructure and would like to cost effectively expand their offering of Layer 2 services
- Service providers that currently offer circuit-based Layer 2 and IP-based Layer 3 services throughout separate infrastructures and would like to join the two to increase profitability

Introducing Enhanced Layer 2 VPNs

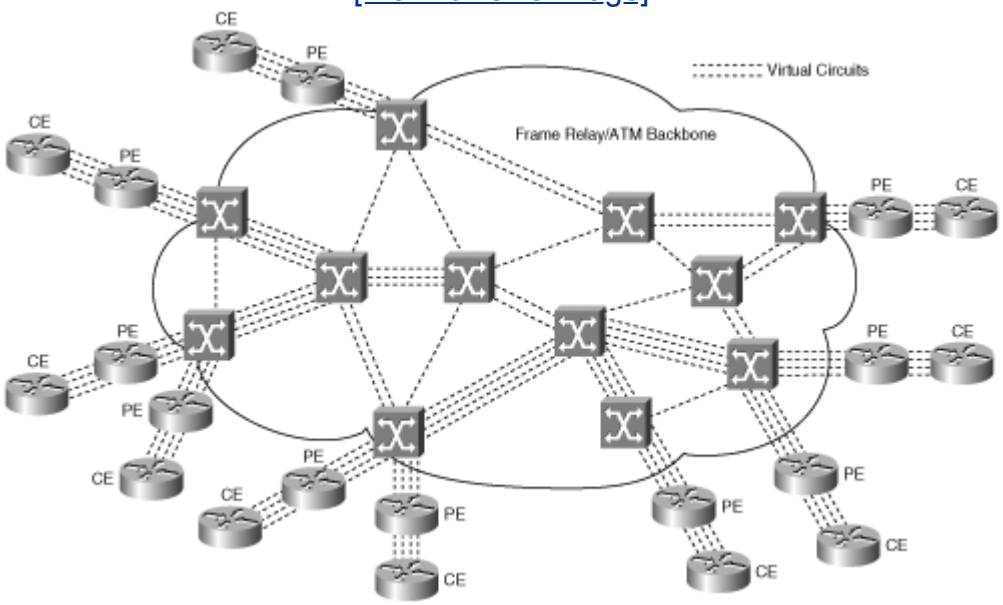
A solution has been developed to address the desire to consolidate the Layer 2 and IP/MPLS-based Layer 3 VPNs. New, enhanced Layer 2 VPNs allow offering a traditional Layer 2 service, such as Frame Relay, by employing an IP/MPLS network infrastructure. This might decrease the cost of providing a comparable service using a dedicated Layer 2 network. In contrast with Layer 3 VPNs, Layer 2 VPNs are capable of carrying multiprotocol (IP and non-IP alike) transport across a common infrastructure. Another drawback of Layer 3 VPNsthe need for edge routers to support routing tables of every connected VPNis eliminated with enhanced Layer 2 VPNs because customer routing tables are not stored on the provider's network. Instead, they are transparently switched site-to-site to the customer's own infrastructure, which reduces complexity.

Even though a Layer 2 service over IP/MPLS might cost the same as a dedicated ATM/Frame Relay-based Layer 2 network, the ability to offer new value-add services is one of the most compelling reasons to move to a packet-based network.

Figure 1-3 illustrates a sample topology with Layer 2 VPN service. Instead of building a separate, private IP network and running traffic across it, enhanced Layer 2 VPNs take existing Layer 2 traffic and send it through point-to-point tunnels on the IP/MPLS network backbone.

Figure 1-3. *Layer 2-Based VPN Services*

[View full size image]



Both enhanced Layer 2 VPNs and Layer 3 VPNs rely on IP/MPLS transport through the core. The principal difference lies in how PE-CE router relations are handled. In an enhanced Layer 2 VPN, the PE router is not a peer to the CE router and does not maintain separate routing tables. Rather, it simply maps incoming Layer 2 traffic onto the appropriate point-to-point tunnel.

Enhanced Layer 2 VPNs use the privacy of Frame Relay and ATM and the flexibility and scalability of IP/MPLS. They deliver network services over routed IP/MPLS networks. Higher efficiency and scalability are achieved because service decisions are made at the VPN and tunnel endpoints and switched without requiring additional provisioning.

With enhanced Layer 2 VPNs, service providers can offer such services as VPNs with managed Internet, intranet, and extranet without the complexity that they required in the past. The new Layer 2 VPN services do not require additional equipment spending because they are available by upgrading Cisco IOS Software. By reducing customer networking complexity and cost, the new Layer 2 VPNs allow service providers to expand their customer base to small and medium-sized businesses.

Layer 2 services have proven to be steady revenue-generating resources because the provider is not required to participate in customer Layer 3 services. Therefore, although service providers are branching into IP/MPLS-based core networks, they continue to maintain an extensive network of Layer 2-based equipment and services. By combining Layer 2 transport with Layer 3, enhanced Layer 2 VPNs offer an attractive alternative and convergence point for Layer 2 and Layer 3 infrastructures.

Some of the key advantages of enhanced Layer 2 VPNs over other VPN techniques include the following:

- **Simple design and implementation** Because these Layer 2 VPNs are based on the IP/MPLS model, the simplicity of IP/MPLS reduces the amount of administration involved in deploying and maintaining the Layer 2 VPN services.

- **New services facilitation** Additional revenue is realized by selling more services to existing customers or by offering new services. By introducing services such as Pseudowire Emulation Edge to Edge (PWE3) (discussed in Chapter 2, "Pseudowire Emulation Framework and Standards"), other revenue-generating services are easy to add. Service providers can sell more bandwidth and better performance to their existing Layer 2 customers.

By utilizing enhanced Layer 2 VPNs, service providers can do the following:

- Lower the cost of providing legacy Layer 2 services through new generation IP/MPLS cores.

- Expand their present Layer 2 networks without having to further invest in their legacy networks.

- Reduce service provider's capital expenditures (capex) and operational expenses (opex) associated with offering numerous services to a customer through service consolidation across a shared infrastructure. Implement transparent LAN and IP/MPLS functionality for IP/MPLS VPN services by providing a simple tunneling mechanism.

- Preserve current investment while building Layer 2 VPN support.

- Transport Layer 2 and Layer 3 protocols.

In addition, new Layer 2 VPNs enable service providers to broaden the geographic scope of their established Layer 2 service to places where their Layer 2 infrastructures are not currently present. By using the IP/MPLS core, traditional Layer 2 services can extend as far as the core.

Enhanced Layer 2 VPNs offer service providers several major cost reductions on their existing infrastructure, which leads to higher profitability. First, by consolidating networks, service providers reduce operational costs by migrating to a single infrastructure, rather than supporting and investing in multiple infrastructures. Second, enhanced Layer 2 VPNs eliminate the need to provision multiple infrastructures (such as Layer 2 and Layer 3) across the core, reducing expensive configuration and maintenance costs.

Service providers can also continue to make money from their existing investments. Existing investments represent expenses not only in equipment, but also in configuration (such as creating circuits, security, and service levels). Although new Layer 2 VPNs offer high return on investment (ROI) when you are buying a routing platform because they integrate with the existing infrastructure, they also help maximize the ROI on the existing infrastructure by working with it, rather than replacing it. By aggregating traffic from ATM, Frame Relay, or Ethernet edge platforms, equipment and configuration investments continue to generate revenue, rather than create more cost or end their return.

On the customer side, enhanced Layer 2 VPNs offer the following advantages:

- Simple to configure

- Provide connectivity of non-IP protocols, both routable and bridged

With enhanced Layer 2 VPNs, customers can independently maintain their routing and security policies. Deployed edge platforms connecting to customer networks continue to create the circuits and interface with customer networks, whereas the Layer 2 VPN-enabled IP/MPLS routing platform essentially creates an intelligent "pipe" to move the traffic through the core, emulating the customer circuit. A VPN that is based on Layer 2 eliminates the need for end users to exchange routing information with service providers, thus reducing the network management, complexity, and associated costs. Additional investment in equipment is unnecessary because the existing customer hardware is sufficient.

Some of the features of enhanced Layer 2 VPNs are as follows:

- The configuration is simplified because only two endpoints must be configured and the rest is signaled across the core, unlike with traditional Layer 2 networks in which you must provision hop by hop.
- The transition from a traditional Layer 2 VPN from the customer's point of view is uncomplicated.
- The customer is responsible for its own routing. All the provider needs to show is that CE-to-CE connection is single hop.
- Because the service provider does not take part in the routing process, the customer's routing privacy is preserved from the provider.
- Layer 2 VPN does not require storing a routing table for each site on the service provider's end.
- A misbehaving CE can, at worst, flap its interface, as opposed to an MPLS VPN, whereby an interface flapping can affect performance of the provider's edge router because of BGP peering.

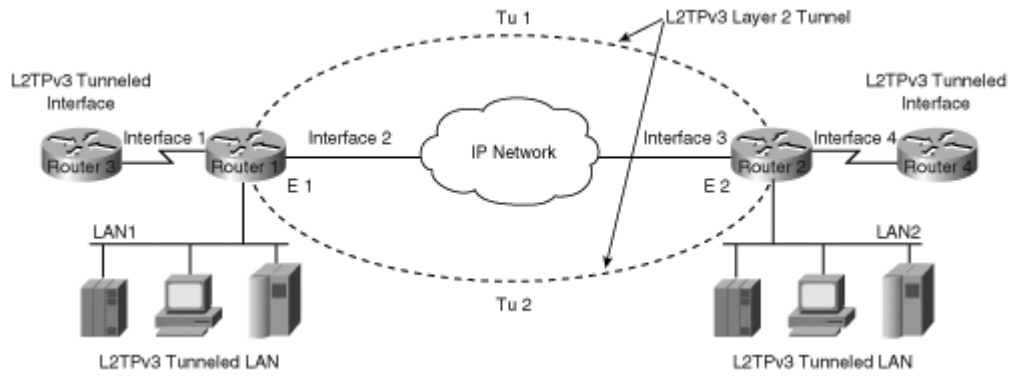
Several enhanced Layer 2 VPN techniques have been developed. One such technique, defined in an IETF draft, is known as Any Transport over MPLS (AToM), which has been designed to allow an MPLS-enabled network to transport Layer 2 frames. Another emerging technology within the IETF is the Layer 2 Tunneling Protocol Version 3 (L2TPv3).

Both AToM and L2TPv3 have the common objective of transmitting packet-switched traffic (Frame Relay, ATM, and Ethernet) across a packet-switched network (PSN). What separates the two is the fact that AToM transports Layer 2 traffic over an MPLS-enabled network, whereas L2TPv3 transports it over a native IP network core. Both L2TPv3 and AToM are offered as part of the new Cisco Unified VPN Suite.

Figure 1-4 shows a sample enhanced Layer 2 VPN topology. The Layer 2 VPN tunnels provide the transport to make routers 3 and 4 appear to be directly connected to Packet over SONET (POS) interfaces (interfaces 1 and 4).

Figure 1-4. *Enhanced Layer 2 VPN Example*

[View full size image]



Supported Layer 2 encapsulations include 802.1Q VLAN, Cisco High-Level Data Link Control (HDLC), Ethernet, Frame Relay, POS, ATM, and PPP.

The first phase of Layer 2 VPN development in Cisco IOS Software supports like-to-like connectivity. This requires that the same transport type be at each end of the network. In the second phase, Layer 2 VPNs were enhanced to provide interworking functions that can connect disparate transport types at each end, such as Frame Relay at one end connecting to Ethernet VLAN at the other.

Note

Subsequent chapters refer to "enhanced Layer 2 VPNs" as "Layer 2 VPNs" for simplicity.

Summary

The enhanced Layer 2 VPN approach is the preferred approach for networks to extend and scale legacy Layer 2 VPN deployments, transport-oriented carriers in general, or any situation that has few VPN sites.

Service providers that are building on the vision of extensible and efficient packet-based infrastructures need a deployable migration solution to maximize and protect their existing investments and revenues. Development of the new Layer 2 VPN technologies such as AToM and L2TPv3 enables the consolidation of Layer 2 and 3 networks while building the value-added IP service portfolios.

Chapter 2. Pseudowire Emulation Framework and Standards

This chapter covers the following topics:

- Pseudowire emulation overview
- Pseudowire emulation standardization

Chapter 1, "Understanding Layer 2 VPNs," introduced Layer 2 virtual private network (VPN) concepts and the problems it is designed to solve. It also highlighted the technology and deployment differences between Layer 2 VPNs and Layer 3 VPNs.

Among the new packet-based Layer 2 VPN architectures, pseudowire emulation forms the foundation for transporting Layer 2 traffic across IP/Multiprotocol Label Switching (MPLS) networks. This chapter describes the general architecture for pseudowire emulation and networking solutions proposed in standardization organizations that sometimes compete with one another. It also highlights other Layer 2 VPN architectures that are based on pseudowire emulation.

Pseudowire Emulation Overview

Pseudowire emulation is essentially a mechanism that re-creates the characteristics of a Layer 1 or Layer 2 circuit service, such as time-division multiplexing (TDM) or Frame Relay, over a packet-switched network (PSN). Pseudowires are emulated circuits that carry service-specific protocol data units (PDU) from one customer device to another through the service provider network. To end customers and their devices, it is transparent that the circuit service is provided through pseudowire emulation. In other words, if the transit network is migrated from a circuit-based legacy network to a packet-based IP/MPLS network, end customers do not perceive any change in services offered by the service provider.

The motivation for pseudowire emulation comes from the desire to have a converged network that delivers multiple services that are currently provided by parallel or overlay networks. Each of these parallel networks offers a specific service. Parallel networks are not only expensive in terms of capital expense and operational costs, but they also make it difficult to expand and maintain network infrastructure and services.

Because IP traffic has increasingly become the majority of the overall network communication, many service providers realize the benefit of investing in packet-based core networks either by expanding the existing PSNs or migrating from their legacy circuit-based networks. Although aiming at providing new packet-based services such as voice over IP (VoIP) and video on demand with this new network infrastructure, service providers also look for ways to migrate the existing services to the new infrastructure to maximize the return on capital and operational investment without impact to the existing revenue streams. Pseudowire emulation makes it possible to achieve this objective.

The next sections describe the fundamental concepts of pseudowire emulation and the processes involved in its deployment, as follows:

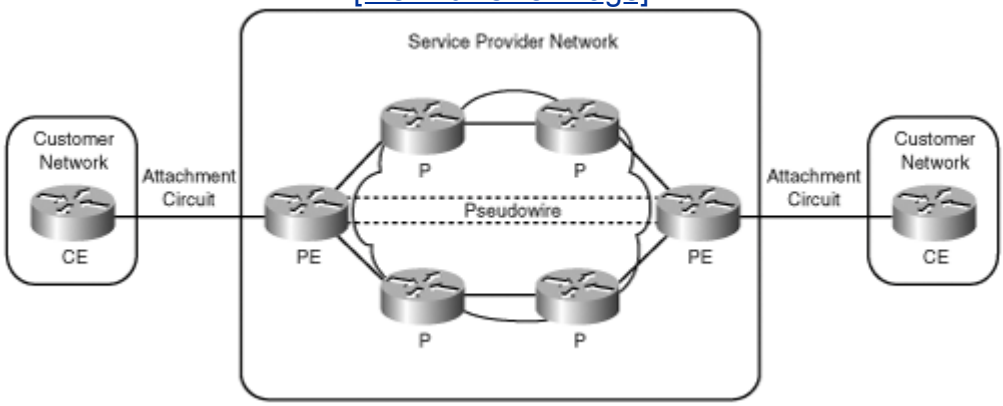
- Network reference model
- Protocol layer and system architecture
- Transporting over PSNs
- Pseudowire setup

Network Reference Model

Despite different Layer 2 VPN solutions and deployment models, a common network reference model can be applied to illustrate the general properties of pseudowire and other network components in the pseudowire emulation architecture, as shown in Figure 2-1.

Figure 2-1. ***Pseudowire Emulation Network Reference Model***

[View full size image]



A provider edge (PE) device is in the service provider administrative domain. It provides pseudowire emulation service to a customer edge (CE) device that belongs to the administrative domain of the customer.

One or more attachment circuits are used to connect a CE to the PE. An attachment circuit can be an Ethernet port, an Ethernet VLAN, a PPP session, a High-Level Data Link Control (HDLC) link, a Frame Relay data-link connection identifier (DLCI), an ATM virtual path identifier (VPI)/virtual connection identifier (VCI), and so on.

A pseudowire is a virtual circuit between two PE devices that interconnects two attachment circuits. You can set it up through manual configuration or automatic signaling. After you establish a pseudowire between two PE devices, native frames received from an attachment circuit are encapsulated into pseudowire PDUs and sent over pseudowire to the peering PE. When pseudowire PDUs arrive at the receiving PE device, they are changed back into the native form and forwarded to the corresponding attachment circuit.

Provider (P) devices form the packet-switched core network and are transparent to CE devices. They are unaware of pseudowires and pseudowire traffic, which PE devices manage. This kind of transparency alleviates the design complexity of the core network. Therefore, you can optimize the core network for core routing and packet forwarding performance without being constrained by the complexity of edge services. This transparency also helps to scale the number of emulated circuits. You need to provision only the edge devices for new circuits; you can leave the core devices alone.

Protocol Layer and System Architecture

Pseudowire emulation involves three protocol layers:

- PSN layer
- Pseudowire encapsulation layer
- Payload layer

The PSN layer specifies the network addressing information of PE devices, which can be IPv4 addresses, IPv6 addresses, or MPLS labels. Network devices use the PSN layer to determine the forwarding path of pseudowire packets. You can think of this path as a packet-switched tunnel that carries pseudowire packets.

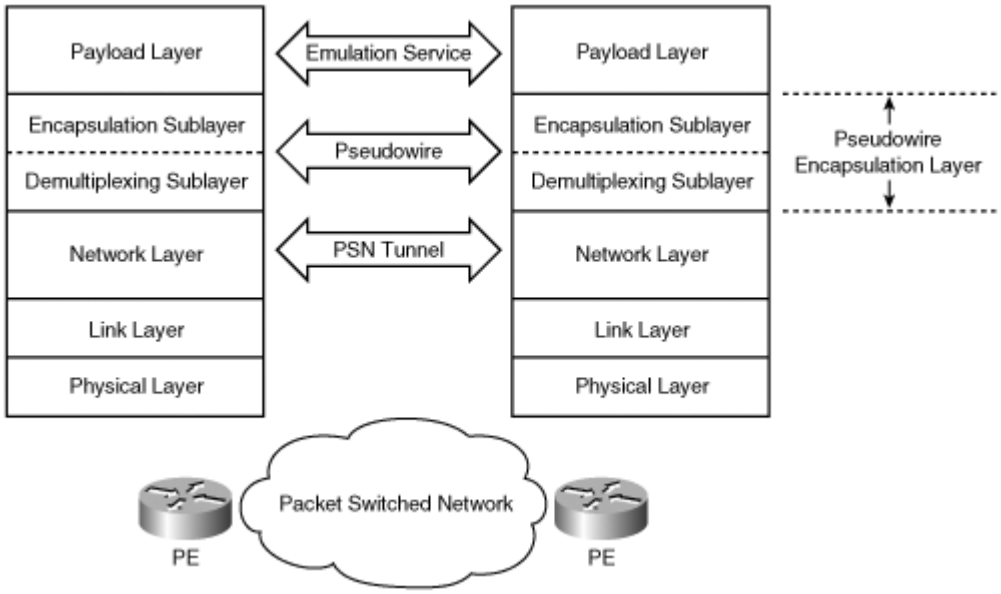
The pseudowire encapsulation layer consists of a pseudowire demultiplexing sublayer and an encapsulation sublayer. The pseudowire demultiplexing sublayer provides a means to carry multiple pseudowires over a single packet-switched tunnel. Each pseudowire has a demultiplexing value that is unique within a tunnel. The encapsulation sublayer carries payload encapsulation information that is removed at the ingress PE device so that the receiving PE device can reconstruct the payload into its native form before sending it to the attached CE device. For example, when the sublayer is transporting Frame Relay traffic over MPLS networks, it removes the Frame Relay header. Payload encapsulation information, such as the backward explicit congestion notification (BECN) bit and discard eligible (DE) bit, must be placed in the encapsulation sublayer. If necessary, this sublayer also carries sequence numbers that are used for in-order packet delivery.

The payload layer carries the pseudowire payload in various forms. For example, it can be Frame Relay packets in the native form or simplified form, ATM AAL5 packets, ATM cells, Ethernet packets, and so on.

Figure 2-2 illustrates the interaction of pseudowire protocol layers that reside on two peering PE devices. Each layer on one PE communicates with the same layer on the other PE through the lower layers, and the lower layers provide services to the upper layers.

Figure 2-2. *Pseudowire Emulation Protocol Layers*

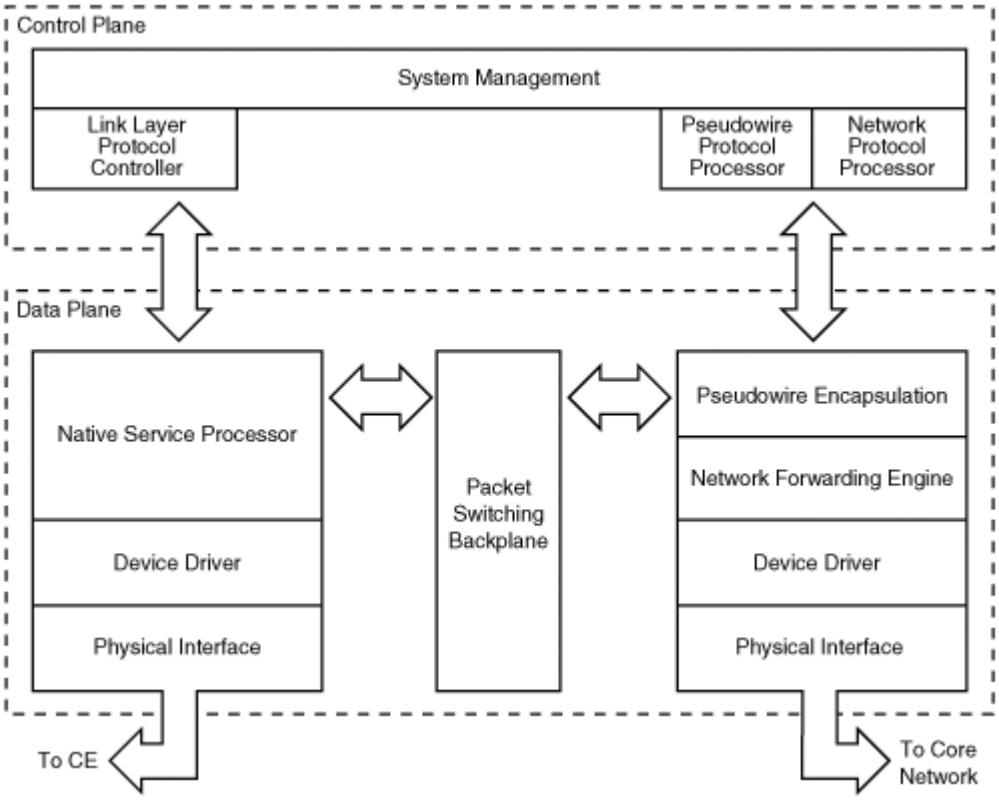
[View full size image]



PE devices play the key role in pseudowire emulation. In fact, the conversion between native circuits and emulated circuits is performed mostly inside PE devices. Therefore, you can benefit from having a high-level understanding of the system architecture of a PE device. Figure 2-3 shows an example of the general system architecture.

Figure 2-3. *PE Device System Architecture*

[View full size image]



The PE device system architecture is divided into the control plane and the data plane. The data plane components include the following:

- **Physical interfaces** Convert bits into electronic signals back and forth on the physical media.
- **Device drivers** Serve as the intermediate layer that constructs media-specific framing for the physical interface and provides a media-independent interface to the upper layer.
- **Native service processor and pseudowire encapsulation** System modules that deal with the data packet manipulation, which is discussed in detail in the following sections.
- **Network forwarding engine** When a data packet is passed to the network forwarding engine from the pseudowire encapsulation module, a destination network address is also provided. Depending on the type of the PSN that carries the pseudowire traffic, the network forwarding engine looks up the address in the IPv4, IPv6, or MPLS forwarding tables. If it finds an outgoing interface, it encapsulates the packet with the appropriate link encapsulation and sends the packet out of the output interface. Otherwise, it discards the data packet.

The control plane components include the following:

- **Link layer protocol controller** Performs line protocol signaling, such as Frame Relay Local Management Interface (LMI) and ATM Integrated Local Management Interface (ILMI), which is needed for setting up attachment circuits.

- **Pseudowire protocol processor and network protocol processor** Perform pseudowire and routing protocol signaling procedures respectively. PE devices use these procedures to establish pseudowires and packet forwarding paths, as illustrated in Figure 2-2. The forwarding information that is obtained through the signaling procedures is distributed to the data plane so that the forwarding table can be populated.

Native Service Processing

In some deployment scenarios, data packets that arrive from attachment circuits are forwarded into pseudowires in their native form. In other cases, you must process native packets before applying the pseudowire encapsulation.

Often, different types of attachment circuits require different native service processing procedures. Furthermore, attachment circuits of the same type might require slightly different processing depending on the configuration that is associated with each attachment circuit. Native service processing occurs on a per-attachment circuit basis.

The native service processor (NSP) can manipulate packets in whichever way is necessary as the packets pass through it. For example, when a PPP packet arrives from a PPP attachment circuit that uses HDLC framing, the NSP removes the HDLC header so that the remaining PPP payload can be in a media-independent format. When the pseudowire encapsulated PPP payload arrives at the far-end PE device, the payload is passed to the NSP after the pseudowire encapsulation is removed. Then the NSP associated with the outgoing attachment circuit determines whether media-specific framing needs to be applied to the PPP payload.

When Ethernet VLAN tags are used as service delimiter, they usually have only local significance. The role of NSP is to remove the service-delimiting VLAN tag when receiving a packet from the VLAN attachment circuit and to add a local service-delimiting VLAN tag when it receives a packet from the pseudowire.

The NSP also normalizes certain control information in different native packet encapsulation into a unified representation for pseudowire operation. For example, besides the Ethernet native frame format, Ethernet packets from CE devices can arrive in other native encapsulations, such as Frame Relay or ATM bridged encapsulations. By normalizing the different forms of native packets into a single Ethernet frame format, you reduce the complexity of pseudowire processing.

Pseudowire Encapsulation Processing

After going through native service processing, the payload is ready for pseudowire encapsulation processing. The NSP might gather some payload-specific control information and pass it to the pseudowire encapsulation processor (PEP), typically through an out-of-band mechanism. The rationale behind the out-of-band mechanism is that in this way, the PEP can treat the payload as an opaque data object; therefore it is relieved from payload-protocolspecific operation.

The payload control information is used for per-packet signaling that is necessary for certain services. Besides this information, the pseudowire encapsulation might include timing for real-time traffic or sequencing for out-of-order detection.

For point-to-point pseudowire emulation, a one-to-one relationship exists between attachment circuits and pseudowires. In other words, given an attachment circuit, the PEP has a corresponding pseudowire and vice versa. A pseudowire consists of a transmitting and a receiving demultiplexer. The transmitting demultiplexer is applied to the payload along with other control information and sent to the network forwarding engine for the remote PE device to identify the pseudowire. When the network forwarding engine passes a pseudowire packet to the PEP, the PEP uses the receiving demultiplexer in the packet header to determine to which attachment circuit and NSP it needs to redirect after removing the pseudowire encapsulation.

Transporting over the PSN

Depending on the PSN infrastructure, you can use either IP or MPLS to transport pseudowire traffic. The PSN infrastructure not only determines the network layer encapsulation for pseudowire packets, but it also usually determines the format of the pseudowire demultiplexer. For instance, if you use IP as the underlying transport, the demultiplexer can be some kind of IP tunnel protocol field that provides demultiplexing capability. If you are using MPLS, you can employ an MPLS label in the label stack for such a purpose.

Some might argue that the pseudowire demultiplexer is not part of the pseudowire encapsulation because of its association with PSN tunneling protocols. This book categorizes pseudowire demultiplexer as part of the pseudowire encapsulation based on its functionality for pseudowires, not how it is implemented in a tunneling protocol. Moreover, the type of pseudowire demultiplexer can differ from the type of the underlying PSN. For example, in some deployment scenarios, it is necessary to carry pseudowire payload over MPLS and then over IP. That is usually because of the existence of a hybrid IP and MPLS network infrastructure for administrative or migration purposes. In this case, pseudowires use MPLS labels as the demulitplexer but IP as the PSN. The protocol details of transporting pseudowire over IP and MPLS are discussed in Chapter 3, "Layer 2 VPN Architectures."

Setting Up a Pseudowire

Prior to establishing an emulated service, you need to set up a pseudowire between two PE devices. You can trigger this setup through one of the following methods:

- Manual configuration
- Dynamic protocol signaling
- An autodiscovery mechanism

The manual setup process is much like provisioning ATM permanent virtual circuits (PVC) in traditional ATM-based Layer 2 VPNs. Essentially, network operators determine all parameters that are needed to set up pseudowires. Then they configure them on the PE devices manually or through network management tools. This can be a labor-intensive process.

Dynamic protocol signaling relieves network operators from many of the operations that are required in the manual setup by exchanging pseudowire information and negotiating the parameters automatically. Some initial provisioning has to be done manually even with dynamic protocol signaling, such as addresses of peering PE devices and identification of remote attachment circuits.

An auto-discovery mechanism utilizes an existing network distribution scheme that is designed for large-scale network operation and management, such as a distributed directory database or an interdomain routing protocol like BGP, to advertise the emulated services. When PE devices learn about the emulated services from each other, they automatically establish pseudowires among them accordingly. Ideally, an auto-discovery mechanism has the minimal amount of manual involvement for pseudowire setup. Although auto-discovery definitely helps in some situations, especially during service migration, the nature of Layer 2 services always incurs a fair amount of manual provisioning compared to Layer 3 services.

Pseudowire setup often requires creating new protocols or extending existing protocols to signal pseudowire information. Creating and extending pseudowire emulation protocols is a hotly debated area in the networking industry and standardization bodies, as described in the next section.

Pseudowire Emulation Standardization

Whenever a major new technology emerges, many companies and organizations get involved in the standardization process and try to push the proposals that are favorable to their business interests.

Pseudowire emulation is no exception. Organizations such as Internet Engineering Task Force (IETF), IEEE, International Telecommunication Union (ITU), ATM Forum, and MPLS Forum have produced many technical proposals and documents on pseudowire emulation. Because the majority of vendor and operator support and activity of pseudowire emulation happen in the IETF, this section focuses on the standard process of the IETF.

IETF Working Groups

The IETF is the predominant organization that standardizes protocols and solutions based on the Internet architecture. The technical work of the IETF is divided by topic into several areas, such as internet, routing, and transport. Under each technical area are several working groups where the actual work is done.

Working groups form as the result of popular interests of solving a particular problem from the networking community and disband when the problem is resolved. Sometimes the charter of a working group changes when a new problem arises or the original problem evolves.

Prior to becoming a working group, an informal discussion group, known as Birds of a Feather (BOF), is formed to measure the scope of the problem and the level of interests in finding the solutions.

In the IETF, the following working groups are discussing proposals that are related to pseudowire emulation:

- **Pseudowire Emulation Edge-to-Edge (PWE3) working group** The goal of this working group is to develop standards for the encapsulation and service emulation of pseudowires. That is, the goal is to encapsulate service-specific PDUs received on one ingress port and carry them across a tunnel, and to emulate the behaviors and characteristics of the service as closely as possible. The two most debated proposals on pseudowire emulation"draft-martini" and "draft-kompella"first surfaced in this PWE3 BOF session. Both drafts are discussed later in this chapter.

- **Layer 2 VPN working group** This working group is responsible for three major Layer 2 VPN solutions or architectures: Virtual Private LAN Service (VPLS), Virtual Private Wire Service (VPWS), and IP-only Layer 2 VPNs. The working group focuses on Layer 2 VPN signaling and provisioning rather than Layer 2 native service emulation, which is the responsibility of the PWE3 working group.

- **Layer 2 Tunneling Protocol working group** This working group is responsible for protocol extensions that support carrying multiple Layer 2 services over IP networks.

Layer 2 VPN Architectures on Pseudowire Emulation

During the Circuit Emulation over Transport BOF (later renamed to PWE3) at the 49th IETF meeting in 2000, two Internet drafts made their debut and stirred up waves of commotion and heated debates. They were known as "draft-martini" and "draft-kompella."

Both drafts addressed the question of how to achieve pseudowire emulation over packet-based networks, but the solutions that each proposed were vastly different. The two drafts were focused on achieving pseudowire emulation over MPLS-based packet networks, and each solution had its advantages and disadvantages. Members of the networking community quickly divided themselves into two camps based on the different design philosophies that were embedded in the two drafts.

> **Note**
>
> The terms *draft-martini* and *draft-kompella* have become synonyms for the two different network architectures that they represent. The actual drafts do not exist in IETF anymore, but the ideas behind them are making their ways toward becoming standards. However, these informal names are still widely used in the networking community to identify the doctrine of each vendor implementation. This section lists the pros and cons of each architecture and does not intend to advertise one method over the other.

draft-martini

The most significant characteristic of draft-martini is its simplicity and straightforwardness. Using Figure 2-1 as reference, the draft describes how to establish a pseudowire between two attachment circuits that are located on two peering PE devices. It also specifies the encapsulation methods for each Layer 2 service. The Label Distribution Protocol (LDP) distributes MPLS labels for various MPLS applications, including pseudowire emulation. The architecture is concerned with creating and managing individual point-to-point pseudowires, which have no correlation to one another.

Before initiating a pseudowire to a remote PE, you need to provision the local PE with a virtual circuit (VC) ID or pseudowire ID shared by both the local and remote attachment circuit, and an IP address of the remote PE. Because the baseline LDP does not readily have the necessary protocol element for pseudowire signaling, the

draft defines a pseudowire extension for LDP. A pseudowire is considered established when the peering PE devices exchange label information for the pseudowire. Using LDP terminology, this means that each PE device sends and receives a label mapping message for a given pseudowire.

Network operators can provision pseudowires by using the architecture that is defined in draft-martini manually or through some sort of network management system. It is much like provisioning traditional Frame Relay or ATM PVCbased Layer 2 VPNs. However, someone could perceive this as either a good attribute or a bad one. Some like the architecture because this is a familiar business, and much of the experience and tools developed in the traditional Layer 2 VPNs can be leveraged. Others think the architecture suffers the same set of problems, such as scalability, as those of the traditional Layer 2 VPNs.

This Layer 2 VPN architecture supports point-to-point Layer 2 services, including Frame Relay, ATM AAL5, ATM Cell, Ethernet, Ethernet VLAN, PPP, and HDLC, in addition to Layer 1 service, such as TDM.

draft-kompella

The architecture that is proposed in draft-kompella does not resemble that of the draft-martini or the traditional Layer 2 VPNs. To a certain degree, it shares some characteristics of Layer 3 dynamic routing. Unlike draft-martini, it involves complex signaling procedures and algorithms, and the provisioning scheme, which is somewhat tricky, works better with some Layer 2 services than others.

One major objective of draft-kompella is to tackle the inherent scalability problem of the traditional Layer 2 VPNs. As shown earlier in Figure 2-1, a pseudowire is needed to connect two CE devices that attach to two different PE devices. To have full connectivity among the CE devices when the number of CE devices increases, the number of pseudowires that needs to be established and managed grows exponentially.

In addition, every time you add a new CE device or move an existing CE device to attach to a different PE device, you must reconfigure all the PE devices that are participating in this VPN to maintain the full-mesh connectivity. This can become a dauntingly labor-intensive task for network operators. The draft attempts to solve the scaling problem by over-provisioning the number of attachment circuits needed for current CE devices so that the existing CE and its PE devices do not need to be reconfigured when adding a new CE to a VPN. The basic premise for over-provisioning is that the attachment circuits between CE and PE devices are relatively cheap.

To provision a Layer 2 VPN using the architecture that is defined in draft-kompella, each CE that belongs to the VPN is given a CE ID, and each CE is configured with a maximum number of CE devices that it can connect to. This is also known as the *CE range*. Each attachment circuit between a CE and a PE is given an index value, which corresponds to a particular remote CE ID in this VPN. By such an arrangement, each CE can derive which attachment circuit connects to which remote CE. Each PE is then configured with the VPNs in which it participates. Each VPN is

denoted by a VPN ID. The PE is provisioned with a list of CE devices that are members of a given VPN. The PE also knows the CE ID, CE range, and the index values for the attachment circuits of each CE.

When a PE is configured with all the necessary information for a CE, it allocates a contiguous range of MPLS labels that corresponds to the CE range. The smallest value in this label range is called the *label base*. For each CE, the PE then advertises its own router ID, VPN ID, CE range, and label base through BGP update messages, which are broadcast to all other PE devices. Even though some PE devices might not be part of the VPN, they can receive and keep this information just in case a CE that is connected to the PE joins the VPN in the future. Because the baseline BGP does not readily have the necessary protocol element for pseudowire signaling, the draft defines a pseudowire extension for BGP.

This architecture solves the scaling problem by making the provisioning task of adding a new CE device a local matter. That is, whenever a new CE device is added, only the CE and the PE to which it is attached need to be configured. Remote CE and PE devices do not need reconfiguration because they can calculate which spare attachment circuit should be used to communicate with the new CE. Remote PE devices can also learn about the new CE through BGP update messages. The broadcast nature of BGP makes it easy to automatically discover PE devices that are participating in Layer 2 VPNs, which further reduces the configuration on PE devices.

The weakness of this architecture comes from the validity of the assumptions it is based on. For example, the low cost of attachment circuits is valid when the CE and PE are directly connected through virtual circuits such as Frame Relay and ATM PVCs, but not when they are connected through a switched Frame Relay or ATM network, or the attachment circuits are individual physical links and ports, such as PPP and HDLC links. In the latter case in which the cost of individual attachment circuits is expensive, over-provisioning becomes impractical. Also, the typical Layer 2 VPNs deployed today are rarely fully meshed because having a fully meshed flat network creates scaling problems for Layer 3 routing, where hierarchy is desired. If a Layer 2 VPN consists only of sparse point-to-point connections, advertising the information of a CE to all other PE devices and keeping it on these PE devices waste network resources because such information is only interesting to a single remote PE.

Not exhaustively, Table 2-1 compares the most noticeable characteristics of the two Layer 2 VPN architectures that are defined by draft-martini and draft-kompella.

Table 2-1. *Pseudowire Emulation Architecture Comparison*

	draft-martini	**draft-kompella**
Network topology	Individual point-to-point pseudowires	Fully meshed point-to-point pseudowires

	draft-martini	draft-kompella
Complexity	Low	High
Scalability	Poor with fully meshed topology; fair with arbitrary topology	High with fully meshed topology; poor with arbitrary topology
Applicability	High; works well on both Layer 2 and Layer 1 services	Fair; works better on directly connected Frame Relay and ATM PVCs than other types
Signaling protocol	LDP	BGP
Discovery protocol	Do not support autodiscovery	BGP
Support base	Wide vendor support	Limited vendor support
Standardization progress	Proceed to PWE3 working group document status	Obsolete

Even though draft-martini has made a lot of progress in standardization and deployment, its primitivenesssuch as the lack of support in Layer 2 VPN autodiscoveryis recognized. New solutions have since been worked on to overcome the issues found throughout product development and network deployment. To find out more about the latest development in the standardization process, refer to the IETF web site at http://www.ietf.org.

Other Layer 2 VPN Architectures

The Layer 2 VPN architectures on pseudowire emulation generally define the procedures for setting up individual pseudowires and encapsulation methods for different Layer 2 services. They are the foundation and building blocks for other types of Layer 2 VPN architectures.

VPWS is directly derived from pseudowire emulation. A VPWS is essentially a network of point-to-point pseudowires that interconnect CE devices of a Layer 2

VPN. Besides the basic pseudowire emulation service, VPWS defines the specifications for point-to-point Layer 2 VPN service in broader terms, such as quality of service (QoS), security, redundancy, VPN membership discovery, and so on. VPWS in some way is designed as a replacement for the traditional Frame Relay or ATM-based Layer 2 VPN.

VPLS also uses the basic pseudowire emulation service, but it is a very different architecture from VPWS. The objective of VPLS is to emulate Transparent LAN Service (TLS) in a packet-based network, which is typically seen in Layer 2 switched Ethernet networks. Instead of acting as a point-to-point cross-connect between the attachment circuit and pseudowire, a VPLS PE functions as an Ethernet bridge. When receiving an Ethernet frame from a CE, the PE looks up the destination MAC address of the frame in its bridging table. If it finds a match, it forwards the frame to the output interface that is specified in the bridging table. Otherwise, it learns and stores the source MAC address in the bridging table, and it floods the Ethernet frame to all output interfaces in the same broadcast domain. Whereas VPWS requires one dedicated attachment circuit for each remote CE device, VPLS allows a single attachment circuit to transmit frames from one CE to multiple remote CE devices. In this respect, VPLS resembles the characteristics of Layer 3 VPN more than VPWS.

IP-only LAN Service (IPLS) is similar to VPLS, but it is tailored and simplified for carrying IP traffic only. In IPLS, a CE device can be a host or a router but not a switch, whereas VPLS has no such a restriction.

Summary

Pseudowire emulation is an emerging networking technology that aims at transitioning traditional Layer 2 services to much leveraged PSNs for operating cost reduction and new value-added services.

Within the network reference model, PE devices are the key components that provide pseudowire emulation services. A PE device consists of the control plane that establishes and maintains pseudowires among PE devices and the data plane that converts frames from their native encapsulation to pseudowire encapsulation back and forth.

This chapter outlined the pseudowire protocol and encapsulation layering. It further explained the various stages of processing in a pseudowire emulation system, such as signaling, native service, pseudowire encapsulation, and tunnel encapsulation.

Even with the fast-growing deployment of pseudowire emulation, the standardization process is an ongoing effort. The IETF and its working groups are the most active and widely respected standardization organizations that develop frameworks and solutions for pseudowire emulation and Layer 2 VPN technology in general. This chapter compared the most debated proposals on pseudowire emulation architectures and highlighted other Layer 2 VPN architectures that are built on top of pseudowire emulation.

Chapter 3. Layer 2 VPN Architectures

This chapter covers the following topics:

- Legacy Layer 2 virtual private network (VPN)
- Any Transport over MPLS (AToM)
- Layer 2 Tunnel Protocol version 3 (L2TPv3)

The previous chapter highlighted different Layer 2 VPN architectures proposed by the network industry and Internet Engineering Task Force (IETF) working groups. In the past few years, significant progress has been made both in designing the Layer 2 VPN protocol specifications and realizing such innovations in a suite of new products. Pseudowire emulation serves as the fundamental building block for different Layer 2 VPN architectures.

A handful of network equipment vendors have developed products that support various levels of pseudowire emulation. The deployment of pseudowire emulation has started growing in the service provider space.

As part of the Unified VPN Suite Solution offering, Cisco IOS Software introduces two flavors of pseudowire emulation:

- AToM
- L2TPv3

To understand the functionalities and characteristics pertaining to these products, you need to first know the inherent properties and operations of the traditional, or legacy, Layer 2 VPNs.

Legacy Layer 2 VPNs

Many types of legacy Layer 2 VPNs exist. The most commonly seen legacy Layer 2 VPNs are based on the following technologies:

- Frame Relay
- ATM
- Data-link switching (DLSw)
- Virtual private dial-up network (VPDN)

Frame Relay and ATM

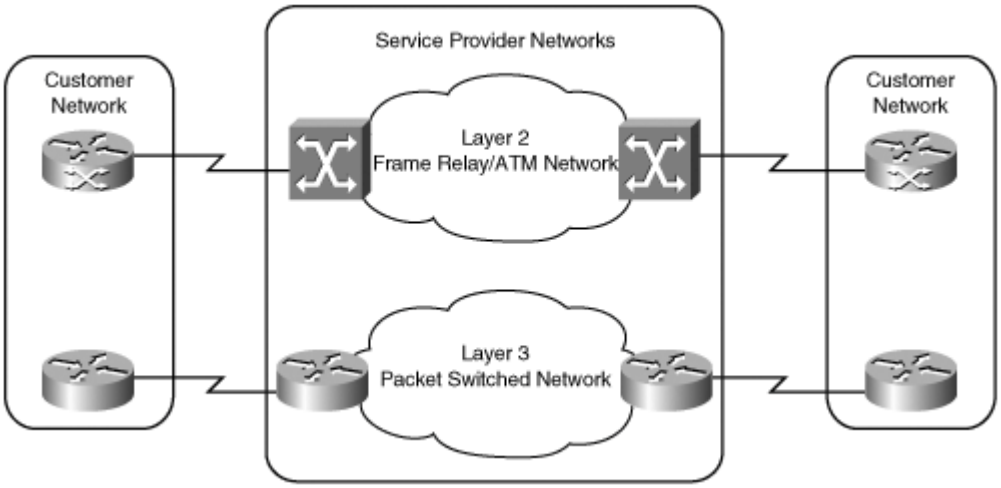
Initially, Layer 2 VPNs were built using leased lines. Frame Relay and ATM are the cost-effective alternatives to the expensive and dedicated leased line service. Service providers can offer these lower cost services to their customers because the Frame Relay and ATM network infrastructure can be shared among many customers while maintaining a comparable level of functionality and guarantee as the leased line service. Frame Relay and ATM also provide link separations among different customers like the leased line service.

One of the most appealing features that Frame Relay and ATM support is bandwidth oversubscription, which the leased line service normally does not provide. Frame Relay and ATM customers typically purchase a committed information rate (CIR) that allows traffic burst to access the service provider network. The CIR is the guaranteed minimal bandwidth when the network is congested. With the bandwidth oversubscription feature, Frame Relay and ATM customers can use more bandwidth than the CIR during traffic bursts as long as the network has available capacity. Frame Relay and ATM also provide circuit multiplexing capability that carries multiple logic or virtual circuits over a single physical link, and the virtual circuits can be used to connect to different remote sites.

Frame Relay and ATM have been the most popular and expansive form of legacy Layer 2 VPN deployment. They are a huge revenue-generating source for service providers. However, Frame Relay and ATM networks are still relatively expensive to build and operate. Service providers often have to maintain separate and parallel networks for Layer 2 and Layer 3 traffic. Figure 3-1 illustrates such parallel networks that offer Layer 2 and Layer 3 services.

Figure 3-1. ***Parallel Network Infrastructures***

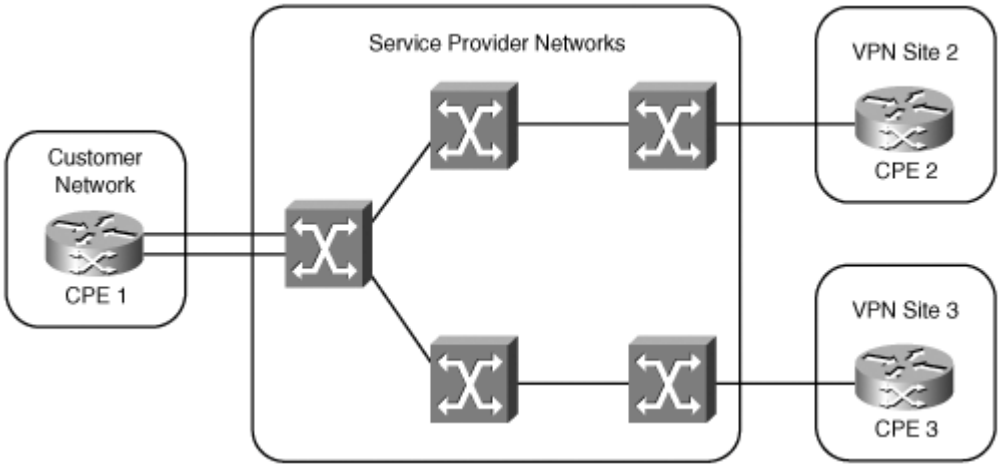
[View full size image]



When building a Layer 2 VPN in a Frame Relay or ATM network, you need to provision edge switches that connect to customer devices with individual virtual circuit mappings, and provision core switches to provide edge-to-edge connectivity for the virtual circuits (VC). Figure 3-2 illustrates a Layer 2 VPN built using a Frame Relay or ATM network. The links that are depicted in the diagram represent logical connections.

Figure 3-2. *Frame Relay or ATM-Based Layer 2 VPN*

[View full size image]



Data Link Switching

DLSw provides a method to transport legacy and nonroutable protocolssuch as Systems Network Architecture (SNA), Network Basic Input/Output System (NetBIOS), and NetBIOS Extended User Interface (NetBEUI)over IP. It has better functionality and scalability than Remote Source Route Bridging (RSRB), but it has limited protocol support.

Virtual Private Dial-Up Network

Many aspects of pseudowire emulation resemble those of VPDN. In this sense, you can think of VPDN as the predecessor of the modern Layer 2 VPN architectures.

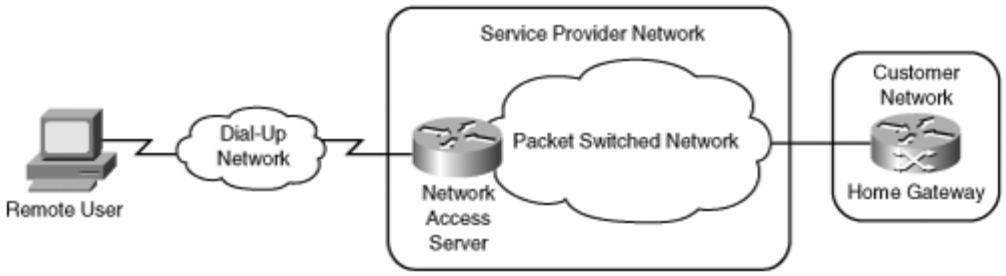
VPDNs are commonly used in wholesale remote-access environments. Without VPDNs, enterprises have to purchase and manage dial-up lines and network access servers for their employees to access internal enterprise resources remotely. The operating and upgrading cost can be prohibitively expensive for small and medium-sized companies. For large companies, VPDNs require a substantial expense when deploying dedicated remote access networks in a widespread geographic environment.

VPDNs enable enterprises to outsource their remote access infrastructures and operations to wholesale service providers. The service providers offer remote access facilities to enterprise remote users from the nearest point of presence (PoP) and backhaul the remote access connections to the enterprise home gateways. The enterprises only need to manage a small number of home gateways for all their remote users. Ultimately, VPDNs lower the overall network operating cost for the enterprises.

For service providers, VPDNs are a new source of revenue serving multiple business and individual customers with the same remote access network infrastructure. When the total number of users increases, service providers can add or upgrade their remote access network capacity in a more economic fashion because all users benefits from it. Figure 3-3 depicts a VPDN network topology.

Figure 3-3. *Virtual Private Dial-Up Network*

[View full size image]



The protocols that support VPDN include the following:

- Point-to-Point Tunneling Protocol (PPTP)
- Layer 2 Forwarding (L2F) Protocol
- Layer 2 Tunnel Protocol Version 2 (L2TPv2)

These protocols tunnel PPP packets between network access servers and home gateways, and PPP is the only Layer 2 protocol they transport. However, because PPP can encapsulate multiple network protocols, such as IP, Internetwork Packet Exchange (IPX), and AppleTalk, many applications find VPDN sufficiently useful.

Note

L2TPv2 is described in the IETF standard RFC 2661. It is a consensual product of the L2TP Extension working group and is derived from the proprietary tunneling protocols PPTP and L2F from Microsoft and Cisco, respectively.

The following is a brief description of how VPDN protocols operate:

1. A remote user or a remote end station initiates a PPP connection to the service provider using either an analog telephone line or an ISDN line.
2. The network access server receives the connection request from the remote user.
3. (Optional) The network access server authenticates the remote user using the specified authentication method, such as Password Authentication Protocol (PAP), Challenge Handshake Authentication Protocol (CHAP), or interactive terminal session.
4. After the remote user is authenticated, an authorization process determines whether the user should be locally terminated or tunneled to a home gateway.
5. If the remote user needs to be tunneled to a remote home gateway, one of the VPDN protocols establishes a tunnel between the network access server and the home gateway, and an optional authentication step can validate the identification of the tunnel endpoints.
6. The user PPP connection is encapsulated into a VPDN session from the network access server to the home gateway.
7. The home gateway authenticates the remote user carried in the VPDN session. Upon successful authentication, the home gateway terminates the PPP connection and grants predefined network access privileges to the remote user.

8. Now PPP frames can pass between the remote user and the home gateway.

For detailed configuration tasks and examples of the legacy Layer 2 VPNs, refer to Cisco.com. Table 3-1 lists some characteristics of the legacy Layer 2 VPNs.

Table 3-1. ***Legacy Layer 2 VPN Comparison***

Legacy Layer 2 VPN	**Payload Type**	**Transport Type**
Frame Relay	Bridged or routed encapsulation	Frame Relay, ATM
ATM	Bridged or routed encapsulation	ATM
DLSw	SNA, NetBIOS, NetBEUI	IP, Frame Relay, Direct
VPDN	PPP	IP, Frame Relay, ATM *
PPTP control packets use IP TCP and data packets use IP GRE. L2F packets use IP UDP. L2TPv2 packets use IP, IP UDP, Frame Relay, and ATM.		

Any Transport over MPLS Overview

In recent years, Multiprotocol Label Switching (MPLS) has had a phenomenal growth in the service provider space, especially where network infrastructures are based on ATM. One of the driving forces for MPLS is to utilize dynamic routing protocols to establish end-to-end virtual connections instead of manually provision ATM VCs hop by hop on each ATM switch. In classic IP over ATM, packets are forwarded based on the predefined ATM VC mappings instead of routing algorithms, which results in suboptimal routing. MPLS resolves this problem by using routing protocols to dynamically create ATM VCs.

MPLS also makes it easy to consolidate the parallel networks into a single MPLS-enabled network. This converged MPLS infrastructure can provide both Layer 2 and Layer 3 services that previously had to rely on separate networks. This section examines how AToM replaces legacy Layer 2 VPNs and the new features it offers.

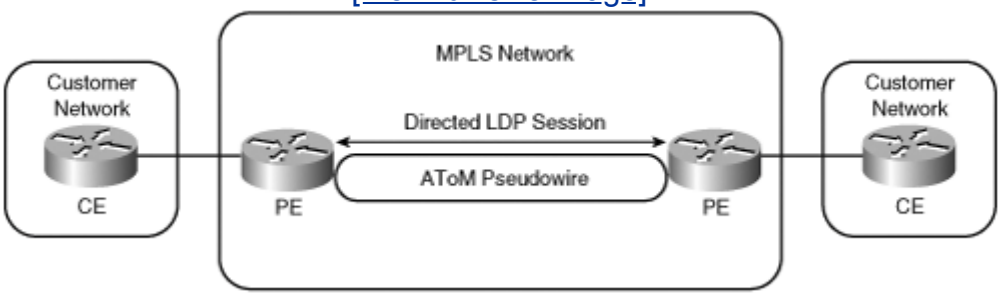
AToM is a pseudowire emulation application that is part of the Unified VPN Suite Solution that Cisco offers to transport Layer 2 traffic over an MPLS network. Besides providing the end-to-end connectivity of the same Layer 2 protocol, AToM is capable of interconnecting disparate Layer 2 protocols through Layer 2 interworking. AToM derives from a series of efforts by service providers and network equipment vendors in an attempt to minimize the impact to existing Layer 2 VPN services and create new service offerings with MPLS-enabled networks.

In the Layer 2 VPN network reference model depicted in Chapter 2, "Pseudowire Emulation Framework and Standards," AToM is enabled on the provider edge (PE) routers, which play a similar role as the edge switches in Frame Relay or ATM-based L2VPNs or the network access server in VPDN. In a Frame Relay or ATM-based Layer 2 VPN, the edge switch maps a Frame Relay or ATM VC connecting to the customer device to a PVC connecting to a core switch by the data-link connection identifier (DLCI) or virtual path identifier (VPI)/virtual connection identifier (VCI) values. In VPDN, the network access server binds a PPP connection from the remote user to a VPDN session. With AToM, the PE router maps an attachment circuit of any supported Layer 2 encapsulation from the customer edge (CE) router to an AToM pseudowire.

An AToM pseudowire is made of a pair of MPLS label-switched paths (LSP). Because an MPLS LSP is inherently unidirectional, to have bidirectional connectivity, a pseudowire is formed by establishing two LSPs in the opposite directions. Different MPLS applications might use different ways to distribute labels. Some use the dedicated Label Distribution Protocol (LDP), whereas others use extensions of existing protocols, including routing protocols. AToM utilizes targeted LDP sessions between PE routers to exchange MPLS labels that are used for pseudowires. You establish a targeted LDP session by sending unicast hello packets rather than multicast hello packets during the LDP discovery phase. LDP also supports TCP message digest, also known as TCP MD5, as its authentication method. Figure 3-4 illustrates the network components of AToM.

Figure 3-4. *AToM Network Components*

[View full size image]



The next sections provide an overview of AToM from the following aspects:

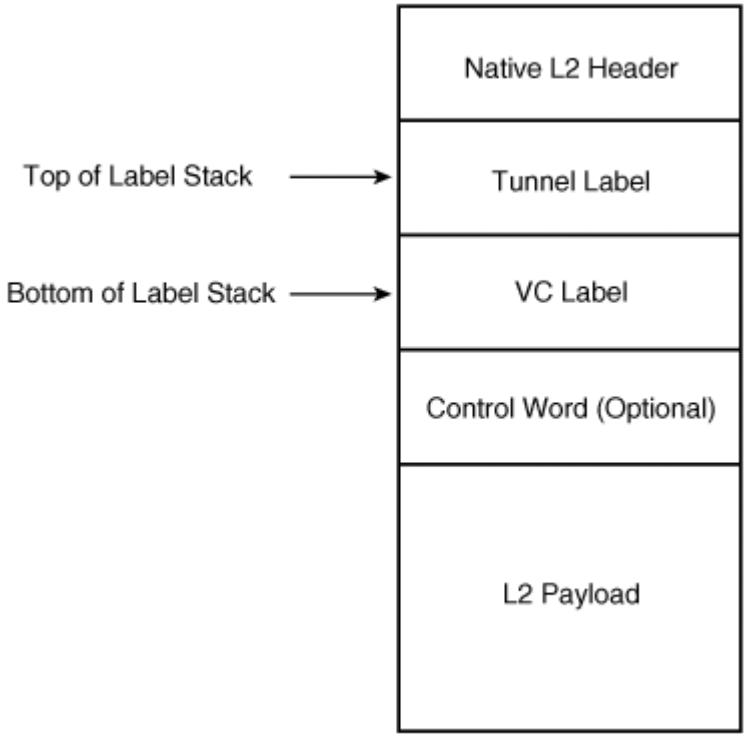
- Label stacking hierarchy in AToM
- Supported Layer 2 protocols
- Decision factors whether to use AToM in your network, such as installation base, advanced features, interoperability, and complexity.

Using Label Stacking in AToM

One common technique that many MPLS applications utilize is label stacking. MPLS label stacking is documented in IETF RFC 3032, "MPLS Label Stack Encoding." The basic idea is to create layers or hierarchies of MPLS labels; each label corresponds to a particular layer in the network architecture. Creating such hierarchies allows aggregation and multiplexing, which improve scalability. It also simplifies the operations on the transit routers, which make forwarding decisions based on the topmost label in the label stack.

The semantics of labels in a label stack might vary from one MPLS application to another. For example, in MPLS traffic engineering, the top label in the label stack represents the traffic-engineered path, and the bottom label represents the original Interior Gateway Protocol (IGP) path. In MPLS Layer 3 VPN, the top label in the label stack represents the IGP path to the next-hop Border Gateway Protocol (BGP) router, which is normally the PE router that originates the VPN routes. The bottom label represents a specific or aggregated VPN route. In Layer 2 VPN, the LDP top label usually represents the IGP path to the peering PE router, and the bottom label represents a *Layer 2 VPN forwarder* on the peering PE router. A Layer 2 VPN forwarder is an abstract entity that switches Layer 2 traffic back and forth between the pseudowire and itself. In the context of pseudowire emulation, the Layer 2 VPN forwarder is usually some sort of attachment circuit. Figure 3-5 shows the overview of an AToM packet.

Figure 3-5. *AToM Packet*



The top label is usually known as the tunnel label or the IGP label. The bottom label is usually known as the VC label or the pseudowire label. The optional control word is not part of the MPLS label stack, but pseudowire encapsulation.

Note

The semantics of labels in a label stack might be different from the previous description when multiple MPLS applications are deployed and integrated in the same MPLS network. For example, use AToM in conjunction with MPLS traffic engineering. You can find examples in Chapter 9, "Advanced AToM Case Studies."

Using label stacking in AToM improves scalability when compared to the scalability of legacy Layer 2 VPNs built on top of Frame Relay or ATM. As you learned in Chapter 2, every time you add a new end-to-end virtual connection or relocate an existing one to a different edge switch, you must ensure that a virtual path extends from one edge switch to the other. If none exists, you need to provision the edge and core switches along the path. With a large number of virtual connections in a typical

Layer 2 VPN, this task amounts to a significant portion of the overall operation cost structure.

Instead of statically provisioning the virtual paths hop by hop, AToM takes advantage of routing protocols to dynamically set up virtual paths across the core network. Only PE routers need to maintain and manage the pseudowire labels for the virtual connections. The pseudowire labels are at the bottom of the label stack, so they are not visible to the transit routers, also known as the Provider (P) routers. The P routers forward packets using the top label and are unaware of the existence of pseudowires.

Many pseudowires can be multiplexed in a single MPLS tunnel LSP. In such a way, the core network is spared from managing and maintaining forwarding information for each pseudowire.

Layer 2 Protocols Supported by AToM

AToM supports a wide range of Layer 2 protocols, including PPP, High-Level Data Link Control (HDLC), Ethernet, Frame Relay, and ATM.

PPP over MPLS operates in the *transparent mode*, in which case PPP sessions are between CE routers, and PE routers do not terminate PPP sessions. In other words, CE routers are the only PPP speakers that process PPP frames through the PPP protocol stack, and PE routers do not participate in PPP protocol exchange.

HDLC over MPLS allows transportation of Cisco HDLC frames over an MPLS network. Like PPP over MPLS, HCLC over MPLS operates in the transparent mode, which is the only mode it supports.

Two types of Ethernet frames are supported in Ethernet over MPLS:

- Untagged Ethernet frames
- IEEE 802.1q tagged Ethernet VLAN frames

PE routers classify Ethernet frames that are received from CE routers into different pseudowires based on the receiving interface or the VLAN tag carried in the Ethernet VLAN frames. Bridging protocol support varies depending on the deployment model. Chapter 7, "LAN Protocols over MPLS Case Studies," has in-depth case studies on running bridging protocols over MPLS networks.

With Frame Relay over MPLS, PE routers forward Frame Relay frames to different pseudowires based on the receiving interface and the DLCI value, and they also provide Local Management Interface (LMI) signaling to CE routers. To Frame Relay customers, the migration in the service provider network is completely transparent. The Frame Relay header is removed at the ingress PE router and added back at the egress PE router. The flags in the Frame Relay headerssuch as backward explicit congestion notification (BECN), forward explicit congestion notification (FECN), discard eligible (DE), and command/respose (C/R)are carried in the pseudowire

control word, which is mandatory for Frame Relay over MPLS. The operation details are described in Chapter 6, "Understanding Any Transport over MPLS."

ATM over MPLS includes two types of ATM services:

- ATM AAL5
- ATM Cell

With ATM AAL5, PE routers either receive ATM AAL5 packets or reassemble ATM cells into ATM AAL5 packets from CE routers and forward them to different pseudowires based on the receiving interface and the VPI or VCI values. The ingress PE router drops all other packets except operations, administration, and maintenance (OAM) cells. The ATM flags, such as explicit forward congestion indication (EFCI) and cell loss priority (CLP), are carried in the pseudowire control word, which is also mandatory for ATM AAL5 over MPLS. ATM Cell over MPLS can encapsulate a single ATM cell at a time or pack multiple ATM cells into one MPLS packet. Both ATM services can be offered in VC mode, VP mode, or port mode. These modes determine the granularity of how ATM packets and cells should be classified and mapped to pseudowires.

Deciding Whether to Use AToM

When determining whether AToM is the right choice for your company, you need to consider several factors, including the following:

- Existing network installation base
- Advanced network services
- Interoperability
- Network operation complexity

The next sections describe how each of these factors can help you determine whether AToM is feasible for your networking environment.

Existing Network Installation Base

For those service providers that have separate parallel networks for Layer 2 and Layer 3 services, an MPLS-enabled network is a natural candidate for converging all services onto a single network infrastructure.

With appropriate software and hardware upgrades, many existing Frame Relay and ATM switches can readily support dynamic routing protocols and perform MPLS label switching. Such a migration allows the service providers to expand their network

capacity and service portfolios and protect their investment on the existing network infrastructure. Transitioning to the packet-based AToM pseudowire emulation has minimal impact to the existing Layer 2 VPN services.

Advanced Network Services

Besides the basic MPLS features such as routing optimization and network consolidation, AToM can leverage advanced MPLS features for enhanced network services, such as MPLS traffic engineering, QoS guarantee, and fast rerouting.

The efficiency with which a service provider utilizes its network infrastructure has a significant impact on the cost structure of its business. The more efficient the use of network resources, the less capital investment that a service provider has to make to provide the desired level of service offering. Traffic engineering aims at solving the problem that some parts of the network are highly congested while others are underutilized. MPLS solves the traffic engineering problem that plain IP routing cannot solve by using MPLS constraint-based routing. Constraint-based routing is essentially a set of algorithms designed to find an optimal path with given routing metrics while confined to the pre-established constraints. The constraints can be performance or administrative requirements imposed by network operators. This book does not go into details about why plain IP routing is insufficient for traffic engineering.

Note

If you are interested in learning more about traffic engineering, you might want to read the following books:

- *MPLS and VPN Architecture*s, Volume I, by Ivan Pepelnjak and Jim Guichard: Cisco Press, 2000.
- *MPLS and VPN Architectures*, Volume II, by Ivan Pepelnjak, Jim Guichard, and Jeff Apcar: Cisco Press, 2003.
- *MPLS: Technology and Applications* by Bruce S. Davie and Yakov Rekhter: Morgan Kaufmann Publishers, 2000.
- *Traffic Engineering with MPLS* by Eric Osborne and Ajay Simha: Cisco Press, 2002.

MPLS traffic engineering helps redirect trafficincluding Layer 2 trafficto less congested parts of the network. Layer 2 services typically come with service-level agreements (SLA). An SLA is a service guarantee that a service provider agrees to offer to its customer on availability, guaranteed bandwidth, burst bandwidth, and so on. The service provider can use an MPLS QoS guarantee to enforce SLAs. The level

of service guarantee is usually associated with the premium that a customer subscribes to. For instance, an SLA with a higher premium might provide more guaranteed bandwidth than an SLA with a lower premium. MPLS constraint-based routing is again used to provide QoS guarantees. It allocates the necessary network resources, such as buffer space and link bandwidth, along the specific path that is established through traffic engineering. Although both MPLS traffic engineering and MPLS QoS guarantee use MPLS constraint-based routing, the difference is that traffic engineering does not require all the bandwidth allocation and queuing mechanisms that are required to provide QoS guarantees.

Another important advanced MPLS feature that AToM can rely on is the ability to reroute traffic to an alternate path in a short period when a failure occurs along the original path, typically within 50 ms. With hop-by-hop, destination-based plain IP routing, the network convergence time is usually seconds upon network failure, which results in packet loss before the network converges. To reduce packet loss during routing transitions, MPLS fast rerouting constructs a protection LSP in advance for a given link by explicitly establishing an alternate path that circumvents the possible failing link. Because the alternate path is set up prior to the link failure, rerouting can take place rather quickly.

Interoperability

A rapidly growing number of service providers and network equipment vendors have become involved in the development and interoperability testing for the MPLS-based pseudowire emulation products.

AToM is the Cisco product for pseudowire emulation over MPLS networks. As the protocol specification and implementation have matured over the past couple of years, the standards-based pseudowire emulation products from different equipment vendors have achieved an excellent level of interoperability. In the service provider space, the deployment has gained significant momentum.

Network Operation Complexity

The previous sections highlighted the advanced features that AToM can offer as the MPLS-based pseudowire emulation. However, they come with a substantial level of complexity in network design and operation, which involves more than just enabling new protocols in the network. Making effective use of these features requires fine-tuning on the network parameters according to the network characteristics.

When an operating problem occurs, AToM also requires highly sophisticated expertise and skills to troubleshoot the issue. For example, LDP is an out-of-band signaling protocol. For a single pseudowire, the control packets might take a different path from the data packets. Therefore, the liveliness of the control plane does not serve as a good indication for that of the data plane, in which case you need more sophisticated diagnosis methods to verify the data plane connectivity, such as MPLS ping.

Establishing AToM pseudowires successfully requires the maximum transmission unit (MTU) settings of both attachment circuits connecting through the pseudowire to match. In addition, the network MTU between the PE routers must accommodate the resulting MPLS-encapsulated packets that carry Layer 2 payload. Because these packets generally do not have an IP header, fragmentation is difficult. That is why packets exceeding the network MTU are dropped. MTU settings need to be carefully engineered throughout the network to avoid connectivity problems.

Layer 2 Tunnel Protocol Version 3 Overview

Although AToM can provide Layer 2 VPN services with advanced network features, you might need an alternative to provide Layer 2 VPN services if your network is not MPLS enabled or you do not want to deploy MPLS technologies. Like many other networking problems, you have multiple options. Depending on your short-term and long-term goals, you can choose the appropriate solution for your needs.

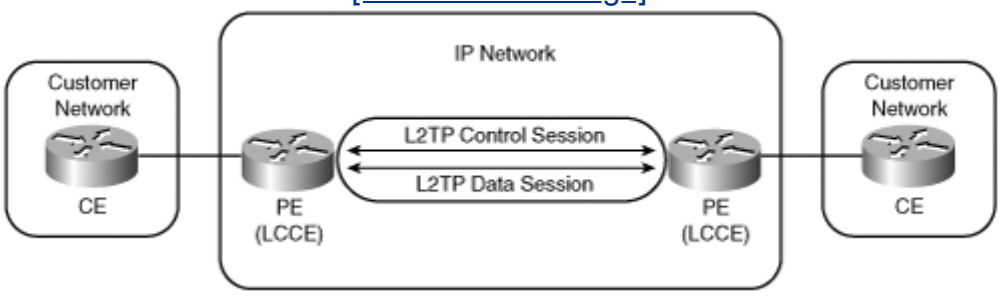
For example, if your goal is to move toward an MPLS-enabled network eventually but you need a time-to-market solution to provide Layer 2 VPN services on top of the existing IP infrastructure, you might choose AToM for Layer 2 VPN services, but you have to overlay AToM pseudowires over IP tunnels, such as generic touting encapsulation (GRE) tunnels. In this way, you can deploy MPLS-based Layer 2 VPN services in a relatively short period of time without being forced to migrate the entire core infrastructure to MPLS immediately. However, if the goal is to ultimately provide Layer 2 VPN services with a pure IP infrastructure, you have the option of choosing an IP-based Layer 2 VPN solution: L2TPv3.

L2TPv2 was originally designed for remote access solutions, and it only supports one type of Layer 2 frames: PPP. Retaining many protocol specifications of version 2, L2TPv3 enhances the control protocol and optimizes the header encapsulation for tunneling multiple types of Layer 2 frames over a packet-based network. L2TPv3 and its supplementary specifications, such as the Ethernet and Frame Relay extensions, describe the requirements and architectures that are applicable to pseudowire emulation using L2TPv3.

L2TPv3 consists of a control plane that uses an in-band and reliable signaling protocol to manage the control and data connections between L2TP endpoints, and a data plane that is responsible for pseudowire encapsulation and provides a best-effort data-forwarding service. In the L2TPv3 network reference models, L2TPv3 is implemented and deployed between a pair of L2TP Control Connection Endpoints (LCCEs). Figure 3-6 illustrates the network components of L2TPv3. The LCCEs are the equivalent of the PE routers in the generic Layer 2 VPN network reference model. For the sake of consistency, this book uses PE router in place of "LCCE" in the context of L2TPv3.

Figure 3-6. *L2TPv3 Network Components*

[View full size image]



Because an L2TPv3 session is inherently bidirectional, an L2TPv3 pseudowire is essentially an L2TPv3 session carrying a particular type of Layer 2 frame. In other words, a one-to-one mapping exists between sessions and pseudowires. During the process of L2TPv3 session establishment, the peering PE routers exchange session IDs. An L2TPv3 session ID is equivalent to an AToM pseudowire label. A session or pseudowire consists of two session IDs. Fundamentally, L2TPv3 pseudowires and AToM pseudowires are set up in a similar fashion. The difference is that the baseline L2TPv3 protocol specification is responsible for constructing such a bidirectional pseudowire, whereas AToM relies on an application-level mechanism that is built on top of the LDP protocol specification for the same function. From the end user's point of view, this difference is insignificant.

> **Note**
>
> In certain deployment scenarios such as interworking between different Layer 2 protocols, AToM and L2TPv3 might carry Layer 3 packets directly. However, because the forwarding decision is still based upon Layer 2 information, such cases belong to the general Layer 2 VPN framework.

Besides using the L2TPv3 control messages to set up pseudowires dynamically, you can use manual configuration to provision the necessary session parameters. When you use manual configuration, you do not need to establish control connection between PE routers.

The next sections give an overview of L2TPv3 from the following aspects:

- L2TPv3 operations
- Supported Layer 2 protocols
- Decision factors whether to use L2TPv3 in your network, such as installation base, advanced features, and complexity

L2TPv3 Operations

Even though L2TP is labeled as an IP-based technology, it is in fact a transport-independent protocol. L2TPv2, which is mostly deployed for remote access applications, specifies mechanisms to tunnel Layer 2 frames over UDP, ATM AAL5, and Frame Relay. L2TPv3 defines the specifications to tunnel Layer 2 frames over IP and UDP.

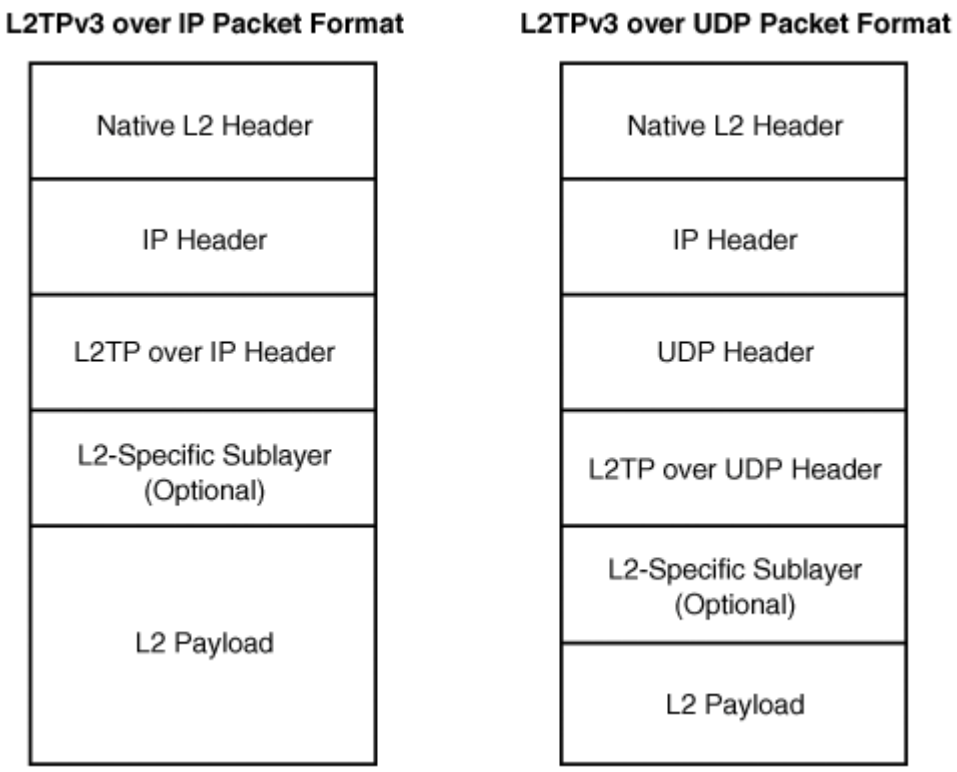
The tunneling mechanism is essentially accomplished by inserting an L2TP header between the IP or UDP header and the Layer 2 payload. A well-known IP protocol

number or UDP port number differentiates L2TP packets from other types of IP traffic. The destination IP address of an L2TP packet is an address of the PE router on the other side of the tunnel. Sessions that are destined to the same PE router are multiplexed by session IDs into a common IP or UDP header.

L2TP control packets are transmitted in-band along with data packets. Therefore, the tunnel endpoints need to have a deterministic way to distinguish one type from the other. For L2TP over UDP, the first bit in the L2TP header indicates whether it is a control packet or a data packet. However, L2TP over IP has a different L2TP header that does not have a field for such indication. Instead, the L2TP header uses the reserved session ID value zero for control packets and nonzero session IDs for data packets.

The discrepancy of the two L2TP header formats is a result of optimization weighted toward different deployment models. The UDP transport mode is friendlier for the cases that require using IPsec to protect L2TP traffic, or traversing Network Address Translation (NAT) and firewalls. The IP transport mode is more tailored for implementing L2TP packet processing and forwarding in high-speed hardware architectures. Figure 3-7 shows an overview of the two formats of an L2TPv3 packet.

Figure 3-7. *L2TPv3 Packet Overview*



L2TP implements a low-overhead reliable delivery mechanism for control packets at the underlying transport layerthat is, IP or UDP. The upper-level functions of L2TP do not have to deal with retransmission or ordering of control packets. L2TP also uses a sliding window scheme for control packet transmission to avoid overwhelming the receiver. In addition, it provides an optional message digest-based authentication to guarantee control packet integrity, and an optional keepalive mechanism to ensure connectivity between tunnel endpoints.

Layer 2 Protocols Supported by L2TPv3

L2TPv3 supports the same set of Layer 2 protocols that AToM does, including PPP, HDLC, Ethernet, Frame Relay, and ATM.

PPP over L2TPv3 mostly operates in the *transparent mode*, in which case CE routers are the only PPP speakers that process the PPP frames through the PPP protocol stack, and PE routers merely forward PPP frames between the peering CE routers transparently.

HDLC over L2TPv3 is similar to PPP over L2TPv3. It allows transportation of Cisco HDLC frames over an IP network in the transparent mode, which is the only mode it supports.

Two types of Ethernet encapsulation are supported in Ethernet over L2TPv3:

- Untagged Ethernet frame
- IEEE 802.1q tagged Ethernet VLAN frame

PE routers classify Ethernet frames that are received from the CE routers into different pseudowires using the receiving interface or the VLAN tag. Bridging protocol support varies depending on the deployment model. Chapter 11, "LAN Protocols over L2TPv3 Case Studies," discusses the details of running bridging protocols over the IP network.

With Frame Relay over L2TPv3, PE routers forward Frame Relay frames to different pseudowires based on the receiving interface and the DLCI number. PE routers also provide LMI signaling to CE routers as if they are Frame Relay switches. Unlike Frame Relay over MPLS, the Frame Relay header is kept intact at the ingress PE router with Frame Relay over L2TPv3; therefore, the egress PE router does not need to reconstruct the Frame Relay header before forwarding the packets to the CE router.

ATM over L2TPv3 also supports ATM AAL5 and ATM Cell services. With ATM AAL5, PE routers receive ATM AAL5 packets or reassemble ATM cells into ATM AAL5 packets from CE routers and forward them to different pseudowires based on the receiving interface and the VPI or VCI numbers. The ATM flags, such as EFCI and CLP, are carried in the L2TPv3 ATM-specific sublayer, which serves a similar purpose to the AToM control word. ATM Cell over L2TPv3 can encapsulate a single ATM cell at a time or pack multiple ATM cells into one L2TPv3 packet. Both ATM services can be offered

in VC mode, VP mode, or port mode. These modes determine the granularity of how ATM packets and cells should be classified and mapped to pseudowires.

Deciding Whether to Use L2TPv3

For organizations and companies that decide to stay with their existing IP-based network infrastructures for the long term and do not intend to migrate to MPLS-enabled networks, choosing L2TPv3 to provide Layer 2 VPN services is obvious. For those who have not decided which technology to choose, consider the following factors to gauge the feasibility and applicability of using L2TPv3 for Layer 2 VPN services.

Existing Network Installation Base

For service providers that do not have parallel legacy networks and those that traditionally provide only Layer 3 services, the problem of maintaining separate networks does not apply to them directly because they do not have the problem to start with. They have little incentive to invest in a new technology unless it brings new revenue opportunities.

As telecommunication deregulation has taken place, these service providers have started eyeing lucrative Layer 2 VPN services. The fastest and least expensive way to provide Layer 2 VPN services in an IP-based infrastructure is to use L2TPv3. AToM relies on a ubiquitous MPLS presence throughout the network infrastructure. If the network is not already MPLS enabled, it has to be migrated to MPLS first. L2TPv3 imposes minimal impactif anyon the core network infrastructure. It only requires the PE routers that provision Layer 2 VPN services to be aware of L2TPv3. In some cases, existing edge routers can readily provide Layer 2 VPN services with proper software upgrades. This is particularly attractive to service providers that are interested in creating new revenue streams with minimal initial investment.

Without L2TPv3, enterprises rely on service providers to provision and manage their Layer 2 network connections among geographically dispersed locations. Not only is the Layer 2 service expensive, but interprovider Layer 2 circuits must also be provisioned when these locations are not covered by a single service provider. The feasibility of provisioning interprovider Layer 2 circuit is constrained by whether these providers have such an interprovider Layer 2 connectivity agreement.

L2TPv3 can be an attractive cost-cutting and easy-to-manage alternative. Instead of getting expensive Layer 2 circuits from service providers, each site can purchase the best and least expensive IP service from a local service provider without worrying about the interprovider agreement issue because IP connectivity always exists among service providers. Each site then enables L2TPv3 on a CPE router and provisions Layer 2 connections to other sites without involving service providers.

Advanced Network Services

Because L2TPv3 uses IP or UDP as its transport layer, integrating with advanced IP-based network services, such as IPSec, is easy. If service providers manage Layer 2 VPN services for their customers, the strong security guarantee that is provided within the service provider network can be sold as a value-added feature. If enterprises manage Layer 2 VPN services, this combination gives them not only site-to-site Layer 2 connectivity but data integrity and privacy when transporting sensitive information across public or shared network infrastructures. With AToM, Layer 2 frames are encapsulated with an MPLS label stack, and there is no IP header in the resulting packet. Therefore, it is quite difficult to apply IPSec features to AToM packets.

Whenever possible, you should set the MTU of both attachment circuits that are connected through a pseudowire to the same value, and set the network MTU to accommodate the resulting L2TPencapsulated packets that carry the Layer 2 payload. If this is not possible, the Cisco IOS L2TPv3 implementation also supports Path MTU discovery and fragmentation options. These make use of the Don't Fragment (DF) bit in the IP header and ICMP messages to discover appropriate MTU settings for pseudowires. When the resulting L2TP packets exceed the pseudowire MTU, users can either choose to drop or fragment the packets.

Plain IP routing and forwarding do not provide advanced network features such as traffic engineering and fast reroute. By deploying IP differentiated services (diffserv), classifying different types of traffic diligently, overprovisioning network bandwidth strategically, and other fine-tunings on routing, you can achieve a fairly high level of service guarantees for Layer 2 VPN services.

Interoperability

L2TPv2 is a widely deployed and highly interoperable protocol, especially in remote access, wholesale dial and broadband networks. It has a large vendor support base.

L2TPv3 evolved from L2TPv2 and has kept many of the major characteristics and specifications of L2TPv2. The control plane procedures are almost identical in both versions. One of the main differences lies in the L2TP header format, which has more impact on the data plane. Another significant change is that the baseline protocol no longer defines the actions for each Layer 2 protocol that is carried in L2TP. Furthermore, it is up to each Layer 2 application to specify the appropriate actions. Because of these differences, the two versions of L2TP implementation are not interoperable.

Network Operation Complexity

L2TPv3 is a relatively simple network protocol as compared to the more sophisticated routing protocols and MPLS protocols. It requires little change to an existing IP-based network and is relatively easy to manage and troubleshoot.

As described in the previous sections, AToM uses LDP as the out-of-band signaling protocol, which means the control packets might take a different path from the data packets. Thus, the control plane connectivity cannot provide a reliable indication for

the data plane connectivity. L2TPv3 uses an in-band TCP-like reliable control connection to set up and tear down data connections. That is why its liveliness serves as a good indication for that of the data plane.

Summary

Prior to pseudowire emulation, Layer 2 VPN services were provided by legacy Layer 2 VPN technologies, such as ATM, Frame Relay, and VPDN, which rely on overlapping parallel network infrastructures.

Pseudowire emulation is the fundamental building block of the new generation of Layer 2 VPN architectures. Many complex network applications that are to replace legacy Layer 2 VPNs or facilitate new requirements are built on top of some form of pseudowire emulation. Cisco offers AToM and L2TPv3 for pseudowire emulation. They are not designed as competing technologies; rather they are optimized for MPLS- and IP-based network infrastructures, respectively. Before determining which product to adopt, consider the technical and business factors and find the right balance between features and manageability. Each has its own merits and implications, some of which are outlined in Table 3-2.

Table 3-2. *Cisco Pseudowire Emulation Products Comparison*

Product Name	**AToM**	**L2TPv3**
Network Infrastructure	IP/MPLS	IP
Signaling Protocol	Directed LDP	L2TPv3
Transport Layer Encapsulation	MPLS label encoding	IPv4
Supported Layer 2 Protocols	PPP, HDLC, Ethernet, Ethernet VLAN, Frame Relay, ATM AAL5, ATM Cell	PPP, HDLC, Ethernet, Ethernet VLAN, Frame Relay, ATM AAL5, ATM Cell
Authentication	TCP MD5	Shared Secret with Message Digest

Product Name	AToM	L2TPv3
Keepalive Mechanism	Unreliable out-of-band LDP keepalive; requires new protocol extensions for reliable connectivity report	Reliable and simple in-band keepalive
Advanced Services	Traffic engineering, QoS guarantee, fast rerouting	IPSec, IP Diffserv, Path MTU discovery, IP fragmentation
Interoperability	Wide vendor and carrier support, good and improving interoperability	Limited vendor and carrier support

Part II: Layer 2 Protocol Primer

Chapter 4. LAN Protocols

This chapter covers the following topics:

- Ethernet background and encapsulation overview
- Metro Ethernet overview
- Metro Ethernet service architectures
- Understanding Spanning Tree Protocol
- Pure Layer 2 Implementation
- 802.1q Tunneling

Now that you've learned the fundamental concepts behind Layer 2 VPNs and pseudowire emulation described in Part I, "Foundation," you need to familiarize yourself with Layer 2 protocols. This chapter describes LAN protocols. Here, you get an overview of Ethernet technology as well as read about the technological and business requirements of both enterprise customers and service providers that are driving the implementation of Metro Ethernet. Finally, you discover the Layer 2 technologies that are available today for transporting traffic over higher-bandwidth circuits.

Ethernet Background and Encapsulation Overview

The motivation behind the development of a computer network came from Xerox wanting to interconnect some of the first personal computers at its Palo Alto Research Center (PARC) and the world's first laser printer so that all of the PARC's computers could print with the same printer. Prior to this, the number of computers at any single facility never exceeded two or three. This was the first instance in which hundreds of computers in the same building required intercommunication.

Ethernet solved two of Xerox's challenges:

- The network had to connect hundreds of computers within the same building.
- It had to be fast enough to support the fast, new laser printer.

Later on, because of the combined efforts of Digital Equipment, Intel, and Xerox, Ethernet became a standard. Ethernet is now the world's most widely used LAN protocol.

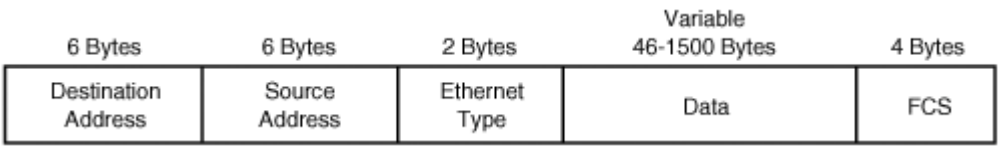
Ethernet uses carrier sense multiple access collision detect (CSMA-CD). The different parts of this protocol are as follows:

- **Carrier sense** Before transmitting data, stations check whether other stations are already transmitting over the multiaccess wire. If other stations are not transmitting, the station can transmit data or wait.
- **Multiple access** All stations are connected to a single physical wire or a single data path.
- **Collision detect** If a collision is detected because two stations transmitted data into the wire simultaneously, both stations stop transmitting, back off, and try again later after a random delay.

The base of Ethernet technology is the Ethernet frame. An IP datagram, for instance, is encapsulated and transmitted in a standard Ethernet (Type II) frame. The frame header is 14 bytes long6 bytes of destination address + 6 bytes of source address + 2 bytes of frame typefollowed by the data portion and completed by 4 bytes of the frame check sequence (FCS). Figure 4-1 shows the fields of the original Ethernet Type II (Ethernet II) frame format.

Figure 4-1. *Ethernet Type II Frame*

[View full size image]



The following explains the fields in an Ethernet II frame:

- **Destination Address** The destination address can be a broadcast address 0xFFFFFFFFFFFF, a specific 48-bit unicast address based on the destination node's MAC address, or a multicast address. You can discover this MAC address from the source address field of a message during protocol synchronization.
- **Source Address** The source address is the sender's 48-bit MAC address.
- **Ethernet Type** The Ethernet Type field is used for higher protocol identification.
- **Data** The Data field contains encapsulated data (such as an IP packet). The valid length ranges for Ethernet II are between 46 and 1500 bytes.
- **FCS** The FCS field contains a 32-bit cyclic redundancy check (CRC) value, which checks for damaged frames.

Note

Originally, Ethernet II was also referred to as DIX after its corporate sponsors Digital, Intel, and Xerox.

The original Ethernet II frame format had some shortcomings. To allow collision detection, the 10-Mbps Ethernet required a minimum packet size of 64 bytes. That meant you needed to pad short frames with 0s. Thus, higher-layer protocols needed to include a Length field to discriminate the actual data from the padding. As a consequence, the original Ethernet frame was changed to include a Length field and to allow for Ethernet to interwork with other LAN media.

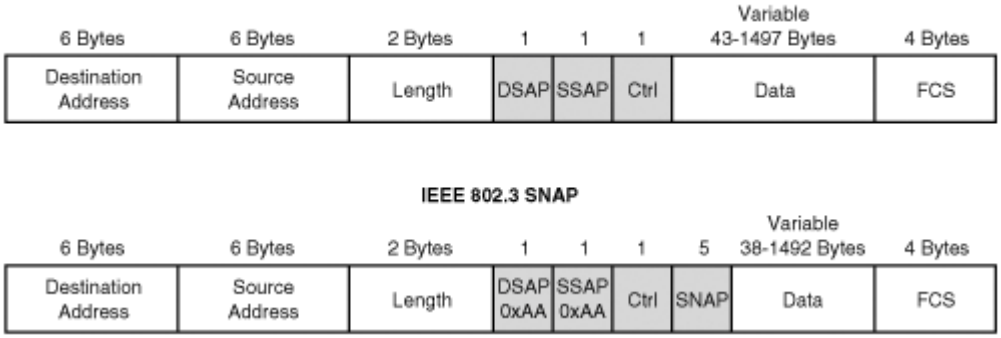
Fortunately, the values assigned to the Ethernet Type field (0x0600 XNS [Xerox], 0x0800 IP [Internet Protocol], and 0x6003 DECNET) were always higher than the maximum frame size with a decimal value of 1500. The 802 committee solution to the task of providing a standard that did not depend on the behavior or characteristics of higher layer protocols was *802.3*. 802.3 replaced the Ethernet Type field with a 2-octet length field. The way to distinguish an Ethernet II from an 802.3 frame is by inspecting the Type/Length field:

- If the value of the field is higher than 1500 decimal, the field represents an Ethernet Type and is Type II.
- If the value of the field is lower than or equal to 1500 decimal, the field represents a length and is 802.3.

In addition, a new form of packet type field was needed, so a Logical Link Control (LLC) header with destination and source service access point (DSAP and SSAP, respectively) and control fields follow the Length field for higher-protocol identification (see Figure 4-2).

Figure 4-2. *802.3 Frame Format*

[View full size image]



The new fields are as follows:

- **Length** This is the length of the frame excluding the preamble, FCS, addresses, and length field.
- **DSAP** A value of 0xAA indicates Subnetwork Access Protocol (SNAP).
- **SSAP** A value of 0xAA indicates SNAP.
- **Control** The Control field specifies the type of LLC frame.

Figure 4-2 also shows an IEEE 802.3 SNAP frame format that is indicated by the DSAP and SSAP values and includes the SNAP field. The SNAP header includes 3 bytes of vendor code and 2 bytes of local code. A vendor code of 0s (0x000000) indicates that the local code is an Ethernet Type II for backward compatibility. This new format moves the Ethernet Type field 8 bytes from its original location in Ethernet II.

Note

The Ethernet Type II frame format is often referred to as *ARPA frame*. The IEEE 802.3 frame format is also called 802.3 LLC to differentiate it from 802.3 SNAP.

Note how the size range for the Data field varies between the different encapsulations (see Table 4-1).

Table 4-1. *Size of the Data Field for Different Ethernet Encapsulations*

Encapsulation	Minimum Size	Maximum Size
Ethernet II	46	1500
802.3 LLC	43	1497
802.3 SNAP	38	1492

Note that you can send IP datagrams smaller than 46 bytes over Ethernet II because IP contains a Total Length field. When you are sending, for example, 36-byte IP datagrams using Ethernet II encapsulation, a 10-byte trailer with all zeroes is appended to the IP datagram. When you are sending the same IP datagram over 802.3 SNAP, the trailer is only 2 bytes, because 8 bytes are used for the LLC + SNAP headers (1 DSAP + 1 SSAP + 1 Control + 3 OUI + 2 Ethertype).

Metro Ethernet Overview

The speed, cost, and availability of Ethernet makes it an attractive transport service option for providers to offer to their customers. The Ethernet family of technologies operates at the following rates:

- 10 Mbps (Ethernet)
- 100 Mbps (Fast Ethernet)
- 1000 Mbps (Gigabit Ethernet)
- 10,000 Mbps (Ten Gigabit Ethernet)

The total price per bit per second is lower for Ethernet compared to other technologies, such as Packet over SONET (POS).

Many providers already offer Ethernet connectivity to their customers; others are considering Ethernet, either as a Layer 2 transport technology or to invest in their existing SONET, Frame Relay, or ATM services with IP virtual private network (VPN) offerings. In both cases, providers must consider the means in which they can use Ethernet to better serve their customers.

Metro Ethernet is a service to connect physically dispersed Ethernet LANs that belong to the same corporation. In the metro environment, you can use Ethernet in the following situations:

- To provide transparent LAN services (TLS) for customers' LAN connectivity to their remote sites
- To provide Layer 2 Ethernet VPNs that virtually unite geographically dispersed servers to appear as though they are located on the same LAN

Enterprises might be interested in Ethernet VPNs because of their ability to offer faster transport between campuses or a main campus and remote branches. This idea of joining geographically dispersed networks with Ethernet becomes possible because Ethernet VPNs are mimicking the way enterprises today might use their ATM, private line, or Frame Relay service. When Ethernet VPNs are involved, the enterprise is serviced with a high-bandwidth network connection at both ends of the network.

Metro Ethernet deployments offer enterprise customers an effective metropolitan-area network (MAN) alternative to current WAN networks. The next sections detail some of the requirements that enterprises and providers impose on Metro Ethernet.

Metro Ethernet Service Architectures

In the most general sense of the term, Metro Ethernet services are network services in which connectivity is provided to the customers via a standard Ethernet User-to-Network Interface (UNI). This is a broad definition because multiple technologies can provide Metro Ethernet services from optical cross-connects (OXC), pure Layer 2 networks, or packet-switched networks.

Because of the broad definition, it is critical to categorize these services. Several taxonomies for Metro Ethernet produce an eclectic portfolio of services. You can categorize Metro Ethernet services as follows:

- Based on connectivity type:

  Point-to-point Similar to a permanent virtual circuit (PVC).

  Multipoint Similar to a cloud.

- Based on service types:

  Wire services A port does not have multiplexing. A customer port connects to a single remote customer port. This is similar to a leased line.

  Relay services Service multiplexing is available based on VLAN, such that different customer VLANs within a customer port can connect to different sites. This is similar to a Frame Relay port.

Combining these two categorizations, you have the first four sets of Metro Ethernet services:

- **Ethernet Wire Service (EWS)** A nonmultiplexed point-to-point service.
- **Ethernet Relay Service (ERS)** A VLAN-multiplexed point-to-point service.
- **Ethernet Multipoint Service (EMS)** A nonmultiplexed point-to-cloud service. An example is Virtual Private LAN Service (VPLS), which is covered in Chapter 15, "Virtual Private LAN Service."
- **Ethernet Relay Multipoint Service (ERMS)** A VLAN-multiplexed point-to-cloud service. The service provider cloud has VLAN mapping. An example is VPLS, which is covered in Chapter 15.

Other Metro Ethernet services are as follows:

- **Ethernet Private Line (EPL)** Similar to EWS service, except it is provided at Layer 1 by OXCs.
- **Layer 2 VPN access** Layer 2 access to Multiprotocol Label Switching (MPLS) VPNs.
- **ATM to Ethernet over MPLS or Ethernet over L2TPv3 Interworking** This topic is covered in Chapter 14, "Layer 2 Interworking and Local Switching."
- **Frame Relay to Ethernet over MPLS or Ethernet over L2TPv3Interworking** This topic is covered in Chapter 14.

Table 4-2 summarizes the characteristics of different Metro Ethernet services.

Table 4-2. *Metro Ethernet Services*

Metro Ethernet Service	Architecture	Service Definition	Connectivity
EPL	Layer 1	Transparent (nonmultiplexed)	Point-to-point
EWS	VPWS[1]	Transparent (nonmultiplexed)	Point-to-point
ERS	VPWS	Multiplexed	Point-to-point
EMS	VPLS	Transparent (nonmultiplexed)	Multipoint-to-multipoint
ERMS	VPLS	Multiplexed	Multipoint-to-multipoint
ATM/Frame Relay Ethernet Interworking	VPWS	Multiplexed	Point-to-multipoint

[1] VPWS = Virtual Private Wire Service

All these categories vary in implementation, but some generalities exist. Table 4-3 shows the interface type of the PE devices both toward the customer and toward the core. The PE devices can be logical devices that are distributed among different physical PEs. In such a case, the user-facing PE is called U-PE, and the network-facing PE is called N-PE.

Table 4-3. *Metro Ethernet Services*

Metro Ethernet Service	U-PE <-> Customer	N-PE <-> Service Provider
EPL	QinQ[1]	WDM[2] wavelength SONET/SDH[3] circuits
EWS	QinQ	EoMPLS[4]
ERS	802.1q Trunk	EoMPLS
EMS	QinQ	Ethernet or EoMPLS
ERMS	802.1q Trunk	EoMPLS
ATM/Frame Relay Ethernet Interworking	Frame Relay or ATM	EoMPLS

[1] QinQ = Stands for 802.1q in 802.1q and is also referred to as 802.1q tunneling

[2] WDM = wavelength division multiplexing

[3] SDH = Synchronous Digital Hierarchy

[4] EoMPLS = Ethernet over MPLS

Note that the transparent services use QinQ facing the customer to provide "VLAN bundling" in a port-based service and achieve transparency for customer bridge protocol data units (BPDUs). On the other hand, the relay services use 802.1q trunking facing the customer in a VLAN-based service to provide the VLAN-multiplexed UNI; thus, they are opaque to customer BPDUs.

Understanding Spanning Tree Protocol

The methods that enterprises use to deploy LAN networks have changed considerably over the years. At the onset of switching technology in the mid-1990s, enterprises used the so-called "VLANs everywhere" model in their network design. In this model, an enterprise would separate its users into workgroups, with each workgroup assigned a VLAN of its own. For instance, networks would have an engineering VLAN, a marketing VLAN, a production VLAN, and so on. Although clients in each VLAN could be located anywhere within the enterprise, these VLANs had to span and be trunked across the entire network. Trunking enables traffic from several VLANs to be carried over a point-to-point link between two devices. Today, the use of VLANs is more restricted, and the "VLANs everywhere" model is no longer preferred. Instead of campus-wide VLANs, Layer 3 switches are preferred.

To facilitate this early design model, the Spanning Tree Protocol (STP) that is specified in the IEEE 802.1d standard was used. STP and the spanning-tree algorithm protect Ethernet networks from broadcast storms by detecting loops. The forwarding nature of Ethernet for broadcasts, multicast, and unknown unicasts can create loops. Broadcast storms are caused by loops. The scenario can become complex when using VLANs, so the role of STP is more critical with VLANs. Loops occur when redundant paths are implemented on the network. Redundant paths serve as a backup in case of link failure, which means they are important for the overall health of the network. Unfortunately, redundant links cause packets to loop between the switches that these links interconnect. To solve the loop problem, while preserving redundancy, you can implement STP. The next sections examine spanning-tree operation and implementation drawbacks a bit more closely.

Spanning-Tree Operation Overview

When you use STP on all switches in an internetwork, it puts one of the redundant links between switches in a blocked state, whereas the other stays in the forwarding state. As a result, only one link is operational at any given time. After the forwarding link experiences a failure, STP recalculates a new path, and a formerly blocked link takes over.

In a spanning-tree environment, the switches elect a root bridge. All subsequent decisions on blocking or forwarding states of ports are made from the perspective of this root bridge. The root selection is made in the following manner:

1. When the switch first comes up, it sends out a BPDU to its directly connected neighbor switch.

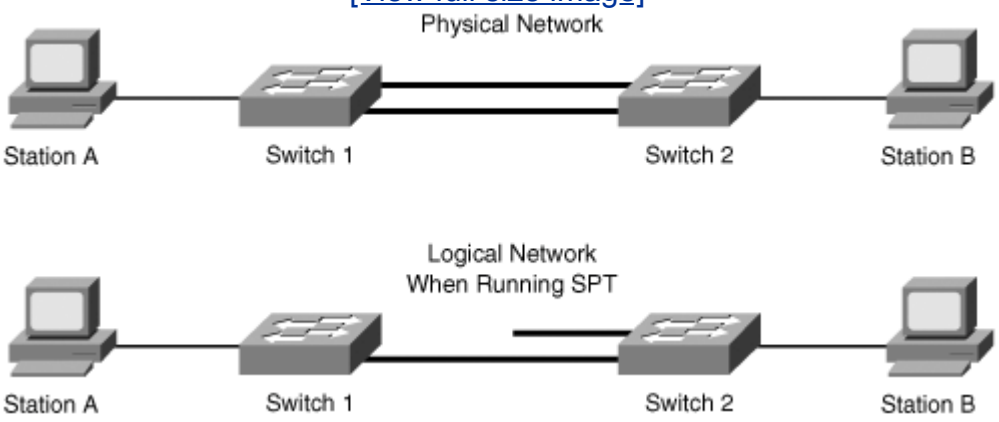
2. Switches link BPDUs and create their own BPDUs to propagate STP information. As the spanning-tree information is propagated through the network, each switch compares the BPDU from its neighbors to its own. Election of a root switch is the result of this comparison. The lower the 2-byte priority of a switch, the higher the chances of its selection as root. Therefore, the switch that has the lowest priority becomes the root bridge. The priority is

a configurable value. When the priorities are the same, the 6-byte or Root ID is used as a tiebreaker.

Consider the sample network in Figure 4-3, in which two redundant links connect switches 1 and 2. These redundant links create the possibility of bridging loops, because broadcast or multicast packets that are transmitted from Station A and destined for Station B ping-pong back and forth between both switches. When STP is enabled in both switches, one of the parallel links is blocked, eliminating the possibility of bridging loops. The logical network with the link blocked is shown in Figure 4-3.

Figure 4-3. *Spanning-Tree Scenario*

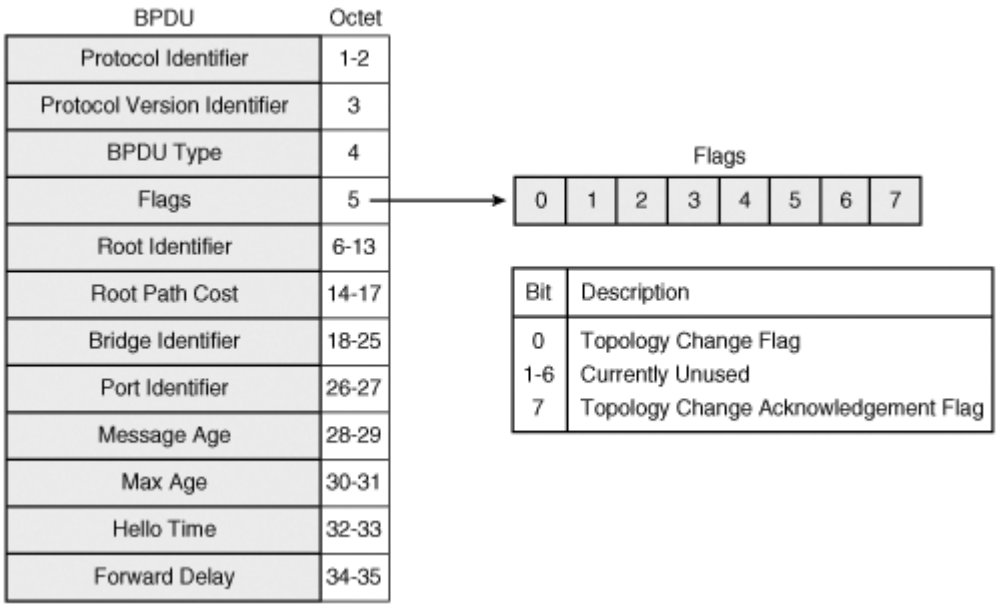
[View full size image]



A BPDU is defined in the IEEE 802.1d MAC Bridge Management protocol, which is the standard implementation of STP. The IEEE 802.1d flag or bit field consists of 8 bits. It is illustrated in Figure 4-4, along with the complete BPDU.

Figure 4-4. *802.1d BPDU Format*

[View full size image]



The BPDU fields are as follows:

- **Protocol Identifier** Identifies the spanning-tree algorithm and protocol, such as STP (0x0000).
- **Protocol Version Identifier** Identifies the protocol version, such as Spanning Tree, Rapid Spanning Tree, and Multiple Spanning Tree.
- **BPDU Type** Identifies the BPDU type, such as Configuration (0x00), Topology Change Notification (0x80), and Rapid/Multiple Spanning Tree (0x02).
- **Flags** Bit flags include the following:

  Bit 7 Topology Change Acknowledgement flag.

  Bit 0 Topology Change flag.

- **Root Path Cost** Multiple of the root cost.
- **Bridge Identifier** MAC address of the bridge.
- **Port Identifier** Port priority (smaller number denotes higher priority).
- **Message Age, Max Age, Hello Time, Forward Delay** These four timer values have times ranging from 0 to 256 seconds.

The root switch dictates that the root bridge will have all its ports in the forwarding mode. On each LAN segment, the switches elect the designated switch that is used for transporting data from that segment to the root switch. On the designated switch, the port that connects to the LAN segment that the switch serves is put in a forwarding mode. You must block all other switch ports across the network. The blocking of ports concerns only a switch-to-switch connection. Ports that are connected to workstations are not involved in a spanning-tree process and are left in a forwarding mode.

Drawbacks of a Spanning-Tree Implementation in Today's Networks

Although serving an important purpose, STP has inherent drawbacks. STP is CPU intensive and vulnerable to network volatility. When a portion of the VLAN experiences a link failure, all switches that carry that VLAN have to learn, process, and forward BPDUs. This problem escalates if several VLANs are involved, in which case the processor can overload. If the processor overloads, it might start dropping BPDUs, thereby weakening and destabilizing the network. STP also has serious convergence issues when implemented in a VLAN-concentrated network with high redundancy. This results in poor scalability of STP networks. STP is triggered when some failure prevents the neighbor switch from receiving the periodic BPDUs sent out by STP forwarding ports. The result is that STP must recalculate and redetermine the STP topology.

To avoid STP issues, one of the more successful solutions that many enterprise customers implement is migration to Layer 3 switching services, made possible with high-performance Layer 3 switches (such as the Cisco Catalyst 6500). Campus-wide VLANs then became obsolete.

With Layer 3 switching, you can forward network traffic based on the Layer 3 address (such as the IP address). In this method, VLANs segment an IP subnet at Layer 2 by mapping a subnet to a separate VLAN. User VLANs then terminate at a Layer 3 switch, and the LAN essentially functions as a routed network.

Similarly, most service providers try to avoid using STP in their core infrastructure. They do this by utilizing technologies such as AToM to route packets that contain Layer 2 frames in the service provider core instead of switching them at Layer 2. This means terminating STP locally on the edge-facing CE. The service provider core is then free of STP, utilizing a full mesh of pseudowires with split-horizon forwarding to prevent loops. Each PE sees all the other PEs as in a point-to-multipoint view. Even in these scenarios in which you avoid STP in the core by using a full mesh of EoMPLS pseudowires and split horizon, you sometimes need STP in an aggregation layer in the distributed PE between U-PE and N-PE.

Pure Layer 2 Implementation

It is important to understand the difference in challenges posed by Metro Ethernet versus traditional Layer 2 network technologies from the service provider's standpoint. Ethernet has little intelligence. If the source and destination are known, the packet is forwarded. If the destination is unknown, the packet is flooded. If the source was previously unknown, the address is learned and the packet is forwarded. The rules look simple, but looks are deceiving. For instance, if a loop occurs, a packet can keep traversing the network forever, which can ultimately bring down the network.

As mentioned in the previous section, STP (IEEE 802.1d) protects the network against loops. Although STP is a CPU-intensive protocol that takes, on average, 30 to 50 seconds to reconverge, many service providers accustomed to Frame Relay's convergence of up to 60 seconds will find it acceptable. Moreover, Cisco has developed several enhancements to STP, and the new Rapid Spanning Tree Protocol that is specified in IEEE 802.1w can further minimize the convergence period.

Metro-wide VLANs with STP require a careful implementation strategy. Ideally, this implementation involves a deterministic topology with a small amount of redundant connections and VLANs spanning as few switches as possible. Good planning, however, can enable a Layer 2 Ethernet transport network for the MAN to offer reliable, high-bandwidth services to the enterprise.

In the pure Layer 2 model, which is a switched (not routed) core, described in this section, the enterprise network forwards untagged frames to the service provider.

> **Note**
>
> The term untagged means without an 802.1q header and refers to the Ethernet II frame you saw in Figure 4-1 or the 802.3 frame you saw in Figure 4-2. 802.1q encapsulation is discussed in the "802.1q Tunneling" section later in this chapter.

In this scenario, the enterprise is not using STP through the service provider's core. The service provider maps the enterprise's subnet to a VLAN. This VLAN is trunked throughout the entire service provider network and ends at the destination enterprise. As far as the enterprise is concerned, the routers appear directly connected, and the data transport is completely transparent. The upcoming "802.1q Tunneling" section of this chapter discusses this topic in more detail.

Utilizing pure Layer 2 solutions for Metro Ethernet is relatively simple and inexpensive. Complications arise, however, when you deal with the inherent Layer 2 scalability issues. Service providers cannot afford to underestimate cautious planning and deployment when it comes to spanning tree and VLAN distribution issues. Most likely, service providers will want to implement redundancy. Because

spanning tree is required to protect against loops in the network, an increase in the number of customer VLANs and locations can spin out of control and result in network failure. Furthermore, it can complicate troubleshooting of a problem.

Cisco has developed some tools to aid administrators with the Layer 2 management to resolve some of the Layer 2 issues with VLANs, STP, and scalability. These tools include the following:

- **VLAN Trunking Protocol (VTP)** VPT is a Layer 2 messaging protocol that manages the addition, deletion, and renaming of VLANs on a networkwide basis. It voids the necessity of having to do these tasks manually.

- **Dynamic Trunking Protocol (DTP)** DTP gives a switch port the ability to automatically negotiate the trunking method with the other network device.

- **STP Root Guard** STP root guard forces a Layer 2 LAN interface to become a designated port. If any device that is accessible through the interface becomes the root bridge, STP Root Guard puts the interface into the root-inconsistent (blocked) state.

- **BPDU Guard** BPDU Guard is an enhancement to STP that capitalizes on the predictability of STP in certain network environments and disables BPDU forwarding on designated ports.

In addition, Cisco uses the highest performance processors available to handle the STP processing. To avoid the "VLANs everywhere" model, the service provider might offer the enterprise multiple VLANs, one to each site.

The next section covers another Layer 2 technologyQinQthat can be used as a transport mechanism in Metro Ethernet networks.

802.1q Tunneling

One of the enterprise's business requirements can entail sending multiple VLANs across the service provider's Metro Ethernet network. The enterprise can accomplish this via 802.1q tunneling, also known as QinQ. This chapter uses both names interchangeably.

802.1q tunneling is a tunneling mechanism that service providers can use to provide secure Ethernet VPN services to their customers. Ethernet VPNs using QinQ are possible because of the two-level VLAN tag scheme that QinQ uses. The outer VLAN tag is referred to as the service provider VLAN and uniquely identifies a given customer within the network of the service provider. The inner VLAN tag is referred to as the customer VLAN tag because the customer assigns it. QinQ's use of double VLAN tags is similar to the label stack used in MPLS to enable Layer 3 VPNs and Layer 2 VPNs. It is also possible for multiple customer VLANs to be tagged using the same outer or service provider VLAN tag, thereby trunking multiple VLANs among customer sites. Note that by using two VLAN tagsouter and inner VLANyou achieve a demarcation point between the domain of the customer and the domain of the service provider. The service provider can use any VLAN scheme it decides upon to identify a given customer within his provider network. Similarly, the enterprise customer can independently decide on a VLAN scheme for the VLANs that traverse the service provider network without consulting the service provider.

In summary, 802.1q tunneling allows service providers to use a single VLAN to support multiple VLANs of customers, while preserving customer VLAN IDs and keeping traffic in different customer VLANs segregated. At the same time, it significantly reduces the number of VLANs required to support the VPNs. QinQ encapsulates the VLANs of the enterprise customers into a VLAN of the service provider.

QinQ accomplishes the following:

- Enterprise customers receive transparent Layer 2 links between sites within a metro area, such as a link from a branch office to a main campus.
- Service providers can separate or group traffic on a per-customer basis using outer VLAN tags as it traverses the common infrastructure so that the same infrastructure can provide service to multiple customers.
- The VLAN ID of the enterprise and the VLAN ID of the service provider do not have to match.
- The customers can treat the switching infrastructure in a remote site as if it were part of the local site. They can use the same VLAN space and run protocols such as STP across the provider infrastructure through 802.1q.

The QinQ model allows the customer edge switch on each side of the tunnel to view the service provider infrastructure as nothing more than a transparent bridge. The

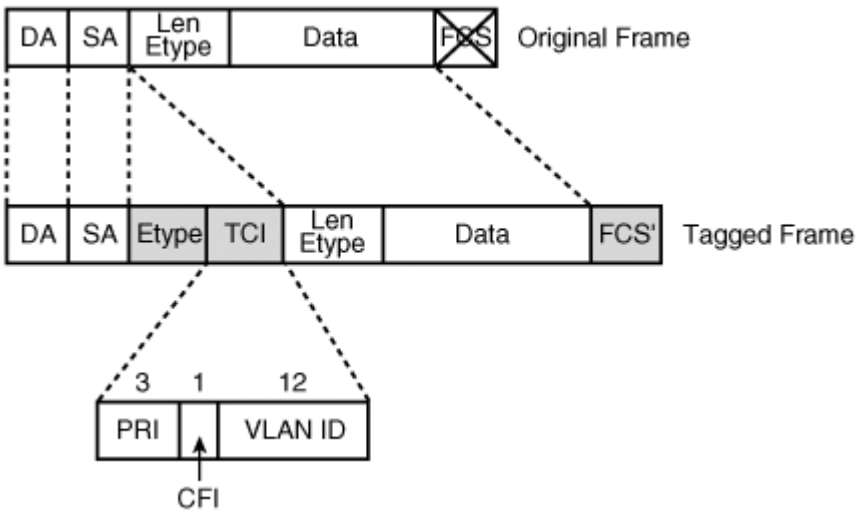
following sections talk about the 802.1q tunneling underlying processes.

802.1. q and 802.1p Tagging

802.1q tagging refers to modifications made to the original Ethernet frame described earlier in the chapter. In 802.1q tagging, additional bytes are inserted into the Ethernet frame.

Altogether, the Ethernet frame is inserted with four additional bytes that turn it into the 802.1q frame, and FCS is recalculated. The new fields are illustrated in Figure 4-5.

Figure 4-5. *802.1q Frame*



Following are the new fields inserted by "tagging":

- **Ethertype** 2 bytes that identify an 802.1q frame and equal 0x8100. Ethertype is also called Tag Protocol Identifier (TPID).
- **TCI** 2 bytes of Tag Control Information that in turn contain the following:

  Priority 3 bits that define the 802.1p user priority. They are also referred to as the class of service (CoS) bits.

  CFI 1-bit Canonical Format Identifier (CFI) for compatibility issues between Ethernet-type networks and Token Ringtype networks.

  VLAN ID A 12-bit field that identifies the VLAN.

IEEE 802.1p is a supplement to the IEEE 802.1d specification. It is intended for QoS implementation on LANs, analogous to the three precedence bits in IP. 802.1p describes mechanisms in switches for handling the time-sensitive traffic and reducing the impact of high-bandwidth traffic within a LAN.

The IEEE 802.1p is needed because Ethernet, unlike Token Ring, does not inherently provide support for priority levels in frames. Based on the MAC frame information, 802.1p provides an in-band QoS signaling method for traffic classification. 802.1p also provides an optional mechanism in switches for supporting end-to-end time-critical frame delivery.

Under IEEE 802.1p, eight CoSs are supported. The higher the value is, the higher the priority of the frame. Zero, the lowest, stands for routine service with no priority specified. You can configure switches in a LAN and different ports of a switch for several different priority levels.

Sometimes high-speed LANs do not require QoS capabilities. However, when backbone networks are involved, QoS methods become necessary on service provider and enterprise networks. You will learn more of the QoS in Layer 2 VPN implementations in Chapters 9, "Advanced AToM Case Studies," and 13, "Advanced L2TPv3 Case Studies." Now it is time to examine the innerworkings of 802.1q tunneling.

Understanding How 802.1q Tunneling Works

A tunnel port is a port that is configured to support 802.1q tunneling. Each customer comes in on a dedicated customer-facing port on the service provider switch where a VLAN that is dedicated to tunneling is assigned. The service provider assigns each customer an outer VLAN tag or a service provider VLAN tag that uniquely identifies him within the network. The service provider VLAN also keeps the customer traffic isolated from other customer traffic that is traversing the same service provider network. That service provider VLAN supports all the VLANs of the customer.

802.1q tunneling refers to multiple tagging of dot1Q frames as they enter a service provider switch from a client switch. QinQ can tag or untag any frames that it receives from the customer tag. 802.1q also has native VLAN frames that are untagged. The service provider switch adds the outer VLAN tag.

Tagged and untagged customer traffic comes from a port on a customer device and enters the service-provider edge switch through a tunnel port. Each customer edge port that is connected to an 802.1q tunnel port is typically configured as a trunk port. The customer trunk port is unaware of the provider 802.1q tunnel and can communicate with all of its other trunk ports that are connected to the metro network of the provider as if they were directly connected. This makes the process transparent to the switching network of the enterprise.

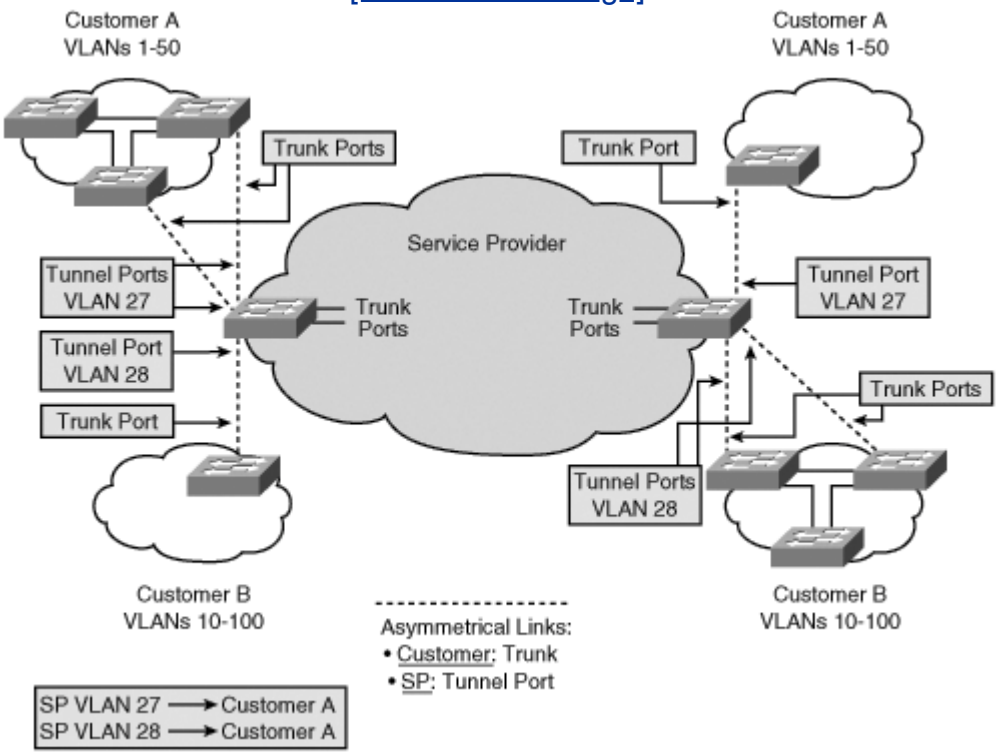
A hub customer edge might have connectivity to two remote spoke sites and have only half of the VLANs from the hub site go to one site and the remaining to the second remote site. This is possible using two service provider VLANs for this

enterprise customer when certain sites need to see only some and not all of the VLAN traffic from the hub site.

The link between the 802.1q trunk port on a customer device and the tunnel port is known as an asymmetrical link. One end is designated as an 802.1q trunk port, whereas the other end is configured as a tunnel port. The tunnel port is configured with an access VLAN ID that is unique to a customer, as shown in Figure 4-6.

Figure 4-6. ***Port Designation in a Service Provider Network***

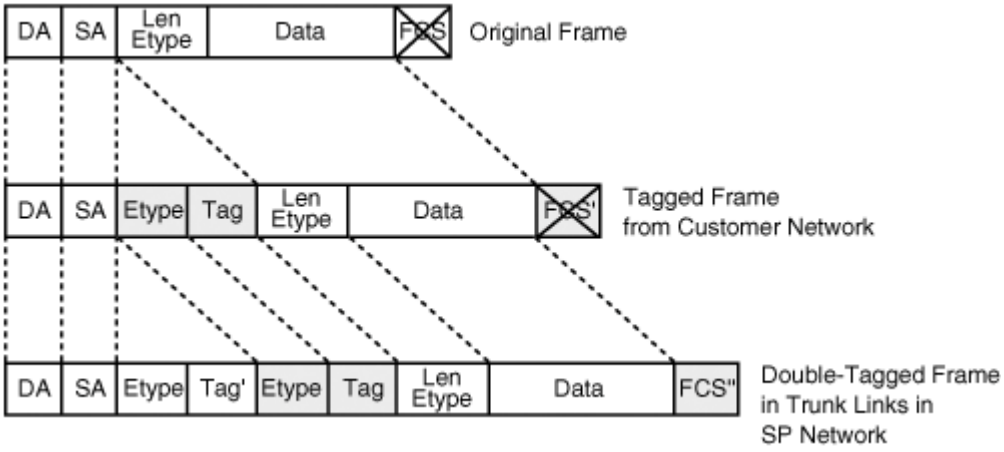
[View full size image]



When a tunnel port receives tagged customer traffic from an 802.1q trunk port, it does not strip the existing VLAN tag (imposed by the customer switch) from the frame header. Instead, it leaves the 802.1q tag intact and adds a 2-byte Ethertype field (0x8100) followed by a 2-byte field containing the priority (CoS) and the VLAN ID. The tunnel port treats the new tagged frame as a Layer 2 frame where the Ethertype is not known to the service provider because it is the bottom of the tag stack. It uses the outer or top VLAN tag for subsequent switching inside the service provider infrastructure. The tagging process is demonstrated in Figure 4-7. First, you see an original untagged frame (described in Figures 4-1 and 4-2), followed by a customer VLAN tagged frame. Finally, you see the addition of a provider's 802.1q tag.

Figure 4-7. *802.1q Tag Addition*

[View full size image]



The tunnel port then puts the received customer traffic into the service provider VLAN that is assigned to the tunnel port. Subsequently, that VLAN transports the customer traffic to the next tunnel device. The customer VLAN (customer 802.1q tagged frames) is tunneled traffic that is carried in a service provider VLAN 802.1q tunnel. The ports in the tunnel are the ingress or egress points of the tunnel. The tunnel ingress and egress ports are not necessarily located on the same device. To reach a remote site in the customer network in the egress tunnel port, the tunnel can traverse multiple network links and multiple network devices (as many as required for a particular customer support).

When the frame reaches the other end of the provider network, an egress tunnel port at the edge switch strips the outermost tag before sending it to the customer network. Then the switch transmits the traffic out of the egress tunnel port with the original 802.1q tag of the enterprise to an 802.1q trunk port on a customer device. The 802.1q trunk port on the customer device strips the 802.1q tag and removes the traffic from the tunnel.

Note

An 802.1q trunk has an untagged native VLAN. When the port is in 802.1q trunk mode, the native VLAN is used for untagged traffic. Therefore, the native VLAN and all VLANs need to stay the same on both sides of the trunk.

802.1q Tunneling Guidelines and Restrictions

When you are configuring 802.1q tunneling, keep the following in mind:

- Because 802.1q tunneled packets are processed as non-IP packets, Layer 3 packet classification does not apply. You can consider only Layer 2 match criteria (for instance, VLANs, source and destination MAC addresses, and 802.1p CoS bits) when filtering tunnel traffic. (Untagged packets that are sent to a tunnel do not have to adhere to this restriction inside the provider network.) Therefore, QoS for tunnel traffic can be provided only for Layer 2.

- Dot1Q tunnel ports are essentially access ports that support double-tagging of incoming packets. Therefore, as far as Dynamic Trunking Protocol (DTP) is concerned, the port mode of an 802.1q tunnel port is not negotiable. Hence, DTP does not work with asymmetrical links because only one port on the link is configured as a trunk.

- VTP does not work on an asymmetrical link or through a tunnel. To enable VTP between two customer ports across a tunnel, configure the protocol tunneling on the tunnel ports.

- An asymmetrical link supports the following Layer 2 protocols:

  UniDirectional Link Detection (UDLD) Allows devices to detect when a unidirectional link exists. Because unidirectional links can cause spanning-tree loops, UDLD shuts down a link when it detects unidirectional traffic.

  Port aggregation protocol (PAgP) Used in the automatic creation of Fast EtherChannel links.

  Cisco Discovery Protocol (CDP) Disabled by default on a QinQ tunnel port to prevent the service provider switch and the enterprise switch from seeing each other. To use CDP between customer edge devices across the provider tunnel, configure protocol tunneling for CDP on the tunnel ports.

- As mentioned, traffic in the native VLAN is untagged and cannot be tunneled correctly. Therefore, make sure that the native VLAN of the 802.1q trunk port in an asymmetrical link does not carry traffic. Tag egress traffic in the native VLAN with 802.1q tags.

- You can tunnel jumbo frames (that is, Ethernet frames in excess of the Ethernet frame MTU and up to 9216 bytes in length) in the core. However, you need to support them in a tunneled network (both in 802.1q tunnel ports and trunk ports in the provider network) for tunneling to work correctly with all packet sizes. Also, the total length of the frame plus 802.1q tag cannot exceed the maximum frame size.

- If the VLAN of the tunnel port does not match the native VLAN of the 802.1q trunk, CDP reports a native VLAN mismatch. Because the 802.1q tunnel feature does not require that the VLANs match, you can ignore these messages in this case.

- Enterprise and service provider switches should not participate in each other's STPs. To ensure this does not happen, STP BPDU filtering is enabled by default on 802.1q tunnel ports and access ports on provider switches. This makes BPDUs from the enterprise network invisible to the provider and vice versa. On the flip side, self-loops from back-to-back connection of the tunnel ports go undetected. To resolve this, all those ports on provider edge switches that interface with a customer should have the Root Guard feature enabled. This way, a customer switch does not mistakenly become an STP root. When you configure protocol tunneling on the customer edge ports, customer switches on either end of the tunnel can see STP BPDUs from other switches of that customer.

- The maximum number of VLANs that the 802.1q standard allows in a Layer 2 domain is 4096, because the VLAN ID field is 12 bits and therefore permits 4096 variations (2^{12} = 4096). Thus, the entire pure Layer 2 solution is bound to that number. It might or might not become a significant hindrance depending on the requirements placed on a particular service provider.

Summary

Pure Layer 2 and 802.1q tunneling network architectures offer practical solutions for enterprises that are looking to receive and service providers that are looking to offer Metro Ethernet connectivity. Examine carefully the inherent limitations of pure Layer 2 implementation if you are considering this type of service and creating its design. Even though enterprises have moved on from pure Layer 2 networks to those that are Layer 3 switched, Layer 2 still holds a great value for an Ethernet solution for a service provider.

Many enterprises and service providers are considering whether QinQ or 802.1q tunneling is right for them. This technique solves the transparency problems for enterprises and enables service providers to offer the desired Layer 2 services at the same time. However, because of some of the issues described in this chapter, QinQ might not work for everyone.

Chapter 5. WAN Data-Link Protocols

This chapter covers the following topics:

- Introducing HDLC encapsulation
- Introducing PPP encapsulation
- Understanding Frame Relay
- Understanding ATM

Before proceeding with a detailed examination of Layer 2 Tunneling Protocol Version 3 (L2TPv3) and Any Transport over MPLS (AToM), it is critical to understand the protocols that these tunneling mechanisms transport. The four WAN data link layer protocols covered include High-Level Data Link Control (HDLC), PPP, Frame Relay, and ATM.

This chapter introduces specific components of these Layer 2 protocols that are relevant to the L2TPv3 and AToM pseudowire protocols explored later in this book. For each of these protocols, the chapter examines the encapsulation format, any relevant control/management protocols that the pseudowire protocols might have to emulate, and any inherent traffic-management characteristics where applicable.

Introducing HDLC Encapsulation

In 1974, IBM developed one of the first bit-oriented synchronous protocols, known as Synchronous Data Link Control (SDLC). After IBM submitted the protocol to the ISO for international standardization, the ISO adapted the protocol and renamed it HDLC. HDLC is covered under the ISO standards ISO 3309 and ISO 4335. During the same period, the Consultative Committee for Telegraph and Telephone (CCITT), now known as International Telecommunication Union (ITU-T), adopted HDLC for the X.25 Link Access Procedure when developing standards for X.25 Data Transmission.

The frame formats between the ISO and ITU-T versions of HDLC share many similarities and have also served as the basis for future protocols such as Frame Relay and PPP.

HDLC was defined to operate in the following three modes:

- **Normal Response Mode (NRM)** Employs a master/slave relationship, whereby secondary (slave) station(s) can transmit only when the primary (master) station permits.
- **Asynchronous Response Mode (ARM)** Is similar to NRM mode, but it differs in that the secondary station(s) does not have to wait for permission from the primary station to send data. The primary station is responsible for link initialization, link teardown, and error recovery.
- **Asynchronous Balanced Mode (ABM)** All stations have equal status and do not require instruction from their peers to perform a task.

In addition to the ITU-T and ISO standards, Cisco modified the HDLC encapsulation and adapted it for use as a serial line encapsulation protocol. The Cisco implementation is used in full duplex point-to-point links and operates in ABM mode. Unlike the standardized HDLC implementation, the nonstandard Cisco version does not perform windowing or retransmission and uses an Ethertype field to indicate the Layer 3 payload.

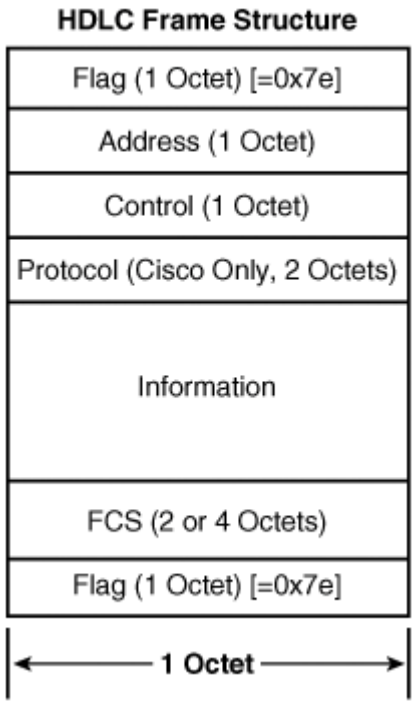
Note

This discussion examines the HDLC frame format as the ISO 3309 standard defines it. Where applicable, any differences between the ISO standard and Cisco implementation are highlighted.

As Chapter 8, "WAN Protocols over MPLS Case Studies," and Chapter 12, "WAN Protocols over L2TPv3 Case Studies" highlight, both L2TPv3 and AToM interaction

with HDLC are limited to HDLC frame transport when dealing with Layer 2 pseudowire emulation. As such, this HDLC section focuses on the HDLC frame format. Figure 5-1 illustrates the HDLC frame structure.

Figure 5-1. *HDLC Frame Format*



Each field is described as follows:

- **Flag** The beginning and end of every HDLC frame must contain a 1-byte Flag Sequence field to delimit the frame. The flag sequence used is 01111110 (0x7E). Because these flags must be unique, it is critical that a 0x7E does not show up in the Data field. To avoid this scenario on synchronous links, HDLC uses a method known as bit stuffing, as defined in American National Standards Institute (ANSI) T1.618, to differentiate this sequence from a flag delimiter. If five consecutive 1s are detected, the bit stuffing technique inserts a 0 bit to avoid having six consecutive 1s in a row. Upon inspection of the frame, the receiving end removes the 0 bit when it detects five consecutive 1s to restore the original sequence. Two alternate flag fields include 0xFF to indicate an IDLE flag and 0x7F for an Abort flag.

  ### Note

  On asynchronous links, HDLC uses *byte stuffing* (sometimes referred to as *character stuffing* or *escaping*) to transform illegal byte values into a

set of legal characters. The receiving end reverses this mechanism to obtain the original values.

- **Address** The Address field uniquely identifies each of the stations on the HDLC link. Depending on the operational mode (NRM, ARM, or ABM), the Address field could contain the primary or secondary station's address when sending command and response messages. ISO standard 3309 can be referenced for more detail on the use of the Address field.

  In Cisco HDLC encapsulation, instead of uniquely identifying a station, the Address field indicates the frame type.

  Valid values include these:

  0x0F for unicast frame

  0x80 for broadcast frame

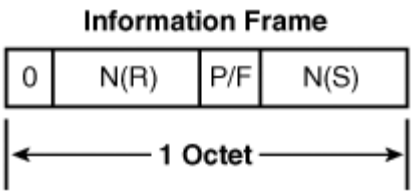
  0x20 for compressed frame

  0x40 for padded frame

- **Control** The Control field contains sequence number information and command or response messages depending on the frame type. Three control frame types are defined in HDLC as follows:

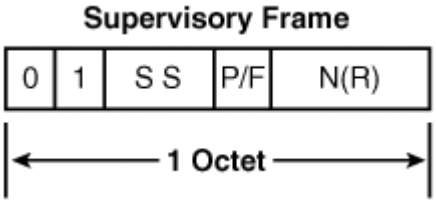
  Information frame Figure 5-2 lays out the Control field octet for an information frame. The first bit of the control octet set to 0 indicates that the frame is an information frame. The N(S) and N(R) are 3-bit fields containing the transmitter's send and receive sequence numbers respectively. The P/F bit indicates whether this is a command request or response.

Figure 5-2. *Control Field FormatInformation Frame*



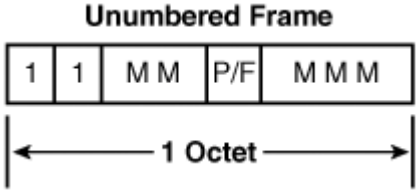
Supervisory frame Figure 5-3 lays out the Control field octet for a supervisory frame. The supervisory frame has a similar format to the information frame except that the first two bits are set to 0 and 1 to distinguish this frame as a supervisory frame, and bits 3 and 4 are supervisory function bits. The remaining fields have the same meaning as in the information frame.

Figure 5-3. *Control Field FormatSupervisory Frame*



Unnumbered frame Figure 5-4 lays out the Control field octet for an unnumbered frame. The first two bits are set to 1 to distinguish this frame as an unnumbered frame. The M bits convey different commands/responses. For information regarding the different M-bit encoding for various command/response message types, refer to the ISO 4335 standard. The P/F bit functions the same way as in other HDLC Control fields.

Figure 5-4. *Control Field FormatUnnumbered Frame*



Cisco HDLC encapsulation does not use the control octet. It sets it to 0.

Protocol The Protocol field is specific to Cisco HDLC encapsulation. The value of the Protocol field identifies the upper-layer protocol stored in the succeeding Information field. Cisco adopted standard Ethertype values to identify most protocols (see Table 5-1), but it also developed additional protocol values for Layer 3 protocols that normally do not exist on Ethernet (see Table 5-2).

Table 5-1. ***Ethernet Standard Values for Cisco HDLC Protocol Field***

Protocol Type	**Protocol Field Value**
PARC Universal Protocol (PUP)	0x0200
Xerox Network Systems (XNS)	0x0600
IP	0x0800
Chaos	0x0804
RFC 826 Address Resolution Protocol (ARP)	0x0806
Virtual Integrated Network Service (VINES) IP	0x0BAD
VINES ECHO	0x0BAF
DECnet Phase IV	0x6003
Apollo Domain	0x8019
Cisco SLARP	0x8035
Digital Equipment Corporation (DEC) Bridge Spanning Tree Protocol	0x8038
Apple Ethertalk	0x809b
AppleTalk ARP	0x80f3
Novell Internetwork Packet Exchange (IPX)	0x8137
Multiprotocol Label Switching (MPLS) Unicast	0x8847

Table 5-2. ***Cisco Invented Values for Cisco HDLC Protocol Field***

Protocol Type	**Protocol Field Value**
Frame Relay ARP	0x0808
IEEE Bridge Spanning Protocol	0x4242
Bridged Ethernet/802.3	0x6558
ISO Connectionless Network Protocol (CLNP)/International Organization for Standardization (ISO) End System-to-Intermediate System (ES-IS) destination service access point (DSAP)/SSAP	0xFEFE
Novell IPX, Standard Form	0x1A58
ES-IS	0xEFEF
RSRB Raw	0x1996
STUN Serial Tunnel	0x1997
Compressed TCP	0x1999

- **Information** The Information field contains data that is to be transmitted and only appears when the Control field is set to be an information frame. The length of this field is variable and depends on the Layer 3 protocol to be carried.

- **Frame check sequence (FCS)** The FCS value contains a 2- or 4-octet cyclic redundancy check (CRC). If the receiver's CRC calculation differs from the value in the frame, the frame is flagged in error.

Note

The Cisco version of HDLC can optionally utilize a simple keepalive mechanism that tracks the sequence numbers of messages that the two endpoints generate locally. Although this control/management protocol exists, neither Layer 2 Tunnel Protocol Version 3 (L2TPv3) nor Any Transport over MPLS (AToM) interact with this keepalive mechanism; instead, they carry these messages across transparently.

Introducing PPP Encapsulation

In 1993, the Internet Engineering Task Force (IETF) defined the requirements for a serial line data encapsulation protocol in RFC 1547. The protocol that eventually grew out of this RFC was PPP. Following are some of the more important goals that RFC 1547 outlined:

- **Protocol multiplexing** PPP was required to support multiple higher level protocols. This, in turn, required PPP to support a Protocol field to allow for multiplexing several network layer protocols over the same point-to-point link.
- **Error detection** PPP must be able to detect errors in the encapsulated frames.
- **Network layer address negotiation** PPP must support dynamic learning and negotiation of network layer addresses.
- **Transparency** PPP cannot restrict the network layer protocols to avoid certain characters or bit patterns. Instead, PPP must be fully transparent to higher layer protocols. Furthermore, PPP must handle any character or bit pattern restrictions through means such as bit or escaping.

The finalized RFC also described features that PPP explicitly did not require, such as these:

- **Error correction** PPP is not expected to perform error correction, only error detection. Instead, the error correction responsibility is left to upper transport layer protocols.
- **Flow control** PPP is not required to support a flow control mechanism; instead, it is expected to receive packets at the full rate possible depending on the physical layer protocol. The transport layer is left with the responsibility of end-to-end flow control.
- **Sequencing** Although some network layer protocols require sequenced frame delivery, PPP is not required to deliver frames in the same order that these frames were sent.

To meet these requirements, PPP's implementation is primarily composed of four components:

- **Encapsulation method** This is an encapsulation method (RFC 1662) based on HDLC for datagram transmission over point-to-point links.

- **Link Control Protocol (LCP)** LCP allows for establishment of link connectivity and dynamic configuration and testing of the data link connection. RFC 1661 refers to this phase as the Link Establishment phase.

- **Authentication phase[optional]** The Authentication phase is an optional step whereby the peer is authenticated via some mechanism such as Challenge Handshake Authentication Protocol (CHAP) or Password Authentication Protocol (PAP).

- **Network Control Protocol (NCP)** An NCP is defined for each upper layer network protocol to allow for dynamic negotiation of its properties. Internet Protocol Control Protocol (IPCP) as defined in RFC 1332 is an example of an NCP for IP and negotiates parameters such as IP address assignment. A host of other NCPs exist for alternate Layer 3 protocols such as IPXCP (IPX) and ATCP (AppleTalk). RFC 1661 refers to this phase as the Network Protocol Phase.

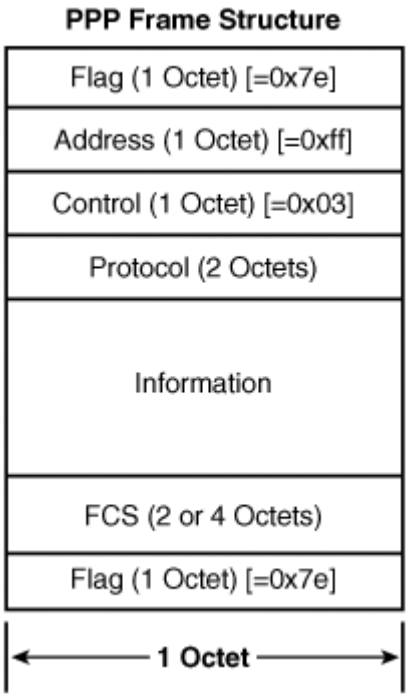
You can find the IETF standards for PPP encapsulation in RFC 1661, "Point-to-Point Protocol (PPP)," and RFC 1662, "PPP in HDLC-Like Framing." Subsequent RFCs were defined to augment PPP's capabilities. RFC 1570 defines additional extensions to the LCP mechanism, and RFC 1990 defines Multilink PPP (MLPPP), which provides a means for link aggregation.

Note

This discussion primarily examines the basic format of PPP data encapsulation as defined in RFCs 1661 and 1662. This discussion does not delve into the negotiation aspects of LCP and the various NCPs because the pseudowire emulation protocols do not interact with PPP at that level.

Figure 5-5 shows the typical PPP frame structure.

Figure 5-5. *PPP Frame Structure*



Each component of the PPP frame is described as follows:

- **Flag** The beginning and end of every PPP frame must contain a 1-byte Flag Sequence field to delimit the frame. The flag value is 01111110 (0x7e). Like HDLC, PPP uses bit stuffing on synchronous links to fulfill the requirement for network layer transparency.

  ### Note

  Similar to HDLC, PPP utilizes byte stuffing on asynchronous links.

- **Address** The Address field is set to an All Stations address of 0xff.

- **Control** The Control field is a single byte that is set to 0x03.

  ### Note

  When you perform Address and Control Field Compression (ACFC), the Address and Control fields are omitted. Furthermore, the Protocol field is reduced to a single octet when performing Protocol Field Compression

(PFC). Both ACFC and PFC are negotiated in the Link Establishment phase. You can obtain additional details on this in RFC 1661.

- **Protocol** The Protocol field is a 2-octet field which identifies the encapsulated protocol. Internet Assigned Numbers Authority (IANA) administers the assigned PPP protocol numbers. You can find the numbers at http://www.iana.org/assignments/ppp-numbers. Note that the IANA-assigned PPP protocol values do not match Cisco-assigned protocol values used in HDLC.

- **Information** The Information field is a variable-length field that contains upper layer protocol data.

- **FCS** The FCS is a 2-octet CRC calculated over the Address, Control, Protocol, Information, and Padding fields. If negotiated, PPP also supports a 4-octet FCS field.

Understanding Frame Relay

During the late 1980s, there was a rapid increase in demand for high-bandwidth WAN services. The proliferation of LAN-based end devices and growth in high-bandwidth applications fueled the need for higher speeds, low delay, and low-cost LAN interconnection.

Frame Relay was developed as a Layer 2 protocol that avoided some of the drawbacks of WAN services at the time, as follows:

- The increased availability of error-free transmission lines reduces the need for protocols such as X.25 that perform hop-by-hop error correction. Instead, Frame Relay relies on higher level protocols to perform end-to-end error correction and flow control.

- Whereas X.25 requires packet processing at Layer 3, Frame Relay strictly operates at Layer 2. This reduces the switching requirements drastically and reduces per-hop delay.

- Unlike time-division multiplexing (TDM)based circuits, which provide fixed point-to-point connectivity, Frame Relay provides network connectivity via packet switching over logical virtual circuits that are similar to X.25. Frame Relay accomplishes this by using a data-link connection identifier (DLCI) to uniquely distinguish virtual circuits on a physical link. This allows for more flexibility and more efficient data transmission through statistical multiplexing.

Frame Relay has subsequently been extended further in the Frame Relay Forum (FRF) standards body to support multiple features such as MultiLink Frame Relay (MLFR), defined in FRF.16, and Frame Relay Fragmentation, defined in FRF.11 and FRF.12.

As you will learn in subsequent chapters that explore L2TPv3 and AToM transport of Frame Relay, both pseudowire protocols interact with three aspects of Frame Relay:

- Frame Relay encapsulation to transport the Frame Relay frame on the pseudowire

- Control Management/Protocol such as Operation, Administration, and Maintenance (OAM) to properly reflect attachment circuit and pseudowire state

- Traffic management to emulate Frame Relay's inherent traffic management capabilities

This section explores these aspects of Frame Relay as a reference for later chapters.

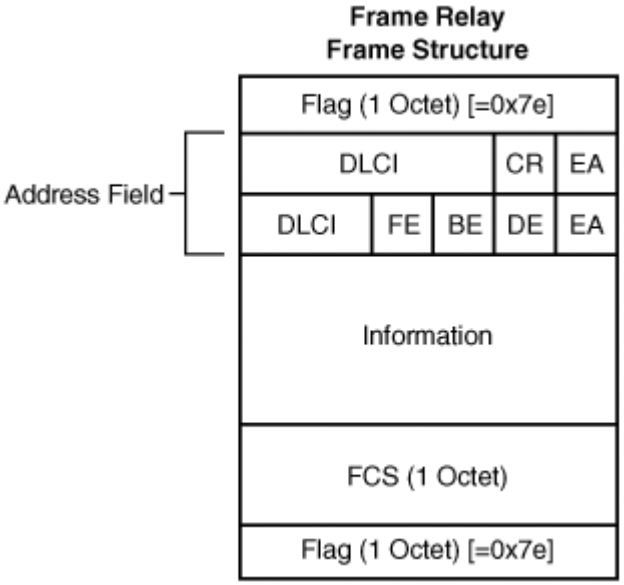
Encapsulation

To better understand how Frame Relay provides its functionality, this section examines Frame Relay Frame structure. The Frame Relay frame format is standardized in two separate standard bodies:

- Internationally via the ITU-T (formerly known at the CCITT) Q.922 Annex A specification
- Domestically in the United States via the ANSI T1.618 specification

Figure 5-6 illustrates the basic Frame Relay encapsulation per the Q.922 Annex A specification.

Figure 5-6. *Frame Relay Frame Structure*



The Frame Relay fields are described as follows:

- **Flag** Like HDLC and PPP, the beginning and end of every Frame Relay frame is delimited with a 01111110 (0x7E).
- **Address** The Address field is a 2-byte header that contains several subfields:

  DLCI The DLCI is a 10-bit field that uniquely represents a virtual connection on the physical channel. If extended addressing is used, a 17- and 23-bit DLCI address is supported.

  CR The Command/Response bit is not defined and not used.

EA The Extended Address bit is the last bit in each header byte. A value of 0 indicates that another header byte follows, whereas a value of 1 indicates that this is the last header byte. This definition allows Frame Relay to support larger DLCI values.

FECN (FE) The forward explicit congestion notification bit is set to 1 to indicate to the receiver that the frame encountered network congestion. The FECN is set on traffic sent from the sender to the receiver.

BECN (BE) The backward explicit congestion notification bit is set to 1 to indicate to the sender that the frame encountered network congestion. Because the BECN bit is set on frames traveling in the opposite direction of the frames that experienced congestion, there must be return traffic toward the sender from the receiver to accomplish this feedback loop. Both FECN and BECN bits should signify to upper layer protocols to perform some action upon indication of congestion.

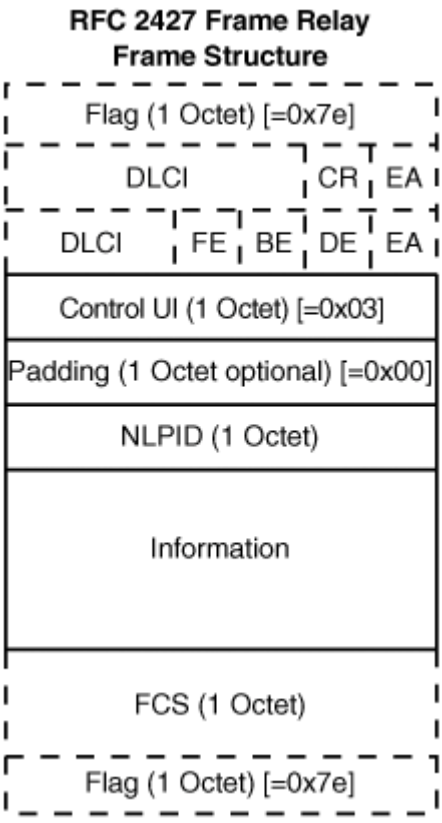
DE The discard eligible bit indicates whether this frame can be dropped in response to network congestion. A value of 0 indicates a higher priority frame versus a frame marked with a DE value of 1.

- **Information** The Information field is a variable length field from 5 to 4096 octets that contains upper layer protocol data.

- **FCS** The FCS is a 16-bit CRC calculated against the Frame Header and Data fields to detect errors.

One of the items lacking in the ITU-T Q.922 Annex A and ANSI T1.618 Frame Relay frame structure is a field indicating the type of Layer 3 data stored in the Information field. In RFC 1490 (made obsolete by RFC 2427), the IETF extended the Frame Relay structure that was defined in previous standards to support a method of multiprotocol transport on Frame Relay. The Frame Relay format was extended, as illustrated in Figure 5-7.

Figure 5-7. *RFC 2427, "Frame Relay Frame Structure"*

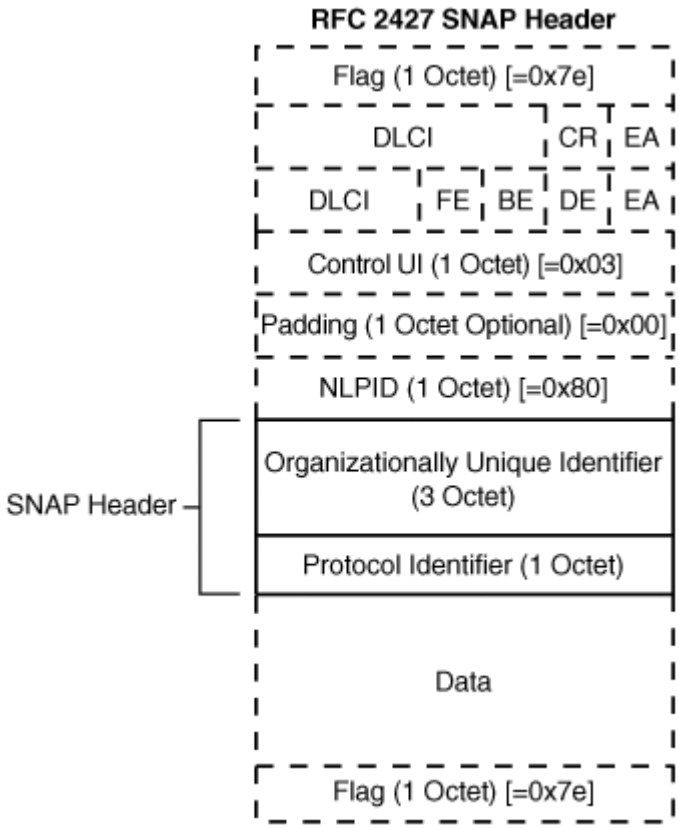


In addition to the Flag, Address, Information, and FCS fields, RFC 2427 defines the following fields:

- **Control** The Control field contains a value of 0x03 to indicate that this is an Unnumbered Information (UI) frame unless it is negotiated otherwise.
- **Padding** You can add an optional 1-byte pad to alter the frame size to an even value.
- **Network Layer Protocol Identifier (NLPID)** The NLPID identifies the Layer 3 protocol that is stored in the Information field. ISO/IEC TR 9577 defines the NLPID values. The NLPID is only 1-byte long, so the number of protocols that this field can represent is limited.

In those cases in which the NLPID is not defined for a protocol, the NLPID is set to 0x80 and an additional Subnetwork Access Protocol (SNAP) is added. Figure 5-8 illustrates the SNAP header format.

Figure 5-8. *SNAP Header*



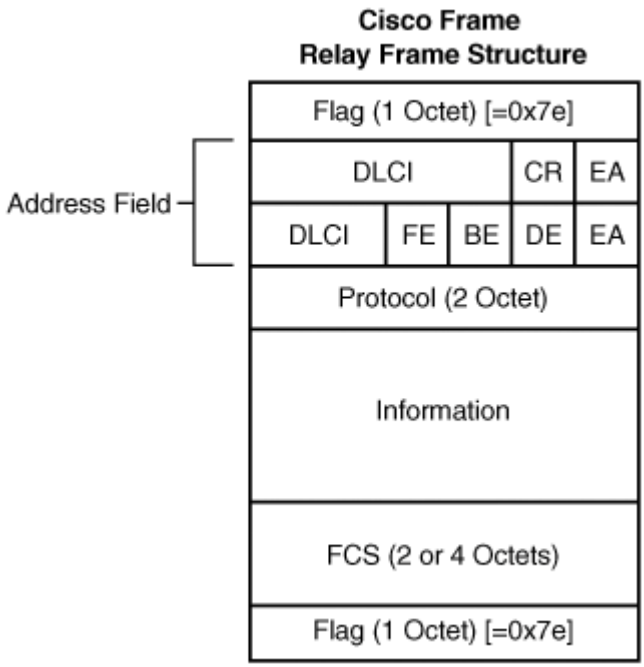
The SNAP fields are described as follows:

- **OUI** The Organizationally Unique Identifier is a 3-octet field identifying the organization that administers the succeeding 1-byte Protocol Identifier (PID). Routed PDUs use an OUI of 0x000000 whereas Bridged PDUs use OUI of 0x0080C2.

- **Protocol Identifier (PID)** PID is a 1-octet field managed by the organization identified in the preceeding OUI. The OUI and PID values together represent a unique protocol.

Cisco has an alternative Frame Relay encapsulation method to identify the Layer 3 payload. Instead of using an NLPID, Cisco uses a Protocol field to perform a similar function. Figure 5-9 illustrates the Cisco format. The Protocol field is a 2-byte field that is equal to the IEEE Ethertype or Cisco-invented codes to represent the protocol that is stored in the Information field.

Figure 5-9. ***Cisco Frame Relay Frame Structure***



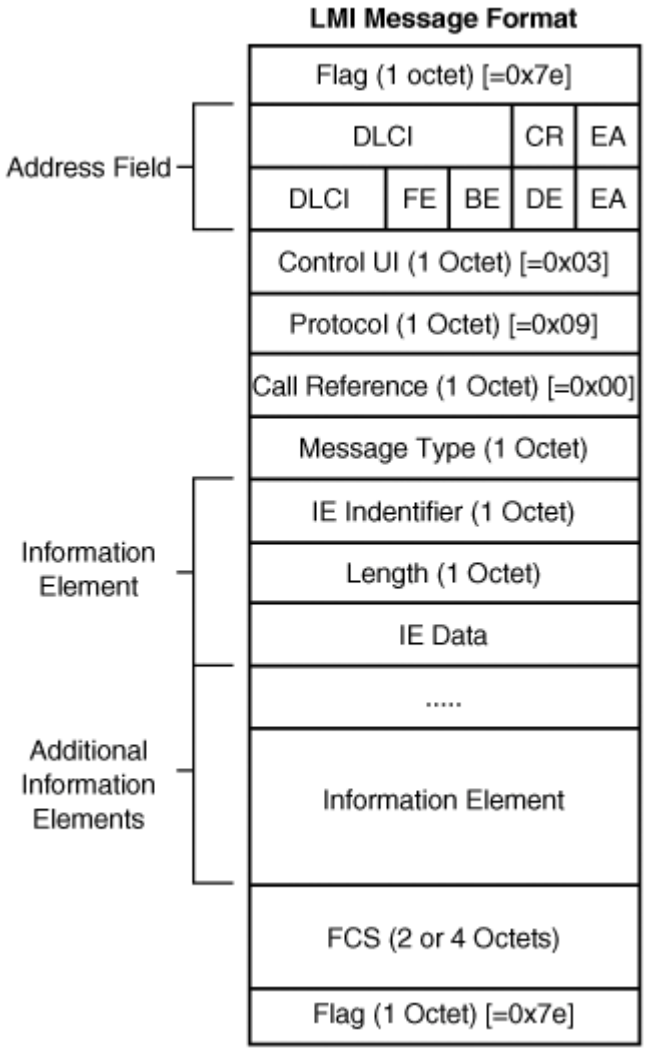
Frame Relay Link Management Interface Protocol

The initial Frame Relay standards by ANSI and ITU-T failed to provide a means to allow the Frame Relay network and the Frame device to communicate their status. In 1990, Cisco, Nortel, DEC, and StrataCom (known as the Gang of Four) developed an interim specification known as Local Management Interface (LMI) to meet this requirement. The goal of LMI was to primarily allow for the exchange of information regarding the link/device status and the notification of logical circuit status. LMI accomplishes this goal by having the Frame Relay end device (data terminal equipment, or DTE) send polls while the Frame Relay network (data communication equipment, or DCE) responds to these polls over a predetermined DLCI. Although LMI refers to the Gang of Four specification, it is also the general term for this data-link mechanism. The ITU-T (Annex A Q.933A) and ANSI (Annex-D T1.617) developed and standardized subsequent variations of LMI.

This section explores the Gang of Four LMI implementation, sometimes referred to as Cisco LMI, and later addresses the differences when compared to the ANSI and ITU-T standards.

The LMI message contains a 5-byte header, a 1-byte message type, one or more information elements of variable length, and a 2-byte CRC as illustrated by Figure 5-10.

Figure 5-10. *LMI Message Format*



The first two bytes of the header are the same as the standard Frame Relay header shown in Figure 5-9. They contain the DLCI, CR, FECN, BECN, DE, and EA bits. However, the Gang of Four implementation uses a fixed DLCI value of 1023 to communicate.

The LMI Message Format fields are described as follows:

- **Control** The Control field is fixed at 0x03 to indicate that this is an unnumbered frame.
- **Protocol** The Protocol field is fixed at 0x09 to indicate that this is an LMI frame.
- **Call Reference** The Call Reference field is unused and is fixed at 0x00.

- **Message Type** The Message Type field is a 1-byte value corresponding to the category of message that is being sent.

  The three common message types are as follows:

  Status Enquiry 0x75

  Status 0x7D

  Update Status 0x7B

- **Information Element** The Information Element is a variable length field that is composed of three additional fields:

  One Byte Information Element Identifier

  One Byte Length field

  A variable length Information Element Data field

  The type of Information Element passed depends on the preceding message type.

  The type of Frame Relay interface determines the LMI messages and the order of the message exchange. Two types of interfaces exist:

  User-to-Network Interface (UNI) A UNI connects a Frame Relay end device (DTE) to the Frame Relay network device (DCE).

  Network-to-Network Interface (NNI) An NNI connects two distinct Frame Relay network devices (DCEs).

In a UNI environment, the DTE periodically sends status enquiry messages, and the DCE end responds with status messages. In an NNI environment, both devices send status enquiry and status messages. An update status message is an optional message that the Frame Relay network sends to the Frame Relay device. The next sections describe each message frame and compare the Gang of Four LMI with the Annex A and Annex D formats.

Status Enquiry Message Frame

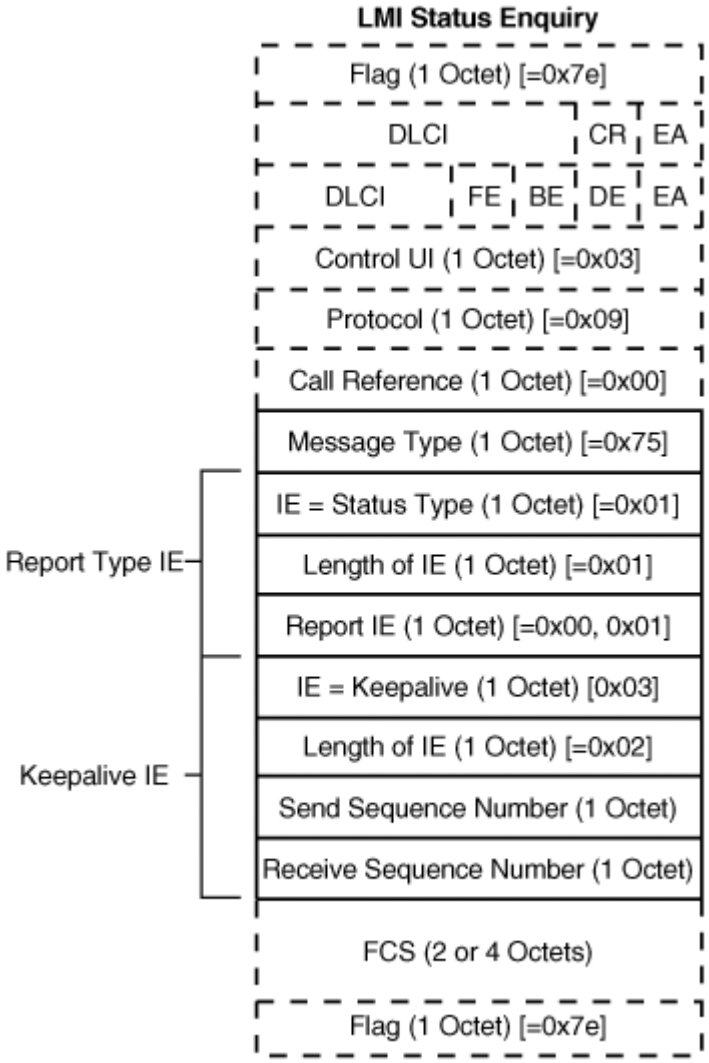
A status enquiry message requests two types of information from the Frame Relay network:

- A link integrity verification record request, which requests a sequence number exchange

- A full status record request, which requests a status report on all logical permanent virtual circuits (PVC) on this port, in addition to a sequence number exchange

Figure 5-11 shows the format for a status enquiry message.

Figure 5-11. *Status Enquiry Message Frame*



Regardless of whether the status enquiry message is a link integrity verification or a full status record, the message contains the following three components:

- **Message Type** This byte field indicates the type of message that is sent. In the case of a status enquiry, this value is 0x75.

- **Report Information Element** The first information element identifies the type of request: link integrity verification (0x01) or full status record (0x00).

- **Keepalive Information Element** The second information element exchanges the sequence number values. The Send Sequence octet should contain the sender's current sequence number, whereas the Receive Sequence octet contains the last sequence number that the sender received. The Frame Relay end device increments its sequence number with every status enquiry message that is sent. Similarly, the Frame Relay network device increments its sequence number with every status message that is sent.

Status Message Frame

Status messages have a similar format to status enquiry messages; however, the number of information elements differs depending on the type of report information element. In response to a link integrity verification request, the status message consists of a message type, a report information element, and a keepalive information element. However, in response to a full status record, PVC status information elements are also sent:

- **Message type** This 1-byte field indicates the type of message that is sent. In the case of a status message, this value is 0x7d.

- **Report information element** The first information element identifies the type of request: link integrity verification (0x01) or full status record (0x00).

- **Keepalive information element** The second information element exchanges the sequence number values and contains the same information as described earlier in the status enquiry message.

- **PVC status information element** In a full status record, an additional PVC status information element is sent for each PVC on the port. In addition to the two octets after the length, which dictate the DLCI that this information element is reporting on, an additional octet indicates the PVC status. The first 4 bits indicating PVC state are as follows:

    - **N** New bit. The New bit indicates if the PVC was newly added since the last full status report (1) or if the PVC was provisioned (0) since the last full status report.

    - **D** Deleted bit. The Deleted bit is not used in a status message.

    - **A** Active bit. The Active bit indicates whether the PVC is active (1) or failed (0).

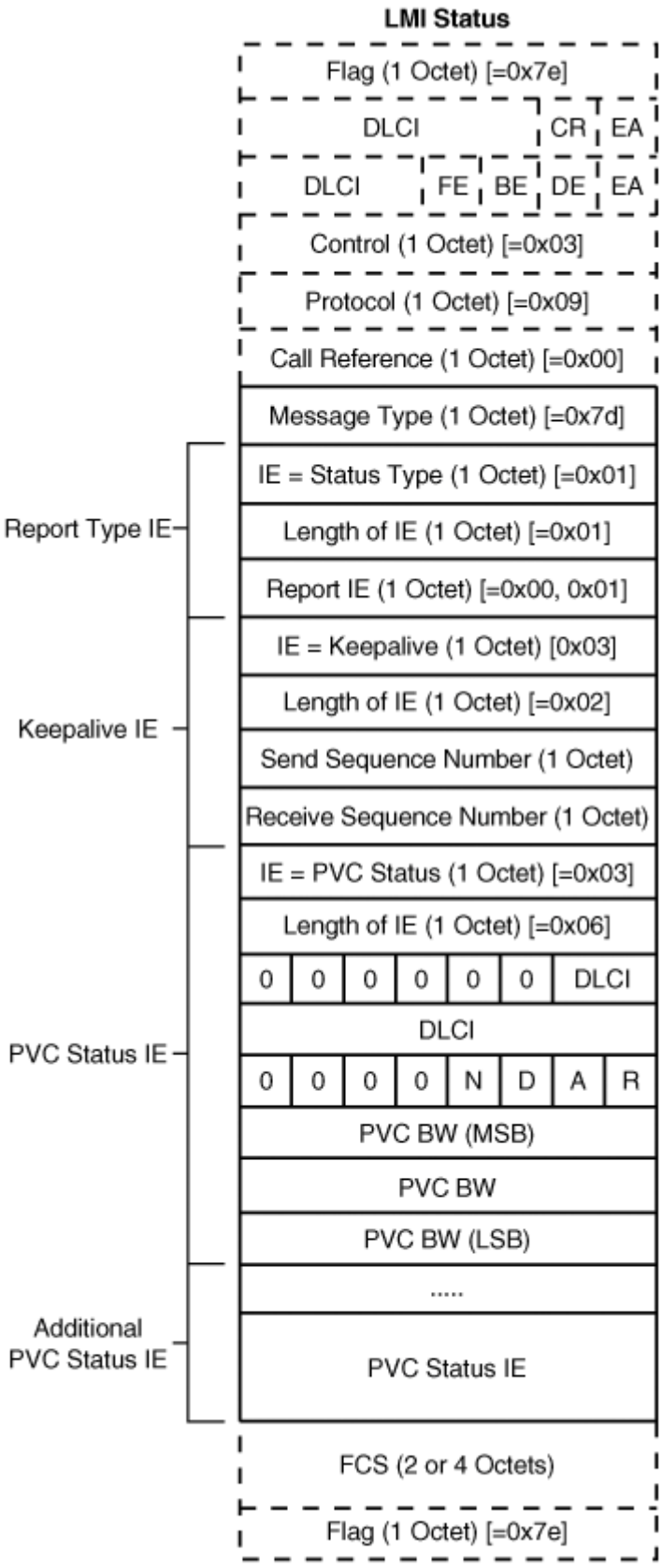
    - **R** Receiver bit. The Receiver bit is an optional implementation that provides a simple flow control mechanism to signal the end device to

stop sending traffic to this particular PVC.

The latter three octets of the PVC status information element are optional. They indicate the bandwidth of the PVC and are specific to Gang of Four LMI.

Figure 5-12 shows the format of a status message containing a full status record.

Figure 5-12. *Status Message Frame*

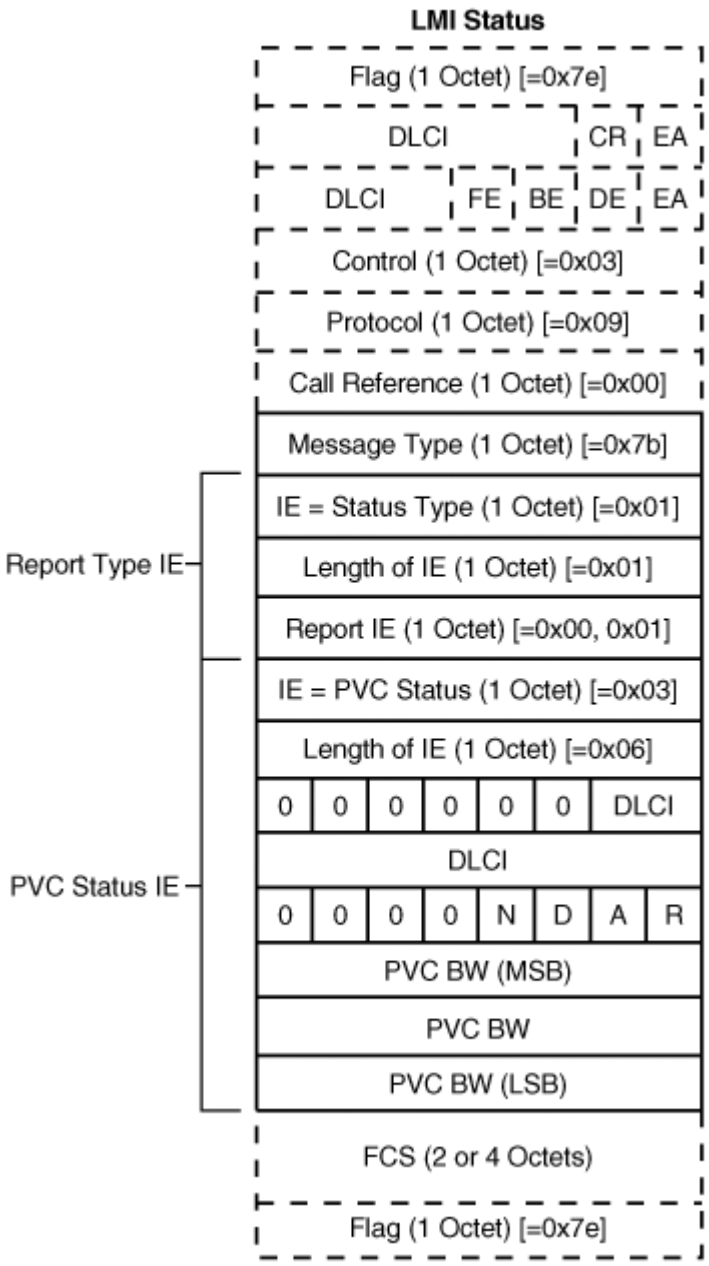


Update Status Message Frame

Unlike a status message, which is sent in response to a status enquiry, an update status message (sometimes referred to as an asynchronous status update) can be sent from the Frame Relay network device to the Frame Relay end device to convey changes in the interface's PVC state.

The format of an update status message is similar to a status message with a few minor differences (see Figure 5-13). The update status message consists of a message type, report type information element, and a PVC status information element for only those PVCs that have changed state. No keepalive information element exchanges sequence numbers. The Delete bit in the PVC Status octet is set in this message to indicate PVC removal; however, the New bit cannot be set in the update status.

Figure 5-13. *Update Status Message Frame*



Comparing Gang of Four LMI with Annex A and Annex D

Several notable differences distinguish the Gang of Four LMI and Annex A and Annex D LMI:

- The Gang of Four LMI uses DLCI 1023, whereas both Annex D and Annex A uses DLCI 0.

- Reserved DLCI ranges differ among Annex A, Annex D, and the Gang of Four implementation. For a 10-bit DLCI address, Annex A and Annex D define a DLCI user range from 16991, whereas the Gang of Four range is 161007. Annex A and Annex D are supported in an NNI environment, whereas the Gang of Four LMI is supported only on UNI.

- Annex A and Annex D update status messages can contain only a single PVC status information element. When multiple PVC states change, a separate update status message must be sent for each PVC that is affected.

- Gang of Four LMI supports optionally carrying the PVC bandwidth in the PVC status information element.

- The Gang of Four LMI utilizes a lower error threshold timer of 2, compared to the Annex D and Annex A threshold timer value of 3 for failed status message replies. A full description of the different timers and their standard defined values for each LMI type is listed in Table 5-3.

Table 5-3. *Frame Relay LMI Timers*

Title	Description	Cisco LMI	Annex D	Annex A
N391 Full Status Polling Counter	Number of cycles at which a full status record request is made.	6	6	6
N392 Error Threshold	Number of failed events out of N393 monitored events before declaring the port in alarm.	2	3	3
N393 Monitored Events Count	Number of events monitored by the port used to determine port alarm state.	4	4	4
T391 Link Integrity Polling Verification Timer	Time (in seconds) between status enquiry messages.	10	10	10

Title	Description	Cisco LMI	Annex D	Annex A
T392 Polling Verification Timer	Time interval (in seconds) at which a status message is expected in reply to a status enquiry message. If it is not received in time, an N392 error is logged.	15	15	15

Managing Traffic

Frame Relay services typically provide traffic throughput guarantees per PVC. To meet those guarantees, it is critical that a mechanism be in place to provide traffic management capabilities. Frame Relay employs Frame Relay policing to determine ingress traffic admission policy and Frame Relay shaping for egress traffic management.

Frame Relay Traffic Policing

Frame Relay traffic policing is a quality of service (QoS) mechanism that is applied on ingress into the network as a means of admission control to limit the amount of traffic that an end device can send into the network.

Frame Relay policing can be represented as a token-based abstraction known as a *leaky bucket model*. Essentially, the leaky bucket model determines whether a frame is compliant or noncompliant based on the fate of a frame's associated token. The leak rate of these buckets represents the admission rate of traffic.

To check for compliancy, every incoming frame has an associated token that is placed in the bucket. If the incoming rate exceeds the leak rate, the total number of tokens will eventually exceed the depth of the bucket. Nonconforming traffic occurs when the frame's associated token exceeds this bucket depth. Noncompliant traffic is either dropped or tagged appropriately. If the token is allowed to pass, the associated frame is admitted. Frame Relay policing employs a similar model with dual buckets, known as a dual leaky bucket model. This model utilizes the following parameters:

- **CIR** The time-averaged leak rate that the Frame Relay network agrees to support a logical connection. In the dual leaky bucket model, this is the leak rate of the first bucket.

- **Excess information rate (EIR)** The average excess rate allowed above CIR. In the dual leaky bucket model, this is the leak rate of the second bucket.

- **Committed Rate Measurement Interval (Tc)** The time interval in which the leaky bucket is replenished with committed burst (Bc)/CIR worth of tokens: Tc=Bc/CIR.

- **Bc** The amount of committed traffic allowed during a Tc interval. This is represented in the dual leaky bucket model as part of the depth of the first leaky bucket: Bc= CIR*Tc.

- **Excess burst (Be)** The amount of excess traffic allowed during a Tc interval. This is represented in the dual leaky bucket model as part of the depth of the second leaky bucket. Be can be set to 0 to cause all noncompliant frames in the second bucket to be discarded: Be=Tc*EIR.

In the dual leaky bucket model, Bc and Be tokens are replenished at every Tc interval for the first and second bucket respectively. When receiving a frame without the DE bit set, Frame Relay policing checks the frame for compliancy by determining whether the associated token of the frame is admitted through the first bucket. If the DE bit is set to 1, Frame Relay policing checks the frame for compliancy against the second bucket.

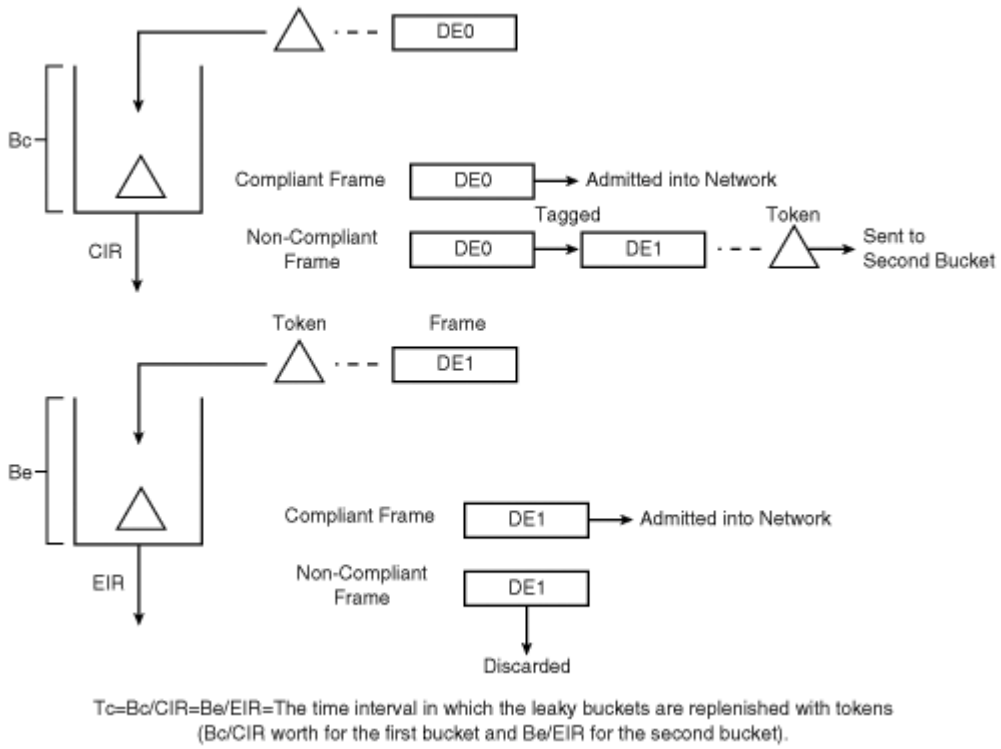
Frames that conform to the CIR of the first bucket are admitted into the network. Frames that are not CIR conformant (that is, the associated token exceeds the depth of the first bucket) are marked to be DE and are sent to the second bucket for EIR conformance.

Frames with the DE bit set are checked for EIR conformance in the second bucket. If the frame is EIR rate compliant, it is queued for transmission; otherwise, the frame is discarded.

Figure 5-14 shows the Frame Relay policing model and illustrates the different outcomes based on compliancy and token availability at the time.

Figure 5-14. *Frame Relay Policing Leaky Bucket Model*

[View full size image]



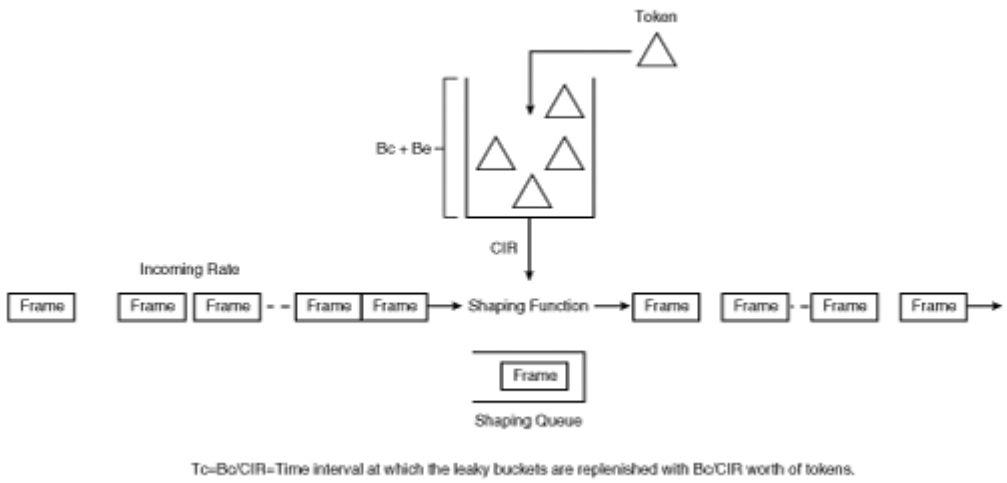
Frame Relay Traffic Shaping

Frame Relay traffic shaping is a traffic management capability offered on a per-PVC basis on egress. Whereas Frame Relay traffic policing is an ingress admission policy, Frame Relay traffic shaping is intended to smooth outgoing traffic to a mean rate and, if necessary, queue traffic for transmission.

In Frame Relay traffic policing, the incoming rate of traffic was never adjusted and the packets were never queued; instead, packets were admitted at their incoming rate based on token availability. In the Frame Relay traffic policing case of noncompliancy, the traffic is potentially dropped. Frame Relay traffic shaping, on the other hand, buffers packets for later transmission in the case of noncompliancy to enforce an average egress rate over time. Frame Relay traffic shaping can be modeled as a leaky bucket, as shown in Figure 5-15.

Figure 5-15. ***Frame Relay Traffic Shaping Leaky Bucket Model***

[View full size image]



Tc=Bc/CIR=Time interval at which the leaky buckets are replenished with Bc/CIR worth of tokens.

Frames are sent through the shaper only if an associated token is available. If no tokens are available, the shaping function queues the frame for later transmission. Tokens are leaked out of the bucket at the CIR. At every Tc (Bc/CIR) interval, Bc/CIR worth of tokens is replenished. The maximum size of the bucket is Bc + Be. The Be component allows a burst capability above the CIR rate. The result of a potentially bursty ingress rate is a smoothed output stream of traffic.

Frame Relay traffic shaping can also adapt its traffic rates based on network conditions. For example, based on indicators of network congestion such as BECNs, the adaptive Frame Relay traffic shaping can reduce the token replenish rate to similarly reduce its outgoing traffic rate.

Understanding ATM

ATM was developed as a high-speed switching solution to handle a variety of traffic types ranging from bursty data services to delay and jitter-sensitive voice. Instead of using variable length frames, ATM utilizes fixed-length cells to transport data. Like Frame Relay, traffic is carried on logical circuits that are uniquely identified by virtual path and circuit identifier fields in the header of each cell.

ATM is probably most well known for its well-developed QoS support because of its strict traffic class definitions. By utilizing a layered protocol architecture, ATM can transport voice, video, and data on the network. Depending on their traffic characteristics, upper layer protocols are processed according to a set of adaptation rules prior to forming each cell.

As subsequent chapters identify, L2TPv3 and AToM interact with three aspects of ATM:

- ATM encapsulation to transport either ATM cells or ATM Adaption Layer (AAL) frames on the pseudowire.
- Control management/protocol, such as OAM, to properly reflect attachment circuit and pseudowire state
- Traffic management features, such as ATM policing and ATM shaping, to emulate ATM's inherent traffic management capabilities

The next section explores these three aspects of ATM as a reference for later chapters.

Encapsulation

To understand ATM encapsulation, it is necessary to describe the lower layers of the ATM protocol stack illustrated in Figure 5-16 and progress through the ATM stack encapsulation from the ATM Adaptation Layer (AAL) and ATM layer with specific focus on AAL5 and ATM cell formats. The lower layers of the ATM protocol stack essentially consist of the following components:

- **Physical layer** The physical layer is composed of two sublayers:

  Transmission Convergence (TC) The TC sublayer handles functions such as cell delineation and error detection/correction. The error detection/correction is accomplished by adding a 1-byte CRC to the ATM cell header.

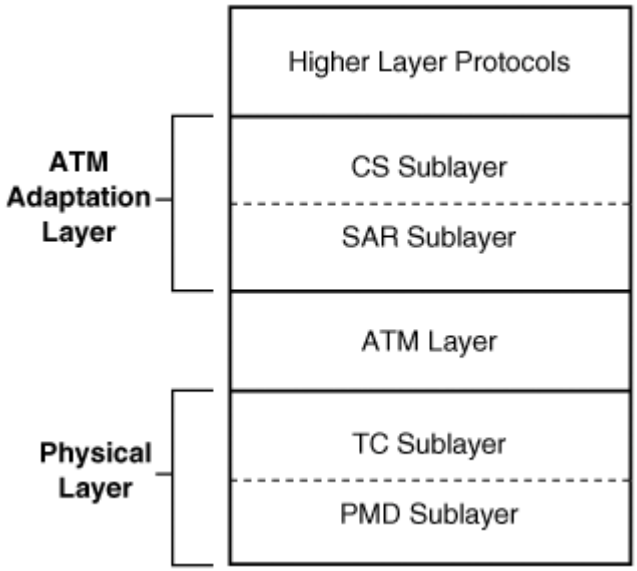
Physical Media-Dependent (PMD) The PMD sublayer is responsible for medium dependent functions such as bit transmission and electro/optical conversion.

- **ATM layer** From a data plane perspective, the ATM layer handles cell header (4 bytes) generation and removal and VPI/VCI translation.
- **AAL** The AAL, defined in ITU-T I.362 and ITU-T I.363, adapts data from various upper layer protocols into the necessary ATM cell payload. The AAL consists of two additional sublayers:

  Convergence sublayer (CS) CS processes data from higher layer protocols into variable length frames known as Convergence Sublayer Protocol Data Units (CS-PDUs).

  Segmentation and Reassembly (SAR) sublayer SAR is responsible for segmenting the CS-PDUs into 48-byte payloads for an ATM cell.

Figure 5-16. ***ATM Protocol Stack***



The next few sections examine the AAL and ATM layer with specific focus on AAL5 and ATM cell formats.

ATM Adaptation Layer

AAL defines multiple AAL formats depending on the traffic type from the upper layer protocols. They include the following:

- **AAL1** AAL1 is intended to carry connection-oriented, constant bit rate traffic with specific timing requirements. Typical AAL1 traffic is Circuit Emulation Services (ATM Forum standard af-vtoa-0078.0000), such as transparently carrying DS-1 and E-1 circuits across an ATM core.

- **AAL2** AAL2 supports payloads that have timing requirements similar to that of AAL1 traffic but that have bursty traffic patterns. Compressed voice and video are examples of AAI2 traffic.

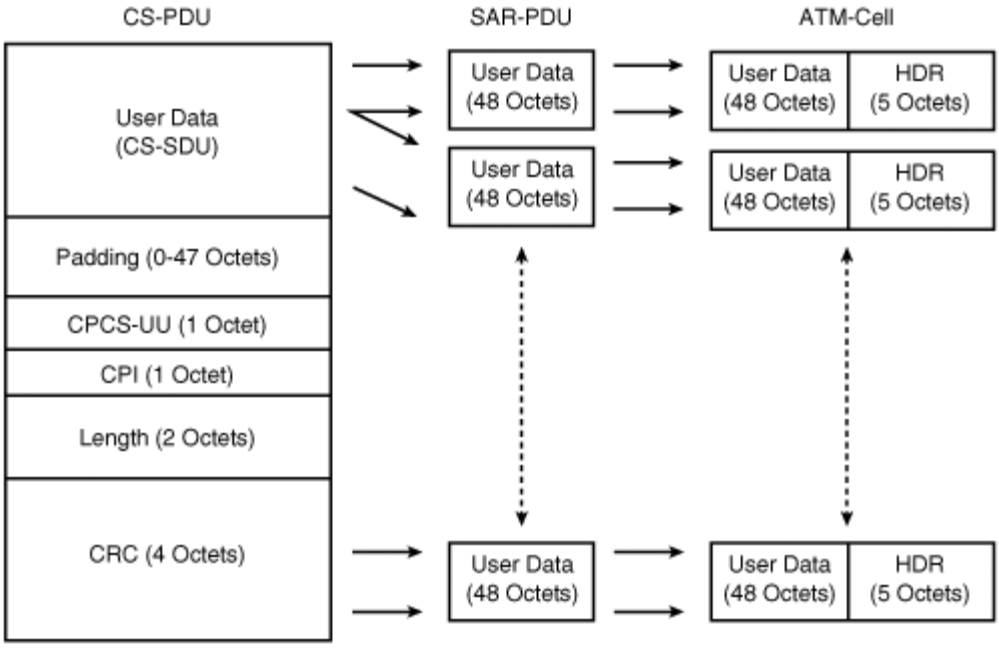
- **AAL3/4** AAL3/4 supports connection-oriented and connectionless variable bit rate traffic. The primary function of AAL3/4 is to carry Switched MultiMegabit Data Service (SMDS) data.

- **AAL5** Because of AAL3/4's large overhead and complexity, AAL5 was developed as a simpler and more efficient adaptation layer to carry connection-oriented and connectionless traffic. AAL5 is the main format used today for carrying IP routed and bridged data.

Figure 5-17 shows the CS-PDU and SAR-PDU structure for AAL5 and the processing involved down to the ATM layer for cell header generation.

Figure 5-17. *AAL5 PDU*

[View full size image]



The CS-PDU is formed by appending a CS-PDU trailer to the CS-SDU. The CS-PDU is composed of the following fields:

- **Padding** The Padding field is added to ensure that the resulting CS-PDU size is a multiple of 48 bytes to present to the SAR function.

- **Common part convergence sublayer user to user (CPCS-UU)** The CPCS-UU field allows upper layer protocols to send information transparently to the AAL5 structure. An example application is FRF 8.1, which uses this octet to transport Frame Relay C/R bit. In other cases such as RFC 2684, "Multiprotocol Encapsulation over ATM Adaptation Layer 5," this field is unused.

- **Common part indicator (CPI)** CPI provides alignment of the CPCS-PDU to 64 bits. The value of this field is set to 0x00.

- **Length** This is a 2-byte field indicating the length of the Payload field.

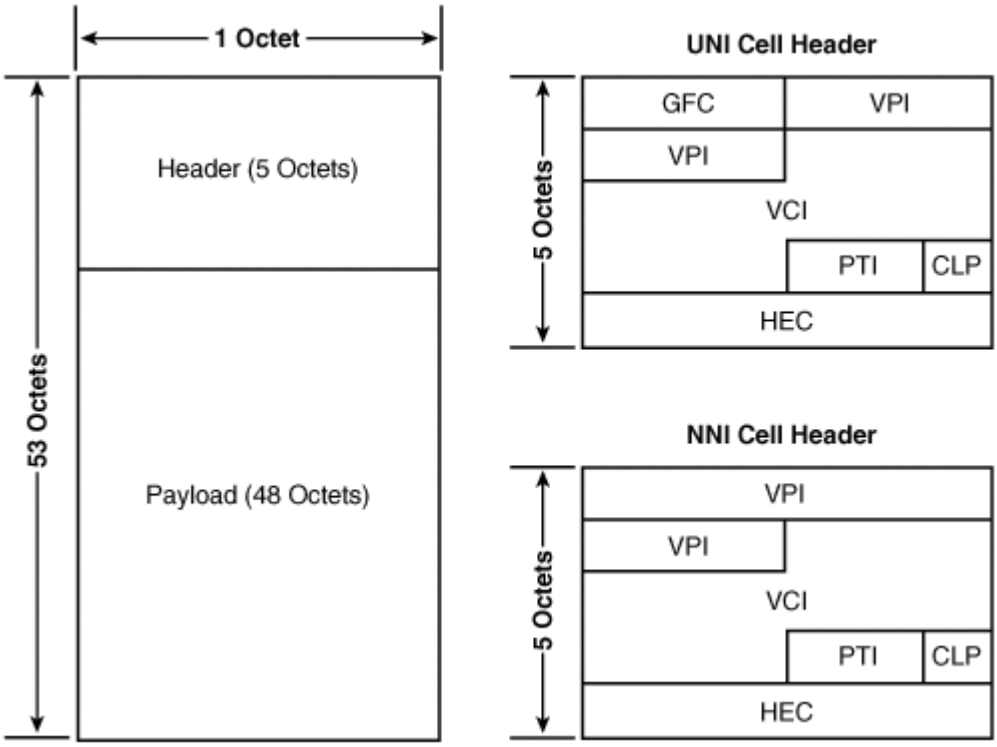
- **CRC** Cyclic redundancy calculated over the entire CS-PDU minus the 4-byte CRC field.

The resulting CS-PDU is presented to the SAR layer, which segments it into 48-byte SARPDUs. The ATM layer generates a 4-byte header for each SAR-PDU. To correctly identify the last cell forming the original AAL5 PDU, the last cell header's third bit in the payload type identifier (PTI), a field in the ATM cell header, is set. The TC sublayer adds the fifth byte to complete the 5-byte header.

ATM Cell Structure

As discussed in the previous section, the ATM layer and the TC sublayer are responsible for the remaining cell header generation and removal. Depending on the interface type (UNI or NNI), the format of the header is slightly different. Figure 5-18 shows the ATM cell format.

Figure 5-18. *ATM Cell Format*

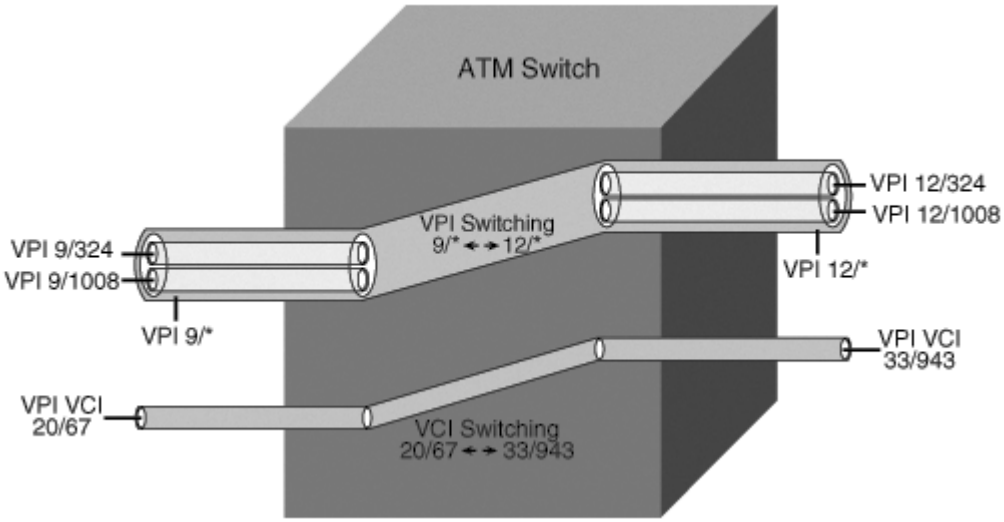


The following are the fields in the ATM cell format:

- **Generic Flow Control (GFC)** The GFC field on the UNI header provides flow control on the particular logical PVC. This field is set to 0 and is not fully standardized.
- **Virtual path identifier/virtual connection identifier (VPI/VCI)** Together, the VPI and the VCI uniquely identify a virtual connection. You can use them together as a switching identifier. Alternately, you can use the VPI alone as a switching field and consider it a logical grouping of the VCIs in that scenario. Figure 5-19 illustrates the difference between VP and VC switching.

Figure 5-19. ***ATM VP and VC Switching***

[View full size image]



- **PTI** PTI is composed of 3 bits that characterize the type of cell and measure congestion. The first bit indicates whether the cell is a management cell (1) or contains user data (0). The remaining two bits are interpreted differently in each of those cases, as follows:

  User data cell The second bit, known as the explicit forward congestion indication (EFCI) field, indicates congestion. The third bit is set to indicate whether this is the last cell in an AAL5 frame.

  Management cell The second bit identifies the cell as an OAM cell (0) or a resource management (RM) cell (1). The third bit distinguishes the OAM cell as an F5 (OAM cell used to convey PVC status) segment (0) or F5 end-to-end flow (1).

- **Cell loss priority (CLP)** The CLP bit prioritizes the cell. In congestion scenarios in which it is necessary to drop traffic, some devices could implement a selective discard mechanism whereby CLP set cells would be dropped before cells without CLP marking.

- **Header error control (HEC)** The TC adds the HEC field, which provides error detection and optionally bit error correction. It is calculated only for the ATM cell header.

ATM Management Protocols: ILMI and OAM

Similar to the Frame Relay environment, ATM provides a signaling mechanism to convey interface and PVC status. The two main mechanisms used are Interim Local Management Interface (ILMI) and OAM cells.

ILMI uses SNMP messages that are encapsulated in AAL5 over VPI/VCI 0/16 to access ILMI MIB variables. This mechanism allows for a variety of information to be conveyed, such as type of signaling used, address registration, and interface and PVC management.

Note

Although the ILMI VPI/VCI default is 0/16, the ILMI specification allows use of an alternate VPI/VCI other than the default value. Also, in VP-tunnel applications, the VPI is set to the VPI of the VP-tunnel.

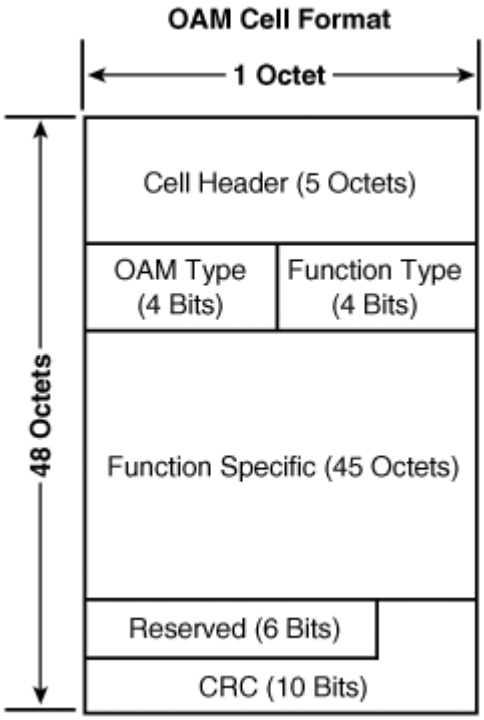
In addition to ILMI, you can use OAM to determine logical circuit status. Two forms of OAM cellsF5 and F4are used depending on the type of logical circuit you are dealing with.

In the case of a PVC, you can use and send OAM F5 cells on the same VPI and VCI as the PVC. The PTI field of a F5 cell not only differentiates the F5 OAM cell from a user data cell, but it differentiates an end-to-end (ATM end-user device to end-user device) OAM or a segment (ATM end-user device to ATM network device) OAM.

F4 OAM cells, on the other hand, convey the status of a permanent virtual path (PVP), a connection switched upon the VPI field alone. F4 OAM cells use the same VPI as the PVP connection that they are representing, but they use VCI 3 for segment OAM and VCI 4 for end-to-end OAM.

Figure 5-20 shows the typical OAM cell format.

Figure 5-20. *OAM Cell Format*



In addition to the typical ATM cell header and CRC field, the OAM fields include the following fields:

- **OAM Type** The OAM Type field determines the management cell's general role:

  Fault management

  Performance management

  Activation/deactivation

- **Function Type** The Function Type field defines the specific function of the cell and is interpreted differently depending on the OAM type.

- **Function Specific** The Function Specific field determines the payload of the OAM cell, which differs based on the OAM Type and Function Type fields. Figure 5-21 illustrates the Function Specific payloads for an alarm indication signal (AIS), far end receive failure (FERF)/remote defect indication (RDI), and loopback function type.

Figure 5-21. ***OAM AIS and Loopback Cell Format***

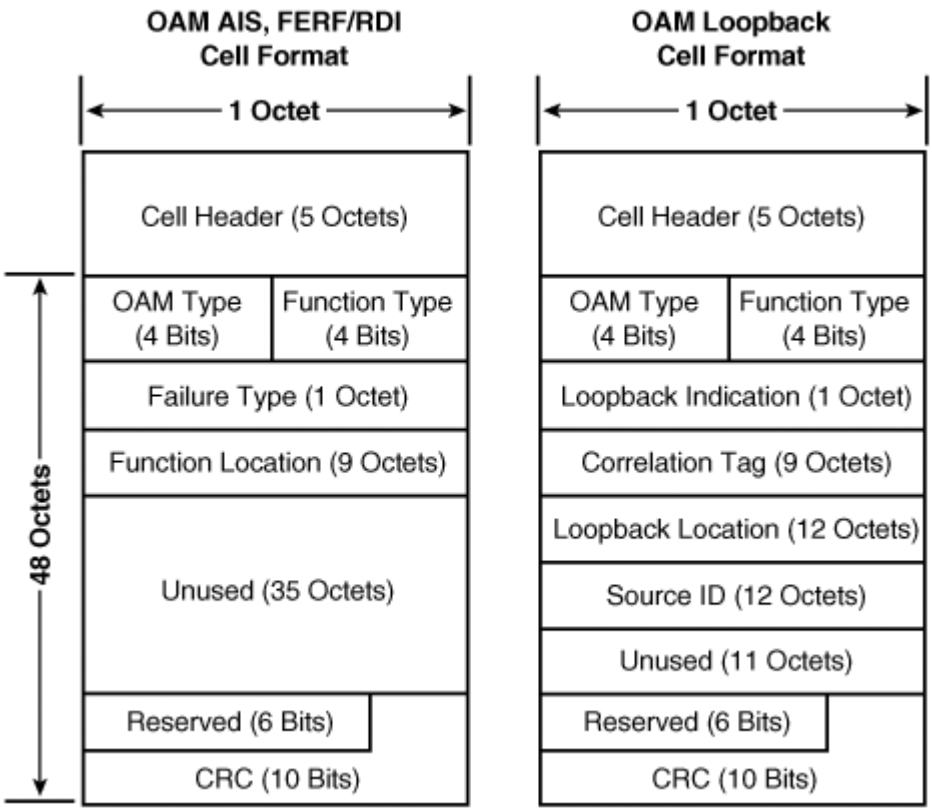


Table 5-4 defines the OAM and Function Type combinations.

Table 5-4. *OAM Type and Function Type*

OAM Type	OAM Type Binary Value	Function Type	Function Type Binary Value
Fault Management	0001	AIS	0000
		RDI/FERF	0001
		OAM Cell Loopback	1000
		Continuity Check	0100

OAM Type	OAM Type Binary Value	Function Type	Function Type Binary Value
Performance Management	0010	Forward Monitor	0000
		Backward Reporting	0001
		Monitoring and Reporting	0010
Activation/Deactivation	1000	Performance Monitor	0000
		Continuity Check	0001

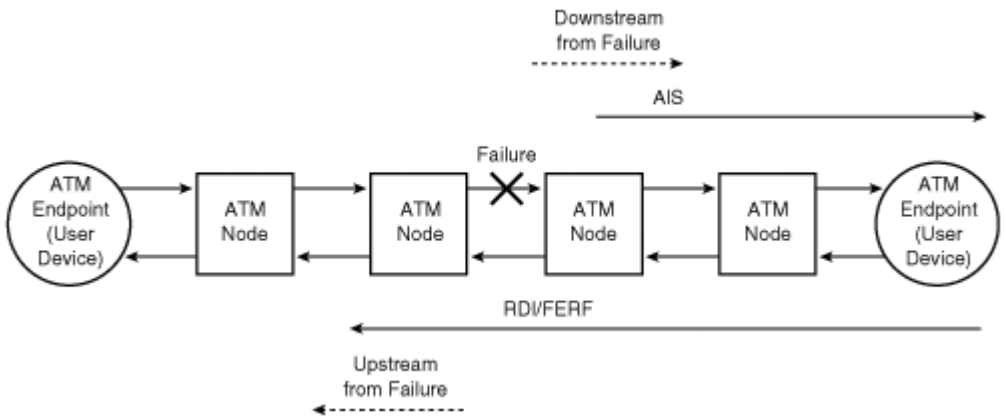
From a fault management perspective, the AIS, RDI/FERF, and Loopback function types are of particular importance in dealing with logical circuit status.

AIS and RDI/FERF indicate to the remote endpoints a failure within the ATM network and function in a similar manner to SONET, DS3, and T1 alarms. An intermediate device that is detecting a link failure to notify downstream nodes generates AIS. RDI/FERF is generated at the intermediate node upon receiving AIS to alert upstream devices. To draw an analogy between T1 alarming and ATM, AIS is similar to a blue alarm, whereas RDI is a yellow alarm.

If an individual VPC or VCC fails in the network, similar VP or VC AIS and FERF/RDI alarms are generated. Figure 5-22 illustrates the AIS and RDI/FERF behavior of ATM nodes and endpoints when dealing with logical circuit failure. The intermediate ATM node, upon detection of a logical circuit breakage, generates AIS in the direction of the failure. The ATM endpoint in turn generates AIS RDI/FERF when receiving the AIS alarm.

Figure 5-22. *Logical Circut Failure AIS and FERF/RDI Alarms*

[View full size image]



OAM Loopback cells are also used as a fault management feature to confirm logical circuit status. When configured, OAM loopback cells are sent and a corresponding loopback cell is received in response. The payload of an OAM loopback cell is shown in Figure 5-18. The Loopback Indicator field first bit is set to 1 on the outgoing cell and set to 0 to indicate a looped response. The Correlation Tag field matches the outgoing OAM loopback cell with the received response cells. A successive number of loopback replies not being returned could indicate to the endpoint that the logical circuit should be declared unusable.

Managing Traffic

ATM is most well known for the QoS capabilities that allow it to carry a variety of traffic classes. The following are the four general ATM traffic classes:

- **Constant bit rate (CBR)** Used for real-time traffic that consumes a fixed amount of bandwidth. Typical applications include real-time voice and circuit emulation.
- **Variable bit rate (VBR)** Reserved for applications that consume a variable amount of bandwidth. VBR traffic that requires tightly constrained delay and delay variation is classified as Real Time VBR (RT-VBR). VBR traffic that does not have such delay requirements is defined as Non-Real Time VBR (NRT-VBR).
- **Available bit rate (ABR)** Used for non-timecritical applications that support a flow control mechanism to allow it to adjust the bandwidth used based on ATM network characteristics. This traffic class is applicable to any data traffic applications that can take advantage of this variable bandwidth allowed through closed loop feedback mechanisms.
- **Unspecified Bit Rate (UBR)** Intended for non-realtime applications that are delay tolerant. Typical applications include best-effort data transport.

ATM networks employ numerous traffic management mechanisms to maintain the necessary QoS guarantees for each customer PVC or PVP.

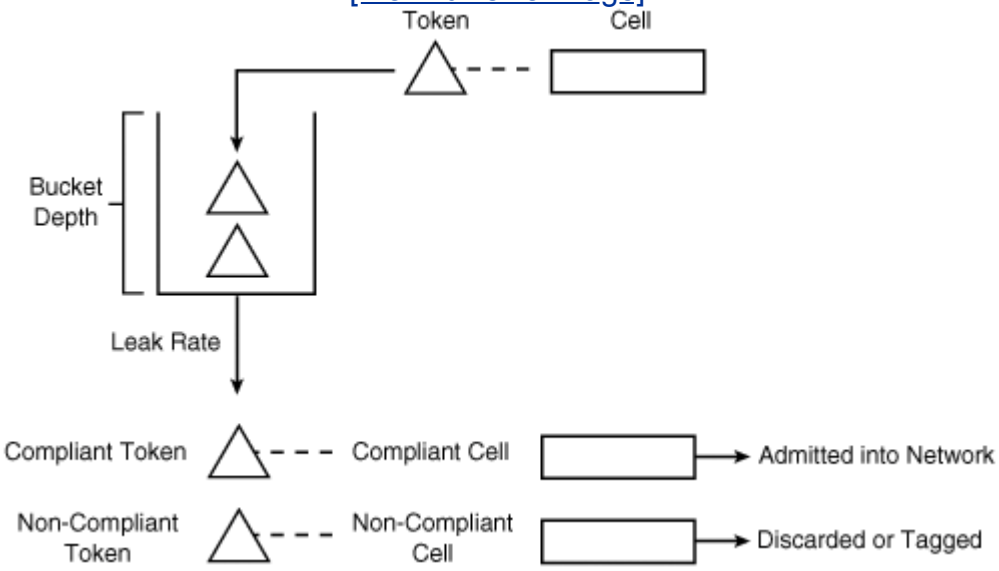
ATM Traffic Policing

One of the methods used to meet those traffic agreements is a feature known as *ATM policing* or *usage parameter control (UPC)*. ATM policing is a mechanism typically performed on ingress into the ATM network to ensure that the traffic received on a logical connection conforms to the defined traffic parameters for that circuit. If the incoming traffic fails to conform, you can discard the data or tag it with a lower priority.

Like Frame Relay policing, ATM policing can be represented as a leaky bucket model, as shown in Figure 5-23.

Figure 5-23. *ATM Policing Leaky Bucket Model*

[View full size image]



In a leaky bucket model, each ATM cell has an associated token whose fate determines whether the ATM cell is considered compliant or noncompliant. The bucket represents the number of tokens that can be stored. If the number of tokens exceeds the size of the bucket, the associated cell is considered noncompliant, and appropriate action, such as discarding or tagging the cell, is performed. The leak rate of the bucket represents the rate at which the tokens are drained from the bucket. If the incoming token rate is greater than the leak rate, the bucket will eventually overflow, and the incoming traffic will be considered noncompliant. More complex traffic-policing contracts use a similar model but employ dual leaky buckets.

The ATM Forum Traffic Management 4.0 standard describes several conformance definitions that determine the type of traffic that is regulated and the action that is performed for compliancy/noncompliancy. Table 5-5 describes the traffic conformance definitions that will be explored in more detail in the following sections: CBR.1, VBR.1, VBR.2, VBR.3, UBR.1, and UBR.2. CBR.1, UBR.1, and UBR.2 can be represented as a single leaky bucket with a leak rate that the peak cell rate (PCR) defines. The VBR.1, VBR.2, and VBR.3 definitions are modeled as a dual leaky bucket, with the first and second bucket leak rate equal to the PCR and sustained cell rate (SCR), respectively. The PCR flow and SCR flow columns define the traffic type that is checked for conformance. For example, CLP (0+1) represents all cells, whereas CLP (0) represents only cells with the CLP bit set to zero. The CLP tagging column defines whether the nonconforming action for that bucket is tagged.

Table 5-5. *ATM Forum Traffic Management 4.0 Traffic Policing Classes*

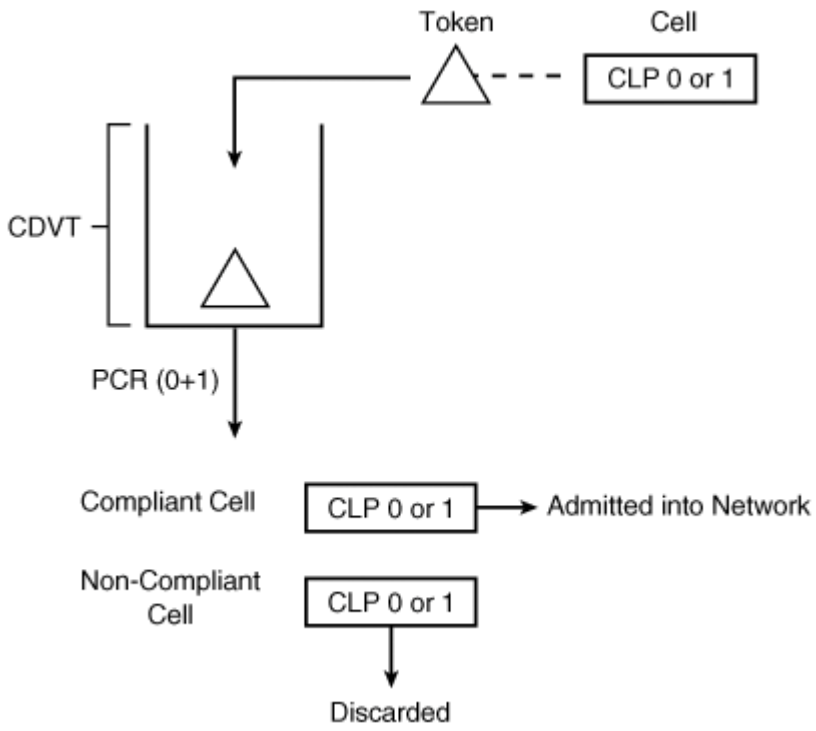
ATM Forum TM 4.0 Spec.	PCR Flow	CLP Tagging for PCR	SCR Flow	CLP Tagging for SCR
CBR.1	CLP (0+1)	No	NA	NA
VBR.1	CLP (0+1)	No	CLP (0+1)	No
VBR.2	CLP (0+1)	No	CLP (0)	No
VBR.3	CLP (0+1)	No	CLP (0)	Yes
UBR.1	CLP (0+1)	No	NA	NA
UBR.2	CLP (0+1)	No	NA	NA

CBR.1 Traffic Policing

The two values that define the CBR.1 traffic policing model are the cell delay variation tolerance (CDVT) and the PCR. In this model, the PCR (0+1) is the leak rate for all cells, CLP 0 and CLP 1 marked cells. The CDVT (0+1) is the depth of the bucket, which allows for some variation in the token rate. If the token rate is less

than or equal to the PCR (0+1), the tokens will be compliant and the associated cells will be allowed into the ATM network. If the token rate is consistently greater than the PCR (0+1) rate, the CDVT bucket depth will eventually be exceeded and those noncompliant tokens, and their associated cells, will be discarded. Figure 5-24 illustrates this process.

Figure 5-24. *CBR.1 Traffic Policing*

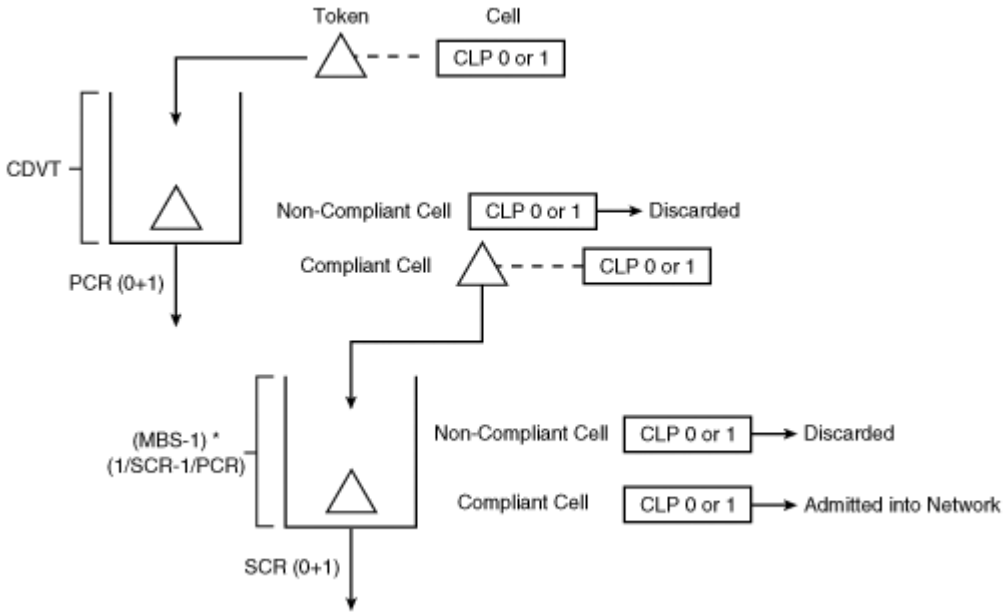


VBR.1 Traffic Policing

Unlike CBR.1, which employs a single leaky bucket model, VBR.1 traffic policing can be modeled as a dual leaky bucket, as shown in Figure 5-25. The first leaky bucket acts like the CBR single leaky bucket with a PCR (0+1) leak rate and a CDVT (0+1) depth. Noncompliant tokens in the first bucket are discarded. All CLP 0 and CLP 1 compliant tokens are then checked against the second leaky bucket whose leak rate is SCR (0+1) and depth is a function of maximum burst size (MBS). Tokens that are noncompliant in the second bucket are discarded. Compliant tokens in the second bucket are allowed into the network.

Figure 5-25. *VBR.1 Traffic Policing*

[View full size image]

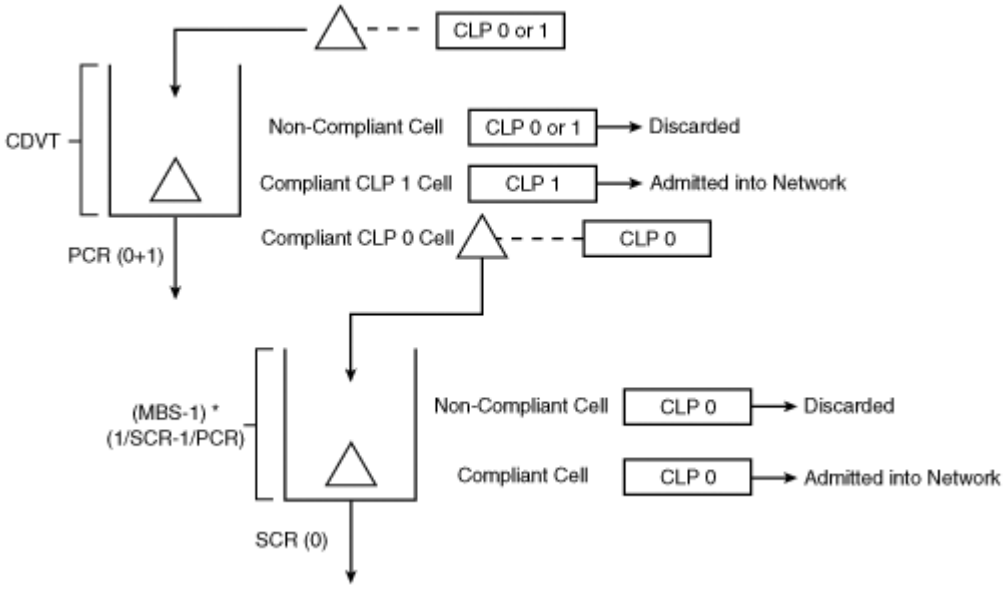


VBR.2 Traffic Policing

VBR.2 traffic policing is modeled as a dual leaky bucket and operates in a similar manner to VBR.1. The difference between the VBR.2 and VBR.1 models is that the second bucket in VBR.2 only checks CLP 0 cells. The compliant CLP 1 tokens from the first bucket are admitted into the network and are not checked for compliance in the second bucket. Figure 5-26 illustrates these differences in the VBR.2 model.

Figure 5-26. *VBR.2 Traffic Policing*

[View full size image]

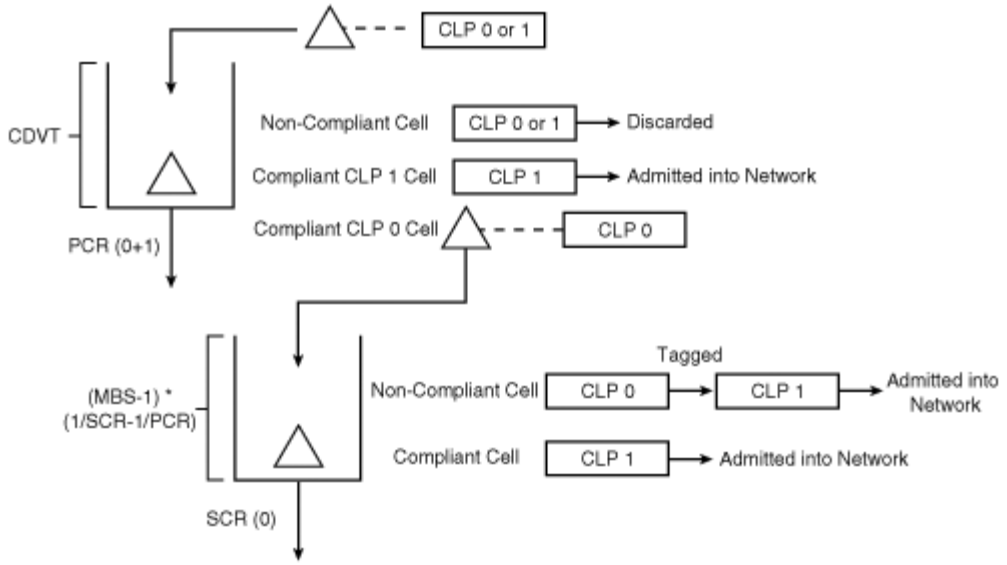


VBR.3 Traffic Policing

VBR.3 policing, illustrated in Figure 5-27, operates in the same manner as VBR.2 except that noncompliant cells in the second bucket are tagged with CLP 1 and admitted into the network instead of being discarded.

Figure 5-27. *VBR.3 Traffic Policing*

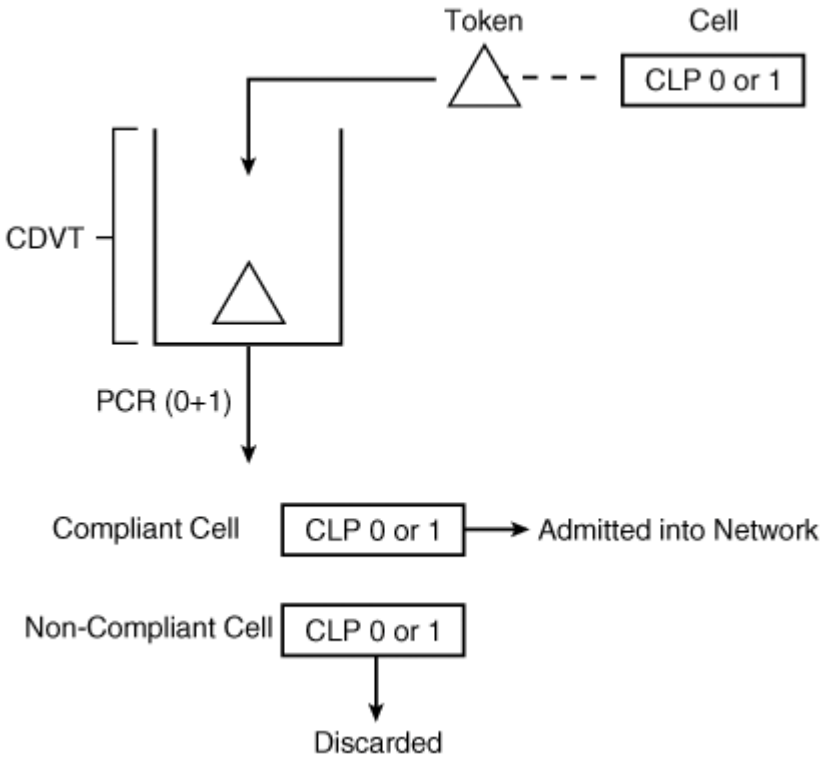
[View full size image]



UBR.1 Traffic Policing

UBR.1 uses a single leaky bucket model with a leak rate of PCR (0+1) and bucket depth of CDVT. Noncompliant cells are discarded, whereas compliant cells are admitted into the network. Figure 5-28 shows the UBR.1 policing model.

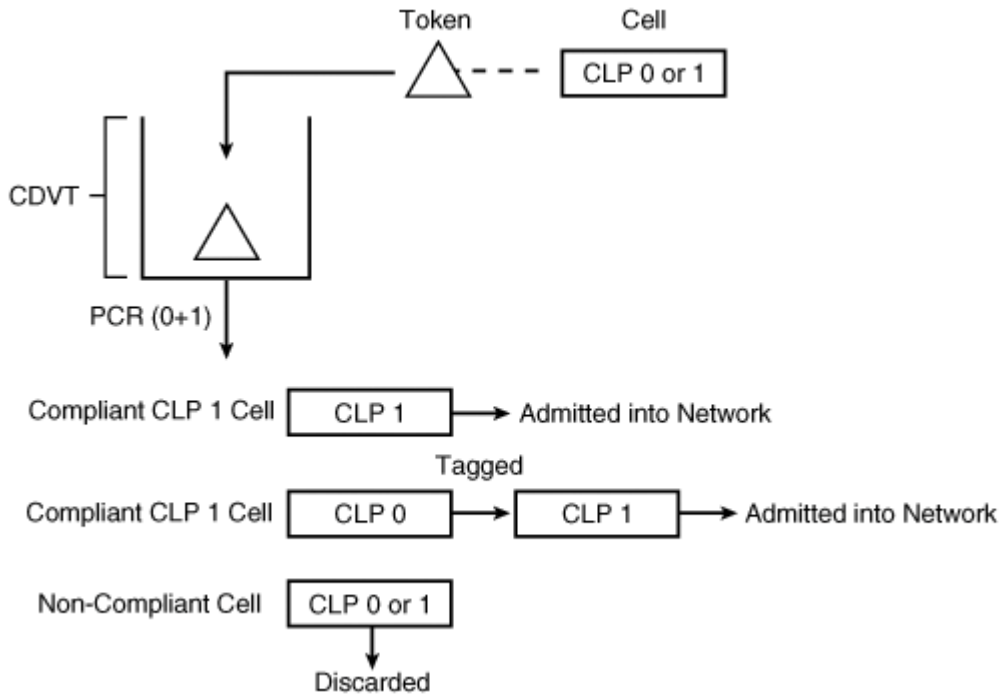
Figure 5-28. *UBR.1 Traffic Policing*



UBR.2 Traffic Policing

As illustrated in Figure 5-29, UBR.2 operates in the same fashion as UBR.1 except that compliant CLP 0 cells are tagged to CLP 1.

Figure 5-29. *UBR.2 Traffic Policing*



ATM Traffic Shaping

ATM traffic shaping is a QoS mechanism that is typically deployed on egress out of an ATM node or end device used to enforce a long-term average rate for a logical circuit. Unlike ATM traffic policing, in which noncompliant traffic is either dropped or marked to a lower priority, ATM traffic shaping queues nonconforming traffic to restrain data bursts and smooth data rates to comply within the defined traffic contract.

Figure 5-30 illustrates the general concept as a leaky bucket model.

Figure 5-30. *ATM Traffic Shaping*

[View full size image]

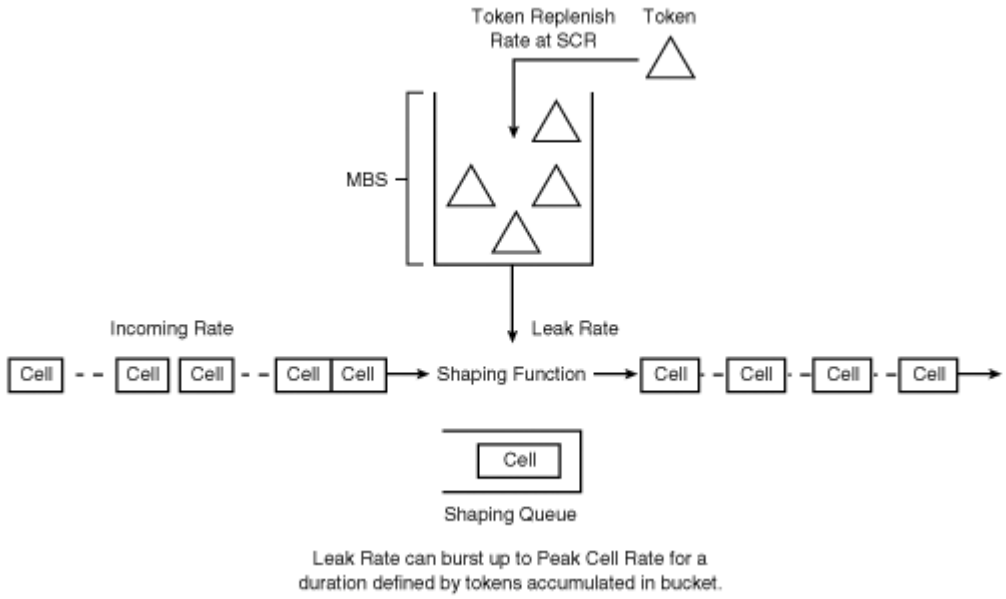


Figure 5-30 defines three parameters:

- **Sustained cell rate (SCR)** The average cell rate that traffic should conform to. This is illustrated in Figure 5-30 as the rate at which tokens are replenished.
- **Peak cell rate (PCR)** The maximum cell rate that the traffic cannot exceed. This is represented in the model as the maximum rate at which tokens can leak out of the token bucket.
- **Maximum burst size (MBS)** The number of cells that the device can transmit up to at the PCR rate. The MBS is the depth of the token bucket.

Similar to Frame Relay traffic shaping, cells are transmitted as long as a corresponding token allowing the transmission is available. Traffic is queued for later transmission if a token is not available. If the incoming rate is less than the SCR, tokens are accumulated up to the MBS depth of the bucket. At some later time, if the incoming rate is bursty and exceeds the SCR for a short time interval, the traffic can use the accumulated tokens to send up to the PCR rate. If the incoming rate continues to exceed the SCR rate, the accumulated tokens will eventually be depleted and the cells will only be able to send at SCR, the token replenish rate. Excess traffic will need to be queued and potentially dropped if the incoming rate does not subside.

This generic model applies differently depending on the nature of the traffic class. VBR defines a PCR, SCR, and MBS and follows the general leaky bucket model. On the other hand, CBR's long-term average rate is defined as its PCR and has some form of transmission priority to meet a strict CDVT based on the nature of the traffic it has to support: real-time applications. UBR PVCs typically are not shaped and

burst up to the ATM port rate. However, you can optionally define a PCR to limit the maximum transmission rate. ABR is unique compared to the other traffic classes because of its ability to adapt its traffic rate based on indicators of network congestion states such as EFCI or via RM cells. ABR shaping defines a PCR, a minimum cell rate (MCR), the minimum rate that the PVC can send at, and some additional parameters that define its rate adaptation factors.

Summary

This chapter reviewed some of the basic properties of several Layer 2 protocols. As mentioned in the chapter introduction, this chapter was not meant to be an exhaustive examination of each of these protocols; instead, it examined the relevant aspects of HDLC, PPP, Frame Relay, and ATM in the context of pseudowire emulation.

Following are several key aspects to take away from this chapter:

- HDLC uses a simple framing mechanism to encapsulate its data. Cisco HDLC encapsulation is a modified form of HDLC and adds a Protocol Identifier field to determine the Layer 3 protocol that is stored in the HDLC payload.

- PPP utilizes a framing mechanism that is similar to HDLC. Although PPP has a rich set of negotiation protocols such as LCP, different optional authentication methods and various NCPs, this was not discussed because they are transparent to the pseudowire emulation protocols.

- Frame Relay adopts an encapsulation format that is similar to HDLC and PPP. A DLCI identifier in the Frame Relay header distinguishes logically distinct Frame Relay circuits.

- Frame Relay LMI conveys circuit status information between network devices through periodic status messages.

- Frame Relay has the option of performing Frame Relay policing to enforce a traffic contract for traffic that is inbound to the network from the customer. You can either discard noncompliant traffic or mark it with a lower priority by setting the DE bit. Conversely, you can apply Frame Relay shaping on network egress toward the customer. Shaping queues traffic that is nonconforming to meet a long-term average rate.

- ATM has a well-defined protocol stack that takes upper layer protocol data and processes it through the AAL, ATM, and Physical layers. ATM also uses VPI/VCI to uniquely represent a logical ATM circuit.

- From a fault management perspective, ATM OAM indicates faults within the network and performs end-to-end connectivity checks.

- ATM traffic policing offers a set of admission control options to enforce ingress traffic contracts from end customer devices. You can either discard out-of-contract traffic or mark it with a lower priority through the use of the CLP bit. Conversely, ATM traffic shaping enforces an average rate of traffic on egress toward customer devices and can employ queuing for nonconforming traffic.

Part III: Any Transport over MPLS

Chapter 6. Understanding Any Transport over MPLS

This chapter covers the following topics:

- Label Distribution Protocol (LDP)
- AToM operations

To provide Layer 2 VPN services over an IP/Multiprotocol Label Switching (MPLS) network infrastructure, the Internet Engineering Task Force (IETF) developed a series of solution and protocol specifications for various Layer 2 VPN applications, including pseudowire emulation. Based on the pseudowire emulation specifications, Any Transport over MPLS (AToM) is implemented as part of the Cisco Unified VPN Suite Solution. The Cisco solution also includes alternative pseudowire emulation using Layer 2 Tunnel Protocol Version 3 (L2TPv3). Chapter 3, "Layer 2 VPN Architectures," outlines the benefits and implications of using each technology and highlights some important factors that help network planners and operators determine the appropriate technology.

This chapter starts with an overview of LDP used by pseudowire emulation over MPLS, followed by an explanation of the protocol specifications and operations of AToM. You learn the general properties of the pseudowire emulation over MPLS networks specified in IETF documents. Additional features that AToM supports are also highlighted in this chapter.

Introducing the Label Distribution Protocol

One of the fundamental tasks in the MPLS architecture is to exchange labels between label switch routers (LSR) and define the semantics of these labels. LSRs follow a set of procedures, known as label distribution protocol, to accomplish this task. A label distribution protocol can be an existing protocol with MPLS label extensions or a new protocol that is specifically designed for this purpose. Although the MPLS architecture allows different label distribution protocols, only LDP is used as the signaling protocol for AToM.

> **Note**
>
> In most MPLS literature, it is common to refer to *label distribution protocol* in lowercase when referring to any protocol that performs label distribution procedures and reserve the *abbreviation LDP* for the specific protocol Label Distribution Protocol, as defined in RFC 3036, "LDP Specification."

The next few sections review some fundamental LDP specifications and operations that are relevant to AToM:

- LDP protocol components
- Discovery mechanisms
- Session establishment
- Label distribution and management
- LDP security

LDP Protocol Components

To have a firm understanding of the protocol operations of LDP, you need to be familiar with the key terminology and protocol entities that are defined in LDP.

LDP peers are two LSRs that use LDP to exchange label information. An LSR might have more than one LDP peer, and it establishes an *LDP session* with each LDP peer. An LDP session is always bidirectional, which allows both LDP peers to exchange label information. However, using a bidirectional signaling session does not make the label-switched path (LSP) bidirectional. As described in Chapter 3, an LSP is unidirectional, and a pseudowire consists of two LSPs of the opposite directions.

Besides directly connected LSRs, LDP sessions can be established between non-directly connected LSRs, which are further explained in the later section titled "LDP Extended Discovery."

Label space specifies the label assignment. The two types of label space are as follows:

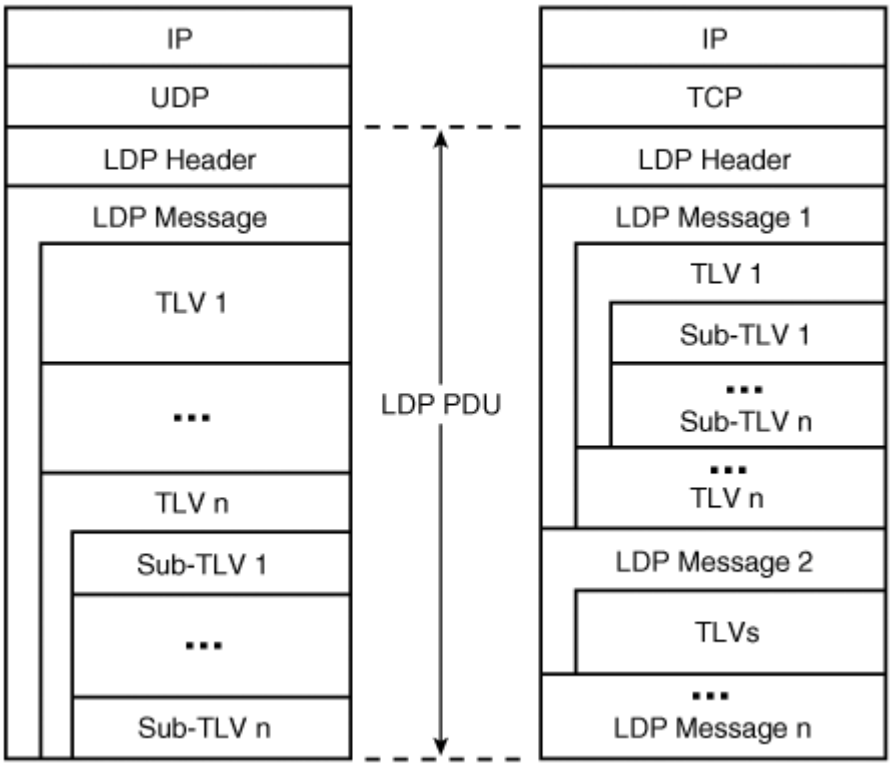
- **Per-interface label space** Assigns labels from an interface-specific pool of labels. This space typically uses interface resources for labels. For example, a label-controlled ATM interface uses virtual path identifiers (VPI) and virtual circuit identifiers (VCI) as labels.

- **Per-platform label space** Assigns labels from a platform-wide pool of labels and typically uses resources that are shared across the platform. Hop-by-hop best-effort IP/MPLS forwarding is an example of using the per-platform label space.

In Chapter 3, the AToM overview explains the use of label stacking. To recap, the label stack of AToM typically consists of two labels: tunnel label and pseudowire label. Tunnel labels can be from either per-interface label space or per-platform label space depending on whether the LSRs perform IP/MPLS forwarding in cell mode or frame mode. Pseudowire labels are always allocated from the general-purpose per-platform label space.

LDP uses User Datagram Protocol (UDP) and TCP to transport the protocol data unit (PDU) that carries LDP messages. Figure 6-1 illustrates the structure of an LDP packet. Each LDP PDU is an LDP header followed by one or more LDP messages. All LDP messages have a common LDP message header followed by one or more structured parameters that use a type, length, value (TLV) encoding scheme. The Value field of a TLV might consist of one or more sub-TLVs.

Figure 6-1. *LDP Packet Structure*



The LDP header consists of the following fields, as depicted in Figure 6-2:

- **Version** The Version field is 2 octets containing the version number of the protocol. The current LDP version is version 1.
- **PDU Length** The PDU Length field is a 2-octet integer specifying the total length of this PDU in octets, excluding the Version and PDU Length fields. The maximum PDU Length can be negotiated during LDP session initialization.
- **LDP Identifier** An LDP Identifier consists of 6 octets and identifies an LSR label space. The first 4 octets are a globally unique value that identifies the LSR. The globally unique value is usually the 32-bit router ID of the LSR. The last 2 octets identify a specific label space within the LSR. A zero value of the last 2 octets represents the platform-wide label space. When an LSR uses LDP to advertise more than one label space to another LSR, it creates a separate LDP session for each label space.

Figure 6-2. *LDP Header Format*

Version (2 Octets) [=1]
PDU Length (2 Octets)
LDP Indentifier (6 Octets)

Four categories exist for LDP messages:

- **Discovery messages** Provide a mechanism in which LSRs indicate their presence in a network by sending Hello messages periodically. Discovery messages include the LDP Link Hello message and the LDP Targeted Hello message. You learn more about discovery messages in the next section "Discovery Mechanisms."

- **Session messages** Establish, maintain, and disconnect sessions between LDP peers. Session messages are LDP Initialization messages and Keepalive messages. You learn more about session messages in the section "Session Establishment" later in this chapter.

- **Advertisement messages** Create, update, and delete label mappings. All LDP Address messages and LDP Label messages belong to advertisement messages.

- **Notification messages** Provide advisory information and signal error information to LDP peers.

Except for discovery messages that use UDP as the underlying transport, LDP messages rely on TCP to ensure reliable and in-order delivery of messages. All LDP messages have the format that is depicted in Figure 6-3.

Figure 6-3. *LDP Message Format*

U	Message Type (15 Bits)
Message Length (2 Octets)	
Message ID (4 Octets)	

Mandatory Parameters (Variable Length)
Optional Parameters (Variable Length)

The following fields make up the LDP message format:

- **Unknown Message Bit (U-Bit)** The U-bit tells the receiver of the message what action to take if he does not understand the message. If the U-bit is set to 0, the receiver needs to respond to the originator of the message with a notification message. Otherwise, the receiver should silently ignore this unknown message.
- **Message Type** The Message Type field identifies the type of message.
- **Message Length** The Message Length field specifies the total number of octets of the Message ID, Mandatory Parameters, and Optional Parameters.
- **Message ID** The Message ID field is a 4-octet value that identifies individual messages.
- **Mandatory Parameters** The Mandatory Parameters field is a set of required parameters with variable lengths that pertain to this message. Some messages do not have mandatory parameters.
- **Optional Parameters** The Optional Parameters field is a set of optional parameters that have variable lengths. Many messages do not have optional parameters.

Most information that is carried in an LDP message is encoded in TLVs. TLV provides a generic and extensible encoding scheme for existing and future applications that use LDP signaling. An LDP TLV consists of a 2-bit Flag field, a 14-bit Type field, and a 2-octet Length field, followed by a variable length Value field. Figure 6-4 shows the common TLV encoding scheme.

Figure 6-4. *LDP TLV Encoding*

U	F	Type (14 Bits)

Length (2 Octets)
Value (Variable Length)

Like the unknown message bit, the unknown TLV bit (U-bit) tells the receiver whether it should send a notification message to the originator if the receiver does not understand the TLV. If the U-bit is set to 0, the receiver must respond with a notification message and discard the entire message. Otherwise, the unknown TLV is silently ignored and the rest of the message is processed as if the unknown TLV does not exist.

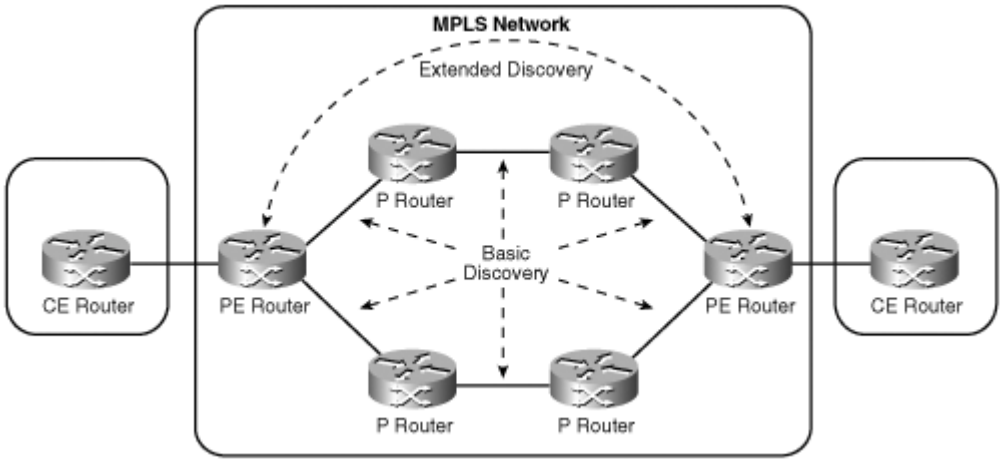
The forward unknown TLV bit (F-bit) applies only when the U-bit is set to 1 and the TLV is unknown to the receiver. If the F-bit is set to 0, the unknown TLV is not forwarded. Otherwise, it is forwarded with the containing message.

Discovery Mechanisms

LSRs use LDP discovery procedures to locate possible LDP peers. The basic discovery mechanism identifies directly connected LDP peers. The extended discovery mechanism identifies non-directly connected LDP peers. LSRs discover LDP peers by exchanging LDP Hello messages. As you learned in the previous section, two types of LDP Hello messages exist. LDP Link Hellos are used for LDP basic discovery, and LDP Targeted Hellos are used for LDP extended discovery. Figure 6-5 illustrates where LDP basic discovery and LDP extended discovery occur in an MPLS network.

Figure 6-5. *LDP Basic and Extended Discovery*

[View full size image]



LDP Basic Discovery

With LDP basic discovery enabled on an interface, an LSR periodically sends LDP Link Hello messages out the interface. LDP Link Hellos are encapsulated in UDP packets and sent to the well-known LDP discovery port 646 with the destination address set to the multicast group address 224.0.0.2. This multicast address represents all routers on this subnet.

An LDP Link Hello message that an LSR sends carries the LDP identifier for the label space that the LSR intends to use for the interface and other information, such as Hello hold time. When the LSR receives an LDP Link Hello on an interface, it creates a Hello adjacency to keep track of a potential LDP peer reachable at the link level on the interface and learns the label space that the peer intends to use for the interface.

LDP Extended Discovery

For some MPLS applications such as AToM, exchanging label information between non-directly connected LSRs is necessary. Before establishing LDP sessions between non-directly connected LSRs, the LSRs engage in LDP extended discovery by periodically sending Targeted Hello messages to a specific address. LDP Targeted Hello messages are encapsulated in UDP packets and sent to the well-known LDP discovery port 646 with a specific unicast address.

An LDP Targeted Hello message that an LSR sends carries the LDP Identifier for the label space that the LSR intends to use and other information. When the receiving LSR receives an LDP Targeted Hello, it creates a Hello adjacency with a potential LDP peer reachable at the network level and learns the label space that the peer intends to use.

When an LSR sends LDP a Targeted Hello to a receiving LSR, the receiving LSR can either accept the Targeted Hello or ignore it. The receiving LSR accepts the Targeted Hello by creating a Hello adjacency with the originating LSR and periodically sending Targeted Hellos to it.

Session Establishment

After two LSRs exchange LDP discovery Hello messages, they start the process of session establishment, which proceeds in two sequential phases:

1. Transport connection establishment
2. Session initialization

The objective of the transport connection establishment phase is to establish a reliable TCP connection between two LDP peers. If both LDP peers initiate an LDP TCP connection, it might result in two concurrent TCP connections. To avoid this situation, an LSR first determines whether it should play the active or passive role in session establishment by comparing its own transport address with the transport address it obtains through the exchange of LDP Hellos. If its address has a higher value, it assumes the active role. Otherwise, it is passive. When an LSR plays the active role, it initiates a TCP connection to the LDP peer on the well-known LDP TCP port 646.

After the LSR establishes the TCP connection, session establishment proceeds to the session initialization phase. In this phase, LDP peers exchange and negotiate session parameters such as the protocol version, label distribution methods, timer values, label ranges, and so on.

If an LSR plays the active role, it starts the negotiation of session parameters by sending an Initialization message to its LDP peer. The Initialization message carries both the LDP Identifier for the label space of the active LSR and the LDP Identifier of the passive LSR. The receiver compares the LDP Identifier with the Hello adjacencies created during LDP discovery. If the receiver finds a match and the session parameters are acceptable, it replies with an Initialization message with its own session parameters and a Keepalive message to acknowledge the sender's parameters. When the sender receives an Initialization message with acceptable session parameters, it responds with a Keepalive message.

When both LDP peers exchange Initialization and Keepalive messages with each other, the session initialization phase is completed successfully and the LDP session is considered operational.

Label Distribution and Management

Label distribution and management consist of different control, retention, and advertisement modes. Even though it is possible to use an arbitrary permutation for

an MPLS application, a certain combination of control, retention, and advertisement modes is usually more preferable or appropriate for a particular MPLS application.

The next few sections explain the following aspects in label distribution and management:

- Label binding
- Label advertisement message
- Label advertisement mode
- Label distribution control mode
- Label retention mode

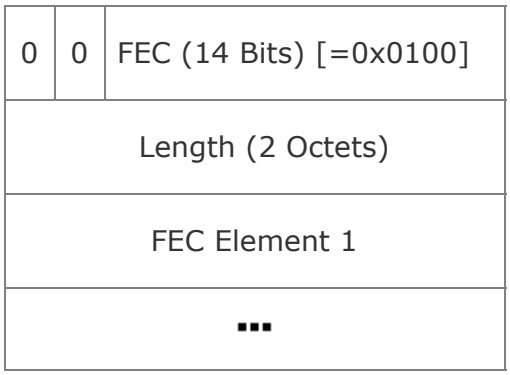
Label Binding

The main focus of an MPLS application is the distribution and management of label bindings. Label bindings are always the centerpiece of information in LDP signaling.

LDP associates a *Forwarding Equivalence Class* (FEC) with each LSP that it creates. An FEC specifies which packets should be forwarded through the associated LSP. Each FEC is defined as a collection of one or more FEC elements. Each FEC element identifies a set of packets that are mapped to the corresponding LSP. For those who are familiar with IP routing, you can consider an FEC as a set of IP routes following a common forwarding path, and an FEC element as a specific IP route prefix.

A label binding is the association between an FEC and a label that represents a specific LSP. The association is created by placing an FEC TLV and a Label TLV in a label advertisement message. Figure 6-6 depicts the FEC TLV encoding.

Figure 6-6. *FEC TLV Encoding*



FEC Element *n*

For FEC element 1 to FEC element *n*, the first octet in the FEC element indicates the FEC element type. The encoding scheme of the FEC element varies depending on the FEC element type, such as address prefix and host address. MPLS pseudowire emulation applications such as AToM use the Pseudowire ID FEC element.

Several different types of Label TLV encodings are available, including the following:

- Generic Label TLV
- ATM Label TLV
- Frame Relay Label TLV

Generic Label TLV carries a label from the platform-wide label space and is the most common encoding among MPLS applications (see Figure 6-7).

Figure 6-7. *Generic Label TLV Encoding*

0	0	Generic Label (14 Bits) [=0x0200]
Length (2 Octets)		
Label (4 Octets)		

Generic Label TLV has a type of 0x0200. A label is a 20-bit label value in a 4-octet Label field.

LDP Advertisement Message

Label bindings are exchanged through LDP advertisement messages. The advertisement messages that are most relevant to pseudowire emulation over MPLS application are these

- Label Mapping
- Label Request
- Label Withdraw
- Label Release

Label Mapping messages advertise label bindings to LDP peers. A Label Mapping message contains one FEC TLV and one Label TLV. Each FEC TLV might have one or more FEC elements depending on the type of application, and each Label TLV has one label.

When an LSR needs a label binding for a specific FEC but does not already have it, it can explicitly request this label binding from its LDP peer by sending a Label Request message. A Label Request message contains the FEC for which a label is being requested. The receiving LSR then responds to a Label Request message with a Label Mapping message for the requested FEC if it has such a binding. Otherwise, it responds with a Notification message indicating why it cannot satisfy the request.

Whereas Label Mapping messages create the bindings between FECs and labels, Label Withdraw messages break them. An LSR sends a Label Withdraw message to an LDP peer to signal that the peer should not continue to use specified label bindings that the LSR previously advertised. A Label Withdraw message contains the FEC for which the label binding is being withdrawn and optionally the originally advertised label. If no Label TLV is included in a Label Withdraw message, all labels that are associated with the FEC are to be withdrawn. Otherwise, only the label that is specified in the Label TLV is to be withdrawn.

An LSR that receives a Label Withdraw message must acknowledge it with a Label Release message. The LSR also uses Label Release messages to indicate that it no longer needs specific label bindings previously requested of or advertised by its LDP peer. A Label Release message contains the FEC for which the label binding is being released and optionally the originally advertised label. If no Label TLV is included in a Label Release message, all labels that are associated with the FEC are to be released. Otherwise, only the label that is specified in the Label TLV is to be released.

Label Advertisement Mode

The MPLS architecture specifies two label advertisement modes. If an LSR explicitly requests a label binding for a particular FEC from the next-hop LSR of this FEC, it uses *downstream on-demand* label advertisement mode. If an LSR advertises label bindings to its LDP peers that have not explicitly requested them, it uses *downstream unsolicited* advertisement mode.

Choosing which label advertisement mode to use depends on the characteristics of a particular MPLS implementation and application. Between each pair of LDP peers, they must have the same label advertisement mode.

Label Distribution Control Mode

Label distribution control determines how LSPs are established initially, and it has two modes: *independent* and *ordered* label distribution control.

With independent label distribution control, each LSR advertises label bindings to its peers at any time. It does not wait for the downstream or next-hop LSR to advertise the label binding for the FEC that is being distributed in the upstream direction. A consequence of using independent mode is that an upstream label can be advertised before a downstream label is received.

When an LSR is using ordered label distribution control, it cannot advertise a label binding for an FEC unless it has a label binding for the FEC from the downstream or next-hop LSR. It has to wait for the downstream LSR to advertise the label binding for the FEC that is being distributed in the upstream direction. As a result, ordered control makes the label distribution of a given LSP occur sequentially from the last hop of the LSP toward the first hop of the LSP.

Label Retention Mode

When an LSR receives a label binding for an FEC from a peer that is not the next hop for the FEC, it has the option to either store or discard the label binding based on the label retention mode in use.

Conservative label retention keeps only the label bindings that will be used to forward packets. The main advantage is that only the labels that are required for data forwarding are allocated and maintained. Because downstream on-demand advertisement mode is mainly employed when the label space is limited, it is normally used with the conservation label retention mode.

With *liberal label retention*, an LSR keeps every label binding it receives from its LDP peers regardless of whether the peers are the next-hop LSRs for the advertised label binding. The main advantage is that an LSP can be updated quickly when the label forwarding information is changed. Liberal label retention is mainly used where the label space is considered an inexpensive resource. When it is used with downstream unsolicited advertisement mode, liberal label retention reduces the total number of label advertisement messages required to set up LSPs. If an LSR is using conservative retention mode in this scenario, it has to send Label Request messages to the peer for the label bindings that it has discarded during the initial label advertisement if that peer becomes the next-hop LSR for the FECs that are being requested.

LDP Security

LDP uses TCP for transport of LDP messages. The LDP specification does not provide its own security measures but leverages the existing TCP MD5 authentication mechanism defined in RFC 1321 and also used by BGP in RFC 2385. MD5

authentication uses a message digest to validate the authenticity and integrity of an LDP message.

A message digest is calculated with the MD5 hash algorithm that uses a shared secret key and the contents of the TCP segment. Unlike clear-text passwords, message digest prevents the shared secret from being snooped. In addition to protecting against spoofing, MD5 authentication provides good protection against denial of service (DoS) and man-in-the-middle attacks.

Understanding AToM Operations

In Chapter 3, you learned how AToM achieves a high degree of scalability by using the MPLS encoding method. You also read an overview of LDP in the previous section. Reading through this section, you will develop a further understanding of how MPLS encapsulation, LDP signaling, and pseudowire emulation work together.

The primary tasks of AToM include establishing pseudowires between provider edge (PE) routers and carrying Layer 2 packets over these pseudowires. The next sections cover the operations of AToM from the perspectives of both the control plane and the data plane as follows:

- Pseudowire label binding
- Establishing AToM pseudowires
- Control word negotiation
- Using sequence numbers
- Pseudowire encapsulation

Pseudowire Label Binding

An AToM pseudowire essentially consists of two unidirectional LSPs. Each is represented by a pseudowire label, also known as a VC label. The pseudowire label is part of the label stack encoding that encapsulates Layer 2 packets going over AToM pseudowires. Refer to Chapter 3 for an overview of an AToM packet.

The label distribution procedures that are defined in LDP specifications distribute and manage the pseudowire labels. To associate a pseudowire label with a particular Layer 2 connection, you need a way to represent such a Layer 2 connection. The baseline LDP specification only defines Layer 3 FECs. Therefore, the pseudowire emulation over MPLS application defines a new LDP extensionthe Pseudowire ID FEC elementthat contains a pseudowire identifier shared by the pseudowire endpoints. Figure 6-8 depicts the Pseudowire ID FEC element encoding.

Figure 6-8. *Pseudowire ID FEC Element*

Pseudowire ID FEC (1 Octet) [=128]	C	Pseudowire Type (15 Bits)	Pseudowire Info Length (1 Octet)

Group ID (4 Octets)
Pseudowire ID (4 Octets)
Interface Parameters (Variable Length)

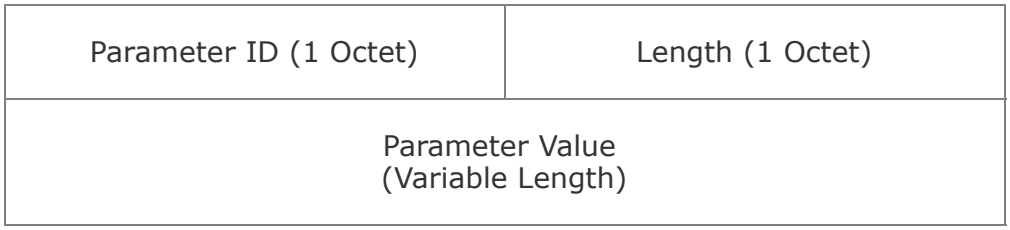
The Pseudowire ID FEC element has the following components:

- **Pseudowire ID FEC** The first octet has a value of 128 that identifies it as a Pseudowire ID FEC element.
- **Control Word Bit (C-Bit)** The C-bit indicates whether the advertising PE expects the control word to be present for pseudowire packets. A control word is an optional 4-byte field located between the MPLS label stack and the Layer 2 payload in the pseudowire packet. The control word carries generic and Layer 2 payload-specific information. If the C-bit is set to 1, the advertising PE expects the control word to be present in every pseudowire packet on the pseudowire that is being signaled. If the C-bit is set to 0, no control word is expected to be present.
- **Pseudowire Type** PW Type is a 15-bit field that represents the type of pseudowire. Examples of pseudowire types are shown in Table 6-1.
- **Pseudowire Information Length** Pseudowire Information Length is the length of the Pseudowire ID field and the interface parameters in octets. When the length is set to 0, this FEC element stands for all pseudowires using the specified Group ID. The Pseudowire ID and Interface Parameters fields are not present.
- **Group ID** The Group ID field is a 32-bit arbitrary value that is assigned to a group of pseudowires.
- **Pseudowire ID** The Pseudowire ID, also known as VC ID, is a non-zero, 32-bit identifier that distinguishes one pseudowire from another. To connect two attachment circuits through a pseudowire, you need to associate each one with the same Pseudowire ID.
- **Interface Parameters** The variable-length Interface Parameters field provides attachment circuit-specific information, such as interface MTU, maximum number of concatenated ATM cells, interface description, and so on. Each interface parameter uses a generic TLV encoding, as shown in Figure 6-9.

Table 6-1. *Pseudowire Types*

Pseudowire Type	Description
0x0001	Frame Relay data-link connection identifier (DLCI)
0x0002	ATM AAL5 service data unit (SDU) virtual channel connection (VCC)
0x0003	ATM Transparent Cell
0x0004	Ethernet VLAN
0x0005	Ethernet
0x0006	High-Level Data Link Control (HDLC)
0x0007	PPP

Figure 6-9. *Interface Parameter Encoding*



Even though LDP allows multiple FEC elements encoded into an FEC TLV, only one FEC elementthe Pseudowire ID FEC elementexists in each FEC TLV for the pseudowire emulation over MPLS application.

Establishing AToM Pseudowires

Typically, two types of LDP sessions are involved in establishing AToM pseudowires. They are the nontargeted LDP session and the targeted LDP session.

The nontargeted LDP session that is established through LDP basic discovery between a PE router and its directly connected P routers is used to distribute tunnel labels. The label distribution and management of tunnel labels pertains to the deployment model of the underlying MPLS network. It can be some combination of downstream on-demand or unsolicited label advertisement, independent or ordered control, and conservative or liberal label retention. Neither pseudowire emulation nor AToM dictates any particular label distribution and management mode for tunnel labels.

Note

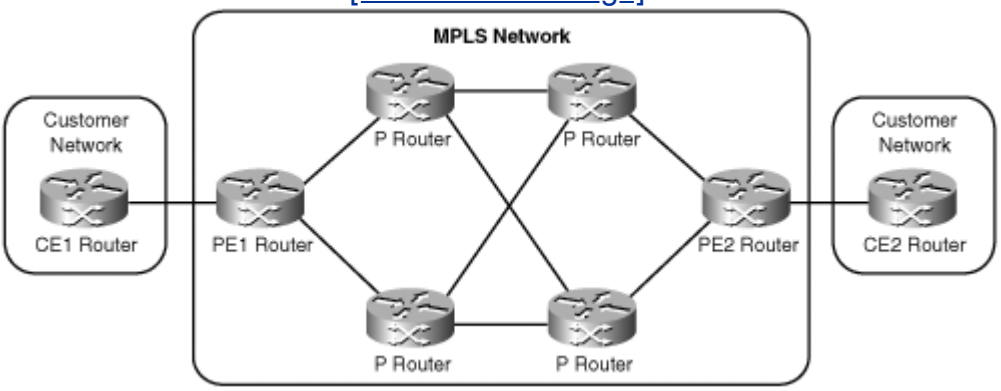
In some MPLS deployment scenarios, tunnel LSPs are set up through Resource Reservation Protocol Traffic Engineering (RSVP-TE) instead of nontargeted LDP sessions.

The other type of LDP sessions are established through LDP extended discovery between PE routers. These sessions are known as targeted LDP sessions because they send periodic Targeted Hello messages to each other. Targeted LDP sessions in the context of pseudowire emulation distribute pseudowire labels. IETF documents on pseudowire emulation over MPLS specify the use of downstream unsolicited label advertisement. In Cisco IOS Software, AToM uses independent label control and liberal label retention to improve performance and convergence time on pseudowire signaling.

Figure 6-10 illustrates an example of AToM deployment.

Figure 6-10. *AToM Deployment Model*

[View full size image]



The following steps explain the procedures of establishing an AToM pseudowire:

1. A pseudowire is provisioned with an attachment circuit on PE1.

2. PE1 initiates a targeted LDP session to PE2 if none already exists. Both PE routers receive LDP Keepalive messages from each other and complete the session establishment. They are ready to exchange pseudowire label bindings.

3. When the attachment circuit state on PE1 transitions to up, PE1 allocates a local pseudowire label corresponding to the pseudowire ID that is provisioned for the pseudowire.

4. PE1 encodes the local pseudowire label into the Label TLV and the pseudowire ID into the FEC TLV. Then it sends this label binding to PE2 in a Label Mapping message.

5. PE1 receives a Label Mapping message from PE2 and decodes the pseudowire label and pseudowire ID from the Label TLV and FEC TLV.

6. PE2 performs Steps 1 through 5 independently.

7. After PE1 and PE2 exchange the pseudowire labels and validate interface parameters for a particular pseudowire ID, the pseudowire with that pseudowire ID is considered established.

If one attachment circuit on one PE router goes down, a Label Withdraw message is sent to the peering PE router to withdraw the pseudowire label that it previously advertised.

Control Word Negotiation

During pseudowire establishment, Label Mapping messages are sent in both directions. To enable the pseudowire, you need to set some interface parameters to certain values that the peering PE router expects. When a mismatch occurs, fixing the problem requires manual intervention or configuration changes. The protocol cannot correct the mismatch automatically. For example, when the interface MTUs of the peering PE routers are different, the pseudowire is not established.

You can negotiate the presence of the control word through protocol signaling. The control word has 32 bits, as shown in Figure 6-11. If it is present, the control word is encapsulated in every pseudowire packet and carries per-packet information, such as sequence number, padding length, and control flags.

Figure 6-11. *AToM Control Word*

Reserved (4 Bits)	Control Flags (6 Bits)	Length (6 Bits)
Sequence Number (16 Bits)		

For certain Layer 2 payload types that are carried over pseudowires, such as Frame Relay DLCI and ATM AAL5, the control word must be present in the pseudowire encapsulation. That means you must set the C-bit in the pseudowire ID FEC element to 1 in both Label Mapping messages. When you receive a Label Mapping message that requires the mandatory control word but has a C-bit of 0, a Label Release message is sent with an Illegal C-bit status code. In this case, the pseudowire is not enabled.

For other Layer 2 payload types, the control word is optional. If a PE router cannot send and receive the optional control word, or if it is capable of doing that but prefers not to do so, the C-bit in the Label Mapping message that the PE router sends is set to 0. If a PE router is capable of and prefers sending and receiving the optional control word, the C-bit in the Label Mapping message it sends is set to 1. When two PE routers exchange Label Mapping messages, one of the following scenarios could happen when the control word is optional:

- Both C-bits are set to the same valuethat is, either 0 or 1. In this case, the pseudowire establishment is complete. The control word is used if the common C-bit value is 1. Otherwise, the control word is not used.

- A PE router receives a Label Mapping message but has not sent a Label Mapping message for the pseudowire, and the local C-bit setting is different from the remote C-bit setting. If the received Label Mapping message has the C-bit set to 1, in this case, the PE router ignores the received Label Mapping message and continues to wait for the next Label message for the pseudowire. If the received Label Mapping message has the C-bit set to 0, the PE router changes the local C-bit setting to 0 for the Label Mapping message to be sent. If the attachment circuit comes up, the PE router sends a Label Mapping message with the latest local C-bit setting.

- A PE router has already sent a Label Mapping message, and it receives a Label Mapping message from a remote PE router. However, the local C-bit setting is different from the remote C-bit setting. If the received Label Mapping message has the C-bit set to 1, in this case, the PE router ignores the received Label Mapping message and continues to wait for the next label message for the pseudowire. If the received Label Mapping message has the C-bit set to 0, the PE router sends a Label Withdraw message with a Wrong C-bit status code, followed by a Label Mapping message with the C-bit set to 0. The pseudowire establishment is now complete, and the control word is not used.

To summarize the previous two scenarios, when the C-bit settings in the two Label Mapping messages do not match, the PE router that prefers the use of the option control word surrenders to the PE router that does not prefer it, and the control word is not used.

Configuring whether the control word is to be used in an environment with many different platforms is sometimes a tedious process. AToM automates this task by detecting the hardware capability of the PE router. AToM always prefers the presence of the control word and utilizes the control word negotiation procedures to reach a common C-bit value between PE routers.

Using Sequence Numbers

Because Layer 2 packets are normally transported over Layer 1 physical media directly, most Layer 2 protocols assume that the underlying transport ensures in-order packet delivery. These protocols might not function correctly if out-of-order delivery occurs. For instance, if PPP LCP packets are reordered, the end-to-end PPP connection is unable to establish.

To avoid out-of-order packets, the best solution is to engineer a reordering-free packet network. Even though this goal is not always easy to achieve, you should make it a priority because no matter what kind of remedy you might use, network performance suffers significantly from out-of-order delivery.

Sequencing that is defined in pseudowire emulation mainly serves a detection mechanism for network operators to troubleshoot occasional out-of-order delivery problems. Implementations might choose to either discard or reorder out-of-order packets when they are detected. Because the latter requires huge packet buffer space for high-speed links and has significant performance overhead, AToM simply discards out-of-order packets and relies on the upper layer to retransmit these packets.

The first step in using sequencing is to signal the presence of the control word, as described in the previous section. The control word contains a 16-bit Sequence Number field. However, the presence of the control word does not mandate sequencing. When sequencing is not used, Sequence Number value is set to 0.

After negotiating the control word, the sequence number is set to 1 and increments by 1 for each subsequent packet that is being transmitted. If the transmitting sequence number reaches the maximum value 65535, it wraps around to 1 again.

To detect an out-of-order packet, the receiving PE router calculates the expected sequence number for the next packet by using the last receiving sequence number (which has an initial value of 0) plus 1, and then mod (modulus) by 2^{16} (2^{16} = 65536). If the result is 0, the expected sequence number is set to 1. A packet that is received over a pseudowire is considered in-order if one of the following conditions is met:

- The receiving sequence number is 0.

- The receiving sequence number is no less than the expected sequence number and the result of the receiving sequence number minus the expected sequence number is less than 32768.

- The receiving sequence number is less than the expected sequence number and the result of the expected sequence number minus the receiving sequence number is no less than 32768.

If none of these conditions is satisfied, the packet is considered out-of-order and is discarded.

Sometimes the sending or the receiving PE router might lose the last transmitting or receiving sequence number because of transient system problems. This router might want to restart the sequence number from the initial value. AToM implements a set of signaling procedures to reliably resynchronize the sequence number. Although the IETF documents do not specify these procedures, the procedures are interoperable with any standard-compliant implementation. The resynchronization procedures in AToM are as follows:

- If the transmitting PE router needs to reset the transmitting sequence number, it must inform the receiving PE router to reset the receiving sequence number. AToM accomplishes this by letting the transmitting PE router send a Label Release message to the receiving PE router, followed by a Label Request message. Because the receiving PE router interprets this as a pseudowire flapping, it resets the receiving sequence number.

- If the receiving PE router needs to reset the receiving sequence number, it must inform the receiving PE router to reset the transmitting sequence number. AToM does so by letting the receiving PE router send a Label Withdraw message to the transmitting PE router, followed by a Label Mapping message. Because the transmitting PE router perceives this as a pseudowire flapping, it resets the transmitting sequence number.

Pseudowire Encapsulation

To properly emulate Layer 2 protocols over pseudowires, you need to encapsulate each Layer 2 payload in such a way that Layer 2 characteristics are preserved as close to what they are in the native form as possible.

Aside from the MPLS label stack, pseudowire encapsulation also contains payload-specific information that varies on a per-transport and per-packet basis. This section discusses the payload-specific part of the encapsulation, which includes the control word and the Layer 2 payload.

The next few sections explain how the following Layer 2 protocols are encapsulated and processed on PE routers:

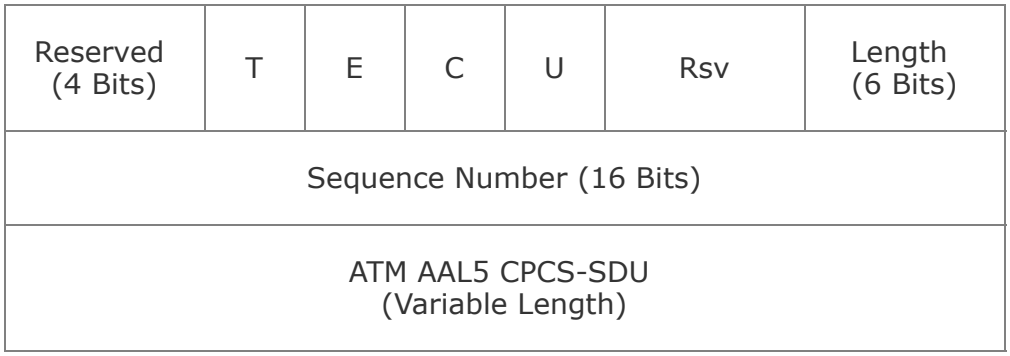
- ATM
- Frame Relay
- HDLC
- PPP
- Ethernet

ATM

AToM supports two types of encapsulation for ATM transport: ATM AAL5 common part convergence sublayer service data unit (CPCS-SDU) and ATM Cell.

The ATM AAL5 CPCS-SDU encapsulation includes a mandatory control word. The ATM AAL5 CPCS-SDU encapsulation requires segmentation and reassembly (SAR) on the CE-PE ATM interface. When an ingress PE router receives ATM cells from a CE router, it reassembles them into an AAL5 CPCS-SDU and copies ATM control flags from the cell header into the control word before sending it over a pseudowire. The AAL5 CPCS-SDU is segmented into ATM cells with proper cell headers on the egress PE router. Figure 6-12 illustrates the AAL5 CPCS-SDU pseudowire encapsulation.

Figure 6-12. *AAL5 CPCS-SDU Pseudowire Encapsulation*



The control flags in the control word are described as follows:

- **Transport Type (T-Bit)** This bit indicates whether the packet contains an ATM Operation, Administration, and Maintenance (OAM) cell or an AAL5 CPCS-SDU. If T = 1, the packet contains an ATM OAM cell. Otherwise, it contains an AAL5 CPCS-SDU. Being able to transport an ATM OAM cell in the AAL5 mode provides a way to enable administrative functionality over AAL5 VC.

- **EFCI (E-Bit)** The E-bit stores the value of the EFCI bit of the last cell to be reassembled when the payload contains an AAL5 CPCS-SDU or that of the ATM OAM cell when the payload is an ATM OAM cell on the ingress PE router. The egress PE router then sets the EFCI bit of all cells to the value of the E-bit.

- **CLP (C-Bit)** This is set to 1 if the CLP bit of any cell is set to 1 regardless of whether the cell is part of an AAL5 CPCS-SDU or is an ATM OAM cell on the ingress PE router. The egress PE router sets the CLP bit of all cells to the value of the C-bit.

- **Command/Response Field (U-Bit)** When FRF.8.1 Frame Relay/ATM PVC Service Interworking traffic is being transmitted, the CPCS-UU Least Significant Bit of the AAL5 CPCS-SDU might contain the Frame Relay C/R bit. This flag carries that bit from the ingress PE router to the egress PE router.

With the ATM Cell encapsulation, ATM cells are transported individually without SAR. The ATM Cell encapsulation consists of the optional control word and one or more ATM cells. Each ATM cell has a 4-byte ATM cell header and a 48-byte ATM cell payload. Figure 6-13 illustrates the ATM cell pseudowire encapsulation.

Figure 6-13. ***ATM Cell Pseudowire Encapsulation***

<table>
<tr><td colspan="4">Optional Control Word (4 Octets)</td></tr>
<tr><td>VPI (12 Bits)</td><td>VCI (16 Bits)</td><td>PTI
(3 Bits)</td><td>C</td></tr>
<tr><td colspan="4">ATM Payload (48 Octets)</td></tr>
<tr><td>VPI (12 Bits)</td><td>VCI (16 Bits)</td><td>PTI
(3 Bits)</td><td>C</td></tr>
<tr><td colspan="4">ATM Payload (48 Octets)</td></tr>
<tr><td colspan="4">▪▪▪</td></tr>
</table>

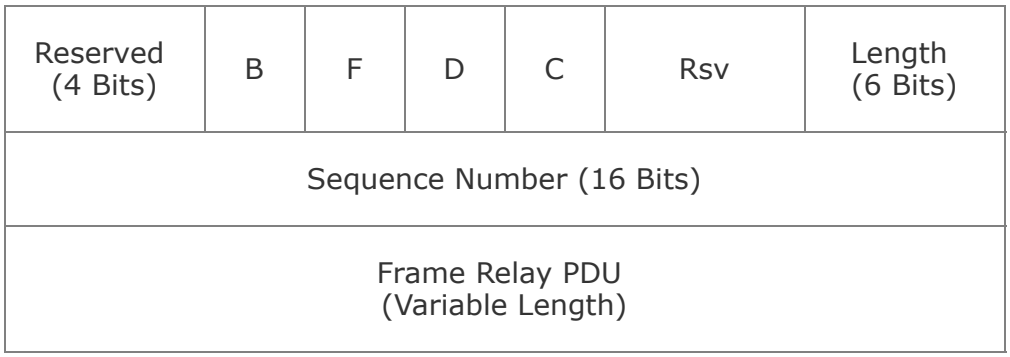
The maximum number of ATM cells that an ingress PE router can fit into a single pseudowire packet is constrained by the network MTU and the number of ATM cells that the egress PE router is willing to receive. This is signaled to the ingress PE

router through the interface parameter "maximum number of concatenated ATM cells" in the Label Mapping message.

Frame Relay

Frame Relay DLCIs are locally significant, and it is likely that two Frame Relay attachment circuits that are connected through a pseudowire have different DLCIs. Therefore, you do not need to include DLCI as part of the Frame Relay pseudowire encapsulation. The control word is mandatory. Control flags in the Frame Relay header are mapped to the corresponding flag fields in the control word. Frame Relay payloads that are carried over pseudowires do not include the Frame Relay header or the FCS. Figure 6-14 illustrates the Frame Relay pseudowire encapsulation.

Figure 6-14. *Frame Relay Pseudowire Encapsulation*



The Frame Relay control flags in the control word are described as follows:

- **Backward Explicit Congestion Notification (B-Bit)** The ingress PE router copies the BECN field of an incoming Frame Relay packet into the B-bit. The B-bit value is copied to the BECN field of the outgoing Frame Relay packet on the egress PE router.
- **Forward Explicit Congestion Notification (F-Bit)** The ingress PE router copies the FECN field of an incoming Frame Relay packet into the F-bit. The F-bit value is copied to the FECN field of the outgoing Frame Relay packet on the egress PE router.
- **Discard Eligibility (D-Bit)** The ingress PE router copies the DE field of an incoming Frame Relay packet into the D-bit. The D-bit value is copied to the DE field of the outgoing Frame Relay packet on the egress PE router.

- **Command/Response (C-Bit)** The ingress PE router copies the C/R field of an incoming Frame Relay packet into the C-bit. The C-bit value is copied to the C/R field of the outgoing Frame Relay packet on the egress PE router.

HDLC

HDLC mode provides port-to-port transport of HDLC encapsulated frames. The pseudowire HDLC encapsulation consists of the optional control word, HDLC address, control and protocol fields without HDLC flags, and the FCS.

You can also use the HDLC mode to transport Frame Relay User-to-Network Interface (UNI) or Network-to-Network Interface (NNI) traffic port-to-port transparently because they use HDLC framing.

PPP

PPP mode provides port-to-port transport of PPP encapsulated frames. The PPP pseudowire encapsulation consists of the optional control word and the protocol field without media-specific framing information, such as HDLC address and control fields or FCS.

When you enable the Protocol Field Compression (PFC) option in PPP, the Protocol field is compressed from two octets into a single octet. PFC occurs between CE routers and is transparent to PE routers. PE routers transmit the protocol field in its entirety as it is received from CE routers.

If the CE-PE interface uses HDLC-like framing, the ingress PE router always strips off HDLC address and control fields from the PPP frames before transporting them over pseudowires. Perhaps two CE routers negotiate Address and Control Field Compression (ACFC). The egress PE router has no way of knowing that unless it snoops into the PPP LCP negotiation between the CE routers, and that is normally undesirable because of system complexities and performance overhead. Therefore, the egress PE router cannot determine whether it should add HDLC address and control fields for PPP frames that are being sent to the CE router.

In Cisco IOS, AToM uses a simple solution to solve this problem without snooping. Basically, the PPP specification says that a PPP implementation that supports HDLC-like framing must prepare to receive PPP frames with uncompressed address and control fields at all times regardless of ACFC. So with AToM, the egress PE router always adds HDLC address and control fields back to the PPP packet if the egress CE-PE interface uses HDLC-like framing. For interfaces that do not use HDLC-like framing, such as PPP over Ethernet, PPP over Frame Relay, and PPP over ATM AAL5, the egress PE router does not add HDLC address and control fields to the PPP packet.

Ethernet

With the Ethernet pseudowire encapsulation, the preamble and FCS are removed from the Ethernet frames on the ingress PE router before sending them over pseudowires, and they are regenerated on the egress PE router. The control word is optional.

Ethernet pseudowires have two modes of operations:

- **Raw mode** In raw mode, an Ethernet frame might or might not have an IEEE 802.1q VLAN tag. If the frame does have this tag, the tag is not meaningful to both the ingress and egress PE routers.

- **Tagged mode** In tagged mode, each frame must contain an IEEE 802.1q VLAN tag. The tag value is meaningful to both the ingress and egress PE routers.

To explain how ingress and egress PE routers process a VLAN tag, it is necessary to define the semantics for the VLAN tag first. For example, when the ingress PE receives an Ethernet frame from a CE router and the frame contains a VLAN tag, there are two possible scenarios:

- The VLAN tag is a service delimiter. The provider uses a *service delimiter* to distinguish one type of customer traffic from another. For example, each service-delimiting VLAN tag can represent a different customer who the provider is serving or a particular network service that the provider wants to offer. Some equipment that the provider operates usually places this VLAN tag onto the Ethernet frame.

- The VLAN tag is not a service delimiter. A CE router or some equipment that the customer operates usually places this VLAN tag. The VLAN tag is not meaningful to the ingress PE router.

If an Ethernet pseudowire operates in raw mode, a service-delimiting VLAN tag, if present, is removed from the Ethernet frame that is received from a CE router before the frame is sent over the pseudowire. If the VLAN tag is not a service delimiter, it is passed across the pseudowire transparently.

If an Ethernet pseudowire operates in tagged mode, each Ethernet frame that is sent over the pseudowire must have a VLAN tag, regardless of whether it is a service-delimiting VLAN tag.

In both modes, the service-delimiting VLAN tags have only local significance. That is, these tags are meaningful only at a particular CE-PE interface. When the egress PE router receives an Ethernet frame from the pseudowire, it references the operation mode and its local configuration to determine how to process this frame before transmitting it to the CE router. If the egress PE is using raw mode, it might add a service-delimiting VLAN tag, but it will not rewrite or remove a VLAN tag that is already present in the frame. If the egress PE is using tagged mode, it can rewrite, remove, or keep the VLAN tag that is present in the frame.

In Metro Ethernet deployment, in which CE routers and PE routers are connected through an Ethernet switched access network, packets that arrive at PE routers can contain two IEEE 802.1q VLAN tags. This type of packet is commonly known as a QinQ packet. When the outer VLAN tag is the service-delimiting VLAN tag, QinQ packets are processed exactly like the ones with a single VLAN tag in both raw mode and tagged mode. When the combination of the outer and inner VLAN tags is used for service-delimiting, it is processed as if it were a single VLAN tag but with an extended range of values.

If you need to take QoS into consideration, the ingress PE router can map the user priority bits in the VLAN header to the MPLS EXP bits in the MPLS label stack. In this way, transit LSRs in the MPLS network can apply QoS policies to the Ethernet frames that are carried over pseudowires.

Summary

This chapter first gave an overview of LDP, including LDP components and operations that are related to pseudowire emulation over MPLS. Then the chapter explained the control signaling and data switching details of AToM.

Despite multiple possible combinations of label distribution and management modes for pseudowire signaling, AToM implements the combination that uses LDP downstream unsolicited advertisement, independent control, and liberal retention modes.

Pseudowire emulation over MPLS introduces new protocol extensions and signaling procedures, such as the Pseudowire ID FEC element in the FEC TLV that is defined to represent a pseudowire that connects two attachment circuits on different PE routers, the negotiation of the control word presence, and sequence number resynchronization.

Pseudowire emulation over MPLS also specifies new encapsulation methods and data switching procedures, such as the control word that is customized for carrying transport-specific information, Layer 2 payload encapsulations, ingress/egress processing optimized for transporting over pseudowires, and the sequence number for detecting out-of-order packets.

Chapter 7. LAN Protocols over MPLS Case Studies

This chapter covers the following topics:

- Understanding Ethernet over MPLS technology
- EoMPLS transport case studies
- Common troubleshooting techniques

Chapter 6, "Understanding Any Transport over MPLS," introduced you to the general concepts of Any Transport over MPLS (AToM). In this chapter, you learn operation and configuration of Ethernet over MPLS (EoMPLS), one of the draft-martini-based AToM technologies. EoMPLS offers a way to connect geographically dispersed Ethernet networks. By deploying EoMPLS in their core, service providers can implement Ethernet VPN services.

This chapter outlines important aspects of the EoMPLS technology and provides step-by-step configuration procedures for enabling EoMPLS, primarily on the service provider's side. As you already know from previous chapters, the Layer 2 VPN is transparent to the end customer, so the configuration required for the enterprise's devices is minimal.

The case studies included in this chapter do not concentrate on configuration specifics that are native to different platforms. Instead, they provide generic configuration for routers and switches.

> **Note**
>
> This chapter relies heavily on knowledge and comprehension of concepts learned from previous chapters, especially Chapter 6. In some cases, simple references to these chapters are provided. In other cases (when necessary), certain concepts are reiterated.

Understanding Ethernet over MPLS Technology

EoMPLS, as specified in the draft-martini discussed in Chapter 6, allows Layer 2 Ethernet frames to be transported across a Multiprotocol Label Switching (MPLS) core network. For the label switch router (LSR) to switch Layer 2 virtual circuits (VC), it must have IP connectivity to transport any Layer 2 attachment services. Thus, the edge LSRs must have the capability to switch Layer 2 VCs. EoMPLS has several mechanisms in place to support such transport. These mechanisms are further explained in the following sections:

- EoMPLS Label Stack
- Supported VC Types
- Label Imposition
- Label Disposition

EoMPLS Label Stack

Most common EoMPLSs use a two-level label stack. (The VC label should always have at least one label, and the packet-switched network (PSN) label should have zero or more label stack entries.) This means that the Ethernet packets transported between the egress and ingress LSRs are sent containing two labels: the top, or outer, and the bottom, or inner. The outer label, also known as the tunnel label or Interior Gateway Protocol (IGP) label, sends packets over the MPLS backbone. The egress LSR assigns the inner label, known as the VC label (or pseudowire) label). The VC label identifies the attachment circuit on the egress LSR. The egress LSR binds the Layer 2 egress interface with the configured VC ID and sends the VC label to the ingress provider edge (PE) by using the targeted Layer Distribution Protocol (LDP) session. For more information about a targeted LDP session, review the "Establishing AToM Pseudowires" section of Chapter 6.

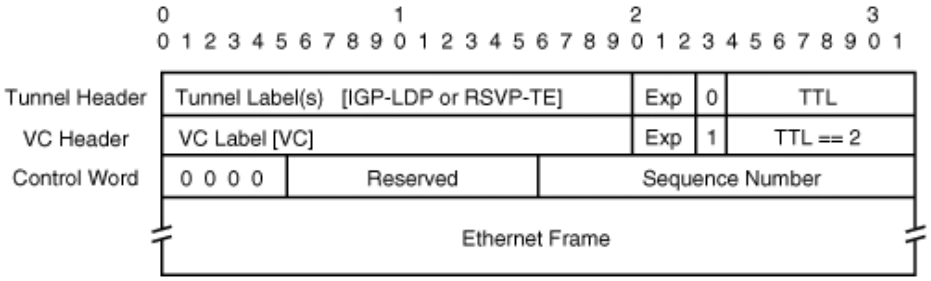
The next two sections describe the EoMPLS packet format and maximum transmission unit (MTU) size requirements.

Packet Format

Figure 7-1 demonstrates the EoMPLS encapsulation format, but it also applies to other forms of transport over MPLS.

Figure 7-1. EoMPLS Packet Format

[View full size image]



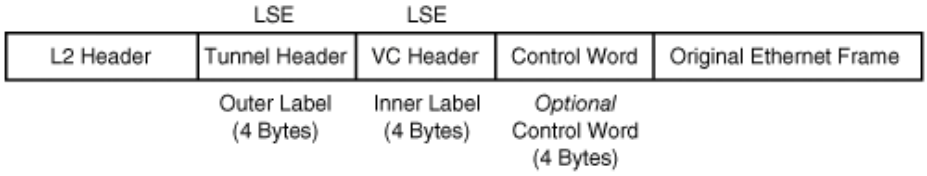
The bottom VC label and the top tunnel label comprise the two levels of the label stack. The tunnel label switches packets from the ingress PE to the egress PE. The ingress LSR sets the VC label's Time to Live (TTL) field to a value of 2 (in this case), and it sets the TTL of the tunnel label to 255. To indicate that the VC label is at the bottom of the stack, the ingress PE marks the VC label's end-of-stack bit with the value of 1.

MTU Size Requirements

In the most common scenario, the two level stack entries of the EoMPLS label stack append 8 bytes to a Layer 2 frame (4 bytes each). In addition, an optional 4-byte control word is always preferred in Cisco routers. Figure 7-2 shows the overhead introduced by the addition of the tunnel and the VC labels plus the optional control word on top of the original Ethernet frame and the L2 header.

Figure 7-2. EoMPLS Label Stack Overhead

[View full size image]



Assuming that Layer 2 in the packet-switched network (PSN) is Ethernet in Figure 7-2, the PSN Layer 2 header contains the following:

- **Destination address (DA)/source address (SA)**12 bits
- **Protocol Ethertype 0x8847**2 bytes

The MPLS label is set to 0x8847, which indicates that the frame carries an MPLS unicast packet. If the PSN uses a different Layer 2 technology, the upper layer identification specifies an MPLS packet, such as Cisco High-Level Data Link Control (HDLC) Type or PPP Data Link Layer protocol.

Note

The preceding fields describe the Layer 2 header in an Ethernet PSN. Do not confuse them with the Ethernet header fields in the transported Layer 2 frame from the customer edge (CE) device.

In addition, Figure 7-2 shows a label stack of 2 label stack entries (LSE), each containing the following:

- 20-bit label
- 3-bit Experimental Field (Exp)
- 1-bit Bottom of Stack Indicator (S)
- 1-byte TTL

Finally, Figure 7-2 shows an optional 4-byte control word and the original Ethernet frame. The original Ethernet frame's header from the CE device that is transported in EoMPLS is at least 14 bytes and contains the following:

- **DA/SA**12 bits
- **Protocol Ethertype**Indicates the upper-layer protocol

In the case of 802.1q Ethernet VLAN transport, the Ethernet overhead is 18 bytes, with the addition of the 4-byte VLAN Tag header, also referred to as the 802.1q header. An Ethertype with a value of 0x8100 indicates that there is a VLAN Tag header between the Ethernet and upper-layer headers. The 802.1q header is as follows:

- 3 priority (P) bits
- Canonical Format Identifier (CFI) bit
- 12-bit VLAN ID (VID)
- 2-byte Ethertype field (indicates the higher-layer protocol, such as 0x0800 for IPv4)

Table 7-1 summarizes the information outlined previously.

Table 7-1. *Summary of Layer 2 and MPLS Components*

Field	Bits	How	Comments

Field	Bits	How	Comments
Destination MAC address	First 6 bytes.	FIB[1] lookup, included in the rewrite string.	From ARP[2].
Source MAC address	Next 6 bytes.	MAC address of the router or switch.	Field is always the same.
Ethertype	Next 2 bytes.	0x8847.	MPLS Unicast indicated by 0x8847.
Tunnel labelMPLS label	20 bits (bits 019 after MAC header).	Derived from PSN label for the remote PE's FEC[3].	Information can be obtained through FIB lookup.
Tunnel labelEXP[4] bits	3 bits. Bits 2022 of tunnel label.	Experimental bits in the tunnel PSN.	Intermediate LSRs can modify field when switching the packet through the PSN.
Tunnel labelS bit	1 bit. Bit 23 of tunnel label.	0.	The S bit in the Tunnel Label field is always set to 0 for the tunnel label to indicate that another LSE[5] follows.
Tunnel labelTTL field	8 bits. Bits 2431 of tunnel label.	Initially set to 255 or can be derived from the transported IP header.	This field is decremented in intermediate LSRs by means of TTL processing.

Field	Bits	How	Comments
VC labelMPLS label	20 bits. Bits 019 after tunnel label.	Derived from the incoming packet's VLAN tag and the ingress port (that is, incoming attachment circuit).	VC label is associated with the egress attachment circuit and advertised through LDP. You obtain this information from the FIB lookup.
VC labelEXP bits	3 bits. Bits 20-22 of VC label.	Same as the tunnel EXP bits.	You can configure this field.
VC labelS bit	1 bit. Bit 23 of VC label.	1.	Because the VC label is the last label in the MPLS label stack, the S bit is always set to 1.
VC labelTTL field	8 bits. Bits 2431 of VC label.	2.	This field is always set to 2.

[1] FIB = Forwarding Information Base

[2] ARP = Address Resolution Protocol

[3] FEC = forward error correction

[4] EXP = Experimental

[5] LSE = label stack entry

You can see a sample EoMPLS packet showing the fields from Example 7-1. The EoMPLS packet over Ethernet PSN that contains a two-label stack and control word transports an 802.1q VLAN frame that contains an Internet Control Message Protocol (ICMP) echo request (ping) sent from a CE device. The different layers of protocols are highlighted.

Example 7-1. ***EoMPLS Frame Sample (Ethernet Frame)***

```
Ethernet II
    Destination: 00:03:a0:19:c0:c2
    Source: 00:03:a0:19:c5:02
    eType: MPLS Unicast (0x8847)
MultiProtocol Label Switching Header
    MPLS Label: 16
    MPLS Experimental Bits: 2
    MPLS Bottom Of Label Stack: 0
    MPLS TTL: 253
MultiProtocol Label Switching Header
    MPLS Label: 16
    MPLS Experimental Bits: 2
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 2
AToM EoMPLS Header
    AToM MPLS Control Word: 0x00000000
Ethernet II
    Destination: aa:aa:aa:aa:aa:aa
    Source: bb:bb:bb:bb:bb:bb
    Type: 802.1Q Virtual LAN (0x8100)
802.1q Virtual LAN
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = CFI: 0
    .... 0000 0110 0100 = ID: 100
    Type: IP (0x0800)
    Trailer: 00000000000000000000
Internet Protocol
    Version: 4
    Header length: 20 bytes
    ! Output omitted for brevity
    Time to live: 255
    Protocol: ICMP (0x01)
    Header checksum: 0xa3fd (correct)
    Source: 10.1.2.203 (10.1.2.203)
    Destination: 10.0.0.201 (10.0.0.201)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xc15c (correct)
    Identifier: 0x000f
    Sequence number: 0x0000
    Data (8 bytes)
```

The overhead incurred when Ethernet frames were transported over MPLS and rules and restrictions were imposed on the MPLS network.

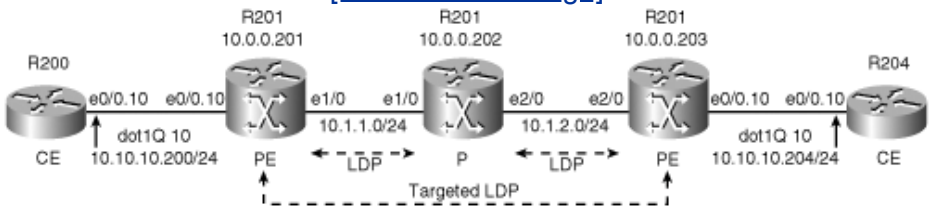
For instance, the MTU configuration of the MPLS network should accommodate the largest expected frame size in the label-switched paths (LSPs) plus the header (Ethernet frame header, in this case), control word, and 8 additional label stack bytes. This includes configuration of the CE and PE links.

Figure 7-3 illustrates a sample network over which you can see the MTU calculation for VLAN-tunneled modes. (Both modes are discussed in the "Supported VC Types" section of

this chapter.) To verify these calculations, you perform pings with different packet sizes from the CE R200 to the CE R204.

Figure 7-3. Calculating MTU Requirements

[View full size image]



Using the network depicted in Figure 7-3, you can see in Example 7-2 the different results when sending pings with the don't fragment (DF) bit set from R200 to R204, with sizes of 1470 and 1471 bytes, respectively.

Example 7-2. *Probing for MTU Limits in EoMPLS*

```
R200#ping ip 10.10.10.204 size 1470 timeout 1 df-bit

Type escape sequence to abort.
Sending 5, 1470-byte ICMP Echos to 10.10.10.204, timeout is 1 seconds:
Packet sent with the DF bit set
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/28/32 ms
R200#
R200#ping ip 10.10.10.204 size 1471 timeout 1 df-bit

Type escape sequence to abort.
Sending 5, 1471-byte ICMP Echos to 10.10.10.204, timeout is 1 seconds:
Packet sent with the DF bit set
.....
Success rate is 0 percent (0/5)
R200#
```

You can use the following formula to calculate MTU requirements for the core:

```
Core MTU >= Edge MTU + Transport Header + AToM Header + (MPLS Label Stack *
MPLS Header Size)
```

The Edge MTU is the MTU that is configured in the CE-facing PE's interface.

This formula uses the following values for VLAN transport and a two-label stack and provides the core MTU needed to transport 1500-byte packets from the CE:

```
Core MTU >= 1470 + 18 + 4 + (2 * 4)
Core MTU >= 1500
```

You input the value of 1470 bytes in the preceding formula as the edge MTU because that is the largest unfragmented packet that was successfully transported. The result of the formula is a core MTU that is greater than or equal to 1500 bytes, which is the actual MTU that is configured in the core.

On the other hand, if you want to transport 1500-byte packets from the CE device, you can substitute that value for the Edge MTU in the general formula to calculate the corresponding Core MTU needed:

```
Core MTU >= Edge MTU + 18 + 4 + (2 * 4)
Core MTU >= 1500 + 18 + 4 + (2 * 4)
Core MTU >= 1530
```

In this case, you need to configure the MTU links to allow for 1530-byte packets.

Table 7-2 outlines the MTU calculation to show that the overhead is 30 bytes. That is why only packets that are up to 1470 bytes with DF bit set are successfully transported in Example 7-2.

Table 7-2. *Calculating MTU Requirements for Ethernet VLAN Transport*

Layer	Description	Core Overhead
Transported	Ethernet VLAN	18 bytes
AToM	Control word	4 bytes
MPLS	MPLS stack entries * MPLS header size	2 headers * 4 bytes/header = 8 bytes
Total		**30 bytes**

Keeping in mind that the transport overhead for VLAN-tunneled is 18 bytes, the transport overhead for port-tunneled is 14 bytes, and that MPLS traffic engineering (TE) fast reroute (FRR) uses an additional label stack entry, you can see the MTU calculations for various cases in Table 7-3. (All values are in bytes.) Note that the sizes in square brackets indicate the values when the optional control word is not used.

Table 7-3. *Calculating MTU Requirements for EoMPLS Cases*

Field	Edge	Transport	AToM (Control Word)	MPLS Stack	MPLS Header	MPLS	Total
EoMPLS Port Mode	1500	14	4 [0]	2 LSEs	4 bytes/LSE	8	**1526 [1522]**
EoMPLS VLAN Mode	1500	18	4 [0]	2 LSEs	4 bytes/LSE	8	**1530 [1526]**
EoMPLS Port + TE FRR	1500	14	4 [0]	3 LSEs	4 bytes/LSE	8	**1530 [1526]**
EoMPLS VLAN + TE FRR	1500	18	4 [0]	3 LSEs	4 bytes/LSE	12	**1534 [1530]**

Under certain circumstances, the egress label edge router (LER) might receive a packet that exceeds the MTU of the egress interface. For instance, a PE might receive a jumbo frame that exceeds the default 1500 MTU in the core. The core must be able to transport larger frame bytes.

If the MTU is not increased between the PE and P routers and the encapsulated packet on the ingress PE exceeds the LSP MTU, the packet is dropped on the PE. Likewise, if a packet arrives at the egress LSR that exceeds the MTU of the egress Layer 2 interface (VLAN), it is dropped. In addition, the MPLS backbone does not support fragmentation of the Layer 2 packets. Therefore, the MTU of all intermediate circuits must be able to handle the biggest Layer 2 packet that is transported.

The MTU configuration of the ingress and egress PEs needs to match. Otherwise, a VC negotiation occurs, but it is rejected and the circuit fails to come up. Resolving this issue requires configuring the same MTU parameters on each side. To avoid packet drops in the core, the **mpls mtu** *number* on interfaces must be increased on P and PE routers to meet requirements.

Supported VC Types

As mentioned in the previous section, EoMPLS can operate in two modes: port-tunneling mode and VLAN-tunneling mode. Port-tunneling is also referred to as port-to-port transport. VLAN-tunneled interfaces are VC type 4, or, 0x0004, and port-tunneled interfaces are VC type 5, or 0x0005 (as specified in the draft-martini). Cisco devices support both VC types.

Note

The newest Cisco implementations provide for auto-sensing of the VC type.

In VLAN-tunneling mode, the ingress information for the VLAN is contained within the dot1Q header of the packet. (Refer to Chapter 4, "LAN Protocols," for more information on dot1Q.) By looking at the VLAN ID in the dot1Q header, the network processor (NP) can determine the next step in processing, described in the "Label Imposition" section of this chapter.

In port-tunneling mode, the packet does not have ingress port information. For inclusion of ingress information, the port-tunneled interface is put into the QinQ mode. A hidden VLAN is then created and added onto the packet. A *hidden VLAN* is a VLAN that is numbered outside the allowed range for VLAN IDs. This is how the NP learns the ingress information. The hidden VLAN concept applies to a switch platform (that is, 6500 and 7600 platforms). In contrast, VLAN-tunneled mode does not require a hidden VLAN. The NP can discern the ingress information from the packet's dot1Q header.

A similar concept applies to routers (VLAN stacking method). It involves the use of subinterfaces.

Another difference between the port-tunneling mode and VLAN-tunneling mode is in the handling of VLAN IDs. In port-tunneling mode, the VLAN ID is transparently passed from the ingress PE to the egress PE over MPLS in a single VLAN. In VLAN-tunneling mode, however, the VLAN ID at each end of the EoMPLS tunnel can be different. To overcome this, the egress side of the tunnel that is mapped to a VLAN rewrites the VLAN ID in outgoing dot1Q packets to the ID of the local VLAN.

Label Imposition

Earlier in this chapter, you learned about the format of the EoMPLS label stack. Adding these labels onto a packet is referred to as *label imposition*. You perform label imposition on the label imposition router or an ingress PE. Routers receive a Layer 2 packet and encapsulates it for the MPLS backbone.

Depending on whether the port-tunneling or VLAN-tunneling mode is used, the interfaces that receive the Layer 2 packets can be either Ethernet port interfaces or VLAN interfaces (or subinterfaces). To impose labels on packets, the ingress PE router maintains a table, which associates an EoMPLS tunnel with an interface/FEC. This table keeps the information needed for sending the packet, such as the outgoing interface and encapsulation.

The label imposition process involves the following:

1. A two-level label stack, described earlier in the "EoMPLS Label Stack" section of this chapter, is learned via the LDP for each VC. An NP also provides a MAC rewrite that contains the outgoing Layer 2 encapsulation in addition to the label stack that changes the encapsulation of the packet to comply with the outgoing interface.

2. The label stack and the output MAC encapsulation are prepended to the packet, and the packet is forwarded to the egress interface for transmission.

3. The outgoing VC label's TTL field is set to 2.

Label Disposition

Label disposition refers to label removal from a packet. It is performed at the egress PE, or the label disposition router. This router receives a packet, strips the bottom (VC) label (and the top [PSN] label if it exists, in cases such as explicit-null), and sends the remaining Layer 2 frame out of the egress attachment circuit interface.

After label imposition, the packets travel across the MPLS core network via standard label switching and arrive at the egress PE. At that point, they might already have their tunnel label removed and be left with only the VC label. This might have occurred because the next to last (penultimate) router "popped" the tunnel label prior to transmitting the packet to the egress PE. This process is commonly referred to as *Penultimate Hop Popping (PHP)*. If hop popping is supported at the penultimate router, the egress PE requests it. It advertises an implicit null label to its directly connected neighbor via BGP, which causes the neighbor to pop the tunnel label when switching the packets to the egress PE.

When the egress PE receives the packet with the VC label, it needs to select an appropriate form of disposition. For this, the egress PE checks the label forwarding information base (LFIB). The LFIB contains information about the binding between the outgoing interface and a given VC ID, which was initially inserted into the LFIB with the VC label. The LFIB lookup informs the PE that EoMPLS disposition will be performed and finds the corresponding egress interface for the VC. The VC label is then popped, the VLAN ID is rewritten (if needed), and the frame is transmitted to the proper outgoing interface.

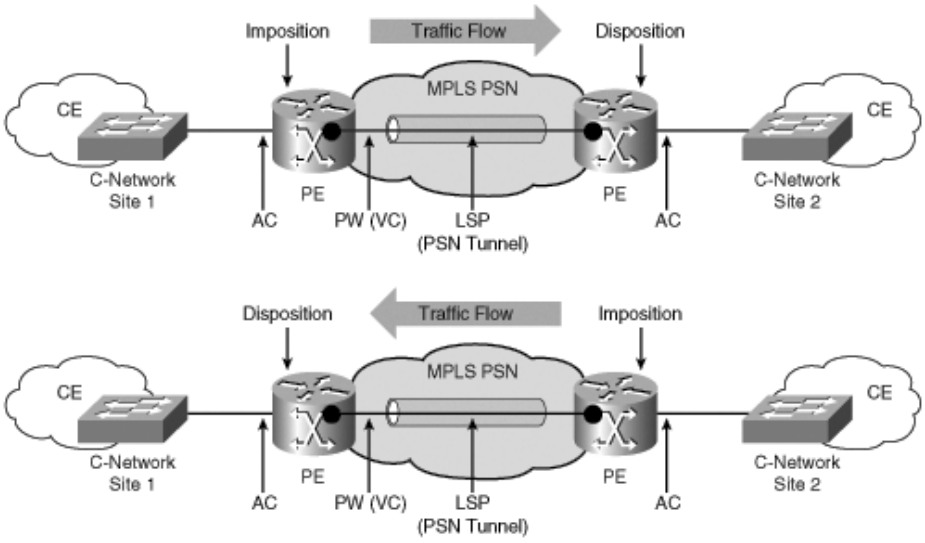
Note

The ingress and egress PEs are the only two routers in the MPLS backbone with knowledge of the Layer 2 transport VCs. No other intermediate hops have table entries for the Layer 2 transport VCs. Therefore, only PEs require EoMPLS functionality.

Figure 7-4 illustrates the process of imposition and disposition, where traffic flow is bound first from Site 1 to Site 2 and then in the opposite direction.

Figure 7-4. Label Imposition and Disposition

[View full size image]



The traffic flow over an EoMPLS VC, between the imposition and disposition PEs, follows the same path across the MPLS backbone, unless routing changes within the network of the provider cause the change.

Note

For an LSP to be present from PE to PE, routes from a PE that its neighbors discover cannot be summarized. They must have a mask of /32.

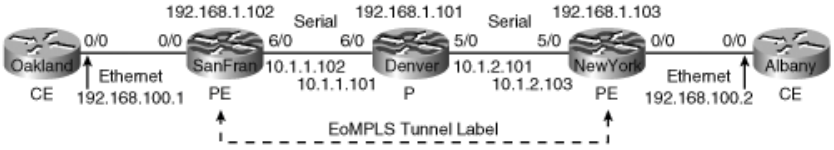
EoMPLS Transport Case Studies

This section demonstrates the configuration that is required to establish connectivity between the customer sites using EoMPLS transport. Although it is impossible to cover all possible situations, a wide range of EoMPLS scenarios is provided. The case studies do not concentrate on specific platforms; rather, they offer generic configuration for routers and switches. The beginning case studies show topologies and configurations that involve routers and move on to case studies that involve switches.

Figure 7-5 shows the general topology used throughout the case studies. Some variations from one case study to the next require configuration or topology modifications. The goal that is common to all case studies is to establish Layer 2 and higher connectivity between the two customer sites (Oakland and Albany) by extending Layer 2 across an MPLS-enabled and routed core network. Routed means that IP traffic is switched at Layer 2 and not bridged across the core.

Figure 7-5. EoMPLS Case Study Topology

[View full size image]



Prior to configuring tasks that are specific to each case study, you must complete configuration that is applicable to all case studies. This configuration involves the three service provider core routers: SanFran, Denver, and NewYork. The configuration tasks are as follows (not necessarily in this order):

- Assign IP addresses to all physical core links.
- Choose an interior IP routing protocol to propagate those networks.
- Configure loopback interfaces. Remember to use the /32 subnet mask because no summarization is allowed into an LDP-targeted session. For instance, the IP addresses that are employed in the case studies are 192.168.1.102 for SanFran, 192.168.1.101 for Denver, and 192.168.1.103 for NewYork.
- Enable CEF (necessary for MPLS to work) with the **ip cef** command. (On some higher-end router platforms, you need to enable distributed CEF [dCEF] instead.)
- Configure MPLS globally using the **mpls ip** command.
- Configure LDP to tell the routers to exchange MPLS labels with the **mpls label protocol ldp** command.
- Specify the loopback interface for the LDP router ID selection via the **mpls ldp router-id loopback# [force]** command.
- Enable MPLS on the router-to-router interfaces by using the **mpls ip** and **mpls label protocol ldp** interface commands.

Example 7-3 demonstrates the preceding configuration for SanFran, Denver, and NewYork, respectively.

Example 7-3. ***Required Preconfiguration***

```
hostname SanFran
!
ip cef
mpls ip
mpls label protocol ldp
mpls ldp router-id Loopback0 force

!
interface Loopback0
ip address 192.168.1.102 255.255.255.255
!
interface Serial6/0
ip address 10.1.1.102 255.255.255.0
no ip directed-broadcast
mpls label protocol ldp

mpls ip
!
router ospf 100
log-adjacency-changes detail
network 0.0.0.0 255.255.255.255 area 0
hostname Denver
!
ip subnet-zero
ip cef
mpls ip
mpls label protocol ldp
mpls ldp router-id Loopback0 force
!
interface Loopback0
 ip address 192.168.1.101 255.255.255.255
 no ip directed-broadcast
!
interface Serial5/0
 ip address 10.1.2.101 255.255.255.0
 no ip directed-broadcast
 mpls label protocol ldp
 mpls ip
!
interface Serial6/0
 ip address 10.1.1.101 255.255.255.0
 no ip directed-broadcast
 mpls label protocol ldp
 tag-switching ip
!
router ospf 100
 log-adjacency-changes detail
 network 0.0.0.0 255.255.255.255 area 0

hostname NewYork
!
ip subnet-zero
ip cef
mpls ip
mpls label protocol ldp
mpls ldp router-id Loopback0 force
!
interface Loopback0
 ip address 192.168.1.103 255.255.255.255
!
interface Serial5/0
 ip address 10.1.2.103 255.255.255.0
```

```
 mpls label protocol ldp
 mpls ip
!
router ospf 100
 log-adjacency-changes detail
 network 0.0.0.0 255.255.255.255 area 0
```

Note

Normally, you would not use network 0.0.0.0 when configuring your OSPF statements. Here, it is used strictly in a practice lab environment.

At this point, you should verify that basic connectivity between the core devices works before moving on to specific EoMPLS configuration. Apply the following verification and troubleshooting principles to each router. For brevity, output for only one router is shown for each step.

Check that the routes are being received via an IGP, as shown in Example 7-4.

Example 7-4. show ip route ospf *Command*

```
SanFran#show ip route ospf
     10.0.0.0/24 is subnetted, 2 subnets
O       10.1.2.0 [110/128] via 10.1.1.101, 00:11:11, Serial6/0
     192.168.1.0/32 is subnetted, 3 subnets
O       192.168.1.101 [110/65] via 10.1.1.101, 00:11:11, Serial6/0
O       192.168.1.103 [110/129] via 10.1.1.101, 00:11:11, Serial6/0
```

Verify that the MPLS-enabled interfaces are operationalin other words, that MPLS is enabled on an interface, as in Example 7-5.

Example 7-5. show mpls interfaces *Command*

```
Denver#show mpls interfaces
Interface            IP         Tunnel   Operational
Serial5/0            Yes (ldp)  No       Yes
Serial6/0            Yes (ldp)  No       Yes
```

Ensure that the PE routers have discovered the P router via the **show mpls ldp discovery** command, as shown in Example 7-6.

Example 7-6. show mpls ldp discovery *Command*

```
NewYork#show mpls ldp discovery
 Local LDP Identifier:
    192.168.1.103:0
    Discovery Sources:
    Interfaces:
        Serial5/0 (ldp): xmit/recv
            LDP Id: 192.168.1.101:0
```

You can also confirm that LDP sessions are established between the routers (see Example 7-7). You can see that NewYork has established a session with Denver.

Example 7-7. show mpls ldp neighbor *Command*

```
NewYork#show mpls ldp neighbor
    Peer LDP Ident: 192.168.1.101:0; Local LDP Ident 192.168.1.103:0
        TCP connection: 192.168.1.101.646 - 192.168.1.103.11004
        State: Oper; Msgs sent/rcvd: 10/10; Downstream
        Up time: 00:02:00
        LDP discovery sources:
          Serial5/0, Src IP addr: 10.1.2.101
        Addresses bound to peer LDP Ident:
          10.1.2.101      192.168.1.101   10.1.1.101
```

Another way of verifying whether the label forwarding table is built correctly is to issue the **show mpls forwarding-table** and **show mpls forwarding-table** detail commands, as in Example 7-8.

Example 7-8. show mpls forwarding-table *and* show mpls forwarding-table detail *Commands*

```
SanFran#show mpls forwarding-table
Local  Outgoing    Prefix           Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id     switched   interface
16     Pop tag     10.1.2.0/24      0          Se6/0       point2point
17     Pop tag     192.168.1.101/32 0          Se6/0       point2point
18     17          192.168.1.103/32 0          Se6/0       point2point
------------------------------------------------------------
SanFran#show mpls forwarding-table detail
Local  Outgoing    Prefix           Bytes tag   Outgoing     Next Hop
tag    tag or VC   or Tunnel Id     switched    interface
16     Pop tag     10.1.2.0/24      0           Se6/0        point2point
       MAC/Encaps=4/4, MRU=1504, Tag Stack{}
       0F008847
       No output feature configured
17     Pop tag     192.168.1.101/32  0          Se6/0        point2point
       MAC/Encaps=4/4, MRU=1504, Tag Stack{}
       0F008847
       No output feature configured
18     17          192.168.1.103/32  0          Se6/0        point2point
       MAC/Encaps=4/8, MRU=1500, Tag Stack{17}
       0F008847 00011000
       No output feature configured
```

You are now ready to begin the specialized EoMPLS case studies. They are as follows:

- Case Study 7-1: Router to RouterPort Based
- Case Study 7-2: Router to RouterVLAN Based
- Case Study 7-3: VLAN Rewrite
- Case Study 7-4: Switch to SwitchVLAN Based

- Case Study 7-5: Switch to SwitchPort Based
- Case Study 7-6: VLAN Rewrite in Cisco 12000 Series Routers
- Case Study 7-7: Map to Pseudowire

Case Study 7-1: Router to RouterPort Based

In this case study, you build on the preconfigured portion of the service provider core routers by using the topology presented in Figure 7-5. Your objective is to transport all customer traffic without utilizing 802.1q. In this case, CE devices are routers. You explore the CE switches scenario in Case Study 7-5, later in this chapter.

The port transparency feature is designed for Ethernet port-to-port transport, where the entire Ethernet frame without the preamble or FCS is transported as a single packet based on the VC type 5.

Configuring Port Transparency

Port transparency configuration involves the two PE routers: SanFran and NewYork. Keep all settings from the previous section and issue the **xconnect** *peer-router id vcid* **encapsulation mpls** command for those Ethernet links that face the customer. This command allows you to make a connection to the peer PE routerthat is, from SanFran to NewYork and vice versa. *peer-router-id* was specified as the loopback interface's address on each of the PEs. vcid is a unique identifier shared between the two PEs. The **vcid** value must match on both routers. The **encapsulation mpls** portion of the command identifies MPLS instead of L2TPv3 as the tunneling method that encapsulates data in the pseudowire.

Example 7-9 shows the new configuration on the SanFran and NewYork routers.

Example 7-9. *Configuring Port Transparency*

```
hostname SanFran
!
!
! Output omitted for brevity
!
interface Ethernet0/0
 xconnect 192.168.1.103 100 encapsulation mpls
 description to Oakland
!

hostname NewYork
!
! Output omitted for brevity
!
interface Ethernet0/0
xconnect 192.168.1.102 100 encapsulation mpls
description to Albany
!
```

Verifying and Troubleshooting Port Transparency Operation

You can take several steps to ensure that your configuration is complete. First, you might want to check the status of the VCs by issuing the **show mpls l2transport vc** command. Example 7-10 shows that VC 100 is up on both SanFran and NewYork.

Example 7-10. show mpls l2transport vc *Command*

```
SanFran#show mpls l2transport vc

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------  --------------  ----------  ----------
Et0/0          Ethernet                192.168.1.103   100        UP
---------------------------------------------------------------------------

NewYorkshow mpls l2transport vc

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------  --------------  ----------  ----------
Et0/0          Ethernet                192.168.1.102   100        UP
```

To view more detailed information on VC 100, type in the **show mpls l2transport vc 100 detail** command on either of the PE routers, as shown in Example 7-11. From the output, you learn the VC and interface status, label values, MTU presets, among other useful facts.

Example 7-11. show mpls l2transport vc 100 detail *Command*

```
SanFran#show mpls l2transport vc 100 detail
Local interface: Et0/0 up, line protocol up, Ethernet up
  Destination address: 192.168.1.103, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 17, next hop point2point
    Output interface: Se6/0, imposed label stack {17 19}
  Create time: 00:07:06, last status change time: 00:06:27
  Signaling protocol: LDP, peer 192.168.1.103:0 up
    MPLS VC labels: local 19, remote 19
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description: to Albany
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 45, send 45
    byte totals:   receive 4806, send 4812
    packet drops:  receive 0, send 0
```

To check the VC type, you can turn on debugging with the **debug mpls l2transport signaling message** command. Try using it immediately followed by the **interface xconnect** command. Your output should match that in Example 7-12.

Example 7-12. debug mpls l2transport signaling message *Command*

```
#SanFran#debug mpls l2transport signaling message
AToM LDP message debugging is on
SanFran(config)#int e 0/0
SanFran(config-if)# xconnect 192.168.1.103 100 encapsulation mpls

00:29:01: %LDP-5-NBRCHG: LDP Neighbor 192.168.1.103:0 is UP
00:29:01: AToM LDP [192.168.1.103]: Sending label mapping msg
vc type 5, cbit 1, vc id 100, group id 0, vc label 19, status 0, mtu 1500
00:29:01: AToM LDP [192.168.1.103]: Received label mapping msg, id 100
```

```
vc type 5, cbit 1, vc id 100, group id 0, vc label 19, status 0, mtu 1500
00:29:02: %SYS-5-CONFIG_I: Configured from console by console
```

In addition, you can use the **show mpls l2transport binding** command to get similar information without debugging.

> **Note**
>
> With the output of some commands, you can determine the VC type. The most common way is by observing the configuration of the interface. A physical interface without a subinterface represents VC type 5. A VLAN or subinterface means VC type 4. In newer Cisco IOS Software implementations, such details become less critical because of auto-negotiation of the VC type.

Finally, you should be able to verify connectivity by looking at the ARP table of the CE routers and sending an ICMP message from one CE router to the other. Example 7-13 displays the output of these commands issued on the Oakland router.

Example 7-13. show arp *and* ping *Commands*

```
Oakland#show arp
Protocol  Address          Age (min)  Hardware Addr   Type  Interface
Internet  192.168.100.1           -   00D0.0c00.6c00  ARPA   Ethernet0/0
_____________________________________________________________________
Oakland#ping 192.168.100.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.100.2, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 16/18/20 ms
Oakland#ping 192.168.100.2
_____________________________________________________

Oakland#show arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  192.168.100.1           -   00D0.0c00.6c00  ARPA   Ethernet0/0
Internet  192.168.100.2           0   00D0.0c00.6f00  ARPA   Ethernet0/0
Oakland#
```

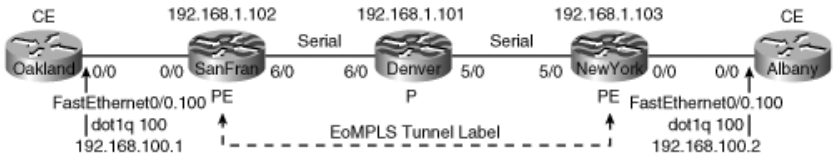
In Example 7-13, notice the difference between the first and the second time that the **show arp** command is used.

Case Study 7-2: Router to RouterVLAN Based

This case study explains how to enable MPLS to transport Layer 2 VLAN packets between the two customer sites. The configuration is based on the topology from Figure 7-6.

Figure 7-6. Router-to-Router VLAN-Based Topology

[View full size image]



Again, for this case study, you can use the preconfigured portion described earlier in the chapter. However, instead of VC type 5, as in the previous case study, you will use VC type 4.

Configuring VLAN-Based EoMPLS on PE Routers

To configure VLAN-based EoMPLS on the imposition/disposition routers, follow these steps:

Step 1. Specify an Ethernet interface that is facing the customer.

Step 2. Specify the Ethernet subinterface on that interface. Set up the subinterface for the label imposition.

Step 3. Specify 802.1q encapsulation for the subinterface with the **encapsulation dot1Q** *vlan-id* command. The *vlan-id* value should match that of the adjoining CE router.

Step 4. Finally, specify the VC for use to transport the VLAN packets with the **xconnect peer-router id vcid encapsulation mpls** command. The purpose of the command was discussed previously in Case Study 7-1. Its use and method of application are the same here.

Example 7-14 demonstrates the SanFran and NewYork configuration needed for this case study.

Example 7-14. *Configuring VLAN-Based EoMPLS on PE Routers*

```
hostname SanFran
!
! Output omitted for brevity
!
interface FastEthernet0/0
!
interface FastEthernet0/0.100
 encapsulation dot1Q 100
 xconnect 192.168.1.103 100 encapsulation mpls
!

hostname NewYork
!
!
!
interface FastEthernet0/0
!
interface FastEthernet0/0.100
 encapsulation dot1Q 100
 no ip directed-broadcast
 no cdp enable
 xconnect 192.168.1.102 100 encapsulation mpls
!
```

Configuring VLAN-Based EoMPLS on CE Routers

VLAN-based EoMPLS also requires configuration of the CE routers. The CE routers must have the same VLAN ID as the PE routers. Use the following configuration on the CE routers to transport Layer 2 VLAN packets:

Step 1. Select an Ethernet interface that is facing the PE.

Step 2. Set up the subinterface. It should have the same VLAN ID as the adjacent PE.

Step 3. Give an IP address to the subinterface.

Step 4. Specify an 802.1q encapsulation for the subinterface, along with the VLAN ID, via the **encapsulation dot1Q** *vlan-id* command.

Example 7-15 shows the customer-side router configuration of Oakland and Albany.

Example 7-15. *Configuring VLAN-Based EoMPLS on CE Routers*

```
hostname Oakland
!
interface Ethernet0/0
 no ip address
 no ip directed-broadcast
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 192.168.100.1 255.255.255.0
 no ip directed-broadcast

------------------------------------------------------------------------
hostname Albany
!
interface Ethernet0/0
 no ip address
 no ip directed-broadcast
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 192.168.100.2 255.255.255.0
 no ip directed-broadcast
```

Verifying and Troubleshooting the Configuration

To ensure the validity of your configuration, you can use the same techniques as in Case Study 7-1. For instance, issue **show mpls l2transport vc** on one of the PE routers to check the status of the VC, as shown in Example 7-16. Note the subinterface in the Local intf column.

Example 7-16. show mpls l2transport vc *Command*

```
SanFran#show mpls l2transport vc

Local intf     Local circuit          Dest address    VC ID      Status
```

```
-------------  ---------------------- --------------- ---------- ----------
Et0/0.100      Eth VLAN 100           192.168.1.103   100        UP
```

The show mpls l2transport vc 100 detail command output from Example 7-17 presents the .100 numbered subinterface, in addition to the Eth VLAN 100 up, indicating the use of the VLAN-based EoMPLS. Compare it to Ethernet up from the same command's output for port-based EoMPLS used in Case Study 7-1.

Example 7-17. show mpls l2transport vc 100 detail *Command*

```
SanFran#show mpls l2transport vc 100 detail
Local interface: Et0/0.100 up, line protocol up, Eth VLAN 100 up
  Destination address: 192.168.1.103, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 17, next hop point2point
    Output interface: Se6/0, imposed label stack {17 16}
  Create time: 00:00:57, last status change time: 00:00:20
  Signaling protocol: LDP, peer 192.168.1.103:0 up
    MPLS VC labels: local 16, remote 16
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 3, send 3
    byte totals:   receive 1627, send 1628
    packet drops:  receive 0, send 0
```

The **show mpls forwarding-table** command in Example 7-18 shows label 16 advertised to the remote PE and used in disposition.

Example 7-18. show mpls forwarding-table *Command*

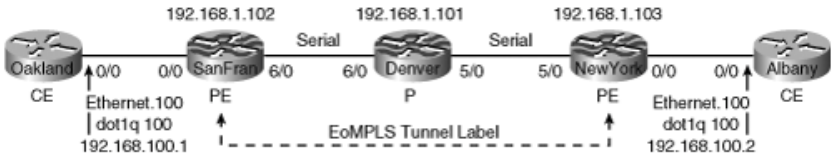
```
SanFran#show mpls forwarding-table
Local  Outgoing    Prefix            Bytes tag   Outgoing    Next Hop
tag    tag or VC   or Tunnel Id      switched    interface
16     Untagged    l2ckt(100)        1603        Et0/0.100   point2point
17     Pop tag     10.1.2.0/24       0           Se6/0       point2point
18     Pop tag     192.168.1.101/32  0           Se6/0       point2point
19     17          192.168.1.103/32  0           Se6/0       point2point
```

Case Study 7-3: VLAN Rewrite

The VLAN rewrite feature is needed when The VLAN IDs on either side of the customer Ethernet network do not match. The network scenario in Figure 7-7 illustrates such a situation.

Figure 7-7. VLAN Rewrite Topology

[View full size image]



To compensate for the mismatch, change VLAN encapsulation on SanFran to match the VLAN ID of NewYork. Following Figure 7-7, modify the Case Study 7-2 configuration on the Oakland router to reflect the new VLAN ID.

Example 7-19 shows the Oakland and Albany interface configurations. Note that in comparison to Case Study 7-2, the settings for Albany remain the same, whereas Oakland displays 200 as its VLAN ID for the dot1Q encapsulation.

Example 7-19. *Configuring CE Routers*

```
hostname Albany
!
interface Ethernet0/0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 192.168.100.2 255.255.255.0
 no ip directed-broadcast
```

```
hostname Oakland
!
interface Ethernet0/0
!
interface Ethernet0/0.200
 encapsulation dot1Q 200
 ip address 192.168.100.1 255.255.255.0
```

On the PE side, reconfigure the VLAN ID for the subinterface to match that of its neighboring CE. According to the topology used in this case study, you do not need to change the VLAN ID on NewYork from your previous configuration because the VLAN ID for Albany remains 100. However, you need to reset the ID for SanFran to 200 to equal Oakland's new ID.

As previously mentioned, vcid values of both PEs need to be the same. Therefore, they will remain 100 in the **xconnect** command, thereby rewriting the 200 VLAN to 100 to meet the requirement. The PE configuration is illustrated in Example 7-20.

Example 7-20. *Configuring VLAN Rewrite on PE Routers*

```
hostname SanFran
!
!
! Output omitted for brevity
!
interface Ethernet0/0
!
interface Ethernet0/0.200
 encapsulation dot1Q 200
 xconnect 192.168.1.103 100 encapsulation mpls
```

```
-------------------------------------------------------------------------
hostname NewYork
!
! Output omitted for brevity
!
!
interface Ethernet0/0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 xconnect 192.168.1.102 100 encapsulation mpls
```

The issue of VLAN mismatch is not as simple when switches are concerned. This is discussed further in Case Study 7-6.

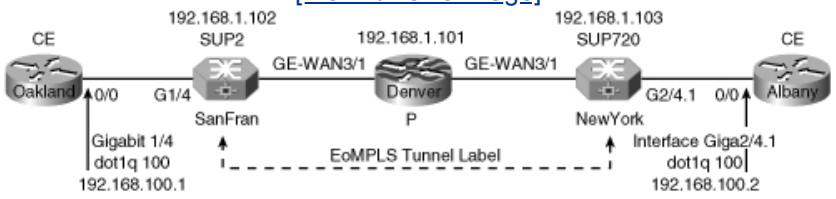
Case Study 7-4: Switch to SwitchVLAN Based

In this case study, the topology differs from the rest in that both PE and CE devices are switches instead of routers. The PEs are 7600 routers with gigabit WAN interfaces facing the MPLS core. Both CE switches connect to the PEs via 802.1q trunks.

The new topology is presented in Figure 7-8.

Figure 7-8. Switch to SwitchVLAN-Based Topology

[View full size image]



Note

EoMPLS can run either on a SUP720-3BXL-based system or a supervisor engine 2-based system. With SUP720-3BXL-based systems, you can implement EoMPLS on either the supervisor engine 720 or on an OSM or a Flexwan module facing the MPLS core network, whereas the supervisor engine 2-based system must be equipped with an OSM module or a Flexwan module facing the MPLS core.

Configuring VLAN-Based EoMPLS on PEs

To start, you do not need to make configuration changes on the Denver router of the service provider network. The majority of the SanFran and NewYork switches' configuration mirrors that of the SanFran and NewYork routers from previous case studies. Example 7-21 shows the configuration of the SanFran and NewYork switches prior to the settings needed for this case study. Notice that the serial interfaces changed to GE-WAN.

Example 7-21. ***SanFran and NewYork's Initial Configuration***

```
hostname SanFran
!
ip cef
mpls ip
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
ip address 192.168.1.102 255.255.255.255
no ip directed-broadcast
!
interface GE-WAN3/1
ip address 10.1.1.102 255.255.255.0
no ip directed-broadcast
mpls label protocol ldp

mpls ip
!
router ospf 100
log-adjacency-changes detail
network 0.0.0.0 255.255.255.255 area 0
```

```
hostname NewYork
!
ip subnet-zero
ip cef
mpls ip
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 192.168.1.103 255.255.255.255
!
interface GE-WAN3/1
 ip address 10.1.2.103 255.255.255.0
 mpls label protocol ldp

 mpls ip
!
router ospf 100
 log-adjacency-changes detail
 network 0.0.0.0 255.255.255.255 area 0
```

Now you can configure specific tasks for this case study.

Supervisor Engine 2-Based System Configuration

Assume that SanFran is a SUP2-based system. Follow these steps to configure SanFran for EoMPLS:

Step 1. Configure a VLAN ID or VLAN range with the **vlan** {*vlan-id* | *vlan-range*} global command. Activate the VLAN with the **state active** command.

Step 2. Configure the physical port facing the CE for switching by issuing the **switchport** interface command.

Step 3. Set the trunk encapsulation to dot1Q when the interface is in trunking mode with the **switchport trunk encapsulation dot1q** command.

Step 4. Change the allowed list for the specified VLANs via the **switchport trunk allowed vlan list** command.

Step 5. Specify a trunking VLAN Layer 2 interface with the switchport mode trunk interface command.

Step 6. Create a VLAN interface with the **interface vlan** *vlan-id* command.

Step 7. Specify the VC for transporting the Layer 2 VLAN packets via the **mpls l2transport route destination vc-id** command.

Example 7-22 demonstrates the configuration added by following the preceding steps.

Example 7-22. *SanFran Additional Configuration for SUP-2*

```
hostname SanFran
!
vlan 100
 state active
!
interface GigabitEthernet1/4
no ip address
switchport
switchport trunk encapsulation dot1q
switchport trunk allowed vlan 100
switchport mode trunk
no shut
!
interface Vlan100
no ip address
no ip mroute-cache
mpls l2transport route 192.168.1.103 100

no shut
!
```

SUP720-3BXLBased System Configuration

The following set of steps applies EoMPLS on the SUP720-3BXL-based system for the NewYork PE:

Step 1. Disable VLAN Trunking Protocol (VTP) using the **vtp mode transparent** global command.

Step 2. Specify the Gigabit Ethernet subinterface with the interface *gigabitethernetslot/interface.subinterface* command. Make sure the subinterface on the adjoining CE switch is on the same VLAN as this PE switch.

Step 3. Enable the subinterface to accept 802.1q VLAN packets via the **encapsulation dot1q vlan-id** command. The subinterfaces between the CE switches that are running EoMPLS should be in the same subnet.

Step 4. Bind the attachment circuit to a pseudowire VC with the **xconnect** command.

Example 7-23 demonstrates the SUP720 configuration, as discussed in the preceding steps.

Example 7-23. *NewYork's Additional Configuration for SUP-720*

```
!
hostname NewYork
!
vtp mode transparent
!
interface GigabitEthernet2/4
no ip address
no shut
!
interface GigabitEthernet2/4.1
encapsulation dot1Q 100
xconnect 192.168.1.102 100 encapsulation mpls
no shut
!
```

Configuring VLAN-Based EoMPLS on the CE Switches

Example 7-24 shows the CE switches configuration for Oakland and Albany. Notice that the CE side is the same, regardless of whether the PE system is SUP2- or SUP720-3BXL-based.

Example 7-24. *Configuring CE Switches for VLAN-Based EoMPLS*

```
hostname Oakland
!
interface GigabitEthernet1/0
negotiation auto
no cdp enable
no shut
!
interface GigabitEthernet1/0.100
encapsulation dot1Q 100
ip address 192.168.100.1 255.255.255.0
no cdp enable
no shut
!
```

```
hostname Albany
!
interface GigabitEthernet4/0
negotiation auto
no cdp enable
no shut
!
interface GigabitEthernet4/0.100
encapsulation dot1Q 100
ip address 192.168.100.2 255.255.255.0
no ip directed-broadcast
no cdp enable
no shut
!
```

Verifying and Troubleshooting the Configuration

This section describes the verification techniques recommended for this case study.

Issue the **gshow vlan brief** command on a PE to check whether the interface can provide two-way communication, as seen in Example 7-25.

Example 7-25. show vlan brief *Command*

```
SanFran#Show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------
1    default                          active
2    VLAN0100                         active
```

The **show mpls ldp discovery** and **show mpls ldp neighbor** commands (as described earlier in the opening paragraphs of the "Case Studies" section of this chapter), in addition to the **show mpls forwarding-table** command, are useful. Although the **show mpls forwarding-table** command was already displayed in Examples 7-6 and 7-16 of this chapter, what makes it different now is the line highlighted in Example 7-26 showing the Layer 2 circuit (VLAN) configured in this case study.

Example 7-26. show mpls forwarding-table *Command*

```
SanFran#show mpls forwarding-table

Local  Outgoing    Prefix          Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id    switched   interface

0      Untagged    l2ckt(100)      133093     Vl100       point2point

! Output omitted for brevity
```

You can check the status of your VCs by issuing the show mpls l2transport vc command.

Note the difference in the output of the **show mpls l2transport vc** command between Example 7-16 (earlier in this chapter) and Example 7-27. The local intf portion that showed Et0/0.100 before now shows VLAN 100.

Example 7-27. show mpls l2transport vc *Command*

```
SanFran#show mpls l2transport vc

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------  --------------  ----------  ----------
Vl100           Eth VLAN 100            192.168.1.103   100         UP
```

To view detailed information about your VC, issue the **show mpls l2transport vc detail** command, as in Example 7-28.

Example 7-28. show mpls l2transport vc detail *Command*

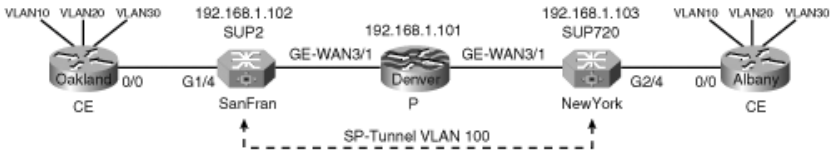
```
SanFran#show mpls l2transport vc detail
Local interface: Vl100 up, line protocol up, Eth VLAN 100 up
  Destination address: 192.168.1.103, VC ID: 100, VC status: up
    Tunnel label: 17, next hop 192.168.1.101
    Output interface: GE3/3, imposed label stack {17 16}
  Create time: 00:00:57, last status change time: 00:00:20
  Signaling protocol: LDP, peer 192.168.1.103:0 up
    MPLS VC labels: local 16, remote 16
    Group ID: local 71, remote 89
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 3, send 3
    byte totals:   receive 1627, send 1628
    packet drops:  receive 0, send 0
```

Case Study 7-5: Switch to SwitchPort Based

In this case study, you learn how to configure port-based EoMPLS in the switch-based environment. As in the preceding case study, SanFran is a supervisor engine 2-based system and NewYork is a SUP720-3BXL-based system. The configuration presented in this case study supports both QinQ and native Ethernet traffic. Figure 7-9 shows the topology for this case study.

Figure 7-9. Switch to SwitchPort-Based Topology

[View full size image]



Note

You could allow for port-based EoMPLS without 802.1q support. This would include all basic configuration and exclude all dot1Q-related tasks. Because this is a simpler approach with the same basic configuration, it does not warrant its own case study in this book.

Configuring Port-Based EoMPLS on the SanFran Switch

To set up the SanFran SUP2-based system for port-based EoMPLS with QinQ support, follow these steps:

Step 1. Enter the VLAN ID with the **vlan** {*vlan_id* | *vlan_range*} command.

Step 2. Enable dot1Q tagging for all VLANs in a trunk via the **vlan dot1q tag native** command.

Step 3. Configure the physical port facing the CE for switching by using the **switchport** interface

command.

Step 4. Set the trunking mode to tunneling with the **switchport mode dot1qtunnel** interface command.

Step 5. Specify a VLAN whose traffic will be accepted by the port through the **switchport access vlan** *vlan_id* command.

Step 6. Configure bridge protocol data unit (BPDU) filtering on an interface with the **spanning-tree bpdufilter enable** command to prevent a port from sending and receiving BPDUs to protect the provider's side from potential spanning-tree attacks.

Step 7. Create a virtual VLAN interface via the **interface vlan** *vlan_id* command.

Step 8. Specify the VC to transport the VLAN traffic with the **mpls l2transport route** *destination vc_id* command.

Example 7-29 demonstrates the SanFran EoMPLS port-based configuration for transporting QinQ traffic described in the preceding steps.

Example 7-29. *SanFran Port-Based EoMPLS Configuration*

```
hostname SanFran
!
vlan 100
!
vlan dot1q tag native
!
interface GigabitEthernet1/4
no ip address
switchport
switchport access vlan 100
switchport trunk encapsulation dot1q
switchport mode dot1q-tunnel
no cdp enable
spanning-tree bpdufilter enable
no shut
!
interface Vlan100
no ip address
no ip mroute-cache
mpls l2transport route 192.168.1.103 100
no shut
```

Note

Be careful when enabling BPDU filtering on an interface, because it effectively disables the spanning tree for that interface. If used incorrectly, bridging loops can occur.

Configuring Port-Based EoMPLS on the NewYork Switch

NewYork is a SUP720-3BXL-based system that requires the following configuration:

Step 1. Enter the Gigabit Ethernet interface.

Step 2. Bind the attachment circuit to a pseudowire VC with the **xconnect** command, as shown in previous case studies.

As you can see, essentially two commands exist: old (**mpls l2transport route**) and new (**xconnect**) to enable port mode EoMPLS on a SUP720-3BXLbased system. This is because significant improvements were made to the system in an effort to simplify command-line interface (CLI).

Example 7-30 demonstrates the needed configuration on the NewYork switch.

Example 7-30. *NewYork Port-Based EoMPLS Configuration*

```
hostname NewYork
!
interface GigabitEthernet2/4
no ip address
xconnect 192.168.1.102 100 encapsulation mpls
no shut
!
```

CE switches do not require special configuration. You have the choice of enabling or forgoing dot1Q on the CEs. You can review Example 7-24 from the preceding case study for Oakland and Albany's settings.

To verify your configuration, use the techniques described in Case Study 7-4. In the outputs of these commands, you will not find specifics that indicate the difference between VLAN and port modes. However, you will see whether your configuration is working.

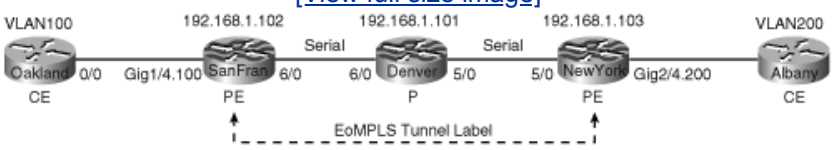
Case Study 7-6: VLAN Rewrite in Cisco 12000 Series Routers

As mentioned earlier in this chapter, the VLAN ID rewrite feature on routers and switches allows the use of different VLAN IDs on VLAN interfaces at two ends of the tunnel.

Some router platforms perform VLAN rewrites automatically on the disposition router on output; therefore, no special configuration is needed. When you are using certain line card combinations of the Cisco 12000 routers, however, manual configuration of the VLAN rewrite feature is required. In this case study, the SanFran and NewYork PEs are the Cisco 12000 series routers. The topology is shown in Figure 7-10.

Figure 7-10. VLAN ID Rewrite Topology

[View full size image]



When you are configuring the VLAN rewrite on the Cisco 12000 series platforms, keep in mind that because of the difference in functionality, additional configuration might be required if the ends of the EoMPLS connections are not provisioned with the same line cards.

Some examples of the difference in system flow between different line cards with VLAN rewrite are outlined next:

- For example, a 4-port Gigabit Ethernet line card is used, traffic flows from VLAN 100 on Oakland to VLAN200 on Albany. As the frame reaches the edge-facing line card of NewYork, the VLAN ID in the dot1Q header changes to the VLAN ID that is assigned to VLAN 200. This is because the 4-port Gigabit Ethernet line card performs a VLAN ID rewrite on the disposition side.

- When a 3-port Gigabit Ethernet line card is used, traffic flows from VLAN 100 on Oakland to VLAN 200 on Albany. But, unlike the preceding example, as the frame reaches the edge-facing line card of SanFran, the VLAN ID in the dot1Q header changes to the VLAN ID that is assigned to VLAN 200. This is because the 3-port Gigabit Ethernet line card performs VLAN ID rewrite on the imposition side.

To configure VLAN rewrite on the PEs with the 3-port Gigabit Ethernet line card scenario, follow these steps:

Step 1. Specify the Gigabit Ethernet subinterface. Ensure that the subinterface on the adjoining CE router is on the same VLAN.

Step 2. Enable the subinterface to accept 802.1q VLAN packets with the **encapsulation dot1q** *vlan_id* command.

Step 3. Bind the attachment circuit to a pseudowire VC with the **xconnect** command.

Step 4. Enable the use of VLAN interfaces with different VLAN IDs at both ends of the tunnel via the **remote circuit id** *remote_vlan_id* command.

Example 7-31 shows the VLAN rewrite configuration on SanFran and NewYork.

Example 7-31. *Configuring VLAN Rewrite on SanFran and NewYork*

```
hostname SanFran
!
interface GigabitEthernet1/4.100
encapsulation dot1Q 100
no cdp enable
no ip directed-broadcast
xconnect 192.168.1.103 100 encapsulation mpls
remote circuit id 200
no shut
!

hostname NewYork
!
interface GigabitEthernet2/4.200
encapsulation dot1Q 200
no cdp enable
no ip directed-broadcast
xconnect 192.168.1.102 100 encapsulation mpls
remote circuit id 100
!
```

When you use the **remote circuit id** command, you are effectively enabling the VLAN Rewrite to be performed at imposition. In this case, the disposition (egress) PE router needs to inform the imposition (ingress) PE router of the VLAN used in the attachment circuit by means of signaling. This remote VLAN signaling is accomplished with the interface parameter Requested VLAN ID (parameter ID 0x06) attached to the LDP mapping message.

In newer Cisco IOS Software releases, the **remote circuit id** command is not required in line cards because the VLAN ID Rewrite feature is implemented automatically, and the 4-port Gigabit Ethernet line card was enhanced to perform VLAN ID rewrite, either on imposition or disposition. This allows you to configure EoMPLS in a consistent manner without checking for line card hardware dependencies.

Verifying and Troubleshooting the Configuration

To determine whether VLAN rewrite is enabled on Cisco 12000 series PEs, issue the **show controllers eompls forwarding-table** [*port*] [*vlan*] command. To execute this command, open a session directly to the line card. Example 7-32 shows the output of the command on SanFran and NewYork.

Example 7-32. show controllers eompls forwarding-table *Command*

```
SanFran

LC-CON0#show controllers eompls forwarding-table 0 100

Port # 0, VLAN-ID # 100, Table-index 100
EoMPLS configured: 1
tag_rew_ptr             = D001BB58
Leaf entry?     = 1
FCR index       = 20
           **tagrew_psa_addr    = 0006ED60
           **tagrew_vir_addr    = 7006ED60
           **tagrew_phy_addr    = F006ED60
        [0-7] loq 8800 mtu 4458  oq 4000 ai 3 oi 04019110 (encaps size 4)
        cw-size 4  vlanid-rew 200
        gather A30 (bufhdr size 32 EoMPLS (Control Word) Imposition profile 81)
        2 tag: 18 18
        counters 1182, 10 reported 1182, 10.
    Local OutputQ (Unicast):     Slot:2 Port:0  RED queue:0  COS queue:0
    Output Q (Unicast):          Port:0         RED queue:0  COS queue:0
_____________________________________________________________________________

NewYork

LC-CON0#show controllers eompls forwarding-table 0 200

Port # 0, VLAN-ID # 200, Table-index 200
EoMPLS configured: 1
tag_rew_ptr             = D0027B90
Leaf entry?     = 1
FCR index       = 20
           **tagrew_psa_addr    = 0009EE40
           **tagrew_vir_addr    = 7009EE40
           **tagrew_phy_addr    = F009EE40
        [0-7] loq 9400 mtu 4458  oq 4000 ai 8 oi 84000002 (encaps size 4)
        cw-size 4  vlanid-rew 100
        gather A30 (bufhdr size 32 EoMPLS (Control Word) Imposition profile 81)
        2 tag: 17 18
        counters 1182, 10 reported 1182, 10.
    Local OutputQ (Unicast):    Slot:5  Port:0  RED queue:0  COS queue:0
    Output Q (Unicast):         Port:0          RED queue:0  COS queue:0
```

Note

Other platforms that do not require manual configuration do not provide VLAN ID rewrite information in their output.

Port VLAN ID Inconsistency Issue

In EoMPLS using CE switches, Spanning Tree Protocol (described in Chapter 4) still runs between the customer end devices, and it is transported across the MPLS core backbone. In a VLAN ID rewrite scenario, Port VLAN ID (PVID) inconsistency stems from the Per VLAN Spanning Tree + (PVST+) BPDU being received on a different VLAN than it was originated. Therefore, when the trunk port on Oakland receives a PVST+ BPDU from the Albany's STP of VLAN 200 with a tag of VLAN 200, you get an error message as soon as the circuit comes up:

```
%SPANTREE-2-BLOCK_PVID_LOCAL : Blocking [chars] on [chars]. Inconsistent local vlan.
```

The listed STP instance is that of the native VLAN ID of the listed interface. The first [chars] is the interface, and the second [chars] is the STP instance. As a result, the trunk port is held in a blocking state (for both VLAN 100 and VLAN 200) until the inconsistency is resolved.

To unblock the interface, change the VLAN IDs on the CEs so that they match. When this is not possible and VLAN ID rewrite is required, you must turn off the STP. This alternative opens a door to bridging loops; therefore, you should use it with extreme caution.

Case Study 7-7: Map to Pseudowire

For EoMPLS configuration, you might choose to configure a pseudowire class template that consists of configuration settings used by all attachment circuits that are bound to the class. Pseudowire was introduced in Chapters 2, "Pseudowire Emulation Framework and Standards," and 6 and is discussed in further detail in the advanced configuration case studies of Chapter 9, "Advanced AToM Case Studies."

Example 7-33 shows configuration of the VC 100 in Ethernet port mode.

Example 7-33. *Pseudowire Class Configuration*

```
hostname SanFran
!
pseudowire-class ethernet-port
  encapsulation mpls

int GigabitEthernet1/4
  xconnect 192.168.1.103 100 pw-class ethernet-port
!
```

Common Troubleshooting Techniques

This section introduces you to the most common troubleshooting techniques for PEs in EoMPLS. You first learn the commands and outputs for the Cisco router PEs and then learn to troubleshoot Cisco 7600 series switches.

Troubleshooting EoMPLS on Routers

The first common step in troubleshooting problems is attempting to discover failure by verifying the status of a VC by issuing the **show mpls l2transport vc** command.

Three conditions must be met so that the VC is UP:

- The disposition interfaces are programmed if the VC has been configured and the CE interface is UP.
- If the IGP label exists, it can be implicit null in a back-to-back configuration.
- The imposition interface is programmed if the disposition interface is programmed and you have a remote VC label and an IGP label (LSP to the peer).

If the status field is marked DOWN (that is, the VC is not ready to carry traffic between the two VC endpoints), as shown in the output of Example 7-34, execute the **show mpls l2transport vc detail** command seen in Example 7-35 for more in-depth information.

Example 7-34. show mpls l2transport vc *Command*

```
NewYork#show mpls l2transport vc

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------- --------------- ---------- ----------
Et0/0          Ethernet                192.168.1.102   100        DOWN
```

Example 7-35. show mpls l2transport vc detail *Command*

```
NewYork#show mpls l2transport vc detail
Local interface: Et0/0 up, line protocol up, Ethernet up
  Destination address: 192.168.1.102, VC ID: 100, VC status: down
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop point2point
    Output interface: Se5/0, imposed label stack {16 16}
  Create time: 00:18:10, last status change time: 00:03:51
  Signaling protocol: LDP, peer 192.168.1.102:0 up
    MPLS VC labels: local 16, remote 16
    Group ID: local 0, remote 0
    MTU: local 1500, remote unknown
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 0, send 0
```

```
    byte totals:    receive 0, send 0
    packet drops:   receive 0, send 78
```

Table 7-4 describes some of the significant fields of the **show mpls l2transport vc detail** command output.

Table 7-4. show mpls l2transport vc detail *Command Output Fields*

Field	Description
Destination address	The IP address of the remote router specified for this VC as part of the **mpls l2transport route** or **xconnect** command.
VC ID	The virtual circuit identifier assigned to the interface on the router.
VC status	The status of the VC. The status can be one of the following:
	UPThe VC is in a state in which it can carry traffic between the two VC endpoints. A VC is UP when both imposition and disposition interfaces are programmed.
	DOWNThe VC is not ready to carry traffic between the two VC endpoints.
	ADMIN DOWNA user has disabled the VC.
Tunnel label	An IGP label that routes the packet over the MPLS backbone to the destination router with the egress interface.
Output interface	The interface on the remote router that has been enabled to transmit and receive Layer 2 packets.
Imposed label stack	A summary of the MPLS label stack that directs the VC to the PE router.

Field	Description
Signaling protocol	The type of protocol that sends the MPLS labels. The output also shows the status of the peer router.
MPLS VC labels	The local VC label is a disposition label, which determines the egress interface of an arriving packet from the MPLS backbone. The remote VC label is a disposition VC label of the remote peer router.
MTU	The maximum transmission unit specified for the local and remote interfaces.
Sequencing	This field describes whether sequencing of out-of-order packets is enabled or disabled.

In the **show mpls l2transport vc detail** command output, pay attention to the remote unknown next to the local MTU. One of the possible causes is a remote interface down or an MTU mismatch. Verify to make sure that MTU on each side is the same. If an EoMPLS tunnel is still down after this and you cannot pass traffic, perform another check by issuing the **show mpls forwarding-table** command, as demonstrated in Example 7-36.

Example 7-36. show mpls forwarding-table *Command*

```
NewYork#show mpls forwarding-table
Local  Outgoing    Prefix           Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id     switched   interface
17     Untagged    10.1.1.0/24      0          Se5/0       point2point
18     Untagged    192.168.1.101/32 0          Se5/0       point2point
19     Untagged    192.168.1.102/32 0          Se5/0       point2point
20     Untagged    l2ckt(100)       4592       Et0/0.100   point2point
```

The Untagged result in the Outgoing tag or VC field indicates that an MPLS label might not be exchanged between the PE and P (Denver) routers.

A couple possible causes exist. Either the **mpls ip** has not been enabled per interface, or CEF is disabled (or not enabled) on the P or PE router. To verify, issue the **show mpls ldp discovery** command, as in Example 7-37.

Example 7-37. show mpls ldp discovery *Command*

```
NewYork#show mpls ldp discovery
 Local LDP Identifier:
```

```
    192.168.1.103:0
    Discovery Sources:
    Targeted Hellos:
          192.168.1.103 -> 192.168.1.102 (ldp): active/passive, xmit/recv
                LDP Id: 192.168.1.102:0
```

The output shows whether you have a direct LDP session open between directly connected MPLS-enabled interfaces. By observing this behavior, you can conclude that MPLS indeed was not enabled per interface facing the core. To solve this problem, enable MPLS on an interface or check whether CEF is enabled.

Two common issues result when the circuit does not come up:

- The remote port is down or not configured.
- The MTU is mismatched.

Examples 7-38 and 7-39 display the output of the **show mpls l2transport vc** *vcid***detail** command with the two conditions, respectively.

Example 7-38. *Remote Port Down or Not Configured*

```
NewYork#show mpls l2transport vc 10 detail
Local interface: FastEternet0/0.10 up, line protocol up, Eth VLAN 10 up
  Destination address: 192.168.1.102, VC ID: 10, VC status: down
    Tunnel label: not ready
    Output interface: unknown, imposed label stack {}
  Create time: 22:31:53, last status change time: 04:02:56
  Signaling protocol: LDP, peer 192.168.1.102:0 up
    MPLS VC labels: local 19, remote unassigned
    Group ID: local 0, remote unknown
    MTU: local 1500, remote unknown
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 1650, send 1743
    byte totals:   receive 552557, send 550044
    packet drops:  receive 0, send 7
```

Example 7-39. *MTU Mismatch*

```
NewYork#show mpls l2transport vc 10 detail
Local interface: FastEternet0/0.10 up, line protocol up, Eth VLAN 10 up
  Destination address: 192.168.1.102, VC ID: 10, VC status: down
    Tunnel label: not ready
    Output interface: unknown, imposed label stack {}
  Create time: 22:36:10, last status change time: 00:00:20
  Signaling protocol: LDP, peer 192.168.1.102:0 up
    MPLS VC labels: local 19, remote 21
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1000
    Remote interface description: *** To SanFran ***
  Sequencing: receive disabled, send disabled
```

```
  VC statistics:
    packet totals: receive 1880, send 1901
    byte totals:   receive 168476, send 155436
    packet drops:  receive 0, send 13
```

The highlighted portions of Examples 7-38 and 7-39 call attention to the faulty conditions. Compare the output to output of the same command in an operational environment from Example 7-40.

Example 7-40. *Working Example*

```
NewYork#show mpls l2transport vc 10 detail
Local interface: FastEternet0/0.10 up, line protocol up, Eth VLAN 10 up
  Destination address: 192.168.1.102, VC ID: 10, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 17, next hop 10.1.1.202
    Output interface: Et1/0, imposed label stack {17 21}
  Create time: 23:06:37, last status change time: 00:30:47
  Signaling protocol: LDP, peer 192.168.1.102:0 up
    MPLS VC labels: local 19, remote 21
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description: *** To SanFran ***
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 1683, send 1777
    byte totals:   receive 565455, send 563328
    packet drops:  receive 0, send 7
```

Example 7-41 presents the verification and configuration sequence of enabling MPLS on the Serial 5/0 interface.

Example 7-41. *Configuring MPLS on an Interface*

```
NewYork#show mpls interfaces
Interface              IP      Tunnel   Operational
Se5/0                  No      Yes      No

configure terminal
interface serial 5/0
mpls ip

NewYork#show mpls interfaces
Interface              IP      Tunnel   Operational
Se5/0                  Yes     Yes      Yes
```

Next, check whether MPLS is directly receiving labels and exchanging LDP between the PE and the P by reissuing the **show mpls forwarding-table** command. The output is provided in Example 7-42.

Example 7-42. show mpls forwarding-table *Command*

```
NewYork#show mpls forwarding-table
Local  Outgoing    Prefix           Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id     switched   interface
17     Pop tag     10.1.1.0/24      0          Se5/0       point2point
18     Pop tag     192.168.1.101/32 0          Se5/0       point2point
19     16          192.168.1.102/32 0          Se5/0       point2point
20     Untagged    l2ckt(100)       5310       Et0/0.100   point2point
```

To ensure that LDP xmit/recv (transmit/receive) is occurring in both directions, use the **show mpls ldp discovery** command again, as in Example 7-43.

Example 7-43. show mpls ldp discovery *Command*

```
NewYork#show mpls ldp discovery
 Local LDP Identifier:
    192.168.1.103:0
    Discovery Sources:
    Interfaces:
        Serial5/0 (ldp): xmit/recv
            LDP Id: 192.168.1.101:0
    Targeted Hellos:
        192.168.1.103 -> 192.168.1.102 (ldp): active/passive, xmit/recv
            LDP Id: 192.168.1.102:0
```

Example 7-44 shows that the VC is now ready and operational and should be able to send traffic from CE to CE.

Example 7-44. show mpls l2transport vc *Command*

```
NewYork#show mpls l2transport vc

Local intf     Local circuit            Dest address    VC ID      Status
-------------  ------------------------ --------------- ---------- ----------
Et0/0          Ethernet                 192.168.1.102   100        UP
```

Verify with the **show mpls l2transport vc detail** command output (shown in Example 7-45) that the packets are being sent and received.

Example 7-45. show mpls l2transport vc detail *Command*

```
NewYork#show mpls l2transport vc detail
Local interface: Et0/0 up, line protocol up, Ethernet up
  Destination address: 192.168.1.102, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop point2point
    Output interface: Se5/0, imposed label stack {16 16}
  Create time: 00:18:10, last status change time: 00:03:51
  Signaling protocol: LDP, peer 192.168.1.102:0 up
    MPLS VC labels: local 16, remote 16
    Group ID: local 0, remote 0
```

```
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 56, send 56
    byte totals:   receive 6019, send 6014
    packet drops:  receive 0, send 78
```

Debugging EoMPLS Operation on PE Routers

You can use several useful debugging commands to verify and troubleshoot EoMPLS operation on PE routers.

For dot1Q operation, look for VC type 4 in output generated by the **debug mpls l2transport signaling message** command, as shown in Example 7-46.

Example 7-46. *Debugging dot1Q*

```
NewYork#debug mpls l2transport signaling message

NewYork(config)#interface ethernet 0/1.100
NewYork(config-subif)#shutdown
00:19:51: AToM LDP [192.168.1.102]: Sending label withdraw msg
vc type 4, cbit 1, vc id 100, group id 0, vc label 20, status 0, mtu 1500
00:19:51: AToM LDP [192.168.1.102]: Received label release msg, id 78
vc type 4, cbit 1, vc id 100, group id 0, vc label 20, status 0, mtu 0

NewYork(config-subif)#no shutdown
00:21:56: AToM LDP [192.168.1.102]: Sending label mapping msg
vc type 4, cbit 1, vc id 100, group id 0, vc label 20, status 0, mtu 1500
```

In troubleshooting the port-based EoMPLS operation, look for VC type 5 in the **debug mpls l2transport signaling message** command output, as shown in Example 7-47.

Example 7-47. *Debugging Port to Port*

```
NewYork#debug mpls l2transport signaling message
AToM LDP message debugging is on
!
NewYork(config)#interface ethernet 0/0
NewYork(config-if)#shutdown
00:08:39: AToM LDP [192.168.1.102]: Sending label withdraw msg
vc type 5, cbit 1, vc id 100, group id 0, vc label 16, status 0, mtu 1500
00:08:39: AToM LDP [192.168.1.102]: Received label release msg, id 34
vc type 5, cbit 1, vc id 100, group id 0, vc label 16, status 0, mtu 0

00:08:41: %LINK-5-CHANGED: Interface Ethernet0/0, changed state to
  administratively down

NewYork(config-if)#no shutdown
00:08:42: AToM LDP [192.168.1.102]: Sending label mapping msg
vc type 5, cbit 1, vc id 100, group id 0, vc label 20, status 0, mtu 1500
00:08:44: %LINK-3-UPDOWN: Interface Ethernet0/0, changed state to up
```

```
00:08:45: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed
  state to up
```

Another helpful command is **debug acircuit event** for information on all attachment circuits, as illustrated in Example 7-48.

Example 7-48. debug acircuit event *Command*

```
NewYork#debug acircuit event
NewYork(config)#int e 0/0
NewYork(config-if)#no shutdown
00:12:59: ACLIB [192.168.1.102, 100]: SW AC interface UP for Ethernet interface
Et0/0
00:12:59: ACLIB [192.168.1.102, 100]: pthru_intf_handle_circuit_up() calling
  acmgr_circuit_up
00:12:59: ACLIB [192.168.1.102, 100]: Setting new AC state to Ac-Connecting
00:12:59: ACLIB: Update switching plane with circuit UP status
00:12:59: ACLIB [192.168.1.102, 100]: SW AC interface UP for Ethernet interface
Et0/0
00:12:59: ACLIB [192.168.1.102, 100]: pthru_intf_handle_circuit_up() ignoring
  up event. Already connected or connecting.
00:12:59: Et0/0 ACMGR: Receive <Circuit Up> msg
00:12:59: Et0/0 ACMGR: circuit up event, SIP state chg fsp up to connected,
  action is p2p up forwarded
00:12:59: ACLIB: pthru_intf_response hdl is 8C000002, response is 2
00:12:59: ACLIB [192.168.1.102, 100]: Setting new AC state to Ac-Connected
00:12:59: AToM LDP [192.168.1.102]: Sending label mapping msg
vc type 5, cbit 1, vc id 100, group id 0, vc label 16, status 0, mtu 1500
00:12:59: Et0/0 ACMGR: Rcv SIP msg: resp peer-to-peer msg, hdl 8C000002,
  sss_hdlB4000003
00:12:59: Et0/0 ACMGR: remote up event, SIP connected state no chg, action is
  ignore
00:13:01: %LINK-3-UPDOWN: Interface Ethernet0/0, changed state to up
00:13:02: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed
state to up
```

Use the **debug mpls l2transport vc event** command to see the AToM event messages about the VCs, as shown in Example 7-49. Watch how the messages reflect the shutdown of the interface and the healthy recovery.

Example 7-49. debug mpls l2transport vc event *Command*

```
NewYork#debug mpls l2transport vc event

NewYork(config)#interface ethernet 0/0
NewYork(config-if)#shutdown
00:14:37: AToM MGR [192.168.1.102, 100]: Local end down, vc is down
00:14:37: AToM MGR [192.168.1.102, 100]: Unprovision SSM segment
00:14:37: AToM SMGR: Submit Imposition Update
00:14:37: AToM SMGR: Submit Disposition Update
00:14:37: AToM SMGR [192.168.1.102, 100]: Event Imposition Disable
00:14:37: AToM SMGR [192.168.1.102, 100]: State [Imposition/Disposition Rdy->
  Disposition Rdy]
00:14:37: AToM SMGR [192.168.1.102, 100]: Event Disposition Disable
```

```
00:14:37: AToM SMGR [192.168.1.102, 100]: State [Disposition Rdy->Provisioned]
00:14:37: AToM SMGR: Submit SSM event
00:14:37: AToM SMGR: Event SSM event
00:14:37: AToM SMGR [192.168.1.102, 100]: sucessfully teardown sss switch for pwid
5A000000
00:14:37: AToM SMGR [192.168.1.102, 100]: sucessfully processed ssm unprovisioning
  for pwid 5A000000
00:14:39: %LINK-5-CHANGED: Interface Ethernet0/0, changed state to administratively
  down
00:14:40: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed state
  to down

NewYork(config-if)#no shutdown
00:15:16: AToM MGR [192.168.1.102, 100]: Local end up
00:15:16: AToM MGR [192.168.1.102, 100]: Validate vc, activating data plane
00:15:16: AToM SMGR: Submit Imposition Update
00:15:16: AToM SMGR: Submit Disposition Update
00:15:16: AToM SMGR [192.168.1.102, 100]: Event Imposition Enable, imp-ctrlflag
83, remote vc label 16
00:15:16: AToM SMGR [192.168.1.102, 100]: Imposition Programmed, Output Interface:
  Se5/0
00:15:16: AToM SMGR [192.168.1.102, 100]: State [Provisioned->Imposition Rdy]
00:15:16: AToM SMGR [192.168.1.102, 100]: Event Disposition Enable, disp-ctrlflag
  3, local vc label 16
00:15:16: AToM SMGR [192.168.1.102, 100]: State [Imposition Rdy->Imposition/
  Disposition Rdy]
00:15:16: AToM SMGR: Submit SSM event
00:15:16: AToM SMGR: Event SSM event
00:15:16: AToM SMGR [192.168.1.102, 100]: sucessfully processed ssm provision
  request pwid 5A000000
00:15:16: AToM SMGR [192.168.1.102, 100]: Send COMPLETE signal to SSM
00:15:16: AToM SMGR [192.168.1.102, 100]: sucessfully setup sss switch for pwid
5A000000
00:15:16: AToM SMGR: Submit SSM event
00:15:16: AToM SMGR: Event SSM event
00:15:16: AToM SMGR [192.168.1.102, 100]: sucessfully processed ssm bind for pw
id 5A000000
00:15:16: AToM MGR [192.168.1.102, 100]: Receive SSM dataplane up notification
00:15:16: AToM MGR [192.168.1.102, 100]: Dataplane activated
00:15:18: %LINK-3-UPDOWN: Interface Ethernet0/0, changed state to up
00:15:19: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed
state to up
```

To verify the flow of packets across the Layer 2 tunnel, use the **debug mpls l2transport packet data** command, as shown in Example 7-50.

Caution

Make sure to use this testing technique in a lab environment only.

Example 7-50. debug mpls l2transport packet data *Command*

```
NewYork#debug mpls l2transport packet data
00:17:44: ATOM disposition: in Se5/0, size 60, seq 0, control word 0x0
```

```
00:17:44: 00 00 0C 00 6C 00 00 00 0C 00 6C 00 90 00 00 00
          ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^
          Dest. Address     Source Address    Etype = 0x9000 = LOOP

00:17:44: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00:17:44: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00:17:44: 00 00 00 00 00 00 00 00 00 00 00 00

00:17:44: ATOM disposition: in Se5/0, size 114, seq 0, control word 0x0
00:17:44: 00 00 0C 00 6F 00 00 00 0C 00 6C 00 08 00 45 00
          ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^...
          Dest. Address     Source Address    |     Begins IP packet

                                              Etype = 0x0800 = IP

00:17:44: 00 64 00 0A 00 00 FF 01 72 3A C0 A8 64 01 C0 A8
00:17:44: 64 02 08 00 28 0D 0E D9 13 FC 00 00 00 00 00 10
00:17:44: 33 58 AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD
00:17:44: ATOM imposition: out Se5/0, size 130, EXP 0x0, seq 0, control word 0x0
00:17:44: 0F 00 88 47 00 01 00 FF 00 01 01 02 00 00 00 00
          ^^^^^ ^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^
          HDLC  |     Tunn. Label VC Label    Ctrl-word

                |     Label=16    Label=16

                |     S=0         S=1

                |     TTL=255     TTL=2

                etype = MPLS Unicast

00:17:44: 00 00 0C 00 6C 00 00 00 0C 00 6F 00 08 00 45 00
          ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^...
          Dest. Address Source Address |Begins IP packet

                                       Etype = 0x0800 = IP

00:17:44: 00 64 00 0A 00 00 FF 01 72 3A C0 A8 64 02 C0 A8
00:17:44: 64 01 00 00 30 0D 0E D9 13 FC 00 00 00 00 00 10
00:17:44: 33 58 AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
00:17:44: AB CD
```

Note

Note that in Example 7-50, the offline hand decoding of the packets is shown in bold.

You can see in Example 7-50 that the disposition packets show only the EoMPLS Payload, whereas the imposition packets also include the EoMPLS header.

In the first packet, the Ethertype of 0x9000 indicates a loopback packet. For this reason, the source and destination MAC addresses are the same. The second packet shows an IP packet transported in EoMPLS. Finally, in the third packet with the imposition operation, you can also see the Tunnel, MPLS, and AToM headers. Specifically, the Layer 2 is Cisco HDLC with an HDLC type of 0x8847, indicating that MPLS follows. A two-level MPLS stack includes the Tunnel label of 16 and a VC label of 16. Note that these two values do not need to be the same. The label stack is followed by a 4-byte control word and finally an Ethernet frame with an Ethertype of 0x0800 transporting an IP datagram with an ICMP packet.

Troubleshooting EoMPLS on Switches

Troubleshooting commands on switches are, for the most part, the same as those on routers. However, the output might be different, as you learn in this section. For instance, use the output of the command **show mpls l2transport vc** from Example 7-51 to get information about the VCs just as you would use this command on routers.

Example 7-51. show mpls l2transport vc *Command on a Switch*

```
Metro-switch#show mpls l2transport vc
Transport Client     VC     Trans Local     Remote     Tunnel
VC ID     Intf       State  Type VC Label   VC Label   Label
200       Vl200      UP     vlan 215        215        implc-null
```

Table 7-5 explains the most pertinent fields of the **show mpls l2transport vc** command.

Table 7-5. *Fields of the show mpls l2transport vc Command*

Field	Description
VC ID	The VC ID configured by the **mpls l2 route** or **xconnect** command; must match on both ends.
Client Intf	Indicates which Layer 2 interface is being used.
VC State	The UP state shows whether the VC ever saw traffic.
Trans Type	The available results include vlan for VLAN based and Ether for port based.

Field	Description
Local VC label	Indicates that the local VC label is being used. For Cisco 7600 series switches, the local label is derived from the VLAN (local label = VLAN + 15).
Remote VC label	The label advertised by the remote PE and used by this PE to reach the CE on the other end of the LSP.
Tunnel Label	Identifies the outer label used to switch the packets between the PEs. An implicit-null label is shown for the back-to-back connection.

For more in-depth information about the VCs, use the **show mpls l2transport vc detail** command, as shown in Example 7-52.

Example 7-52. show mpls l2transport vc detail *Command*

```
Metro-switch# show mpls l2transport vc detail
vcid: 200, local groupid: 24, remote groupid: 102 (vc is up)
client: Vl200 is up, destination: 1.2.2.1, Peer LDP Ident: 1.2.1.1:0
local label: 215, remote label: 215, tunnel label: implc-null
outgoing interface: s0/0, next hop: point2point
Local MTU: 1500, Remote MTU: 1500
Remote interface description: Vlan 200
imposition: LC Programmed
current imposition/last disposition slot: 2/32
Packet totals(in/out): 6246/6159
byte totals(in/out): 536999/444722
```

Table 7-6 describes the various output fields.

Table 7-6. *Fields of the* show mpls l2transport vc detail *Command*

Field	Description
groupid	AToM group ID advertised in the VC FEC

Field	Description
destination	IP address of the targeted LDP session
outgoing interface	Indication of which interface is being used to transmit
next hop	MAC address of the next hop or point2point
remote interface description	Interface on remote PE that connects to the CE
imposition	Indication of whether the line card has been programmed (that is, imposition rewrite resolved)
current imposition	Indication of the slot performing imposition for this VLAN
last disposition	Indication of the last slot performing disposition*
packet or bytes in	Per VC counters for disposition
packet or bytes out	Per VC counters for imposition
*7600 supports per-destination load sharing. If multiple connections to the MPLS cloud exist, the imposition traffic can be transmitted on one interface, and the disposition traffic for the same VLAN can be received in another interface.	

Summary

In this chapter, you learned how to configure and troubleshoot EoMPLS. EoMPLS specifies transport of Ethernet frames across an MPLS core based on the draft-martini. Following are some of the important EoMPLS points to remember:

- To establish an EoMPLS circuit, you must designate a specific physical port for an enterprise customer on a PE.
- EoMPLS VCs are point-to-point circuits.
- Traffic sent between the imposition/disposition routers (PEs) over an EoMPLS VC takes the same path across the IP/MPLS backbone unless the LSP changes because of routing changes inside the provider network.
- A customer can have more than one EoMPLS VC per physical port. In this case, the PE should be able to distinguish between specific 802.1q headers for each EoMPLS VC.

Chapter 8. WAN Protocols over MPLS Case Studies

This chapter covers the following topics:

- Setting up WAN over MPLS pseudowires
- Introducing WAN protocols over MPLS
- Configuring WAN protocols over MPLS case studies
- Advanced WAN AToM case studies

This chapter covers the functional details and implementation of the transport and tunneling of different WAN protocols over Multiprotocol Label Switching (MPLS). Building upon the WAN data-link protocol primer introduced in Chapter 5, "WAN Data-Link Protocols," and the overview of Any Transport over MPLS (AToM) introduced in Chapter 6, "Understanding Any Transport over MPLS," this chapter explores the operation, configuration, and troubleshooting of High-Level Data Link Control (HDLC), PPP, Frame Relay, and AToM. It also covers the different operational modes of these protocols.

Similarly to previous chapters, this chapter walks you through the different building blocks and detailed configuration and operation aspects of WAN protocols over MPLS. Several case studies present the service provider side of the configuration. Although customer premise configurations are included, they do not vary from traditional Layer 2 transport technologies, because WAN over MPLS is transparent to the end user.

Setting Up WAN over MPLS Pseudowires

In the most general sense, the transport and tunneling of Layer 2 WAN protocols is no different from the transport of Ethernet over MPLS. As discussed in detail in previous chapters, draft-martinibased technologies allow the transport of Layer 2 packets over an MPLS network over point-to-point pseudowires. Although the underlying architecture does not change, several Layer 2-specific customizations from the architectural model allow the transport of specific Layer 2 WAN protocols. This section covers some of these similarities and differences.

Control Plane

The setup and maintenance of AToM pseudowires is based on the targeted (also referred to as *directed*) Label Distribution Protocol (LDP) session between a pair of provider edge (PE) routers. You can bind the Layer 2 attachment circuit (AC) to the label by using the LDP Label Mapping message. Several pseudowires that are signaled by the targeted LDP session between PEs use one packet-switched network (PSN) tunnel label-switched path (LSP) signaled by Link LDP (IGP) or another label distribution protocol, such as Resource Reservation Protocol Traffic Engineering (RSVP-TE).

The fact that multiple pseudowires use the same PSN tunnel and that only PE devices participate in pseudowire signaling adds to the scalability of the AToM solution, given that only PEs know about pseudowire to attachment circuit mappings (PW<->AC), whereas the core P routers remain uninformed of them. The core only knows about Interior Gateway Protocol (IGP) layer LSPs.

> **Note**
>
> The terms *virtual circuit* (VC) and *pseudowire* are used interchangeably as the mechanism that transports the elements of an emulated service between PE routers over the MPLS PSN.

One of the special cases of setting up WAN over MPLS pseudowires is the use of specific interface parameters in the Pseudowire ID FEC element, which you learned about in Chapter 6. For example, whereas some interface parameters are applicable to multiple VC types or emulated services (such as maximum transmission unit [MTU] and Interface Description), others are valid only for specific VC types.

The maximum number of concatenated ATM cells interface parameter is applicable only to the different ATM cell transport modes. The Frame Relay DLCI length interface parameter indicates the length of the DLCI field in the Frame Relay header and pertains only to Frame Relay over MPLS.

The following subsections explain more about the transport of WAN protocols over MPLS PSNs:

- Pseudowire types used

- Data plane encapsulation
- Usage of the control word
- MTU size requirements

Pseudowire Types Used

The 15-bit pseudowire type (or VC type) field identifies the type of pseudowire. The different VC types used in the transport of WAN protocols over MPLS are shown in Table 8-1.

Table 8-1. *Pseudowire Types Used in WAN Transport*

Pseudowire Type	Description	Usage
0x0001	Frame Relay DLCI	Frame Relay over MPLS DLCI Mode
0x0002	ATM AAL5 SDU VCC	ATM over MPLS AAL5 SDU Mode
0x0003	ATM Transparent Cell	ATM over MPLS Cell Relay Port Mode
0x0006	HDLC	HDLC over MPLS
0x0007	PPP	PPP over MPLS
0x0009	ATM n-to-one VCC[1] cell	ATM over MPLS Cell Relay VC Mode
0x000A	ATM n-to-one VPC[2] cell	ATM over MPLS Cell Relay VP Mode

[1] Virtual channel connection

[2] Virtual path connection

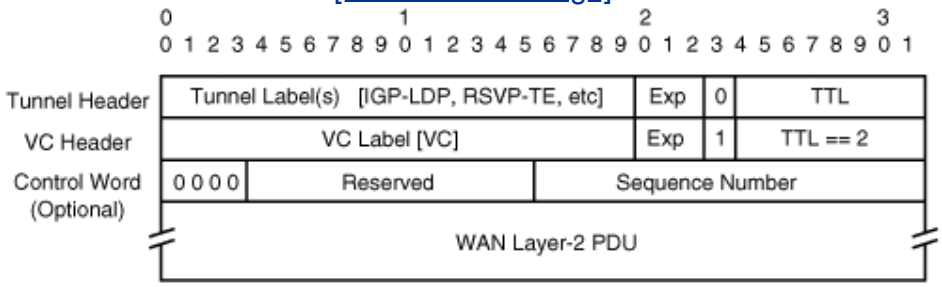
The case study sections later in this chapter reference these pseudowire types.

Data Plane Encapsulation

The encapsulation of Layer 2 WAN protocol data units (PDU) is specified in the encapsulation martini draft and subsequent Pseudowire Emulation Edge-to-Edge (PWE3) working group derivative drafts spawned from it (see Figure 8-1). Essentially, the Layer 2 PDU is encapsulated in an MPLS stack where the inner or bottom label (the VC label contained in the VC MPLS shim header) identifies the Layer 2 attachment circuit and is advertised in the LDP targeted session. The tunnel header is in turn comprised by 0 or more MPLS headers from the PSN tunnel control plane.

Figure 8-1. ***Encapsulation of WAN Protocols over MPLS***

[View full size image]



Note that the depth of the tunnel header in terms of the number of MPLS shim headers can vary along the LSP. The tunnel header has 0 labels in the case of Penultimate Hop Popping (PHP), whereby the egress PE advertises an implicit null label in the link LDP session. In the most common case, the tunnel header is made of one label distributed by the core LDP session. However, the tunnel header can contain more than one label in cases such as traffic engineering (TE) with Fast Reroute (FRR), MPLS-VPN Carrier Supporting Carrier (CSC), or inter-AS (IAS) environments or when using the reserved Router Alert label of 1.

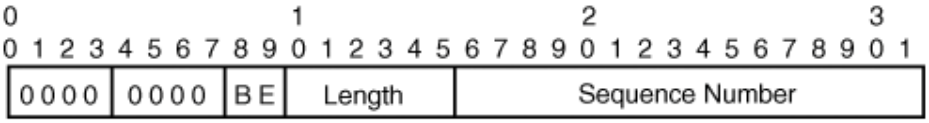
A 32-bit control word might reside between the VC label and the WAN Layer 2 PDU. The control word negotiation and usage are covered in Chapter 6. The upcoming section discusses the control word usage in the context of WAN protocols over MPLS.

Usage of the Control Word

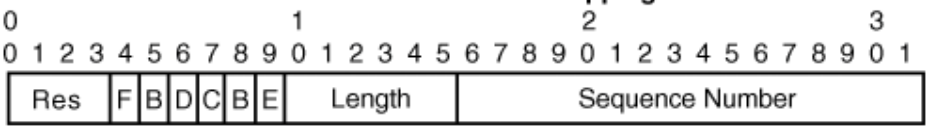
During pseudowire setup, the usage of a control word is negotiated by setting the C bit in the pseudowire ID FEC element. Figure 8-2 compares the control word format for different WAN protocols over MPLS.

Figure 8-2. ***Control Word Format for Different WAN Protocols***

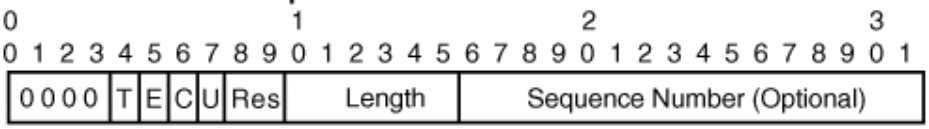
PPP and HDLC



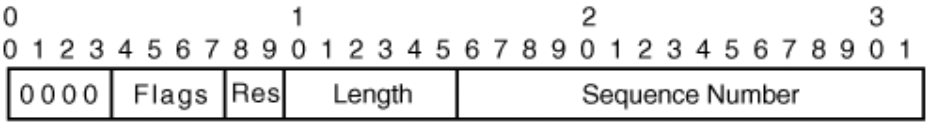
FRoMPLS Header Structure for One-to-One Mapping Mode



AAL5 CPCS-SDU Encapsulation



ATM N-to-One Cell Mode



The following describes each component from Figure 8-2:

- All encapsulations

  First Nibble The first four bits are set to 0x0 to prevent aliasing with IP packets over MPLS. For IP over MPLS (IPoMPLS), the first nibble coincides with the IP Header's version field: 0x4 for IPv4 and 0x6 for IPv6.

  B- and E-Bits These two bits are fragmentation indicators that are used in PWE3 fragmentation and reassembly.

  Length The 6-bit Length field permits values from 0 to 64 only. You use this field when the link layer protocol in the PSN requires a minimum frame length. If the total length of an AToM packet's payloadincluding the control wordis less than 64 bytes, you set the Length field to the length of the AToM packet's payload, including the 4-byte control word. Otherwise, you set it to 0.

- Frame Relay over MPLS (FRoMPLS, also referred to as FRoPW in Internet Engineering Task Force [IETF] documents)

  F-bit FR forward explicit congestion notification (FECN) bit.

  B-bit FR backward explicit congestion notification (BECN) bit.

  D-bit FR DE bit, which indicates the discard eligibility.

C-bit FR frame command/response (C/R) bit.

- AAL5 CPCS-SDU (often referred to as AAL5 SDU over MPLS)

  T-bit Transport type. Indicates ATM admin cell or AAL5 payload.

  E-bit Explicit Forward Congestion Indication (EFCI) bit.

  C-bit Cell loss priority (CLP) bit.

  U-bit Command/Response field.

Although the control word is optional for some encapsulations such as PPP, HDLC, and cell relay (ATM cell mode transport), it is required for Frame Relay and ATM AAL5 over MPLS. This requirement for Frame Relay and ATM AAL5 transport modes is because the control word carries control information. This information is specific for the Layer 2 that is being emulated that is not carried in the AToM payload. For example, you will see in the upcoming section "Frame Relay over MPLS" that the Frame Relay Q.922 header is stripped at the ingress PE at MPLS imposition, so the control word carries the FECN, BECN, and DE bits that were present in the now-stripped header.

MTU Requirements

Every time you encapsulate a PDU with a new protocol header, you need to take into account maximum transmission unit (MTU) considerations. In a Layer 2 VPN, PEs during imposition are encapsulating a customer edge's (CE) Layer 2 PDU to be switched across an MPLS network. You need to calculate a series of associated overheads to properly set up the core MTU. You can subdivide these overheads into three categories:

- **Transport overhead** This is the overhead that is associated with the specific Layer 2 that is being transported. Table 8-2 lists transport overheads for different WAN protocols.

Table 8-2. *Transport Overhead for Different WAN Protocols over MPLS*

Transport Type	Transport Header Size	Transport Header Reason [Bytes]
Frame Relay DLCI, Cisco encapsulation	2 bytes	Ethertype [2]

Transport Type	Transport Header Size	Transport Header Reason [Bytes]
Frame Relay DLCI, IETF encapsulation	2-8 bytes	SNAP => Control [1] + Pad [1] + NLPID [1] + OUI [3] + Ethertype [2]
Cisco HDLC	4 bytes	Address [1] + Control [1] + Ethertype [2]
PPP	2 bytes	PPP DLL Protocol [2]
AAL5	0-32 bytes	Header

- **MPLS overhead** This is the overhead that MPLS headers add (including the VC label). It is equal to 4 bytes times the number of MPLS headers included.
- **AToM overhead** This is the overhead incurred because of the control word. It is equal to 4 bytes.

ATM Cell transport is deliberately left out of Table 8-2. In ATM cell relay over MPLS (CRoMPLS), the packets that are transported are of a fixed length of 52 bytes. They can be concatenated up to a maximum number of cells, making MTU calculation different than for all other Layer 2 transports. The upcoming section "Encapsulations and Packet Format for Cell Transport" covers this topic.

Frame Relay with IETF encapsulation refers to RFC 2427, "Multiprotocol Interconnect over Frame Relay," which makes RFC 1490 obsolete. For Frame Relay with IETF encapsulation DLCI transport, the overhead is considered variable; many packets have a transport overhead of 2 bytes: the control byte of 0x03 and the Network Layer Protocol Identifier (NLPID). This is the minimum overhead in Frame Relay IETF. However, in some other cases (such as when a protocol does not have an NLPID assigned), the NLPID value of 0x80 indicates that a Subnet-work Attachment Point (SNAP) header follows. The Organizationally Unique Identifier (OUI) of 0x000000 indicates that a 2-byte Ethertype follows. In this case, in which the upper layer protocol has no NLPID, the transport overhead is 8 bytes, and you need to use the worst case when setting the core MTU.

On the other hand, for Frame Relay Cisco encapsulation, a 2-byte Ethertype is always used instead of control and NLPID, making the transport overhead permanently 2 bytes.

Tip

When you are studying the packet formats for the different WAN protocols in the upcoming sections, a good exercise is to come back to Table 8-2 and identify the different fields that make up the transport overhead.

You can use the following generic formula to calculate the core MTU from the edge MTU. The edge MTU is the MTU configured in the interface of the CE-facing PE:

```
Core MTU ≥Edge MTU + Transport Header + AToM Header + (MPLS Label Stack *
MPLS Header Size)
```

Introducing WAN Protocols over MPLS

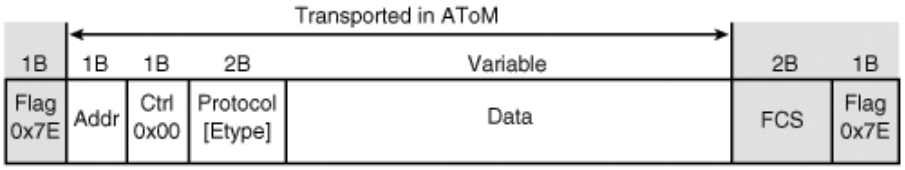
The following sections detail the different WAN protocols over MPLS. This section presents fundamental concepts about the transport of HDLC frames over MPLS (HDLCoMPLS), Point-to-Point Protocol over MPLS (PPPoMPLS), FRoMPLS, and different flavors of Asynchronous Transfer Mode over MPLS (ATMoMPLS).

HDLC over MPLS

As you learned in Chapter 5, Cisco HDLC is proprietary. It differs from the International Organization for Standardization (ISO) standard HDLC in that Cisco HDLC does not perform windowing and retransmission. In addition, the higher layer protocol identification is not standardized. Cisco HDLC uses the Ethernet Type value to identify the higher layer protocol that it carries. The frame format and bit-stuffing techniques used in HDLC are defined in the American National Standards Institute (ANSI) T1.618 standard.

Figure 8-3 depicts the Cisco HDLCoMPLS frame format highlighting the fields that are stripped and removed at AToM imposition.

Figure 8-3. ***Cisco HDLCoMPLS Packet Format***



You can see that, except for the start and the end of the frame flag equal to 0x7E and the frame checksum carried in the frame check sequence (FCS), the complete HDLC frame including all header fields is transported unmodified. Frame flags and FCS fields are removed at ingress and re-created at egress. All header fields are uninspected, meaning that no one attempts to interpret the header and make assumptions.

At this point, you might be thinking that HDLC is too basic, so HDLCoMPLS is basic, too. Why would you want to use HDLCoMPLS? The answer to that question is specific: The strength of HDLCoMPLS is its simplicity. Because someone checks only the flag and FCS fields, you can transport an HDLC-like protocol by using HDLCoMPLS. As long as the Layer 2 protocol contains a 0x7E flag and a frame checksum as a trailer, you can tunnel it by using HDLCoMPLS in a port-to-port or interface-to-interface mode. In fact, that is why you can transport Cisco HDLC over HDLCoMPLS. As far as HDLCoMPLS is concerned, standard HDLC and Cisco HDLC frames are indistinguishable. Other protocols that share HDLC-like framing and can be transported in a port-mode fashion are Synchronous Data Link Control (SDLC), CCITT No. 7 signaling, IBM Systems Network Architecture (SNA), PPP, Frame Relay, and X.25.

PPP over MPLS

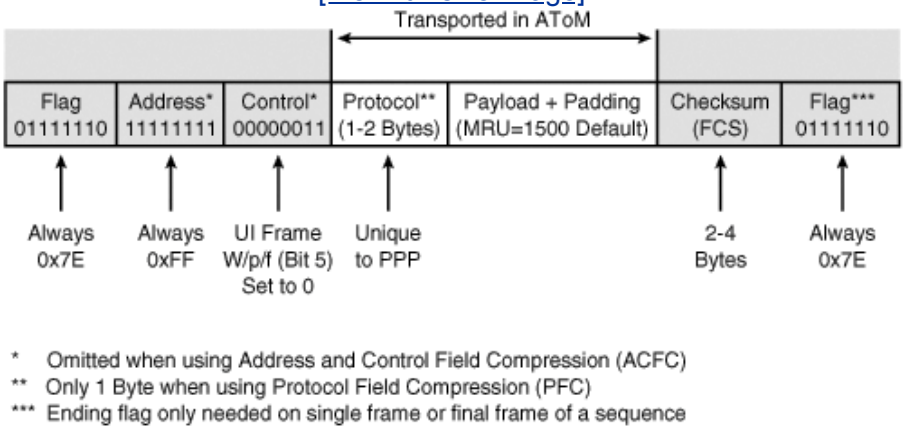
Modeled after the HDLC frame with the addition of the protocol fields, PPP is a standard method for transporting multiprotocol datagrams over point-to-point links.

The transportation of PPP frames over MPLS is quite similar to the transportation of HDLC frames. The two frames share many common characteristics, including the same control word format.

In contrast to HDLCoMPLS, however, PPPoMPLS requires some interpretation of the PPP header. Specifically, in addition to the 0x7E flag and FCS fields being removed at imposition, the Address (0xFF) and Control (0x03) fields are stripped at the imposition router. These fields are not transported in PPPoMPLS packets (that information can be implicitly gleaned because the VC type is PPP) and re-created at the disposition PE before transmitting to the remote CE. Figure 8-4 shows the PPPoMPLS packet format.

Figure 8-4. *PPPoMPLS Packet Format*

[View full size image]



As you can see in Figure 8-4, all media-specific framing information is excluded and not transported. The PPP PDU is transported in its entirety, including the Protocol field (whether compressed using Protocol Field Compression [PFC] or not).

As a result, the following will not work:

- FCS alternatives
- Address and Control Field Compression (ACFC)
- Asynchronous Control Character Map (ACCM)

PFC, however, will work.

Because in PPPoMPLS the Address and Control fields are interpreted and not transported, CEs should not negotiate ACFC. ACFC is not recommended anyway, because the performance penalty in alignment is larger than the bandwidth savings. Using ACFC results in a 2-byte PPP header, and using PFC results in a 3-byte PPP header; in both cases, the PPP PDU does not start after a 32-bit-word. If you want the CEs to perform ACFC anyway, use HDLCoMPLS.

Note

Using either ACFC or PFC (both defined in RFC 1661, "The Point-to-Point Protocol [PPP]") changes the alignment of the network data inside the frame, which in turn decreases switching efficiency in both the ingress and egress CE.

Because the Protocol field is transported unmodified, you can negotiate PFC between PEs. That is not recommended though, because of the same word alignment reasons explained for ACFC.

One important aspect of the transport of PPPoMPLS is the PPP Finite State Machine (FSM) negotiation, specifically between which peers PPP negotiation (that is Link Control Protocol [LCP], authentication, and Network Control Protocols [NCP]) occur. In PPPoMPLS, the PPP negotiation takes place directly between CE devices. In other words, PPP does not actually run or terminate on the PE devices. After you configure an interface for PPP encapsulation in a PE router, PPP leaves the closed state and tries to negotiate LCP and NCP. Then, when a pseudowire is configured for PPPoMPLS, PPP enters a closed state, and LCP and NCP negotiation is nonexistent with the CE.

Frame Relay over MPLS

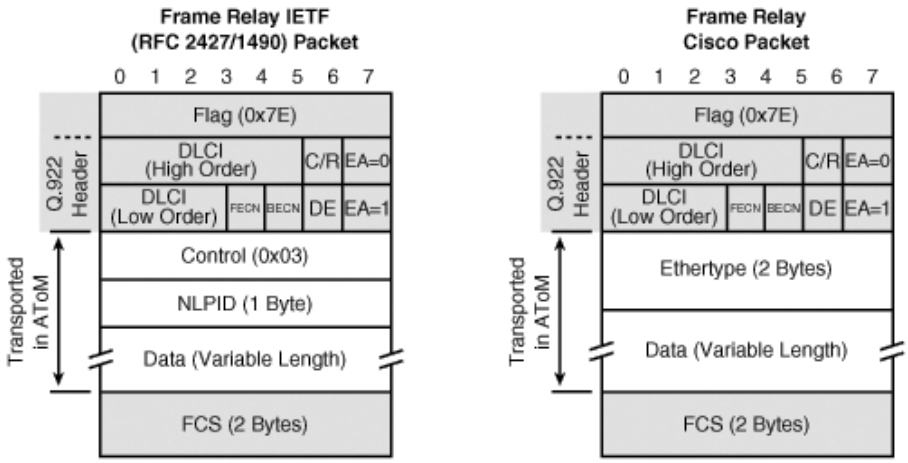
At this point, you have already learned of a way to transport Frame Relay frames in port mode over an MPLS PSN: using HDLCoMPLS. When you are using HDLCoMPLS, Local Management Interface (LMI) runs directly between CE devices and not between PE and CE. Therefore, if the CE devices are Frame Relay switches, use Frame Relay Network-to-Network Interface (NNI); if the CE devices are routers, use Frame Relay data communication equipment-data terminal equipment (DCE-DTE) LMI.

There exists, however, a much more granular way to transport Frame Relay frames over MPLS, and that is by using Frame Relay DLCI mode. FRoMPLS DLCI mode enables you to transport a specific Frame Relay VC over an MPLS cloud. Perhaps even more importantly, it provides the framework for local management between PE and CE device by means of Frame Relay LMI. LMI enables the exchange of Link Status by way of a keepalive mechanism using Status Enquiry and Status messages, in addition to Frame Relay PVC or DLCI Status using a Full Status message. In contrast to PPPoMPLS, FRoMPLS has some PE-CE management exchange because LMI runs between PE and CE devices.

As detailed in Chapter 5, you can configure two different Frame Relay encapsulations on a Cisco router: IETF and Cisco. Figure 8-5 depicts these two encapsulation methods.

Figure 8-5. *FRoMPLS Packet Format*

[View full size image]



From Figure 8-5, you can see that for both encapsulation methods, and similarly to HDLCoMPLS and PPPoMPLS, the Flag of 0x7E and the FCS are stripped at imposition and are not transmitted over AToM. The 2-byte Q.922 header is also stripped at imposition and is not transported in AToM. In consequence, a mechanism should be in place to inform the remote PE of the value of all the fields in the Q.922 header so that the remote PE can re-create it and send a Frame Relay packet to the remote CE without losing information. The following list details the different methods for conveying the Q.922 FR header information to the remote PE:

- **DLCI** PEs do not exchange the DLCI at any moment. It is a local PE responsibility to map the local VC that is exchanged by LDP in the pseudowire ID forward error correction (FEC) to the attachment circuit (that is, the Frame Relay DLCI). Remember that DLCIs are locally significant, and the PSN is acting as a Frame Relay cloud.

- **C/R** The Command/Response bit is sent in the C-bit in the Frame Relay over Pseudowire (FRoPW) Header (that is, control word). Refer to Figure 8-2.

- **FECN** The FECN bit is sent in the F-bit in the FRoPW Header (that is, control word).

- **BECN** The BECN bit is sent in the B-bit in the FRoPW Header (that is, control word).

- **DE** The discard eligible bit is sent in the D-bit in the FRoPW Header (that is, control word).

- **EA** During pseudowire establishment, PEs negotiate the characteristic of a Frame Relay PVC with respect to extended addressing. They do this by including the optional Frame Relay DLCI length interface parameter in the VC FEC element in the FEC TLV inside the LDP Label Mapping message. The optional Frame Relay DLCI length interface parameter (interface parameter type 0x08) indicates the length of the FR Header and can have a value of 2 or 4.

In summary, the C/R, FECN, BECN, and DE are sent on the control word flags on a per-packet basis. In contrast, LDP sends the extended addressing characteristics of the FR PVC on pseudowire establishment P. This implies that on a given Frame Relay PVC, all packets need to use normal addressing (Q.922 header of 2 bytes) or extended addressing (that is, Q.922 header of 4 bytes), but not mixed.

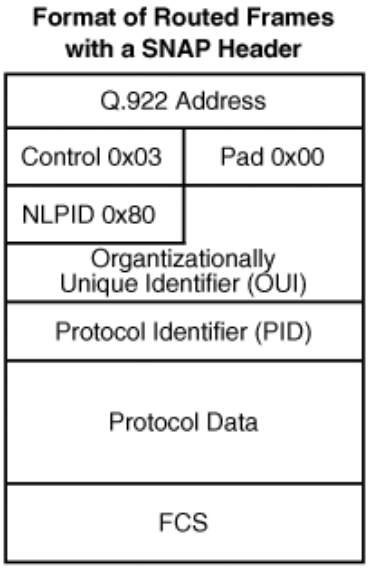
Note

FRoMPLS requires the control word.

As shown in Figure 8-5, the difference between IETF and Cisco encapsulation for Frame Relay is the upper layer protocol identification. You can configure a Cisco router to run either of the two encapsulations.

For IETF encapsulation, the format for routed frames allows an NLPID value of 0x80, indicating that a SNAP header follows (see Figure 8-6). Because not all protocols have an NLPID value assigned (NLPID space is limited), you have to use the SNAP form in such cases. Using the SNAP form increases the transport overhead for Frame Relay IETF to a total of 8 bytes.

Figure 8-6. *Format of Routed Frames with SNAP Header*



One of the direct consequences of the dual encapsulation method using NLPID or SNAP is that IP datagrams over Frame Relay can be encapsulated in two different ways:

- Using an IP assigned NLPID of 0xCC. This is the preferred method specified in RFC 2427 and also the method used in Cisco routers.
- Using an NLPID value of 0x80 indicates that a SNAP header follows. A SNAP header includes an OUI of 0x000000 that indicates an Ethertype follows. The Ethertype of 0x0800 is for IP.

ATM over MPLS

You can categorize the transport of ATMoMPLS as follows:

- **ATM AAL5 over MPLS** Transport of RFC 2684/1483 AAL5 SDUs over MPLS.
- **ATM Cell over MPLS** Relay of ATM cells over MPLS. You can subcategorize ATM Cell Relay over MPLS as follows:

  Port Mode Transport of ATM cells from an ATM interface.

  VP Mode Transport of ATM cells from an ATM virtual path (VP).

  VC Mode Transport of ATM cells from an ATM virtual circuit (VC).

ATMoMPLS presents three different degrees of transport granularity: granularity at the VC, VP, or Port level. If a user wants to transport an ATM VC over MPLS, he has the option of doing AAL5 over MPLS (AAL5oMPLS) or CRoMPLS VC mode. A user has to use CRoMPLS if the ATM frames transported over the VC are not AAL5 but a different adaptation layer, such as AAL2. In the next section, you learn some of the differences between the two. If a user wants to transport an ATM VP (for example, for virtual trunking applications) or an ATM port (for trunking or cell transport applications), the only mode available is CRoMPLS.

Encapsulations and Packet Format for AAL5 Transport

ATM is the Layer 2 WAN technology that involves more data plane complexity than the others. As detailed in Chapter 5, the ATM protocol stack includes the ATM Adaptation Layer (AAL) and the ATM layer. AAL is in turn divided into two sublayers:

- Convergence sublayer (CS)
- Segmentation and reassembly (SAR) sublayer

The first AToM mode for transporting ATM is the AAL5 mode, in which the pseudowire transports AAL5 common part convergence sublayer (CPCS) service data units (SDU).

Figure 8-7 shows that the AAL5 CPCS protocol data unit (PDU) is composed of the CPCS-PDU payload or CPCS-SDU, padding to ensure that the CPCS-PDU length is an integer multiple of 48 bytes for the SAR layer, and a CPCS-PDU trailer. The CPCS-PDU trailer in turn contains a 1-byte User-to-User indication (CPCS-UU) field, a 1-byte common part indicator (CPI), a 2-byte Length indicator that specifies the length of the payload in octets, and a 4-byte cyclic redundancy check (CRC). Only the CPCS-SDUthe CPCS-PDU's payload or the CPCS-PDU without its padding and traileris transported in AAL5oMPLS SDU mode.

Figure 8-7. *AAL5oMPLS Packet Format*

[View full size image]

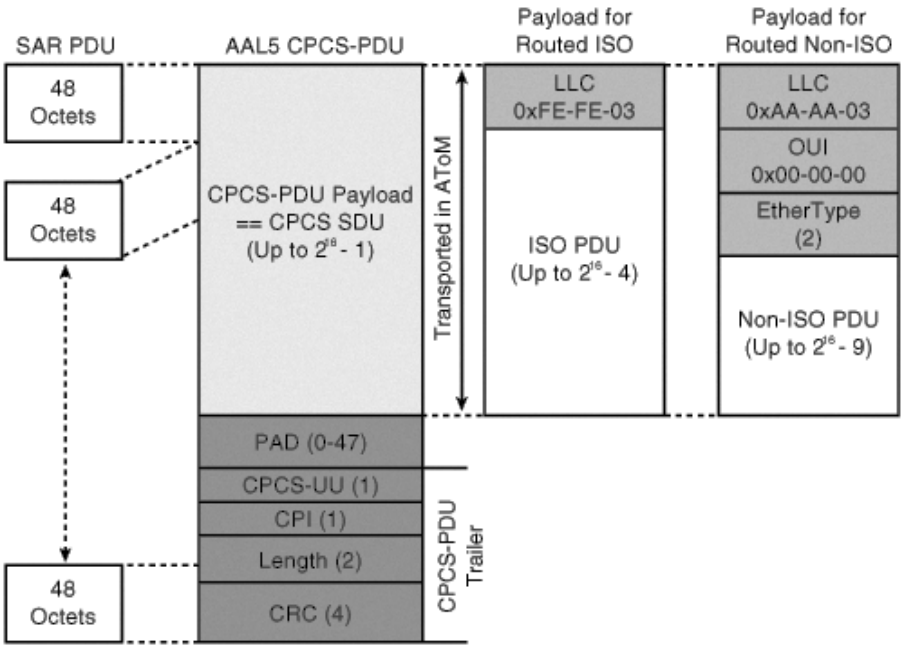
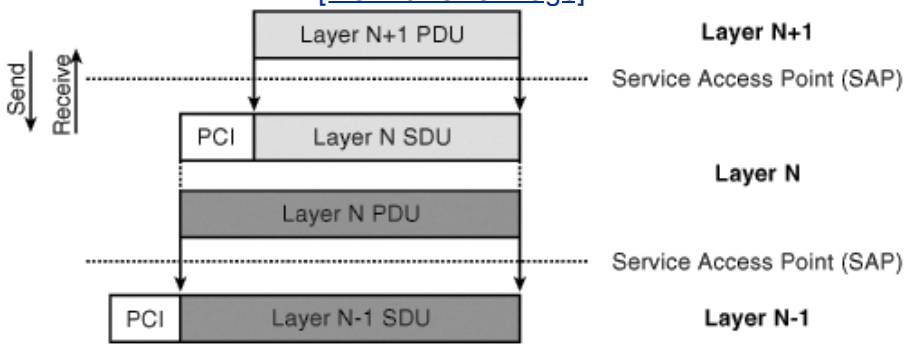


Figure 8-7 shows the format of the AAL5 CPCS-SDU for routed ISO (such as Connectionless Network Service [CLNS]) and non-ISO (such as IP) protocols. These formats are useful in understanding the different overheads that play for different protocols that are transported in AAL5oMPLS.

The only supported AAL5 transport mode over MPLS is AAL5 SDU mode. In protocol layering or protocol encapsulation, a service access point (SAP) exists between two layers (see Figure 8-8). Each protocol sends (down direction) and receives (up direction) data via the SAP. The PDU at a higher layer N+1 becomes the layer N SDU when traversing the SAP. At layer N, protocol control information (PCI) is added to the SDU to form the PDU at that layer N, which in turn becomes the SDU at layer N-1. In summary, at a given layer, PDU = PCI + SDU. PDU includes the protocol control data (PCI) plus the carried data (SDU). The PDU at layer N is the SDU at layer N-1. Any data that enters the AAL5 layer becomes an SDU for AAL5 CS.

Figure 8-8. *Understanding PDU Versus SDU*

[View full size image]



In AAL5 SDU mode, the AToM payload's (AAL5 CPCS-SDU) length need not be an integer multiple of 48 bytes, because the padding and trailer were stripped before AToM encapsulation.

In AAL5oMPLS, the ingress PE receives ATM cells from the customer premises equipment (CPE), and it needs to reassemble them to send an AAL5 SDU over MPLS in a single packet. The cell headers are not transported, so it is critical to understand how the different ATM cell header fields are conveyed to the other end. In AAL5oMPLS, the presence of a control word is required, although its use is optional (refer to Figure 8-2). The following list enumerates the different ATM cell header fields and how they are transported in AAL5 SDU mode:

- **Virtual path identifier (VPI) and virtual circuit identifier (VCI)** PE routers do not carry or exchange the VPI and VCI. The PEs keep the VPI and VCI values in the state of the pseudowire, and the disposition PE router rewrites the VPI and VCI value.
- **Payload type identifier (PTI)** The PTI contains three fields:

  C This field indicates whether the cell contains user data or control (management) data; you can loosely map this field to the transport bit (T-bit) in the AAL5 control word. The T-bit indicates whether an AToM packet contains a cell or an AAL5 SDU. OAM cells on an AAL5 pseudowire are sent as cells and set the T-bit.

  EFCI The EFCI is transported in the E-bit in the control word.

  End of packet (EOP) You do not need the end of the AAL5 packet bit because the cells are reassembled in the ingress PE.

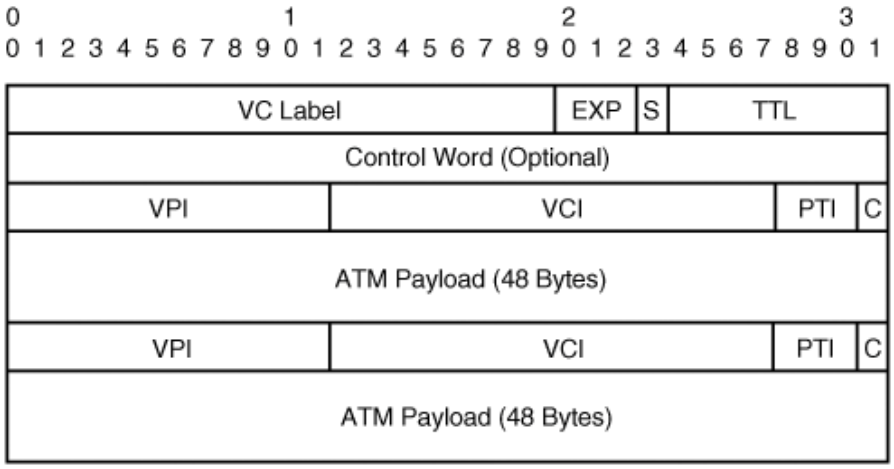
- **CLP** The CLP-bit is carried in the C-bit in the control word.

In AAL5 SDU mode, the ingress PE reassembles the AAL5 CPCS-PDU, strips off the padding and CPCS-PDU trailer, and sends the AAL5 CPCS-SDU in an AToM packet; at the other end, the egress PE regenerates the PAD and AAL5 trailer. Each AAL5oMPLS packet contains an AAL5 SDU or an OAM Cell. AAL5 SDU mode has less overhead compared to AAL5 PDU mode, because the 8-byte trailer and padding are not transported. The padding can be significant for small packets, given that the smallest TCP packet requires at least two cells when transported natively over ATM.

Encapsulations and Packet Format for Cell Transport

In contrast, the different CRoMPLS modes operate at the ATM layer of the ATM reference model. Figure 8-9 depicts the encapsulation of n-to-one CRoMPLS by including two ATM cells in an AToM packet. Cisco implementation of ATM CRoMPLS uses n-to-one cell transport modes.

Figure 8-9. *CRoMPLS Packet Format*



From Figure 8-9, you can see that for cell relay, the control word is optional. However, Cisco implementation of CRoMPLS advertises the disposition capability for the control word (C-bit in pseudowire ID FEC) and uses the control word whenever possible (that is, when the remote side supports the control word as a disposition capability).

In n-to-one ATM cell transport, a complete ATM layer cell header is appended after the control word, followed by 48 bytes of ATM cell payload.

Note

In the ATM reference model, the cell header is only 4 bytes long at the ATM layer. The Transmission Convergence (TC) sublayer calculates and appends the fifth byte (header error control [HEC]), which is a checksum of the ATM cell header in the ATM physical layer. The TC sublayer handles functions such as cell delineation and error detection and correction by adding a 1-byte CRC. Because cell relay acts in the ATM layer, and for alignment and efficiency reasons, the HEC byte is not included in the transport of ATM cells over MPLS.

An ATM cell is transported with 52 bytes in the AToM payload out of the 53 bytes of the ATM cell, thereby carrying VPI, VCI, PTI, and CLP fields.

Note

In contrast to AAL5oMPLS, in CRoMPLS, encapsulation for a user cell and management or OAM cell is the same.

The VPI field in ATM cell encapsulation is 12 bits long to accommodate the NNI ATM header format's VPI range. The field is interpreted as VPI; therefore, if the cell is actually using UNI ATM header format, the Generic Flow Control field (first nibble) is always 0.

In n-to-1 ATM CRoMPLS, an imposition PE can concatenate multiple ATM cells into a single AToM PDU. Concatenation of cells (also called *cell packing*) is optional in transmission at the ingress PE and is supported in ATM cell port, VP, and VC modes. You can view the cell concatenation as a disposition property, in which an egress PE can support disposition for concatenated cells up to a certain number of cells. The imposition PE knows the maximum number of cells that can be linked because it is indicated during pseudowire establishment with a new interface parameter. The Maximum Number of concatenated ATM cells interface parameter (interface parameter type 0x02) specifies the maximum number of cells in a single AToM PDU that you can process at disposition.

> **Note**
>
> Because the interface parameter is included in the Label Mapping LDP message at VC setup, changing its value would tear down and resignal the pseudowire.

The Maximum Number of concatenated ATM cells interface parameter is required for all the different ATM cell transport modes (port, VP, and VC modes). If the egress PE supports concatenated cells, the ingress PE should only link cells up to the Maximum Number of concatenated ATM cells interface parameter received from the remote PE in the pseudowire ID FEC element.

> **Note**
>
> The Maximum Number of concatenated or "packed" cells is configurable and does not need to be identical between the PE routers. The configuration and details for this feature are included in the section titled "Case Study 8-8: Packed Cell Relay over MPLS," later in this chapter.

Naturally, the cell concatenation encapsulation is more bandwidth efficient than single cell relay (that is, sending one ATM cell per AToM packet), because multiple cells share the AToM overhead. Each ATM cell that is encapsulated in an AToM frame is 52 bytes long, and each AToM packet is at least 64 bytes (52 bytes + two MPLS headers + control word). On the other hand, the compromise is that when you are using cell concatenation, the imposition router introduces more delay when waiting for cells to be packed. Even with a value for packed cells greater than 1, OAM cells are transported in a single AToM packet.

In ATM cell transport, the MTU considerations are different from all other Layer 2 transports. In ATM cell transport, you can predict the maximum AToM PDU size differently than with the other Layer 2 technologies, because the packet size is fixed for ATM cells. You can configure the maximum number of cells to be packed into an MPLS packet up to the MTU of the interface divided by 52 bytes for each ATM cell. Alternatively, you can easily calculate the maximum length of an AToM packet from the following formula and set up the core MTU accordingly:

```
Max AToM packet = (52 * max number of packed cells) + AToM Header + (depth of
MPLS label stack * MPLS Header size)
```

In the formula, the AToM Header refers to the 4-byte control word.

Configuring WAN Protocols over MPLS Case Studies

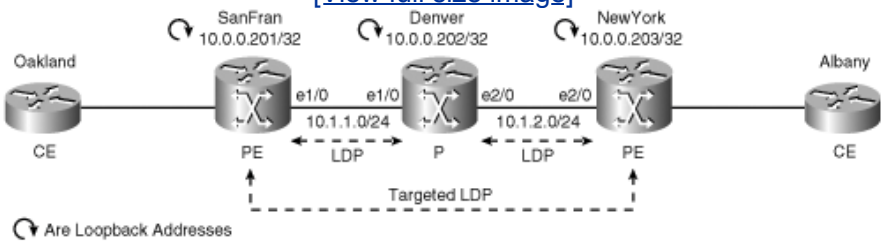
This section covers the configuration required to enable Layer 2 transport using AToM for different WAN protocols. Using a case study approach, this section covers configuration for HDLCoMPLS, PPPoMPLS, FRoMPLS, and ATMoMPLS, concentrating on generic configurations rather than platform specifics.

After the configuration, each case study also covers verification and some troubleshooting for each WAN protocol over MPLS.

All case studies use the same underlying MPLS PSN, shown in Figure 8-10. The objective of all the case studies is to establish Layer 2 transport connectivity between the two customer sites, with host names Oakland and Albany.

Figure 8-10. ***WAN Protocols over MPLS Case Study Topology***

[View full size image]



All case studies require common configuration and verification of the MPLS core. The following list details the required pre-AToM configuration steps on all P and PE routers:

1. Create a loopback interface and assign a /32 IP address to it.
2. Enable IP Cisco Express Forwarding (CEF) globally.
3. Enable MPLS globally and select LDP as the label distribution protocol. Specify the loopback interface's IP address as the LDP Router ID.
4. Assign IP addresses to all physical links connecting the core routers, and enable Link LDP on them.
5. Enable an Interior Gateway Protocol (IGP) among the core routers and include the loopback and the interfaces connecting P and PE routers. These case studies use Open Shortest Path First (OSPF) with a single area 0.

The configuration for the SanFran router is shown in Example 8-1. The configuration for the other two core routers is analogous to this one.

Example 8-1. *Required Preconfiguration*

```
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname SanFran
!
ip cef
mpls ip
mpls label protocol ldp
mpls ldp explicit-null
mpls ldp router-id Loopback0 force
!
interface Loopback0
 ip address 10.0.0.201 255.255.255.255
!
interface Ethernet1/0
 ip address 10.1.1.201 255.255.255.0
 no ip directed-broadcast
 mpls ip
!
router ospf 1
 log-adjacency-changes
 network 10.0.0.0 0.255.255.255 area 0
```

In Example 8-1, you can see **service timestamps** configured for logging and debug with the **msec** option. This is done so that you have more time granularity in the debug and error message output to better understand the protocols and ease troubleshooting.

Example 8-1 also shows **mpls ldp explicit-null** configured to advertise an IPv4 explicit null label (a label with a value of 0) instead of the default implicit null (Pop label operation).

Now you can verify that the core configuration is working as expected. Use the command **show mpls ldp neighbors** in the Denver P router to confirm that the two link LDP sessions are UP. Implicitly, you are validating two other things:

- First, that the LDP neighbors have discovered themselves. LDP discovery is performed by sending LDP Hellos over UDP to the all-routers multicast address (224.0.0.2). It is a prerequisite to LDP session establishment. You can check the LDP discovery status using the command **show mpls ldp discovery**.

- Second, that IP routes for the loopback addresses are being propagated. After LDP session discovery, you establish the LDP session by setting up a TCP session between the addresses that are advertised in the IPv4 Transport address TLV in the Hello messagein this case, the loopback IP addresses. The higher LDP session ID (active) sets up a TCP connection to the lower LDP session ID (passive) at the well-known LDP port of 646. You can also check the IP routing information using the command **show ip route**, and verify the LDP transport address using the command **show mpls ldp discovery detail**.

Example 8-2 shows the Link LDP sessions highlighting that the state is "operational."

Example 8-2. *Verifying the Core LDP Session*

```
Denver#show mpls ldp neighbor
    Peer LDP Ident: 10.0.0.203:0; Local LDP Ident 10.0.0.202:0
!This is NewYork PE
        TCP connection: 10.0.0.203.11022 - 10.0.0.202.646
```

```
        State: Oper; Msgs sent/rcvd: 47/48; Downstream
        Up time: 00:34:06
        LDP discovery sources:
          Ethernet2/0, Src IP addr: 10.1.2.203
        Addresses bound to peer LDP Ident:
          10.0.0.203      10.1.2.203
    Peer LDP Ident: 10.0.0.201:0; Local LDP Ident 10.0.0.202:0
!This is SanFran PE
        TCP connection: 10.0.0.201.646 - 10.0.0.202.11006
        State: Oper; Msgs sent/rcvd: 46/46; Downstream
        Up time: 00:33:57
        LDP discovery sources:
          Ethernet1/0, Src IP addr: 10.1.1.201
        Addresses bound to peer LDP Ident:
          10.0.0.201      10.1.1.201
Denver#
```

You can also verify the MPLS forwarding state in a PE and a P router (see Example 8-3).

Example 8-3. *Verifying the MPLS Forwarding State*

```
SanFran#show mpls forwarding-table
Local  Outgoing    Prefix           Bytes tag Outgoing   Next Hop
tag    tag or VC   or Tunnel Id     switched  interface
16     0           10.1.2.0/24      0         Et1/0      10.1.1.202
17     0           10.0.0.202/32    0         Et1/0      10.1.1.202
18     16          10.0.0.203/32    0         Et1/0      10.1.1.202
SanFran#

Denver#show mpls forwarding-table
Local  Outgoing    Prefix           Bytes tag Outgoing   Next Hop
tag    tag or VC   or Tunnel Id     switched  interface
16     0           10.0.0.203/32    1580313   Et2/0      10.1.2.203
17     0           10.0.0.201/32    1614352   Et1/0      10.1.1.201
Denver#
```

At this point, you are ready for the specific Layer 2 WAN protocols transport configuration. The upcoming sections detail the configuration and verification in the following case studies:

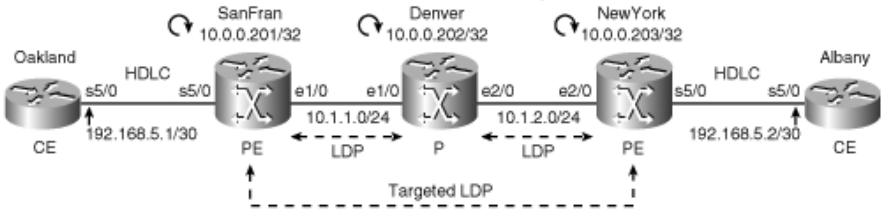
- Case Study 8-1: HDLC over MPLS
- Case Study 8-2: PPP over MPLS
- Case Study 8-3: Frame Relay DLCI over MPLS
- Case Study 8-4: ATM AAL5 SDU over MPLS
- Case Study 8-5: ATM Cell over MPLS

Case Study 8-1: HDLC over MPLS

This section describes the configuration and verification of HDLCoMPLS for transport of HDLC-like frames, including HDLC and Cisco HDLC. See Figure 8-11 for the case study topology.

Figure 8-11. HDLCoMPLS Case Study Topology

[View full size image]



In Figure 8-11, you can see that building from the generic topology in Figure 8-10, you will be using interfaces Serial 5/0 in both PE routers (SanFran and NewYork) and in both CE routers (Oakland and Albany).

Configuring HDLCoMPLS

You know from previous chapters that the AToM states and all Layer 2 transport-specific configuration exist only in the edge routers. This adds to the scalability of the whole AToM solution. Start by configuring HDLCoMPLS on the Serial interfaces in the PE routers SanFran and NewYork, as shown in Example 8-4.

Example 8-4. *HDLCoMPLS PE Configuration*

```
SanFran#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SanFran(config)#interface Serial5/0
SanFran(config-if)#no shutdown
SanFran(config-if)# xconnect 10.0.0.203 50 encapsulation mpls
SanFran(config-if)#
*May 19 01:44:32.328: %LINK-3-UPDOWN: Interface Serial5/0, changed state to up
*May 19 01:44:33.360: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial5/0,
  changed state to up
SanFran(config-if)#end
SanFran#

NewYork#conf t
Enter configuration commands, one per line. End with CNTL/Z.
NewYork(config)#pseudowire-class atom_hdlc
NewYork(config-pw-class)# sequencing transmit
NewYork(config-pw-class)# encapsulation mpls
NewYork(config-pw-class)#interface Serial5/0
NewYork(config-if)#no shutdown
NewYork(config-if)# xconnect 10.0.0.201 50 pw-class hdlc
NewYork(config-if)#end
NewYork#
```

In Example 8-4, the configurations that are applied to both PE routers are slightly different, although both achieve the same result. NewYork uses the **pseudowire-class** configuration, which is more versatile than the one liner **xconnect** configuration and allows for different characteristics to be applied to a pseudowire in a class fashion. This way, you can reuse the pseudowire class across multiple pseudowires.

Also note that the encapsulation for the Serial interfaces is not configured. This is because the default of Cisco HDLC is used.

When you configure the **xconnect** command, Cisco Discovery Protocol (CDP) is automatically disabled on the interface so that CDP packets are not sent to the CE router. The VC ID that is used in the **xconnect** command (50 in this example) needs to match in both ends.

You will analyze the **sequencing transmit** configuration under the pseudowire class in the subsequent "Troubleshooting HDLCoMPLS" subsection.

The CE configuration for the Oakland side is included in Example 8-5 for completeness. The far CE is a mirror of this configuration except for the IP address.

Example 8-5. *HDLCoMPLS CE Configuration*

```
Oakland#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Oakland(config)#interface Serial5/0
Oakland(config-if)# ip address 192.168.5.1 255.255.255.252
Oakland(config-if)#no shutdown
Oakland(config-if)#end
Oakland#
```

At this point, all specific configuration and states that pertain to the transport of Layer 2 frames over MPLS reside in the PE devices. You have not configured the P router Denver.

Verifying HDLCoMPLS

When you configure AToM, the targeted LDP session between PE routers is established, as shown in Example 8-6. Example 8-6 highlights the local and peer LDP identifiers and the targeted discovery source.

Example 8-6. *Verifying the Targeted LDP Session*

```
SanFran#show mpls ldp neighbor 10.0.0.203
    Peer LDP Ident: 10.0.0.203:0; Local LDP Ident 10.0.0.201:0
        TCP connection: 10.0.0.203.11007 - 10.0.0.201.646
        State: Oper; Msgs sent/rcvd: 2215/2209; Downstream
        Up time: 1d08h
        LDP discovery sources:
          Targeted Hello 10.0.0.201 -> 10.0.0.203, active, passive
        Addresses bound to peer LDP Ident:
          10.0.0.203      10.1.2.203
SanFran#
```

Local labels are assigned and distributed using the targeted LDP session. See Example 8-7, which highlights the Layer 2 circuit (l2ckt) local and remote labels.

Example 8-7. *AToM Label Distribution*

```
SanFran#show  mpls forwarding-table
Local  Outgoing     Prefix           Bytes tag  Outgoing  Next Hop
tag    tag or VC    or Tunnel Id     switched   interface
16     0            10.1.2.0/24      0          Et1/0     10.1.1.202
17     0            10.0.0.202/32    0          Et1/0     10.1.1.202
18     16           10.0.0.203/32    0          Et1/0     10.1.1.202
19     Untagged     l2ckt(50)        149313     Se5/0     point2point
SanFran#show mpls l2transport  binding 50
  Destination Address: 10.0.0.203,   VC ID: 50
    Local Label:  19
        Cbit: 1,    VC Type: HDLC,     GroupID: 0
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 19
        Cbit: 1,    VC Type: HDLC,    GroupID: 0
        MTU: 1500,   Interface Desc: n/a            !-+ Signaled as
        VCCV Capabilities: Type 1, Type 2           !-+ Interface Parameter
SanFran#
```

From the output of the command **show mpls forwarding-table**, you can see that local label 19 was assigned for the Layer 2 circuit (l2ckt). From the output of the command **show mpls l2transport binding**, you can see that the remote VC label is also 19. The fact that label 19 is being used on both sides is irrelevant. The local and remote labels are assigned independently and do not need to match. This last command also shows that the VC Type for HDLC is 0x0006 (from Table 8-1), the control word bit is set, the local and remote MTU are 1500, there is no interface description, and virtual circuit connectivity verification (VCCV) types supported are type 1 and type 2. These VCCV capabilities are as follows:

- **Type 1** PWE3 control word (0001b as the first nibble in the control word)
- **Type 2** MPLS Router Alert label (Label == 1)

The command **show mpls l2transport binding** details all information advertised in an LDP Label Mapping message.

Note

Both local and remote attachment circuits' MTUs need to match for the circuit to come up. The MTU is advertised in the MTU interface parameter in the pseudowire ID FEC element through the LDP Label Mapping message.

The interface description is also advertised in an interface parameter. If you add an interface description, it is not automatically sent to the remote peer, because it is included in a label mapping. For the description to appear in the remote PE, you must force the resending of the label mapping, for example, by flapping the interface by means of a **shutdown** followed by a **no shutdown**.

Another useful command is **show mpls l2transport vc 50 detail**, displayed in Example 8-8. The highlighted parts show the VC status and the label stack used in forwarding at imposition.

Example 8-8. *Displaying AToM VC Details*

```
SanFran#show mpls l2transport vc 50 detail
Local interface: Se5/0 up, line protocol up, HDLC up
  Destination address: 10.0.0.203, VC ID: 50, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop 10.1.1.202
    Output interface: Et1/0, imposed label stack {16 19}
  Create time: 1d08h, last status change time: 00:37:20
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 19, remote 19
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  Sequence number: receive 0, send 0
  VC statistics:
    packet totals: receive 1363, send 1402
    byte totals:   receive 237913, send 246646
    packet drops:  receive 0, seq error 0, send 231

SanFran#
```

Among other things, the VC status is displayed in the command output along with the imposed label stack. In this case, 16 is the IGP label and 19 is the VC label. The VC can be in one of three different statuses:

- **UP** VC can carry data between the two endpoints. (Imposition and disposition are programmed.) Two conditions need to hold true:

  Disposition interfaces are programmed The VC is configured, and the CE interface is up.

  Imposition interfaces are programmed The disposition interface is programmed, and you received a remote VC label and an IGP label (LSP to the peer).

- **DOWN** VC is not ready to carry traffic between the two VC endpoints.

- **ADMINDOWN** A user has disabled the VC.

A RAW adjacency, in which the protocol is shown as "raw" because no upper-layer adjacencies exist between the PE and CE, is created out of the attachment circuit (see Example 8-9).

Example 8-9. *AToM HDLC RAW Adjacency*

```
SanFran#show adjacency  serial 5/0 detail
Protocol Interface                 Address
RAW      Serial5/0                 point2point(4)
```

```
                                        0 packets, 0 bytes
                                        Raw        never
                                        Epoch: 0
SanFran#
```

Note that in the case of HDLCoMPLS, the adjacency contains a null encapsulation prepended to the packet that is switched through this adjacency, because the complete HDLC header is transported unmodified. Keep this in mind on the PPPoMPLS case study for comparison.

Finally, you can verify that connectivity between CE routers indeed exists (see Example 8-10).

Example 8-10. *Testing Connectivity Between CE Routers*

```
Oakland#ping 192.168.5.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.5.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/38/60 ms
Oakland#
```

Troubleshooting HDLCoMPLS

So far, you have not configured an MTU in the network. All the PE and CE router's interfaces are using the following default MTU settings:

- MTU == 1500 by default for Serial and Ethernet
- MTU == 4470 by default for High-Speed Serial Interface (HSSI), ATM, and Packet over SONET (POS)

Try to calculate the maximum packet size that you can send between CEs with the default settings. You have the following overheads in place:

- **Transport Header** 4 bytes for HDLC (refer to Table 8-2)
- **AToM Header** 4 bytes of control word
- **MPLS Label Stack * MPLS Header Size** 8 bytes from 2 MPLS headers of 4 bytes each

Based on the preceding list, you can calculate a total of 16 bytes of overhead added to CE frames. This means that only packets up to 1484 bytes from Oakland can reach the remote CE and vice versa. You can test this theory by using an extended ping with a sweep range of sizes that allows you to vary the sizes of the echo packets being sent and verbose output (see Example 8-11).

Example 8-11. *Probing for MTU Between CEs*

```
Oakland#ping
Protocol [ip]:
Target IP address: 192.168.5.2
Repeat count [5]: 1
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Source address or interface:
Type of service [0]:
Set DF bit in IP header? [no]: y
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: v
Loose, Strict, Record, Timestamp, Verbose[V]:
Sweep range of sizes [n]: y
Sweep min size [36]: 1480
Sweep max size [18024]: 1490
Sweep interval [1]:
Type escape sequence to abort.
Sending 11, [1480..1490]-byte ICMP Echos to 192.168.5.2, timeout is 1 seconds:
Reply to request 0 (40 ms) (size 1480)
Reply to request 1 (48 ms) (size 1481)
Reply to request 2 (28 ms) (size 1482)
Reply to request 3 (64 ms) (size 1483)
Reply to request 4 (28 ms) (size 1484)
Request 5 timed out (size 1485)
Request 6 timed out (size 1486)
Request 7 timed out (size 1487)
Request 8 timed out (size 1488)
Request 9 timed out (size 1489)
Request 10 timed out (size 1490)
Success rate is 45 percent (5/11), round-trip min/avg/max = 28/41/64 ms
Oakland#
```

Only packets up to 1484 bytes make it to the other end and back. You either need to increase the core MTU to 1516 bytes using the **mtu** command or reduce the CE interface's MTU to 1484. Make sure you understand the implications to IGP protocols before you change the MTU value.

Finally, capture HDLCoMPLS AToM packets in the SanFran router, from an ICMP Echo (PING) from the Oakland CE. The imposition refers to AToM packets that are sent toward the NewYork PE out of the Ethernet interface, and disposition means AToM packets that are received from Denver and sent to the Oakland CE. Use the command **debug mpls l2transport packet data**, as shown in Example 8-12.

Example 8-12. *Capturing and Decoding HDLCoMPLS Packets*

```
SanFran#debug mpls l2transport packet data
AToM packet data debugging is on
SanFran#
*May 19 02:51:21.095: ATOM imposition: out Et1/0, size 130, EXP 0x0, seq 0,
  control word 0x0
*May 19 02:51:21.095: XX XX XX XX XX XX YY YY YY YY YY YY 88 47 00 01
                      ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
                      SA MAC           DA MAC            |     top_shim-->
                                                         etype = MPLS Unicast
```

```
*May 19 02:51:21.095: 00 FF 00 01 31 02 00 00 00 00 0F 00 08 00 45 00
                      ^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^ ^^ ^^ ^^^^^ ^^^^^...
                <--top_shim VC_Label    Ctrl-word   |  |  |     Begins IP Packet
                   Label=16 Label=19                |  |  etype = IPv4
                   S=0      S=1                     |  Control
                   TTL=255  TTL=2                   Address = Unicast Frame

*May 19 02:51:21.095: 00 64 00 32 00 00 FF 01 30 13 C0 A8 05 01 C0 A8
*May 19 02:51:21.095: 05 02 08 00 03 A7 00 06 00 04 00 00 00 00 07 C1
*May 19 02:51:21.095: 72 D8 AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.095: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.095: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.095: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.095: AB CD
*May 19 02:51:21.143: ATOM disposition: in Et1/0, size 104, seq 2905, control
  word 0xB59
*May 19 02:51:21.143: 0F 00 08 00 45 00 00 64 00 32 00 00 FF 01 30 13
                      ^^ ^^ ^^^^^ ^^^^^...
                      |  |  |     Begins IP Packet
                      |  |  etype = IPv4
                      |  Control
                      Address = Unicast Frame
*May 19 02:51:21.143: C0 A8 05 02 C0 A8 05 01 00 00 0B A7 00 06 00 04
*May 19 02:51:21.143: 00 00 00 00 07 C1 72 D8 AB CD AB CD AB CD AB CD
*May 19 02:51:21.143: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.143: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.143: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 02:51:21.143: AB CD AB CD AB CD AB CD
```

Note

Note in Example 8-12 and in the following examples dealing with packet decoding that the offline hand decoding of the packets is shown in bold.

Analyzing the disposition as dumped in the SanFran PE, you can see that the control word has a non-null value, whereas the control word in imposition is null. This is because you have configured the NewYork endpoint to perform sequencing, and you can see the sequence number increasing in the control word. The sequence number in the control word is included in the rightmost 2 bytes. From Example 8-12, the control word is 0x00000B59 so the sequence number is 0x0B59. That number is 2905 in decimal.

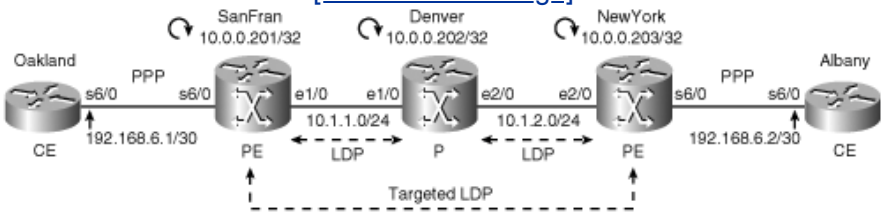
At disposition, only the AToM payload or SDU is displayed. At imposition, the complete AToM packet is dumped.

Case Study 8-2: PPP over MPLS

The case study for PPPoMPLS is quite similar to HDLCoMPLS. In this case study, you concentrate on the differences with Case Study 8-1. For the PPPoMPLS case study, you use interfaces Serial 6/0 in all PEs and CEs (see Figure 8-12).

Figure 8-12. PPPoMPLS Case Study Topology

[View full size image]



Configuring PPPoMPLS

The configuration for PPPoMPLS is analogous to HDLCoMPLS, except that the PPP encapsulation needs to be specified in the Serial interface. Example 8-13 shows the configuration for the two PE devices.

Example 8-13. *Configuring PPPoMPLS*

```
SanFran#show running-config interface serial 6/0
Building configuration...

Current configuration : 188 bytes
!
interface Serial6/0
 description *** To Oakland Serial 6/0 ***
 no ip address
 encapsulation ppp
 no cdp enable
 xconnect 10.0.0.203 60 encapsulation mpls
end

SanFran#

NewYork#show running-config interface serial 6/0
Building configuration...

Current configuration : 187 bytes
!
interface Serial6/0
 description *** To Albany Serial 6/0 ***
 no ip address
 encapsulation ppp
 no cdp enable
 xconnect 10.0.0.201 60 encapsulation mpls
end

NewYork#
```

It is important to note, however, that after you configure the **xconnect** command under the PE interface, PPP is closed. That is, no LCP or NCP takes place between PE and CE, and no PPP state machine runs in the PE device. You can enable **debug ppp negotiation** in the PE to prove this concept (see Example 8-14).

Example 8-14. *Debug PPP Negotiation in the PE Router*

```
SanFran(config-if)# xconnect 10.0.0.203 60 encapsulation mpls
*May 18 17:16:35.583: Se6/0 LCP: State is Closed
*May 18 17:16:35.583: Se6/0 PPP: Phase is DOWN
*May 18 17:16:35.583: Se6/0 PPP: Phase is ESTABLISHING, Passive Open
*May 18 17:16:35.583: Se6/0 LCP: State is Listen
*May 18 17:16:35.583: Se6/0 LCP: State is Closed
*May 18 17:16:35.583: Se6/0 PPP: Phase is DOWN
*May 18 17:16:36.631: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial6/0,
  changed state to up

SanFran(config-if)#do show interface serial 6/0
Serial6/0 is up, line protocol is up
  Hardware is HD64570
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation PPP, loopback not set
  Keepalive set (10 sec)
  Last input 00:03:10, output 00:00:06, output hang never
```

The output of the command **show interface** for the Serial 6/0 shows the encapsulation as PPP, but LCP and NCP are not indicated. For comparison, the same command shows LCP and NCPs state in the CE device (see Example 8-15).

Example 8-15. *PPP State in the PE Devices*

```
Oakland#show interfaces serial 6/0
Serial6/0 is up, line protocol is up
  Hardware is HD64570
  Internet address is 10.10.6.200/24
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation PPP, loopback not set
  Keepalive set (10 sec)
  LCP Open
  Open: OSICP, IPCP, CDPCP
  Last input 00:00:02, output 00:00:02, output hang never
  Last clearing of "show interface" counters never
```

The PPP negotiation happens between CE devices, and the core network acts as a transport.

Verifying and Troubleshooting PPPoMPLS

The first verification that CE-CE communication exists was shown in Example 8-15. From the CE device, you can check the status of PPP negotiation using the command **show interface**.

From the PE device, you can verify that the VC is up (see Example 8-16).

Example 8-16. *Verifying the PPPoMPLS VC Status*

```
SanFran#show mpls l2transport vc 60
```

```
Local intf     Local circuit            Dest address    VC ID      Status
-------------  ------------------------ --------------- ---------- ----------
Se6/0          PPP                      10.0.0.203      60         UP
SanFran#show mpls l2transport vc 60 detail
Local interface: Se6/0 up, line protocol up, PPP up
  Destination address: 10.0.0.203, VC ID: 60, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop 10.1.1.202
    Output interface: Et1/0, imposed label stack {16 20}
  Create time: 12:45:20, last status change time: 12:44:36
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 20, remote 20
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description: *** To Albany Serial 6/0 ***
  Sequencing: receive disabled, send disabled
  Sequence number: receive 0, send 0
  VC statistics:
    packet totals: receive 18291, send 18289
    byte totals:   receive 3577950, send 3403595
    packet drops:  receive 0, seq error 0, send 0

SanFran#
```

From Example 8-16, you can see that the VC is up and the interface parameters of MTU and interface description have been advertised. The VC type is PPP, which has a value of 0x0007. You might wonder, however, how you can see the value of the VC type in real time when it is advertised. The answer is by using the debug command **debug mpls l2transport signaling message**. After you enable the debug in NewYork, you must bounce the remote interface to force withdrawing and remapping of the label (see Example 8-17).

Example 8-17. *Debugging AToM Signaling Messages*

```
NewYork#debug mpls l2transport signaling message
AToM LDP message debugging is on
NewYork#

SanFran(config)#int s 6/0
SanFran(config-if)#shut

*May 19 16:19:44.995: AToM LDP [10.0.0.201]: Received label withdraw msg, id 1822,
graceful restart instance 2
vc type 7, cbit 1, vc id 60, group id 0, vc label 20, status 0, mtu 0
*May 19 16:19:45.203: AToM LDP [10.0.0.201]: Sending label release msg
vc type 7, cbit 1, vc id 60, group id 0, vc label 20, status 0, mtu 0
SanFran(config-if)#no shutdown

NewYork#
*May 19 16:20:40.071: AToM LDP [10.0.0.201]: Received label mapping msg, id 1825,
  graceful restart instance 2
vc type 7, cbit 1, vc id 60, group id 0, vc label 20, status 0, mtu 1500
```

From Example 8-17, you can see that when you shut down the interface in SanFran, an LDP label withdraw message is sent, which is acknowledged with an LDP label release message. When you enable the interface, an LDP label mapping message is sent, including the VC type of 7. More details on this procedure are covered in "Case Study 8-6: Decoding LDP Label Mapping and Pseudowire ID FEC Elements."

It is also useful to capture PPPoMPLS AToM packets to fully understand the encapsulation. You can do this with the command **debug mpls l2transport packet data**, as shown in Example 8-18.

Example 8-18. *Capturing and Decoding of PPPoMPLS Packets*

```
SanFran#debug mpls l2transport packet data
*May 19 17:33:26.916: ATOM imposition: out Et1/0, size 128, EXP 0x0, seq 0,
  control word 0x0
*May 19 17:33:26.916: XX XX XX XX XX XX YY YY YY YY YY YY 88 47 00 01
                      ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
                      SA MAC           DA MAC           |     top_shim-->
                                                        etype = MPLS Unicast
*May 19 17:33:26.916: 00 FF 00 01 41 02 00 00 00 00 00 21 45 00 00 64
                      ^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^ ^^^^^...
                <--top_shim VC_Label    Ctrl-word   |     Begins IP Packet
                   Label=16 Label=20                PPP DLL Protocol# = IPv4
                   S=0      S=1
                   TTL=255  TTL=2
*May 19 17:33:26.916: 00 FA 00 00 FF 01 2D 4B C0 A8 06 01 C0 A8 06 02
*May 19 17:33:26.916: 08 00 6B C8 00 0F 00 02 00 00 00 00 0A E9 07 88
*May 19 17:33:26.916: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.916: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.916: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.916: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.932: ATOM disposition: in Et1/0, size 102, seq 0, control
  word 0x0
*May 19 17:33:26.932: 00 21 45 00 00 64 00 FA 00 00 FF 01 2D 4B C0 A8
                      ^^^^^ ^^^^^...
                      |     Begins IP Packet
                      PPP DLL Protocol # = IPv4
*May 19 17:33:26.932: 06 02 C0 A8 06 01 00 00 73 C8 00 0F 00 02 00 00
*May 19 17:33:26.932: 00 00 0A E9 07 88 AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.932: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.932: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.932: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 17:33:26.932: AB CD AB CD AB CD
```

As noted before, the output of the **debug** command displays the complete packet for imposition operations. It displays only the AToM payload for disposition actions.

Example 8-19 shows the new RAW adjacency that is created for PPPoMPLS.

Example 8-19. *PPPoMPLS RAW Adjacency*

```
SanFran#show adjacency serial 6/0 detail
Proocol  Interface                Address
RAW      Serial6/0                point2point(4)
                                  0 packets, 0 bytes
```

```
                                    FF03
                                    Raw         never
                                    Epoch: 0
SanFran#
```

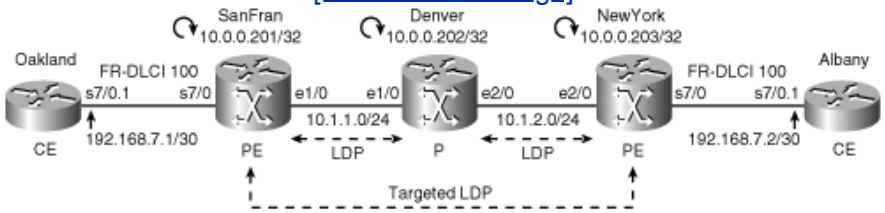
From Example 8-19, you can see that the rewrite that was null for HDLCoMPLS has become 0xFF03. These two bytes are no more than the two bytes (Address and Control) from each PPP packet that are stripped in imposition and regenerated at disposition. They make the encapsulation that is prepended to the packet switched through this adjacency.

Case Study 8-3: Frame Relay DLCI over MPLS

This case study concentrates on Frame Relay DLCI mode over MPLS. This case study is fundamentally different from the two previous case studies because control messaging interaction occurs between PE and CE routers. This case study uses Frame Relay IETF encapsulation. The topology is included in Figure 8-13.

Figure 8-13. Frame Relay DLCI over MPLS Case Study Topology

[View full size image]



In Frame Relay DLCI mode, PE and CE routers run Frame Relay LMI between them. If you instead tunnel and transport Frame Relay in port mode using HDLCoMPLS, the LMI session runs between CE devices. If those CE devices are Frame Relay switches, configure them to run LMI NNI. If the CEs are routers, configure one end as LMI DCE and leave the other as the default LMI DTE. Alternatively, configure both routers as LMI NNI so that the CE can provide status information about its DLCIs to the PE.

Configuring Frame Relay DLCI over MPLS

The general PE configuration for a PE that is running Frame Relay DLCI over MPLS (also known as Frame Relay DLCI-Mode AToM) is shown in Example 8-20. The configuration in the NewYork PE is parallel to this one.

Example 8-20. *FRoMPLS PE Configuration*

```
SanFran#conf t
Enter configuration commands, one per line. End with CNTL/Z.
SanFran(config)#frame-relay switching
SanFran(config)#interface serial7/0
SanFran(config-if)#encapsulation frame-relay ietf
```

```
SanFran(config-if)#frame-relay intf-type dce
SanFran(config-if)#no shutdown
SanFran(config-if)#exit
SanFran(config)#
SanFran(config)#connect frompls serial7/0 100 l2transport
SanFran(config-fr-pw-switching)#xconnect 10.0.0.203 70 encapsulation mpls
SanFran(config-fr-pw-switching)#end
SanFran#
```

In Example 8-20, the global command f**rame-relay switching** is enabled. This is required so that Frame Relay LMI types DCE or NNI can be enabled later.

Next, create a "switched" Frame Relay PVC by using the global command **connect extended** with the **l2transport** keyword. You apply the **xconnect** command under the connect configuration mode (fr-pw-switching).

> **Note**
>
> You can configure the MTU on a switched Frame Relay PVC (pseudowire endpoint) basis by using the command **mtu** under the connect configuration mode.

Example 8-21 depicts a typical CE configuration from the Oakland CE, including Frame Relay quality of service (QoS) parameters.

Example 8-21. *FRoMPLS CE Configuration*

```
interface Serial7/0
no ip address
 encapsulation frame-relay IETF
!
interface Serial7/0.1 point-to-point
 ip address 192.168.7.1 255.255.255.252
 frame-relay interface-dlci 100 IETF
  class myfrpvc
map-class frame-relay myfrpvc
 frame-relay cir 64000
 frame-relay bc 8000
 frame-relay be 0
 frame-relay mincir 32000
```

A Frame Relay **map-class** is defined, including all the desired Frame Relay parameter values. Then this **map-class** is applied to the Frame Relay PVC using the **class** command.

Verifying and Troubleshooting Frame Relay DLCI over MPLS

For FRoMPLS DLCI mode, you can use most verification techniques that were outlined for the previous two case studies. Checking IP layer connectivity between CE devices is the real verification.

The output of the **debug mpls l2transport** signaling message command included in Example 8-22 shows that for Frame Relay DLCI mode, the VC type is 0x0001.

Example 8-22. *VC Type for FRoMPLS*

```
SanFran#debug mpls l2transport signaling message
*May 19 18:10:25.119: AToM LDP [10.0.0.201]: Sending label mapping msg
vc type 1, cbit 1, vc id 70, group id 0, vc label 21, status 0, mtu 1500
```

You can use a few commands on the PE router to verify the FRoMPLS pseudowire status. The most often used commands are included in the upcoming examples. Example 8-23 shows the **show connection** command output.

Example 8-23. *Verifying the Status of the FRoMPLS Connection*

```
SanFran#show connection all

ID    Name              Segment 1             Segment 2               State
================================================================================
4     frompls           Se7/0 100             10.0.0.203 70           UP

SanFran#show connection id 4

FR/Pseudo-Wire Connection: 4 - frompls
  Status     - UP
  Segment 1 - Serial7/0 DLCI 100
    Segment status: UP
    Line status: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
  Segment 2 - 10.0.0.203 70
    Segment status: UP
    Requested AC state: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
SanFran#
```

In Example 8-23, you can observe that a switched connection has two segments:

- **Segment 1** The local attachment circuit out of Serial 7/0 DLCI 100
- **Segment 2** The remote endpoint of the pseudowire in the NewYork PE

Example 8-24 shows the output of the **show mpls l2transport vc** command, which you have seen in the previous case studies for other WAN protocols.

Example 8-24. *Verifying the Status of the FRoMPLS VC*

```
SanFran#show mpls l2transport vc | i Local|---|70
Local intf     Local circuit           Dest address    VC ID      Status
```

```
-------------  ------------------------ --------------- ---------- ----------
Se7/0          FR DLCI 100              10.0.0.203      70         UP
SanFran#show mpls l2transport vc 70 detail
Local interface: Se7/0 up, line protocol up, FR DLCI 100 up
  Destination address: 10.0.0.203, VC ID: 70, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop 10.1.1.202
    Output interface: Et1/0, imposed label stack {16 21}
  Create time: 00:47:09, last status change time: 00:47:08
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 22, remote 21
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  Sequence number: receive 0, send 0
  VC statistics:
    packet totals: receive 317, send 346
    byte totals: receive 110374, send 119708
    packet drops: receive 0, seq error 0, send 0

SanFran#
```

This output is similar to other Layer 2 protocols that are transported over MPLS, but the VC Type is displayed as FR DLCI *DLCI*.

Example 8-25 shows the command **show frame-relay pvc** in PE and CE routers for comparison. See the difference highlighted in the DLCI Usage field (Local for the CE and Switched for the PE) and the additional counters on the PE side. This command also shows the PVC status.

Example 8-25. *Verifying the Status of the FRoMPLS Frame Relay PVC*

```
SanFran#show frame-relay pvc interface serial 7/0 100

PVC Statistics for interface Serial7/0 (Frame Relay DCE)

DLCI = 100, DLCI USAGE = SWITCHED, PVC STATUS = ACTIVE, INTERFACE = Serial7/0

  input pkts 358           output pkts 329          in bytes 121796
  out bytes 112624         dropped pkts 0           in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0          out BECN pkts 0
  in DE pkts 0             out DE pkts 0
  out bcast pkts 0         out bcast bytes 0
  switched pkts 358
  Detailed packet drop counters:
  no out intf 0            out intf down 0          no out PVC 0
  in PVC down 0            out PVC down 0           pkt too big 0
  pvc create time 00:48:13, last time pvc status changed 00:47:53
SanFran#
Oakland#
Oakland#show frame-relay pvc interface serial 7/0 100

PVC Statistics for interface Serial7/0 (Frame Relay DTE)

DLCI = 100, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial7/0.1
```

```
  input pkts 329           output pkts 358          in bytes 112624
  out bytes 121796         dropped pkts 0           in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0          out BECN pkts 0
  in DE pkts 0             out DE pkts 0
  out bcast pkts 348       out bcast bytes 120756
  pvc create time 2d04h, last time pvc status changed 00:47:56
Oakland#
```

The output of the command **show frame-relay pvc** in the SanFran PE router shows one line of DLCI Usage and PVC Status. This is because SanFran's Serial interface is configured as Frame Relay DCE. If a Frame Relay interface is configured for Frame Relay NNI LMI, two separate fields are displayed:

- **LOCAL PVC STATUS** Status of the PVC that is locally configured
- **NNI PVC STATUS** Status of the PVC as learned from the LMI peer

Table 8-3 lists the different values that the Usage and Status fields on the **show frame-relay pvc** command output can take and what they mean.

Table 8-3. Meaning of Frame Relay PVC Usage and Status Fields

Field	Value	Definition
USAGE	LOCAL	If DLCI is configured on the router as a DTE device.
	SWITCHED	If DLCI is configured and the router is acting as a switch.
	UNUSED	If DLCI is not configured on the router but the switch is reporting it.
STATUS	STATIC	If keepalives (Frame Relay LMI) are disabled.
	DELETED	If DLCI is defined on the router (Frame Relay DTE) but not the switch (Frame Relay DCE).
	INACTIVE	If DLCI is defined on the switch (Frame Relay DCE) but the PVC is not up (that is, network, AToM, VC failure).

Field	Value	Definition
	ACTIVE	If DLCI is defined on the switch (Frame Relay DCE) and is enabled.

Specific Frame Relay LMI details such as the LMI type, LMI side, and statistics are available with the **show frame-relay lmi** and **debug frame-relay lmi** commands.

One specific command for troubleshooting Frame Relay pseudowires is **debug frame-relay pseudowire** (see Example 8-26).

Example 8-26. *Debugging Frame Relay Pseudowires*

```
NewYork(config-if)#do debug frame-relay pseudowire
Frame Relay pseudowire events debugging is on
NewYork(config-if)#no shut
NewYork(config-if)#
*Apr 27 14:16:51.247: %LINK-3-UPDOWN: Interface Serial7/0, changed state to up
*Apr 27 14:16:51.247: FRoPW [10.0.0.201, 70]: Local up, sending acmgr_circuit_up
*Apr 27 14:16:51.247: FRoPW [10.0.0.201, 70]: Setting pw segment UP
*Apr 27 14:16:51.263: Se7/0 ACMGR: Receive <Circuit Up> msg
*Apr 27 14:16:51.263: Se7/0 ACMGR: circuit up event, SIP state chg fsp up to
  connected, action is p2p up forwarded
*Apr 27 14:16:51.263: FRoPW [10.0.0.201, 70]: PW nni_pvc_status set ACTIVE
*Apr 27 14:16:51.607: Se7/0 ACMGR: Rcv SIP msg: resp peer-to-peer msg, hdl
  78000004, sss_hdl B000006
*Apr 27 14:16:51.607: Se7/0 ACMGR: remote up event, SIP connected state no chg,
  action is ignore
*Apr 27 14:16:52.267: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial7/0,
  changed state to up
*Apr 27 14:17:01.271: FRoPW [10.0.0.201, 70]: SW AC update circuit state to up
*Apr 27 14:17:01.271: ACLIB: Update switching plane with circuit UP status
```

The highlighted lines show the different state transition changes starting from the attachment circuit being set to up, the connect segment being set to up, the Frame Relay NNI PVC being set to active, and the circuit state being set to up. Example 8-27 shows a capture and decode of an FRoMPLS packet. The packet dump was generated in the same way as the other case studies by using the command **debug mpls l2transport packet data**. Refer to Figure 8-5 for comparison of the Frame Relay packets.

Example 8-27. *Capturing and Decoding of FRoMPLS DLCI Packets*

```
SanFran#debug mpls l2transport packet data
*May 19 19:14:41.080: ATOM imposition: out Et1/0, size 128, EXP 0x0, seq 0,
  control word 0x0
*May 19 19:14:41.080: XX XX XX XX XX XX YY YY YY YY YY YY 88 47 00 01
                      ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
                      SA MAC           DA MAC           |     top_shim-->
                                                        etype = MPLS Unicast
```

```
*May 19 19:14:41.080: 00 FF 00 01 51 02 00 00 00 00 03 CC 45 00 00 64
                      ^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^ ^^ ^^^^^...
                <--top_shim VC_Label    Ctrl-word   |  |   Begins IP Packet
                   Label=16 Label=21                |  NLPID = IP (0xCC)
                   S=0      S=1                     Control = 0x03
                   TTL=255  TTL=2
*May 19 19:14:41.080: 01 11 00 00 FF 01 2B 34 C0 A8 07 01 C0 A8 07 02
*May 19 19:14:41.080: 08 00 BC 31 00 14 00 03 00 00 00 00 0B 45 B6 BC
*May 19 19:14:41.080: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.080: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.080: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.080: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.104: ATOM disposition: in Et1/0, size 102, seq 0, control
  word 0x0
*May 19 19:14:41.104: 03 CC 45 00 00 64 01 11 00 00 FF 01 2B 34 C0 A8
                      ^^ ^^ ^^^^^...
                      |  | Begins IP Packet
                      |  NLPID = IP (0xCC)
                      Control = 0x03
*May 19 19:14:41.104: 07 02 C0 A8 07 01 00 00 C4 31 00 14 00 03 00 00
*May 19 19:14:41.104: 00 00 0B 45 B6 BC AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.104: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.104: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.104: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
*May 19 19:14:41.104: AB CD AB CD AB CD
```

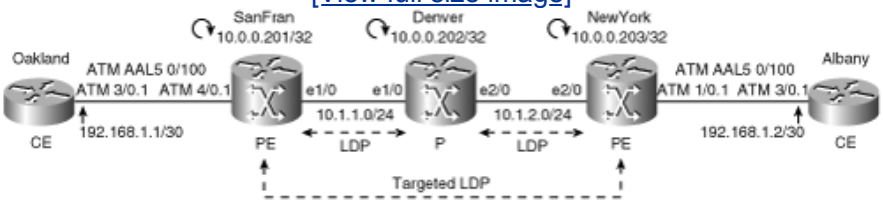
You can see that the IP NLPID of 0xCC is used. For Frame Relay IETF encapsulation, Cisco IOS uses the NLPID value when one is available; otherwise, it uses a SNAP header with NLPID 0x80.

Case Study 8-4: ATM AAL5 SDU over MPLS

The final two case studies explore the transport of ATMoMPLS. In particular, Case Study 8-4 analyzes AAL5 SDU over MPLS. The topology used is shown in Figure 8-14.

Figure 8-14. AAL5 SDU over MPLS Case Study Topology

[View full size image]



Configuring AAL5oMPLS

The configuration of AAL5oMPLS SDU mode only applies to an ATM VC. AAL5 VP or port modes are nonexistent. To configure AAL5oMPLS, you create an ATM PVC with the **l2transport** keyword and then apply the following two configuration steps under an ATM PVC configuration mode:

Step 1. Configure the encapsulation as **encapsulation aal5**.

Step 2. Apply the **xconnect** command with the same format as before.

Example 8-28 shows a sample configuration for the SanFran and NewYork PE nodes.

Example 8-28. *Configuring AAL5oMPLS PEs*

```
SanFran#show running-config interface ATM 4/0.1
Building configuration...

Current configuration : 230 bytes
!
interface ATM4/0.1 point-to-point
 description *** AAL5 SDU AToM to Oakland ***
 pvc 0/100 l2transport
  encapsulation aal5
  xconnect 10.0.0.203 100 encapsulation mpls
!
end

SanFran #

NewYork#show running-config interface ATM 1/0.1
Building configuration...

Current configuration : 229 bytes
!
interface ATM1/0.1 point-to-point
 description *** AAL5 SDU AToM to Albany ***
 pvc 0/100 l2transport
  encapsulation aal5
  xconnect 10.0.0.201 100 encapsulation mpls
 !
end

NewYork #
```

The CE configuration is no different than if the CE routers were connected to a traditional ATM switch (see Example 8-29).

Example 8-29. *Configuring AAL5oMPLS CEs*

```
Oakland#show running-config interface ATM 3/0.1
Building configuration...

Current configuration : 147 bytes
!
interface ATM3/0.1 point-to-point
 ip address 192.168.1.1 255.255.255.252
 pvc 0/100
  oam-pvc manage
 !
end
```

```
Oakland #
Albany#show running-config interface ATM 3/0.1
Building configuration...

Current configuration : 147 bytes
!
interface ATM3/0.1 point-to-point
 ip address 192.168.1.2 255.255.255.252
 pvc 0/100
  oam-pvc manage
 !
end

Albany#
```

In this case, the CEs are configured for unspecified bit rate (UBR) service-type PVCs. You can change the service type and traffic parameters either by explicit configuration under the PVC mode or by defining a VC class. Example 8-29 shows OAM management enabled under the CE PVCs, which will be transported as raw cells over the AToM pseudowire. More details about OAM cell transport over AAL5 SDU Mode pseudowires are covered in the upcoming section titled "Case Study 8-9: Understanding Different ATM Transfer Modes."

Verifying and Troubleshooting AAL5oMPLS

The VC Type for ATM AAL5 SDU VCC is 0x0002. By enabling the **debug mpls l2transport signaling message** command, you can see the targeted LDP label mapping message, including the VC Type in the pseudowire ID FEC TLV (see Example 8-30).

Example 8-30. *Checking the VC Type for AAL5oMPLS SDU Mode*

```
SanFran#debug mpls l2transport signaling message
1d21h: AToM LDP [10.0.0.201]: Received label mapping msg, id 3040
vc type 2, cbit 1, vc id 100, group id 4, vc label 20, status 0, mtu 4470
```

One of the first things you can verify is the pseudowire status and details. You can use the command **show mpls l2transport vc** (see Example 8-31).

Example 8-31. *Verifying the AAL5oMPLS SDU Mode VC*

```
SanFran#show mpls l2transport vc 100

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------  --------------  ----------  ----------
AT4/0.1        ATM AAL5 0/100          10.0.0.203      100        UP
SanFran#show mpls l2transport vc 100 detail
Local interface: AT4/0.1 up, line protocol up, ATM AAL5 0/100 up
  Destination address: 10.0.0.203, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop 10.0.1.203
```

```
    Output interface: Fa0/0, imposed label stack {16 21}
  Create time: 01:10:38, last status change time: 00:12:25
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 20, remote 21
    Group ID: local 4, remote 3
    MTU: local 4470, remote 4470
    Remote interface description: *** AAL5 SDU AToM to Albany ***
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 141, send 141
    byte totals:   receive 8460, send 8460
    packet drops:  receive 0, send 1

SanFran
```

The VC Type is displayed as ATM AAL5, and the VPI/VCI pair is included also. As usual, MTUs need to match for the PVC to come up.

You can also see the l2transport bindings, including all information advertised through the targeted LDP session for this FEC in the LDP label mapping message (see Example 8-32).

Example 8-32. *Displaying the AAL5oMPLS SDU Mode Binding*

```
SanFran#show mpls l2transport binding 100
  Destination Address: 10.0.0.203, VC ID: 100
    Local Label: 20
        Cbit: 1,    VC Type: ATM AAL5,    GroupID: 4
        MTU: 4470,   Interface Desc: *** AAL5 SDU AToM to Oakland ***
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 21
        Cbit: 1,    VC Type: ATM AAL5,    GroupID: 3
        MTU: 4470,   Interface Desc: *** AAL5 SDU AToM to Albany ***
        VCCV Capabilities: Type 1, Type 2
SanFran
```

Together with FRoMPLS, AAL5 SDU mode requires the use of the control word. You can see in Example 8-32 that the C-bit indicating control word disposition capability is set in both LDP advertisements.

Example 8-33 displays the ATM VC information from the Oakland CE and the Albany PE to compare them.

Example 8-33. *Comparing the ATM PVCs in the CE and PE Routers*

```
Oakland#show atm vc interface ATM 3/0.1 detail
ATM3/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 155000
AAL5-LLC/SNAP, etype:0x0, Flags: 0xC20, VCmode: 0x0
OAM frequency: 10 second(s)
InARP frequency: 15 minutes(s)
InPkts: 304, OutPkts: 222, InBytes: 16975, OutBytes: 15897
InPRoc: 304, OutPRoc: 222
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
```

```
Giants: 0
OAM cells received: 340
OAM cells sent: 340
Status: UP
Oakland#

SanFran#show atm vc interface ATM 4/0.1 detail
ATM4/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5 L2transport, etype:0xF, Flags: 0x10000C2E, VCmode: 0x0
OAM Cell Emulation: not configured
Interworking Method: like to like
Remote Circuit Status = No Alarm, Alarm Type = None
InPkts: 496, OutPkts: 216, InBytes: 34359772357, OutBytes: 12677
InPRoc: 0, OutPRoc: 0
InFast: 156, OutFast: 216, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 340
OAM cells sent: 29
Status: UP
SanFran#
```

By contrasting the output of the display of ATM PVC in the PE and CE routers, you can see the following:

- The encapsulation type that is displayed for the CE is AAL5-LLC/SNAP (although it could have been AAL5-MUX or something else), whereas for the PE PVC it is always AAL5 l2transport.
- The CE PVC shows OAM configuration (OAM frequency), whereas the PE PVC displays OAM Cell Emulation configuration. You can enable cell emulation by using the commands **oam-ac emulation-enable** and **oam-pvc manage** so that the PE locally terminates OAM cells (as opposed to transporting them).
- The PE PVC shows AToM-specific information such as the remote pseudowire status and the interworking type.
- The OAM cells received and sent counters have a slightly different interpretation.

On the Oakland CE router, the OAM for cells received and sent display the total number of OAM cells that are received and sent from and to the Albany CE. On the SanFran PE router, OAM counter for received cells displays the total number of OAM cells from the Oakland CE. These cells are encapsulated in an AToM packet as cells (setting the T-bit in the required AAL5oMPLS SDU mode control word) and sent to the remote PE router NewYork. However, the OAM cells sent counter in the PE router does not count the OAM cells received in AToM packets from the remote NewYork PE and sent to the Oakland CE router. It counts the OAM cells that are generated from the SanFran PE, which explains the number discrepancy (see Example 8-34).

Example 8-34. *Alarm Indication Signal (AIS) OAM Cells*

```
SanFran#
*May 19 16:51:40.207: AToM LDP [10.0.0.203]: Received label withdraw msg, id 3340
```

```
vc type 2, cbit 1, vc id 100, group id 3, vc label 21, status 0, mtu 0
*May 19 16:51:40.207: ATM VC alarm condition: remote acircuit DOWN
  forATM4/0.1:VC#1 0/100
*May 19 16:51:40.207: atm oam setstate - VCD#1, VC 0/100: newstate = AIS Xmitted
*May 19 16:51:40.207: F5 OAM alarm: AIS sent, VC#1 0/100 ATM4/0.1
*May 19 16:51:40.207: atm_oam_start_timer VC = 1, curr_q_index = 19016 q_index =
  19047, freq = 1000, cnt = 0 div = 31
*May 19 16:51:40.207: atm_oam_start_timer VC = 1, q_index = 19047, freq = 1000cnt
  = 0
*May 19 16:51:40.207: ATM VC alarm condition: remote acircuit DOWN
  forATM4/0.1:VC#1 0/100
```

From Example 8-34, you can see that when the pseudowire VC is withdrawn because of an MPLS network failure, the SanFran PE router sends OAM AIS cells to the Oakland CE router. The attachment circuit, which in this case is the ATM PVC with VPI/VCI 0/100 in the sub-interface ATM4/0.1 in SanFran, goes into a remote attachment circuit down alarm.

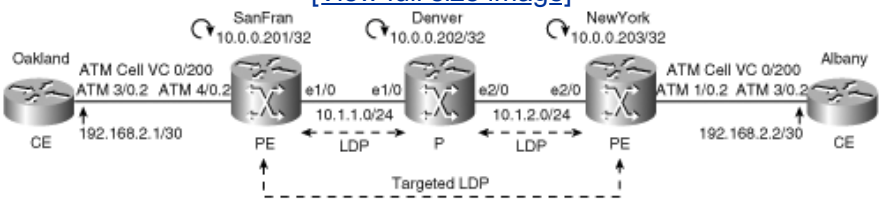
The fault management aspects of the ATM attachment circuits and the relationship between pseudowire status and AIS signals are covered in the next case study.

Case Study 8-5: ATM Cell over MPLS

The other mode of transporting ATMoMPLS is to transport raw cells in Cell Relay mode. This section presents a detailed example of CRoMPLS in VC mode. The topology used is shown in Figure 8-15.

Figure 8-15. CRoMPLS Case Study Topology

[View full size image]



Although this section focuses on CRoMPLS VC mode, it also presents some configuration and verification examples for CRoMPLS in its other two modes of operation: VP mode and port mode.

Configuring CRoMPLS

The configuration steps to set up the cell relay transport of an ATM VC are similar to those required to configure AAL5 transport. The difference is that under the l2transport PVC configuration, the encapsulation is specified as aal0, implying no adaptation layer (raw cells). Example 8-35 lists the configuration of both PE routers.

Example 8-35. *Configuring CRoMPLS PEs*

```
SanFran#show running-config interface ATM 4/0.2
Building configuration...

Current configuration : 229 bytes
!
interface ATM4/0.2 point-to-point
 description *** Cell VC AToM to Oakland ***
 pvc 0/200 l2transport
  encapsulation aal0
  xconnect 10.0.0.203 200 encapsulation mpls
 !
end

SanFran#

NewYork#show running-config interface ATM 1/0.2
Building configuration...

Current configuration : 228 bytes
!
interface ATM1/0.2 point-to-point
 description *** Cell VC AToM to Albany ***
 pvc 0/200 l2transport
  encapsulation aal0
  xconnect 10.0.0.201 200 encapsulation mpls
 !
end

NewYork#
```

Note

In earlier versions of ATM CRoMPLS implementations on the Cisco 12000 series routers with engine 2 ATM linecards, the interface command **atm cell-relay** was required as an initial configuration step. This initial step is no longer required.

You can configure the different modes of CRoMPLS by applying the **xconnect** command under the different configuration modes (see Example 8-36). The context of the **xconnect** command determines the mode:

- **VC mode** Apply the **xconnect** command under **pvc l2transport** configuration submode (cfg-if-atm-l2trans-pvc).
- **VP mode** Apply the **xconnect** command under **atm pvp l2transport** configuration submode (cfg-if-atm-l2trans-pvp).
- **Port mode** Apply the **xconnect** command under the interface configuration mode.

Example 8-36. *Configuring CRoMPLS in VC, VP, and Port Modes*

```
! Configuring Cell Relay over MPLS VC Mode
interface ATM4/0.300 point-to-point
  pvc 0/300 l2transport
    encapsulation aal0
    xconnect 10.0.0.200 300 encapsulation mpls

! Configuring Cell Relay over MPLS VP Mode
interface ATM 5/0
  atm pvp 1 l2transport
    xconnect 10.0.0.200 400 encapsulation mpls

! Configuring Cell Relay over MPLS Port Mode
interface ATM 6/0
  xconnect 10.0.0.200 500 encapsulation mpls
```

You can see from Example 8-36 that you only need the **encapsulation aal0** command in VC mode, where it is necessary to distinguish between AAL5 and cell transport.

Verifying CRoMPLS

This section describes verification of the CRoMPLS configuration. Start by verifying the circuit type of all the CRoMPLS transport modes. Following are the VC types used:

- **CRoMPLS VC Mode** 0x0009 (9)
- **CRoMPLS VP Mode** 0x000A (10)
- **CRoMPLS Port Mode** 0x0003 (3)

The **debug mpls l2transport signaling message** command output provides the circuit type information (see Example 8-37).

Example 8-37. *Verifying the VC Type for All CRoMPLS Transport Modes*

```
! Configuring a remote Attachment Circuit for CRoMPLS Port Mode
SanFran#
*May 19 17:35:07.207: AToM LDP [10.0.0.203]: Received label mapping msg, id 3395
vc type 3, cbit 1, vc id 500, group id 0, vc label 18, status 0, mtu 0

! Configuring a remote Attachment Circuit for CRoMPLS VP Mode
SanFran#
*May 19 17:35:44.511: AToM LDP [10.0.0.203]: Received label mapping msg, id 3397
vc type 10, cbit 1, vc id 400, group id 0, vc label 18, status 0, mtu 0

! Configuring a remote Attachment Circuit for CRoMPLS VC Mode
SanFran#
*May 19 17:36:28.411: AToM LDP [10.0.0.203]: Received label mapping msg, id 3400
vc type 9, cbit 1, vc id 200, group id 0, vc label 18, status 0, mtu 0
```

In Example 8-37, the MTU that is advertised appears as null. The method and reason for this will become clear in this section.

Given that the transport of cells is inherently different from the transport of packets, some differences exist in the AToM pseudowire setup. See the output of the command **show mpls l2transport vc** in Example 8-38.

Example 8-38. *Verifying the ATM Cell over MPLS VC*

```
SanFran#show mpls l2transport vc 200

Local intf     Local circuit            Dest address    VC ID      Status
-------------  ----------------------   --------------  ----------  ----------
AT4/0.2        ATM VCC CELL 0/200       10.0.0.203      200         UP
SanFran#show mpls l2transport vc 200 detail
Local interface: AT4/0.2 up, line protocol up, ATM VCC CELL 0/200 up
  Destination address: 10.0.0.203, VC ID: 200, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop 10.0.1.203
    Output interface: Fa0/0, imposed label stack {16 20}
  Create time: 1d23h, last status change time: 00:08:05
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 21, remote 20
    Group ID: local 0, remote 3
    MTU: local n/a, remote n/a
    Remote interface description: *** Cell VC AToM to Albany ***
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 0, send 1
    byte totals:   receive 0, send 60
    packet drops:  receive 0, send 0

SanFran#
```

The textual version of the VC Type for CRoMPLS in Cisco IOS Software is displayed as follows:

- **VC mode** ATM VCC CELL *VPI/VCI*
- **VP mode** ATM VPC CELL *VPI*
- **Port mode** ATM CELL *Interface_name*

Also note that the local and remote MTU values appear as not available (n/a). In all flavors of CRoMPLS, the MTU value is not advertised in the LDP label mapping, because it does not apply. This can also be noted in Example 8-39.

Example 8-39. *Verifying the ATM Cell over MPLS Binding*

```
SanFran#show mpls l2transport binding 200
  Destination Address: 10.0.0.203,  VC ID: 200
    Local Label:  19
        Cbit: 1,    VC Type: ATM VCC CELL,    GroupID: 0
        MTU: n/a,   Interface Desc: *** Cell VC AToM to Oakland ***
        Max Concatenated ATM Cells: 1
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 20
```

```
        Cbit: 1,    VC Type: ATM VCC CELL,    GroupID: 3
        MTU: n/a,   Interface Desc: *** Cell VC AToM to Albany ***
        Max Concatenated ATM Cells: 1
        VCCV Capabilities: Type 1, Type 2
SanFran#
```

From Example 8-39, you can see again that the MTU value does not apply to the transport of ATM cells over MPLS. The pseudowire comes up even if the MTUs in the two attachment circuits differ. However, a new interface parameter is advertised and displayed in the bindings. This new interface parameter is the maximum number of concatenated ATM cells, also known as the maximum number of cells packed (MNCP). This advertised parameter specifies the maximum number of packed cells that the egress PE can process in a single AToM packet disposition. The MNCP value defaults to 1, meaning that by default only one ATM cell is included in an AToM packet. This subject is covered in more detail in the upcoming section "Case Study 8-8: Packed Cell Relay over MPLS."

From a fault management perspective, the pseudowire status is conveyed to the CE device's ATM endpoints by using AIS of the appropriate hierarchy. The following alarm indications are sent out of the AC for each of the ATM transport modes if the VC label is withdrawn because of an MPLS core network or remote AC failure:

- **VC Mode** F5 (VC-level) AIS OAM cells are sent. Note that this applies to both AAL5oMPLS and CRoMPLS VC Mode.
- **VP Mode** F4 (VP-level) AIS OAM cells are sent.
- **Port Mode** Line (for example Layer 1 SONET-level) AIS is sent.

Advanced WAN AToM Case Studies

This section concentrates on advanced concepts and techniques in deployments of AToM Layer 2 transport of WAN protocols. It covers diverse in-depth topics that involve a higher degree of complexity or understanding.

This section analyzes four additional case studies. The first two case studies present additional details about LDP signaling of pseudowires and specifics about Cisco implementation, including techniques so that you can understand hardware specifics and documentation matrices. Although these two case studies use WAN transport over MPLS examples, they are applicable to all other Layer 2 transports.

Finally, this section includes two advanced cases of ATMoMPLS, namely ATM cell packing and a detailed comparison of different AToM transports for ATM VCs.

Case Study 8-6: Decoding LDP Label Mapping and Pseudowire ID FEC Elements

Chapter 6 presented theoretical aspects of the AToM control plane and details on LDP messages. You have seen examples of using LDP to provide VC label mapping throughout Chapters 7, "LAN Protocols over MPLS Case Studies," and 8, "WAN Protocols over MPLS Case Studies." This section includes the decoding of a real LDP label mapping message captured from the HDLCoMPLS signaling in Case Study 8-1. You can also capture the hexadecimal dump of LDP messages by using the Cisco IOS debug facility with the command **debug mpls ldp session io all**. Example 8-40 shows the decoding of the LDP label mapping message for a pseudowire (VC) FEC highlighting the LDP message, TLVs, and items in the VC FEC element. The decode was obtained using ethereal software.

Example 8-40. *Decoding an LDP Label Mapping Message for a VC*

```
Label Distribution Protocol
    Version: 1
    PDU Length: 77
    LSR ID: 10.0.0.201 (10.0.0.201)
    Label Space ID: 0
    Label Mapping Message
        0... .... = U bit: Unknown bit not set
        Message Type: Label Mapping Message (0x400)
        Message Length: 67
        Message ID: 0x000012d5
        Forwarding Equivalence Classes TLV
            00.. .... = TLV Unknown bits: Known TLV, do not Forward (0x00)
            TLV Type: Forwarding Equivalence Classes TLV (0x100)
            TLV Length: 51
            FEC Elements
                FEC Element 1 VCID: 50
                    FEC Element Type: Virtual Circuit FEC (128)
                    1... .... = C-bit: Control Word Present
                    .000 0000 0000 0110 = VC Type: HDLC (0x0006)
                    VC Info Length: 43
                    Group ID: 5
                    VC ID: 50
                    Interface Parameter: MTU 1500
                        ID: MTU (0x01)
                        Length: 4
                        MTU: 1500
                    Interface Parameter: Description
```

```
                    ID: Interface Description (0x03)
                    Length: 31
                    Description: *** To Oakland Serial 5/0 ***
                Interface Parameter: VCCV
                    ID: VCCV (0x0c)
                    Length: 4
                    CC Type
                        .... ...1 = PWE3 Control Word: True
                        .... ..1. = MPLS Router Alert: True
                    CV Type
                        .... ...0 = ICMP Ping: False
                        .... ..1. = LSP Ping: True
                        .... .0.. = BFD: False
        Generic Label TLV
            00.. .... = TLV Unknown bits: Known TLV, do not Forward (0x00)
            TLV Type: Generic Label TLV (0x200)
            TLV Length: 4
            Generic Label: 19
```

From Example 8-40, you can see that the LDP label mapping message is sent from SanFran (LDP ID 10.0.0.201) and contains two type, length, value (TLV) triplets to provide the FEC-to-label mapping (FEC <-> label):

- **FEC TLV** The FEC TLV includes one FEC element of Type 128 (Virtual Circuit FEC), as discussed in Chapter 6. This FEC element includes the following information:

  Control word present

  VC Type 0x0006 for HDLC

  Group ID 5

  VC ID 50

  A set of interface parameters:

  MTU interface parameter

  Interface description

  VCCV capabilities of control channel (CC) and connectivity verification (CV)

- **Generic Label TLV** This TLV advertises label 19 for the previously referenced FEC.

Note

It is interesting to note the value for the Group ID of 5. The Group ID is an arbitrary 32-bit number that represents a group of pseudowires (a second degree of freedom by creating groups in the VC ID space). These groups are per LDP peer, meaning that VC IDs with the same Group ID belong to the same group if they belong to the same peer. The Group ID provides a superficial incremental benefit when sending one LDP label withdrawal message to a given peer for the group of VCs instead of multiple individual withdrawals for each VC. Earlier releases of Cisco IOS Software used the Interface Index of the main hardware interface descriptor block (IDB) as the Group ID. For example, for a VLAN attachment circuit in interface Gigabit-Ethernet 1/0.100, the Group ID used to be the IfIndex for GigabitEthernet 1/0. This way, wildcard label withdrawals or notifications could be sent on physical port failure. However, because of the limited benefit of the

wildcard withdrawal, current releases of Cisco IOS software set the Group ID to 0 for all pseudowires, and wildcard withdraw messages are not sent. For ATM ACs, a non-zero value is still used, but wildcard withdrawals are not sent.

Case Study 8-7: AToM Hardware Capabilities

Throughout this book, multiple case studies provide generic configuration but do not focus on platform-specific differences. Although remaining hardware agnostic in the case studies is useful, some platform information is required. This section provides a means of checking support on specific platforms without actually listing all platform restrictions.

This section presents an exec command that displays the AToM hardware capability and uses a c7200 VXR series router with two ATM port adapters (PA):

- PA-A1 in slot 3 that does not support AToM
- PA-A3 version 2.0 in slot 4 that does support AToM

First issue the command specifying the ATM PA that does not support ATM transport. See Example 8-41 for abbreviated output. Core functionality refers to a core-facing PE interface, and Edge functionality refers to an edge-facing PE interface.

Example 8-41. *Unsupported AToM Layer 2 Transport Hardware Capability*

```
C7206VXR#show mpls l2transport hw-capability interface ATM 3/0
Interface ATM3/0

!Output omitted for brevity
Transport type ATM AAL5
  Core functionality:
    MPLS label disposition supported
    Control word processing supported
    Sequence number processing not supported
    VCCV Type 1 processing supported
  Edge functionality:
    Not supported

Transport type ATM CELL
  Core functionality:
    MPLS label disposition supported
    Control word processing not supported
    Sequence number processing not supported
    VCCV Type 1 processing not supported
  Edge functionality:
    Not supported
!Output omitted for brevity
Transport type ATM VCC CELL
  Core functionality:
    MPLS label disposition supported
    Control word processing supported
    Sequence number processing not supported
    VCCV Type 1 processing supported
  Edge functionality:
    Not supported
```

```
Transport type ATM VPC CELL
  Core functionality:
    MPLS label disposition supported
    Control word processing supported
    Sequence number processing not supported
    VCCV Type 1 processing supported
  Edge functionality:
    Not supported
C7206VXR#
```

You can see from Example 8-41 that imposition is not supported for any ATM transport type. Therefore, the commands to configure AToM ATM transport do not appear in the router CLI. In contrast, you can issue the same command against an ATM PA that does support AToM ATM transport (see Example 8-42).

Example 8-42. *Supported AToM Layer 2 Transport Hardware Capability*

```
C7206VXR#show mpls l2transport hw-capability interface ATM 4/0
Interface ATM4/0

!Output omitted for brevity

Transport type ATM AAL5
  Core functionality:
    MPLS label disposition supported
    Control word processing supported
    Sequence number processing not supported
    VCCV Type 1 processing supported
  Edge functionality:
    MPLS label imposition supported
    Control word processing supported
    Sequence number processing not supported
    ATM AAL5 forwarding supported
    F5 OAM cell forwarding supported
    CLP bit setting supported
    CLP bit detecting supported
    EFCI bit setting supported
    EFCI bit detecting supported

Transport type ATM CELL
  Core functionality:
    MPLS label disposition supported
    Control word processing supported
    Sequence number processing not supported
    VCCV Type 1 processing supported
  Edge functionality:
    MPLS label imposition supported
    Control word processing supported
    Sequence number processing not supported
    ATM cell forwarding not supported
    ATM cell packing not supported
    F5 OAM cell forwarding supported
    CLP bit setting not supported
    CLP bit detecting not supported
    EFCI bit setting not supported
    EFCI bit detecting not supported
!Output omitted for brevity
C7206VXR#
```

In Example 8-42, you can see that imposition functions are supported for all AToM ATM transport. Specific information about ATM transport features is also included in the command output.

This procedure enables you to check the hardware dependencies without having to check support matrices.

Case Study 8-8: Packed Cell Relay over MPLS

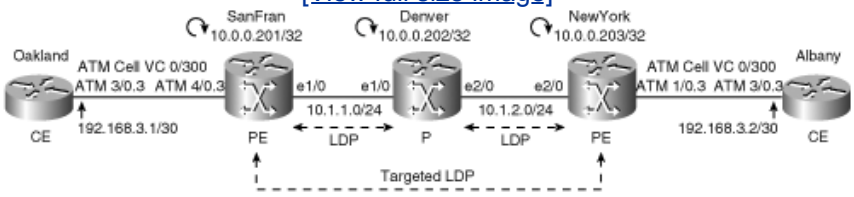
The forthcoming two sections present advanced topics on ATM transport over MPLS. This section introduces ideas and configuration examples on packed cell relay over MPLS, and the next section compares different ATM transports from a data plane perspective.

The concept of cell packing that was presented previously in this chapter is straightforward. It involves concatenating multiple cells into a single AToM packet. The tradeoff is also quite direct: bandwidth efficiency gained by sharing the AToM and lower PCI overhead versus increased latency and jitter.

This section presents a case study on packed cell relay using the topology shown in Figure 8-16.

Figure 8-16. Packed Cell Relay Case Study Topology

[View full size image]



Configuring Cell Packing

Configuring packed cell relay involves two steps:

Step 1. Specifying three timers for the length of time that a PE router can wait for cells to be concatenated into the same MPLS packet. The maximum cell packing timeout (MCPT) configuration is performed at the ATM interface level.

Step 2. Specifying the maximum number of cells to be concatenated in an MPLS packet and the timer that is available for use. You perform this step at the l2transport PVC configuration.

The configuration for the SanFran side is shown in Example 8-43. It highlights the specific cell packing commands.

Example 8-43. *Packed Cell Relay Configuration*

```
SanFran#show running-config interface ATM 4/0
Building configuration...
```

```
Current configuration : 179 bytes
!
interface ATM4/0
 no ip address
 load-interval 30
 atm mcpt-timers 500 800 4095
end

SanFran#
SanFran#show running-config interface ATM 4/0.3
Building configuration...

Current configuration : 296 bytes
!
interface ATM4/0.3 point-to-point
 description *** Packed Cell VC AToM to Oakland ***
 pvc 0/300 l2transport
  encapsulation aal0
  cell-packing 10 mcpt-timer 2
  xconnect 10.0.0.203 300 encapsulation mpls
 !
end

SanFran#
```

Three interface level timers are configured with the **atm mcpt-timers** command. They are shared by all Layer 2 transport VCs and VPs under that interface and its subinterfaces.

Verifying Cell Packing Configuration and Operation

You advertise the max cells to be packed by adding a new interface parameter to the LDP label mapping message (see Example 8-44). As mentioned earlier in the "Encapsulations and Packet Format for Cell Transport" section, the MTU interface parameter does not apply to ATM Cell transport and is not advertised.

Example 8-44. *Packed Cell Relay Verification*

```
SanFran#show mpls l2transport binding 300
  Destination Address: 10.0.0.203, VC ID: 300
    Local Label:  18
        Cbit: 1,    VC Type: ATM VCC CELL,    GroupID: 5
        MTU: n/a,   Interface Desc: *** Packed Cell VC AToM to Oakland ***
        Max Concatenated ATM Cells: 10
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 18
        Cbit: 1,    VC Type: ATM VCC CELL,    GroupID: 2
        MTU: n/a,  Interface Desc: *** Packed Cell VC AToM to Albany ***
        Max Concatenated ATM Cells: 10
        VCCV Capabilities: Type 1, Type 2
SanFran#
```

To fully understand how cell relay packing works, you can perform a simple experiment. First calculate the size of a ping that would fully occupy ten cells without padding. You can use the following formula and refer to the packets shown in Figure 8-7:

```
Number of cells * 48 Bytes/cell - AAL5 PDU Trailer          - SNAP Header for IP =
Number of cells * 48 Bytes/cell - (UU + CPI + Length + CRC) - (LLC + OUI + etype) =
        10      *       48      - (1 + 1 + 2 + 4)           - (3 + 3 + 2) = 464 Bytes
```

You can see that specifying a PING size of 464 bytes requires exactly ten ATM cells. The first part of the exercise consists of sending 10,000 PING packets of 464 bytes and checking the average number of cells per packet (see Example 8-45).

Example 8-45. *Packed Cell Relay Exercise Part 1*

```
SanFran#ping
Protocol [ip]:
Target IP address: 192.168.3.2
Repeat count [5]: 10000
Datagram size [100]: 464
Timeout in seconds [2]:
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 10000, 464-byte ICMP Echos to 192.168.3.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (10000/10000), round-trip min/avg/max = 1/2/16 ms
SanFran#show atm cell-packing
                                average                 average
       circuit                local nbr of cells      peer nbr of cells      MCPT
       type                   MNCP  rcvd in one pkt   MNCP sent in one pkt   (us)
ATM4/0.3        vc 0/300        10        9             10        9           800
SanFran#
```

The counter from Example 8-45 shows nine cells instead of ten because of OAM cells bringing down the average slightly. The second part of the exercise is to clear the counters and send another 10,000 PING packets but now 1 byte longer than before, which is 465 bytes (see Example 8-46).

Example 8-46. *Packed Cell Relay Exercise Part 2*

```
SanFran#clear counters
Clear "show interface" counters on all interfaces [confirm]
SanFran#
*May 27 01:22:25.858: %CLEAR-5-COUNTERS: Clear counter on all interfaces by
  console
SanFran#ping
Protocol [ip]:
Target IP address: 192.168.3.2
Repeat count [5]: 10000
Datagram size [100]: 465
Timeout in seconds [2]:
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 10000, 465-byte ICMP Echos to 192.168.3.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (10000/10000), round-trip min/avg/max = 4/7/24 ms
SanFran#show atm cell-packing
```

```
                                        average                  average
       circuit                    local nbr of cells        peer nbr of cells      MCPT
       type                       MNCP  rcvd in one pkt     MNCP sent in one pkt   (us)
ATM4/0.3         vc 0/300          10        5               10        5            800
SanFran#
```

Example 8-46 shows that the average number of cells packed both in receive and transmit directions dropped drastically to 5. This is because each ICMP echo request and echo packets now require 11 cells instead of 10. Therefore, each PING packet is sent using two MPLS AToM packets, one with 10 cells and the other with just 1 cell, averaging a bit over 5 cells per MPLS packet. Note that the timers are short enough that the second MPLS packet does not have 10 cells. In addition, the second packet is sent with only one cell because the timer expires before the second echo request is received.

Case Study 8-9: Understanding Different ATM Transfer Modes

You have learned about the different AToM encapsulationsin particular the different modes of transporting an ATM PVC (AAL5 CPCS-SDU, single cell relay, and packed cell relay). This section illustrates their similarities and differences with examples to solidify the concepts.

You can highlight some of the differences among ATM transfer modes by sending a 36-byte ping from the Oakland CE routers in the three modes and comparing the capture of those AToM packets in the link between the Denver P and NewYork P routers. You can obtain the capture as output for the command **debug mpls l2transport packet data**. All packets you capture share the same Layer 2 encapsulation (source and destination MAC address and MPLS unicast Ethertype). They also share the absence of a PSN label because of Penultimate Hop Popping (PHP) and a VC MPLS header with a different label. Finally, all the packets share the presence of a control word that has different characteristics.

AAL5 CPCS-SDU Mode

The first case is the AAL5 SDU transport of the ICMP PING. The AAL5 CPCS-SDU consists of the IP/ICMP packet with a SNAP header. Therefore, the contents of the AToM packet are 48 bytes: 4 bytes of control word, 8 bytes of LLC-SNAP Header, and 36 bytes of IP/ICMP packet. The packet is shown in Example 8-47, including inline decoding. The AAL5 CPCS-SDU is highlighted.

Example 8-47. *AAL5 SDU Mode Packet Decode*

```
SanFran#debug mpls l2transport packet data
01:45:36: ATOM imposition: out Fa4/0, size 66, EXP 0x0, seq 0, control word
 0x300000
01:45:36: 00 0C CF 55 24 08 00 04 4E 26 18 70 88 47 00 01
          ^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
          SA MAC            DA MAC            |     VC Label-->
                                              etype = MPLS Unicast
01:45:36: 31 02 00 30 00 00 AA AA 03 00 00 00 08 00 45 00
          ^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^ ^^...
    <--VC_Label Ctrl-word   SNAP                    Begins IP Packet
       Label=19             LLC: AAAA03
       S=1                  OUI: 000000
       TTL=2                etype: 0x0800 (IP)
01:45:36: 00 24 00 14 00 00 FF 01 38 71 C0 A8 01 02 C0 A8
01:45:36: 01 01 00 00 27 F7 00 04 00 00 00 00 00 00 00 70
01:45:36: D7 94
          ^^^^^
```

```
 Ends IP Packet

01:45:40: ATOM imposition: out Fa4/0, size 74, EXP 0x0, seq 0, control word
 0x8380000
```

In Example 8-47, you can see that the contents of the MPLS packets are the 4-byte control word plus the 44-byte AAL5 CPCS-SDU, totaling 48 bytes. You can also see that the value of the control word is 0x00300000, which contains the length equal to 0x30 = 48 bytes.

For completeness, Example 8-47 also includes the first line of the output of the transport of ATM OAM cells in AAL5 CPCS-SDU mode. The value of the control word is now 0x08380000. This depicts a length of 0x38 = 56 bytes (52 bytes of the ATM cell plus 4 bytes of the control word). The T-bit is set to indicate that the contents are an ATM cell. Remember this length to compare it to the single cell relay and packed cell relay modes.

Single Cell Relay Mode

This section describes the ATM cell relay transport VC mode with single cell relay. In this case, the complete AAL5 CPCS-PDU is transported and is made of the CPCS-SDU plus the 8-byte CPCS-PDU trailer and padding if needed. For the 36-byte PING, the CPCS-SDU and the CPCS-Trailer total 52 bytes. Therefore, the AAL5 PDU is segmented into two cells in the SAR layer and padded with 44 bytes (48 * 2 52). Because this case is single cell relay, two AToM packets are generated, each containing one ATM cell (see Example 8-48). For comparison, the AAL5 CPCS-SDU is highlighted.

Example 8-48. *Single Cell Relay Mode Packet Decode*

```
SanFran#debug mpls l2transport packet data
02:27:46: ATOM imposition: out Fa4/0, size 74, EXP 0x0, seq 0, control word
 0x380000
02:27:46: 00 0C CF 55 24 08 00 04 4E 26 18 70 88 47 00 01
          ^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
          SA MAC            DA MAC            |     VC Label-->
                                              etype = MPLS Unicast
02:27:46: 21 02 00 38 00 00 00 00 0C 80 AA AA 03 00 00 00
          ^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^
    <--VC_Label Ctrl-word   ATM Cell    SNAP
       Label=18             Header      LLC: AAAA03
       S=1                  0/200       OUI:  000000
       TTL=2                EoAAL5=0    etype: 0x0800 (IP)
02:27:46: 08 00 45 00 00 24 00 2F 00 00 FF 01 36 56 C0 A8
          ^^^^^^ ^^...
          SNAP   Begins IP Packet
02:27:46: 02 02 C0 A8 02 01 00 00 8C DE 00 0E 00 00 00 00
02:27:46: 00 00 00 97 72 7C 00 00 00 00
                            ^^...
                            CPCS-PDU Padding
02:27:46: ATOM imposition: out Fa4/0, size 74, EXP 0x0, seq 0, control word
  0x380000
02:27:46: 00 0C CF 55 24 08 00 04 4E 26 18 70 88 47 00 01
          ^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
          SA MAC            DA MAC            |     VC Label-->
                                              etype = MPLS Unicast
02:27:46: 21 02 00 38 00 00 00 00 0C 82 00 00 00 00 00 00
          ^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^...
    <--VC_Label Ctrl-word   ATM Cell    CPCS-PDU Padding
       Label=18             Header
       S=1                  VPI=0; VCI = 200
```

```
       TTL=2                EoAAL5 = 1
02:27:46: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02:27:46: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02:27:46: 00 00 00 00 00 2C 2B F1 73 FD
          ...^^ ^^^^^^^^^^^^^^^^^^^^^^
   CPCS-PDU Pad CPCS-PDU Trailer
                UU = 0; CPI = 0;
                Length = 0x2C = 44 Bytes
                CRC = 0x2BF173FD
```

From Example 8-48, you can see the 44 bytes of CPCS-PDU padding, the 8-byte CPCS-PDU trailer, and 4 bytes for each of the ATM-Layer ATM cell headers. The value of the control word is 0x00380000, from which the length equals 0x38 = 56 bytes. These 56 bytes are 52 bytes of ATM cell plus 4 bytes of control word.

Packed Cell Relay Mode

This section includes a capture for ATM packed cell relay VC mode (see Example 8-49). In this case, the two transported cells are concatenated into a single AToM packet.

Example 8-49. *Packed Cell Relay Mode Packet Decode*

```
SanFran#debug mpls l2transport packet data
02:30:27: ATOM imposition: out Fa4/0, size 126, EXP 0x0, seq 0, control word 0x0
02:30:27: 00 0C CF 55 24 08 00 04 4E 26 18 70 88 47 00 01
          ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^ ^^^^^
          SA MAC           DA MAC           |     VC Label-->
                                            etype = MPLS Unicast
02:30:27: 41 02 00 00 00 00 00 00 12 C0 AA AA 03 00 00 00
          ^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^
    <--VC_Label Ctrl-word  ATM Cell     SNAP
       Label=20            Header       LLC: AAAA03
       S=1                 0/300        OUI: 000000
       TTL=2               EoAAL5=0     etype: 0x0800 (IP)
02:30:27: 08 00 45 00 00 24 00 3A 00 00 FF 01 34 4B C0 A8
          ^^^^^^ ^^...
          SNAP   Begins IP Packet
02:30:27: 03 02 C0 A8 03 01 00 00 06 1C 00 12 00 00 00 00
02:30:27: 00 00 06 01 F3 D0 00 00 00 00 00 00 12 C2 00 00
                      ...^^^^^^^^^^^ ^^^^^^^^^^^ ^^...
                  CPCS-PDU Padding  ATM Cell     CPCS-PDU Padding
                                    Header
                                    VPI/VCI = 0/300
                                    EoAAL5=1
02:30:27: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02:30:27: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02:30:27: 00 00 00 00 00 00 00 00 00 2C A6 9C AA FC
                    ...^^ ^^^^^^^^^^^^^^^^^^^^^^^^
              CPCS-PDU Pad CPCS-PDU Trailer
                           UU = 0; CPI = 0;
                           Length = 0x2C = 44 Bytes
                           CRC = 0xA69CAAFC
```

The format of the contents of this single AToM packet is similar to merging the contents of two AToM packets in the previous case. That is, the two cells share the Layer 2, MPLS, and pseudowire (control

word) overheads. The control word presents a value of 0x00000000. The length is not included because it is 108 bytes (52 * 2 + 4), which is greater than 63 bytes, which is the maximum value the length field can take with 6 bits (2^6 - 1 = 63).

The overhead is quite significant in the ATM cell transport because the size of the user data transported is small, and there is additional overhead from the ATM cell header and AAL5 padding and trailer. Table 8-4 lists all the fields of the AToM packet contents and their lengths for the three cases.

Table 8-4. Comparing ATM Transport Overheads

Transport Type	Control Word	ATM Cell Header	LLC-SNAP	IP/ICMP	CPCS-Pad	CPCS-Trailer	Total
AAL5 CPCS-SDU	4 bytes	0	8 bytes	36 bytes	0	0	48 bytes
Single Cell Relay	2 * 4 bytes	2 * 4 bytes	8 bytes	36 bytes	44 bytes	8 bytes	112 bytes
Packed Cell Relay	4 bytes	2 * 4 bytes	8 bytes	36 bytes	44 bytes	8 bytes	108 bytes

From this table, you can see that when you are transporting AAL5 packets, AAL5 CPCS-SDU mode is more efficient, especially when the ATM AAL5 packets are small. When you are transporting other AALs, such as AAL1 for Circuit Emulation Services (CES) or AAL2 for Voice over ATM (VoATM), cell relay is the only option. Similarly, CRoMPLS is the only option when you are trunking using VP or port mode, because AAL5 mode does not exist for VP and port mode ATM transport.

Summary

In this chapter, you learned specifics about the transport of Layer 2 WAN protocols over MPLS using AToM. In particular, this chapter presented theory, configuration, and verification of the transport of several WAN protocols over MPLS. The underlying control plane concept is the same as with Ethernet over MPLS (EoMPLS); however, many encapsulation specifics are based on the martini drafts when adapting AToM to the transport of HDLC, PPP, Frame Relay DLCIs, ATM AAL5 SDUs, and ATM cells.

This chapter discussed the importance of the MTU setting both in the edge devices and the core. For all WAN protocol transport over MPLS except the transport of ATM cells, a pseudowire requires matching MTUs in both ends for it to come up. Even when matching MTUs in the attachment circuits enable control plane success, a conscientious MTU setting in the core is required to avoid data plane problems.

This chapter concluded with four additional case studies. In these case studies, you learned what the exact format of LDP label mapping messages is, how to check for hardware capabilities on a router, and how to decouple the platform specifics. You also learned about packed cell relay over MPLS and the ins and outs of ATMoMPLS VC mode, including a detailed comparison of three different ways of transporting an ATM VC using AToM.

Chapter 9. Advanced AToM Case Studies

This chapter covers the following topics:

- Load sharing
- Preferred path
- AToM pseudowires with MPLS traffic engineering fast reroute
- AToM pseudowire over GRE tunnel
- Pseudowire emulation in multi-AS networks
- LDP authentication for pseudowire signaling
- Verifying pseudowire data connectivity
- Quality of service in AToM

So far, this book has explained the fundamentals of Any Transport over MPLS (AToM) and discussed the basic pseudowire emulation case studies for LAN and WAN protocols. This chapter takes an in-depth look at more complex scenarios in which routing protocols, MPLS applications, and AToM features interact with one another. Because the advanced deployment scenarios cover a wide range of concepts, the format of this chapter varies somewhat from other case study chapters.

Load Sharing

When more than one path exists between the source and destination, load sharing often becomes an important factor for better bandwidth utilization. You can deploy load sharing in many different ways. Load sharing is sometimes completely transparent to pseudowire traffic if it only involves core routers in the network. The case studies in this section focus on the load-sharing scenarios that involve provider edge (PE) routers performing pseudogwire emulation over MPLS. More precisely, an ingress PE router sees more than one path to reach the egress PE router for pseudowire traffic.

Service providers normally deploy standard-based routing protocols in the core networks, such as Open Shortest Path First (OSPF) and Intermediate System-to-Intermediate System (IS-IS), which support Equal-Cost Multipath (ECMP) load sharing. With these routing protocols, only routes that have the least cost to a given destination are shown in the forwarding table. If multiple equal-cost routes go to the same destination, all of them are installed in the forwarding table. Depending on the traffic type and forwarding configuration, packets with the same destination are distributed across these paths.

Note

The total number of equal-cost routes available depends on with Cisco IOS releases and hardware platform specifications.

The load-sharing scheme used in AToM ensures that packets being transmitted over a given pseudowire follow the same path, and packets of different pseudowires are spread across all available equal-cost paths.

The first aspect of this scheme is to minimize the possibility of out-of-order packets because of load sharing. Some Layer 2 protocols or higher layer protocols do not function or their performance is drastically penalized when packets are delivered out of order. Packets that are transmitted over a single pseudowire are generally considered part of the same data flow, which is likely of the same protocol or application. Putting packets of the same flow through the same path reduces the likelihood that they will be misordered in the core network.

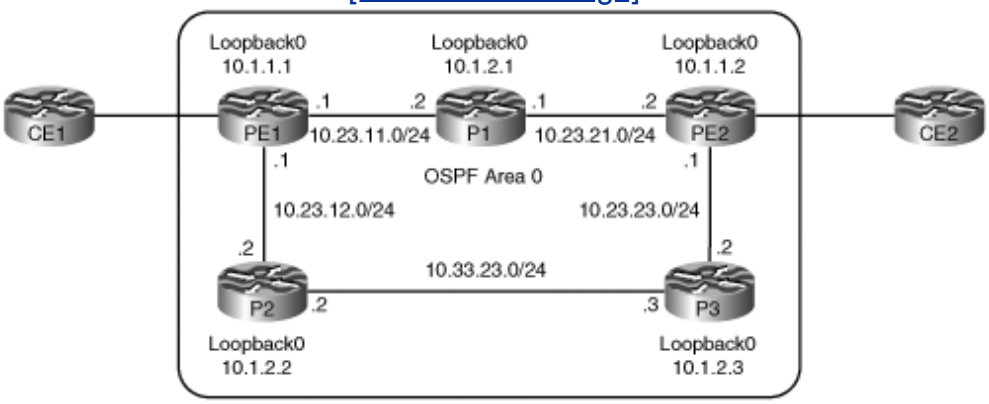
The second aspect of this scheme mandates that the load-sharing scheme used in AToM utilizes a hashing algorithm to assign pseudowires to equal-cost paths. By default, the remote virtual circuit (VC) label of a pseudowire acts as the hash key to calculate the outgoing path. Because of this, pseudowire packets that have the same remote VC label are sent through the same path. Again, the rationale behind this is to keep these packets in the same data flow and to minimize out-of-order packets.

Figure 9-1 shows a network topology with multiple paths between PE1 and PE2. Suppose that the core network is configured with OSPF as the Interior Gateway

Protocol (IGP).

Figure 9-1. ***Multipath Network Topology***

[View full size image]



PE1 has an initial configuration, shown in Example 9-1.

Example 9-1. ***Initial Pseudowire Configuration***

```
hostname PE1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.2 100 encapsulation mpls
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 encapsulation mpls
!
interface Ethernet0/0.3
 encapsulation dot1Q 300
 xconnect 10.1.1.2 300 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.23.12.1 255.255.255.0
```

```
 mpls ip
!
interface Serial3/0
 ip address 10.23.11.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.1 0.0.0.0 area 0
 network 10.23.11.0 0.0.0.255 area 0
 network 10.23.12.0 0.0.0.255 area 0
```

The initial configuration on PE2 is shown in Example 9-2.

Example 9-2. *PE2 Initial Pseudowire Configuration*

```
hostname PE2
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.2 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.1 100 encapsulation mpls
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.1 200 encapsulation mpls
!
interface Ethernet0/0.3
 encapsulation dot1Q 300
 xconnect 10.1.1.1 300 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.23.23.1 255.255.255.0
 mpls ip
!
interface Serial3/0
 ip address 10.23.21.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.2 0.0.0.0 area 0
```

```
 network 10.23.21.0 0.0.0.255 area 0
 network 10.23.23.0 0.0.0.255 area 0
```

The following case study sections discuss how PE routers make path selection decisions for pseudowire traffic when these types of forwarding paths are present:

- Case Study 9-1: Unequal-Cost Multipath
- Case Study 9-2: Equal-Cost Multipath

Case Study 9-1: Unequal-Cost Multipath

With open standard IGP routing protocols such as OSPF and IS-IS, only routes that have the lowest cost are installed in the forwarding table. Therefore, for cases in which multiple paths point to the same destinations but each has a different cost, only one path is selected for packet forwarding. In Example 9-3, PE1 and PE2 are connected through P1 using T1 links and through P2 and P3 using Ethernet links. By default, the OSPF cost is 64 for T1 links and 10 for Ethernet links.

Example 9-3. *SPF Costs on PE1 Interfaces*

```
PE1#show ip ospf interface serial3/0
Serial3/0 is up, line protocol is up
  Internet Address 10.23.11.1/24, Area 0
  Process ID 1, Router ID 10.1.1.1, Network Type POINT_TO_POINT, Cost: 64

PE1#show ip ospf interface ethernet1/0
Ethernet1/0 is up, line protocol is up
  Internet Address 10.23.12.1/24, Area 0
  Process ID 1, Router ID 10.1.1.1, Network Type BROADCAST, Cost: 10
```

To reach the loopback interface on PE1 and PE2, you must take the cost of the loopback interface into account, as shown in Example 9-4.

Example 9-4. *SPF Cost on PE1 Loopback Interface*

```
PE1#show ip ospf interface loopback0
Loopback0 is up, line protocol is up
  Internet Address 10.1.1.1/32, Area 0
  Process ID 1, Router ID 10.1.1.1, Network Type LOOPBACK, Cost: 1
```

The cost for PE1 to go through P1 to reach PE2 is 64 + 64 + 1 = 129, whereas the cost to go through P2 and P3 is 10 + 10 + 10 + 1 = 31. The least-cost path to the destination 10.1.1.2 is through P2 and P3, thus chosen by the routing protocol to forward data packets. You can observe this through the **show** command in Example 9-5.

Example 9-5. *SPF Cost to PE2 Loopback Interface*

```
PE1#show ip route 10.1.1.2
Routing entry for 10.1.1.2/32
  Known via "ospf 1", distance 110, metric 31, type intra area
  Last update from 10.23.12.2 on Ethernet1/0, 00:52:57 ago
  Routing Descriptor Blocks:
  * 10.23.12.2, from 10.1.1.2, 00:52:57 ago, via Ethernet1/0
      Route metric is 31, traffic share count is 1
```

Because Ethernet1/0 is the only outgoing interface that the routing protocol chooses to reach the destination, all AToM pseudowires on PE1 take that interface (see Example 9-6).

Example 9-6. *A Single Lowest-Cost Path Selected on PE1*

```
PE1#show mpls l2transport summary
Destination address: 10.1.1.2, total number of vc: 3
  0 unknown, 3 up, 0 down, 0 admin down
  3 active vc on MPLS interface Et1/0
```

PE2 has a similar result to PE1, as shown in Example 9-7.

Example 9-7. *A Single Lowest-Cost Path Selected on PE2*

```
PE2#show mpls l2transport summary
Destination address: 10.1.1.1, total number of vc: 3
  0 unknown, 3 up, 0 down, 0 admin down
  3 active vc on MPLS interface Et1/0
```

Case Study 9-2: Equal-Cost Multipath

In the previous case study, all AToM pseudowires took a single outgoing path when all feasible outgoing paths had different costs. Example 9-8 uses the same topology and

configuration, but the cost of the link between P2 and P3 is increased so that the overall cost for the path going through P2 and P3 becomes equal to the one through P1. As a result, the routing table on PE1 now shows it has two equal-cost paths to the destination.

Example 9-8. *Two Equal-Cost Paths to PE2 Loopback Interface*

```
PE1#show ip route 10.1.1.2
Routing entry for 10.1.1.2/32
  Known via "ospf 1", distance 110, metric 129, type intra area
  Last update from 10.23.12.2 on Ethernet1/0, 00:00:14 ago
  Routing Descriptor Blocks:
  * 10.23.11.2, from 10.1.1.2, 00:00:14 ago, via Serial3/0
      Route metric is 129, traffic share count is 1
    10.23.12.2, from 10.1.1.2, 00:00:14 ago, via Ethernet1/0
      Route metric is 129, traffic share count is 1
```

In Example 9-9, you see that PE2 has a similar result to PE1.

Example 9-9. *Two Equal-Cost Paths to PE1 Loopback Interface*

```
PE2#show ip route 10.1.1.1
Routing entry for 10.1.1.1/32
  Known via "ospf 1", distance 110, metric 129, type intra area
  Last update from 10.23.23.2 on Ethernet1/0, 00:00:41 ago
  Routing Descriptor Blocks:
  * 10.23.21.1, from 10.1.1.1, 00:00:41 ago, via Serial3/0
      Route metric is 129, traffic share count is 1
    10.23.23.2, from 10.1.1.1, 00:00:41 ago, via Ethernet1/0
      Route metric is 129, traffic share count is 1
```

When equal-cost paths are available, AToM attempts to distribute pseudowires among them. On PE1 and PE2, the **show mpls l2transport summary** command demonstrates that AToM pseudowires are assigned to different output interfaces, as shown in Example 9-10.

Example 9-10. *Pseudowires Load Share Across Equal-Cost Paths*

```
PE1#show mpls l2transport summary
Destination address: 10.1.1.2, total number of vc: 3
  0 unknown, 3 up, 0 down, 0 admin down
  2 active vc on MPLS interface Se3/0
  1 active vc on MPLS interface Et1/0

PE2#show mpls l2transport summary
```

```
Destination address: 10.1.1.1, total number of vc: 3
  0 unknown, 3 up, 0 down, 0 admin down
  2 active vc on MPLS interface Et1/0
  1 active vc on MPLS interface Se3/0
```

Notice that PE1 and PE2 come up with different ideas for how to load share the pseudowires. For all three provisioned pseudowires, PE1 distributes two of them to the serial interface, which takes P1 as the next-hop router towards PE2; PE2 distributes two of the pseudowires to the Ethernet interface, which takes P3 as the next-hop router toward PE1. This means that one of the pseudowires selects the path through P1 in the direction from PE1 to PE2, but it takes the path through P3 and P2 in the return direction from PE2 to PE1. For this particular pseudowire, PE1 transmits its packets to PE2 through P1, and PE2 transmits its packets to PE1 through P3 and P2.

This is not really a problem or mistake. Packets of this pseudowire being transmitted in each direction still follow the same path. IP/MPLS traffic is generally considered unidirectional, and the routing protocol is free to choose different equal-cost paths for packets sent in different directions. It is normal for the core network to route packets under the default hop-by-hop and best-effort forwarding scheme. On the other hand, if the core network engages in traffic engineering techniques and prefers explicit paths for pseudowire traffic, you need to associate AToM pseudowires with these explicit paths. The next section discusses preferred paths in more detail.

Under the default forwarding scheme, the **show mpls l2transport vc detail** command reveals how the underlying load-sharing algorithm works when equal-cost paths are present (see Example 9-11).

Example 9-11. *Load Sharing Selects Different Output Interfaces for Pseudowires*

```
PE1#show mpls l2transport vc detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 100 up
  Destination address: 10.1.1.2, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 17, next hop point2point
    Output interface: Se3/0, imposed label stack {17 22}
  Create time: 2d10h, last status change time: 2d10h
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 21, remote 22
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
!Output omitted for brevity

Local interface: Et0/0.2 up, line protocol up, Eth VLAN 200 up
  Destination address: 10.1.1.2, VC ID: 200, VC status: up
    Preferred path: not configured
    Default path: active
```

```
    Tunnel label: 17, next hop 10.23.12.2
    Output interface: Et1/0, imposed label stack {17 25}
  Create time: 2d10h, last status change time: 2d10h
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 22, remote 25
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
!Output omitted for brevity

Local interface: Et0/0.3 up, line protocol up, Eth VLAN 300 up
  Destination address: 10.1.1.2, VC ID: 300, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 17, next hop point2point
    Output interface: Se3/0, imposed label stack {17 26}
  Create time: 2d10h, last status change time: 2d10h
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 23, remote 26
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
!Output omitted for brevity
```

On PE1, AToM pseudowires of VC ID 100, 200, and 300 have a remote VC label of 22, 25, and 26 respectively. Because two equal-cost paths are available, the algorithm uses the remote VC labels, such as the hash key, to calculate an index value for each pseudowire. Then the index value maps to a hash bucket that corresponds to one of the two output paths. The hash buckets for pseudowires of VC ID 100 and 300 map to the same output interface; therefore, they take the same output path.

Preferred Path

When a given destination has multiple equal-cost paths, the load-sharing algorithm used in AToM distributes pseudowires to all available paths by default, and users do not have the option of choosing which pseudowire goes through which output path.

If network operators want to choose a particular output path for a given pseudowire, they can configure a preferred path and associate it with the pseudowire. AToM provides two options to configure preferred paths for pseudowires: IP routing and MPLS traffic engineering.

Before starting the discussion on preferred path options, it is worthwhile to reiterate that the IP address configured in the **xconnect** *ip_address vc_id* command must always be the router ID that the remote PE router uses for Label Distribution Protocol (LDP) signaling. The router ID is the first four bytes in the LDP ID. The **show mpls ldp neighbor** and **show mpls ldp discovery** commands display the LDP IDs of the local PE router and its neighbors (see Example 9-12).

Example 9-12. *LDP Neighbors and Their LDP IDs*

```
PE1#show mpls ldp neighbor
    Peer LDP Ident: 10.33.23.1:0; Local LDP Ident 10.1.1.1:0
        TCP connection: 10.33.23.1.11000 - 10.1.1.1.646
        State: Oper; Msgs sent/rcvd: 5548/5548; Downstream
        Up time: 3d08h
        LDP discovery sources:
          Ethernet1/0, Src IP addr: 10.23.12.2
        Addresses bound to peer LDP Ident:
          10.33.23.1      10.23.12.2
    Peer LDP Ident: 10.23.11.2:0; Local LDP Ident 10.1.1.1:0
        TCP connection: 10.23.11.2.11142 - 10.1.1.1.646
        State: Oper; Msgs sent/rcvd: 125/126; Downstream
        Up time: 01:30:57
        LDP discovery sources:
          Serial3/0, Src IP addr: 10.23.11.2
        Addresses bound to peer LDP Ident:
          10.23.11.2      10.23.21.1      10.43.11.2
    Peer LDP Ident: 10.1.1.2:0; Local LDP Ident 10.1.1.1:0
        TCP connection: 10.1.1.2.11003 - 10.1.1.1.646
        State: Oper; Msgs sent/rcvd: 136/137; Downstream
        Up time: 01:30:53
        LDP discovery sources:
          Targeted Hello 10.1.1.1 -> 10.1.1.2, active, passive
        Addresses bound to peer LDP Ident:
          10.1.1.2        10.23.21.2      10.23.23.1      10.1.1.200
          10.1.1.201

PE1#show mpls ldp discovery
 Local LDP Identifier:
    10.1.1.1:0
    Discovery Sources:
    Interfaces:
        Ethernet1/0 (ldp): xmit/recv
            LDP Id: 10.33.23.1:0
        Serial3/0 (ldp): xmit/recv
            LDP Id: 10.23.11.2:0
    Targeted Hellos:
```

```
10.1.1.1 -> 10.1.1.2 (ldp): active/passive, xmit/recv
    LDP Id: 10.1.1.2:0
```

Three LDP signaling sessions are available, as shown in the **show mpls ldp neighbor** command output. PE1 establishes two nontargeted LDP sessions with P1 and P2 to exchange IGP labels, which serve as tunnel labels for pseudowire packets. The targeted LDP session between PE1 and PE2 is for pseudowire signaling. The first four bytes of the LDP ID for PE2 are 10.1.1.2, its router ID. The targeted LDP session for pseudowire signaling uses the local and remote router IDs as the source and destination addresses. When no preferred path is configured for a pseudowire, an output path is selected based on the remote router ID in the forwarding table for pseudowire packets.

The preferred path option not only allows pseudowire data packets to flow through a different path from pseudowire control packets, but it also makes it possible to provide differentiated services to pseudowires with different forwarding requirements. For example, you can place pseudowires that carry voice traffic to a special traffic-engineered path with low latency and jitter; you can place pseudowires that remotely back up a large amount of data for file servers to a best-effort path that allows high bursts.

To configure a preferred path, you first need to configure a pseudowire class and associate it with the pseudowire. A pseudowire class configures common attributes for a group of pseudowires. For AToM pseudowires, the encapsulation for the pseudowire class is MPLS, as shown in Example 9-13.

Example 9-13. *Configuring a Pseudowire Class*

```
PE1(config)#pseudowire-class PE1-P1-PE2
PE1(config-pw-class)#encapsulation mpls
```

To associate a pseudowire with a pseudowire class, use the **pw-class** keyword in the **xconnect** command, as shown in Example 9-14.

Example 9-14. *Configuring an* xconnect *Command with a Pseudowire Class*

```
PE1(config)#interface Ethernet0/0.2
PE1(config-subif)#xconnect 10.1.1.2 200 pw-class PE1-P1-PE2
```

The following case studies discuss how to use IP routing and MPLS traffic engineering to select preferred paths for pseudowires.

Case Study 9-3: Configuring Preferred Path Using IP Routing

Whichever method is chosen to configure a preferred path, the basic and usually overlooked prerequisite for a PE to establish network connectivity to a remote PE is a host route entry of the remote PE and its corresponding label in its forwarding tables. The host route, also known as the /32 route, and its label ensure that the local PE has an end-to-end contiguous label-switched path (LSP) to the remote PE. Any non-host route, such as a subnet route or summary route,

does not guarantee the connectivity requirement even if a subnet or summary route network covers the range to which the host route belongs.

Note

The label that corresponds to the host route needs to be explicitly present in the MPLS forwarding table. It cannot be derived from the recursive lookup of the host route's next-hop route. "Case Study 9-9: BGP IPv4 Label Distribution with IBGP Peering," explains this topic in more detail.

Host routes such as router IDs are typically configured on loopback interfaces. A relatively simple way to specify a preferred path for AToM pseudowires is to configure multiple loopback interfaces with different host addresses.

The **preferred-path peer** *host_address* command configures a preferred path using a host address. For example, 10.1.1.200 is a host route configured in a loopback interface on PE2, and PE1 configures a pseudowire class that takes the preferred path to reach 10.1.1.200. When forwarding pseudowire packets to PE2, PE1 uses 10.1.1.200 instead of PE2's router ID 10.1.1.2 to look up the output interface in the forwarding tables (see Example 9-15).

Example 9-15. *Configuring Preferred Path Using IP Routing*

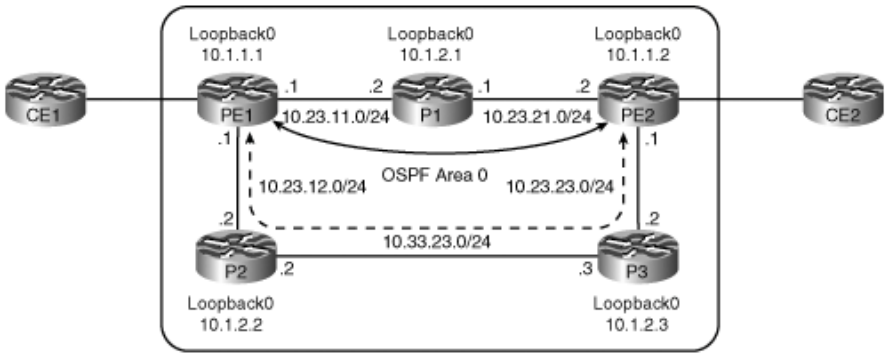
```
PE1(config)#pseudowire-class PE1-P1-PE2
PE1(config-pw-class)#encapsulation mpls
PE1(config-pw-class)#preferred-path peer 10.1.1.200 ?
  disable-fallback disable fall back to alternative route
```

Besides the preferred path, each pseudowire also computes a default path using the remote router ID that is configured in the **xconnect** command. If a preferred path is not found or active, pseudowires that are configured with the preferred path automatically fall back to the default path. The **disable-fallback** option turns off this default behavior so that pseudowires remain down until the preferred path becomes available.

In Case Study 9-2, the default load-sharing hash algorithm assigns pseudowires to different output interfaces. The pseudowire with VC ID 100 takes Interface Serial3/0, pseudowire 200 takes Interface Ethernet1/0, and pseudowire 300 takes Interface Serial3/0. Figure 9-2 shows two available paths between PE1 and PE2. In this case study, you will learn how to configure a pseudowire to associate with a particular output path.

Figure 9-2. *Preferred Path with IP Routing*

[View full size image]



Assume that the initial configuration is identical to that in the "Load Sharing" section. In the following configuration steps, the pseudowire with VC ID 100 still takes the default path assigned by the load-sharing algorithm, but pseudowire 200 takes the preferred path through P1, and pseudowire 300 takes the path through P2 and P3, as shown in Figure 9-2.

Step 1. Configure two additional loopback interfaces and host addresses on PE2:

```
PE2(config)#interface Loopback1
PE2(config-if)#ip address 10.1.1.200 255.255.255.255
PE2(config-if)#exit
PE2(config)#interface Loopback2
PE2(config-if)#ip address 10.1.1.201 255.255.255.255
```

Step 2. Add the host routes into the routing process on PE2:

```
PE2(config)#router ospf 1
PE2(config-router)#network 10.1.1.200 0.0.0.0 area 0
PE2(config-router)#network 10.1.1.201 0.0.0.0 area 0
```

Step 3. Verify that the host routes are present in the routing table on PE1:

```
PE1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
      i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
     ia - IS-IS inter area, * - candidate default, U - per-user static route
      o -  ODR

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
O       10.23.21.0/24 [110/128] via 10.23.11.2, 1d01h, Serial3/0
O       10.1.1.2/32 [110/129] via 10.23.11.2, 1d01h, Serial3/0
                    [110/129] via 10.23.12.2, 1d01h, Ethernet1/0
O       10.23.23.0/24 [110/128] via 10.23.12.2, 1d01h, Ethernet1/0
C       10.1.1.1/32 is directly connected, Loopback0
C       10.23.12.0/24 is directly connected, Ethernet1/0
```

```
C       10.23.11.0/24 is directly connected, Serial3/0
O       10.43.11.0/24 [110/74] via 10.23.11.2, 1d01h, Serial3/0
O       10.33.23.0/24 [110/118] via 10.23.12.2, 1d01h, Ethernet1/0
O       10.1.1.200/32 [110/129] via 10.23.11.2, 00:00:16, Serial3/0
                      [110/129] via 10.23.12.2, 00:00:16, Ethernet1/0
O       10.1.1.201/32 [110/129] via 10.23.11.2, 00:00:12, Serial3/0
                      [110/129] via 10.23.12.2, 00:00:12, Ethernet1/0
```

Step 4. Associate each host route with one particular output interface. You can achieve this in a couple ways, such as by using a static route, policy-based route, or inbound distribute list in OSPF. The following example shows how to configure this with a static route:

```
PE1(config)#ip route 10.1.1.200 255.255.255.255 10.23.11.2
PE1(config)#ip route 10.1.1.201 255.255.255.255 10.23.12.2
```

Caution

When you are using a static route to associate a host route to an outgoing path, configure the next-hop IP address (the IP address of the P router) instead of the output interface. Using output interfaces causes MPLS forwarding to be unable to resolve the outgoing tunnel label, which results in a broken LSP.

Step 5. Verify that each host route is associated with the desired output interface in the routing table, and each has a corresponding label in the MPLS forwarding table.

```
PE1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
      i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
     ia - IS-IS inter area, * - candidate default, U - per-user static route
      o -  ODR

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 10 subnets, 2 masks
O       10.23.21.0/24 [110/128] via 10.23.11.2, 1d01h, Serial3/0
O       10.1.1.2/32 [110/129] via 10.23.11.2, 1d01h, Serial3/0
                    [110/129] via 10.23.12.2, 1d01h, Ethernet1/0
O       10.23.23.0/24 [110/128] via 10.23.12.2, 1d01h, Ethernet1/0
C       10.1.1.1/32 is directly connected, Loopback0
C       10.23.12.0/24 is directly connected, Ethernet1/0
C       10.23.11.0/24 is directly connected, Serial3/0
O       10.43.11.0/24 [110/74] via 10.23.11.2, 1d01h, Serial3/0
O       10.33.23.0/24 [110/118] via 10.23.12.2, 1d01h, Ethernet1/0
S       10.1.1.200/32 [1/0] via 10.23.11.2
S       10.1.1.201/32 [1/0] via 10.23.12.2

PE1#show mpls forwarding-table
Local  Outgoing   Prefix          Bytes tag  Outgoing   Next Hop
tag    tag or VC  or Tunnel Id    switched   interface
16     Pop tag    10.23.21.0/24   0          Se3/0      point2point
17     20         10.1.1.2/32     0          Se3/0      point2point
```

```
       17          10.1.1.2/32     0           Et1/0       10.23.12.2
18     Pop tag     10.33.23.0/24   0           Et1/0       10.23.12.2
19     18          10.23.23.0/24   0           Et1/0       10.23.12.2
20     Pop tag     10.43.11.0/24   0           Se3/0       point2point
21     Untagged    l2ckt(200)      557264      Et0/0.2     point2point
22     23          10.1.1.201/32   0           Et1/0       10.23.12.2
24     21          10.1.1.200/32   0           Se3/0       point2point
25     Untagged    l2ckt(300)      557264      Et0/0.3     point2point
26     Untagged    l2ckt(100)      557631      Et0/0.1     point2point
```

Step 6. Configure preferred paths in the pseudowire-class configuration mode:

```
PE1(config)#pseudowire-class PE1-P1-PE2
PE1(config-pw-class)#encapsulation mpls
PE1(config-pw-class)#preferred-path peer 10.1.1.200
PE1(config-pw-class)#exit
PE1(config)#pseudowire-class PE1-P2-P3-PE2
PE1(config-pw-class)#encapsulation mpls
PE1(config-pw-class)#preferred-path peer 10.1.1.201
```

Step 7. Configure pseudowires 200 and 300 with their respective pseudowire classes:

```
PE1(config)#interface Ethernet0/0.2
PE1(config-subif)#xconnect 10.1.1.2 200 pw-class PE1-P1-PE2
PE1(config-subif)#exit
PE1(config)#interface Ethernet0/0.3
PE1(config-subif)#xconnect 10.1.1.2 300 pw-class PE1-P2-P3-PE2
```

Step 8. Verify that pseudowires 200 and 300 are taking the preferred paths:

```
PE1#show mpls l2transport vc detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 100 up
  Destination address: 10.1.1.2, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 20, next hop point2point
    Output interface: Se3/0, imposed label stack {20 22}
  Create time: 1d01h, last status change time: 1d01h
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 26, remote 22
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 1536, send 1538
    byte totals:   receive 572855, send 573600
    packet drops:  receive 0, send 0

Local interface: Et0/0.2 up, line protocol up, Eth VLAN 200 up
  Destination address: 10.1.1.2, VC ID: 200, VC status: up
    Preferred path: 10.1.1.200, active
    Default path: ready
    Tunnel label: 21, next hop point2point
```

```
      Output interface: Se3/0, imposed label stack {21 25}
    Create time: 1d01h, last status change time: 1d01h
    Signaling protocol: LDP, peer 10.1.1.2:0 up
      MPLS VC labels: local 21, remote 25
      Group ID: local 0, remote 0
      MTU: local 1500, remote 1500
      Remote interface description:
    Sequencing: receive disabled, send disabled
    VC statistics:
      packet totals: receive 1536, send 1537
      byte totals:   receive 572855, send 573230
      packet drops:  receive 0, send 0

  Local interface: Et0/0.3 up, line protocol up, Eth VLAN 300 up
    Destination address: 10.1.1.2, VC ID: 300, VC status: up
      Preferred path: 10.1.1.201, active
      Default path: ready
      Tunnel label: 23, next hop 10.23.12.2
      Output interface: Et1/0, imposed label stack {23 26}
    Create time: 1d01h, last status change time: 1d01h
    Signaling protocol: LDP, peer 10.1.1.2:0 up
      MPLS VC labels: local 25, remote 26
      Group ID: local 0, remote 0
      MTU: local 1500, remote 1500
      Remote interface description:
    Sequencing: receive disabled, send disabled
    VC statistics:
      packet totals: receive 1536, send 1538
      byte totals:   receive 572855, send 573605
      packet drops:  receive 0, send 0
```

At the end of Step 8, the pseudowire with VC ID 100 is going through the default path, and pseudowire 200 is going through the preferred path toward 10.1.1.200, with the output interface Serial3/0 connected to P1. Pseudowire 300 is going through the preferred path toward 10.1.1.201, with the output interface Ethernet1/0 connected to P2.

Example 9-16 is the complete configuration on PE1 for sending pseudowire traffic toward PE2 over preferred paths:

Example 9-16. *Configuration for Preferred Path Using IP Routing*

```
hostname PE1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
pseudowire-class PE1-P1-PE2
 encapsulation mpls
 preferred-path peer 10.1.1.200
!
pseudowire-class PE1-P2-P3-PE2
 encapsulation mpls
 preferred-path peer 10.1.1.201
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
!
```

```
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.2 100 encapsulation mpls
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 pw-class PE1-P1-PE2
!
interface Ethernet0/0.3
 encapsulation dot1Q 300
 xconnect 10.1.1.2 300 pw-class PE1-P2-P3-PE2
!
interface Ethernet1/0
 ip address 10.23.12.1 255.255.255.0
 mpls ip
!
interface Serial3/0
 ip address 10.23.11.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.1 0.0.0.0 area 0
 network 10.23.11.0 0.0.0.255 area 0
 network 10.23.12.0 0.0.0.255 area 0
!
ip route 10.1.1.200 255.255.255.255 10.23.11.2
ip route 10.1.1.201 255.255.255.255 10.23.12.2
```

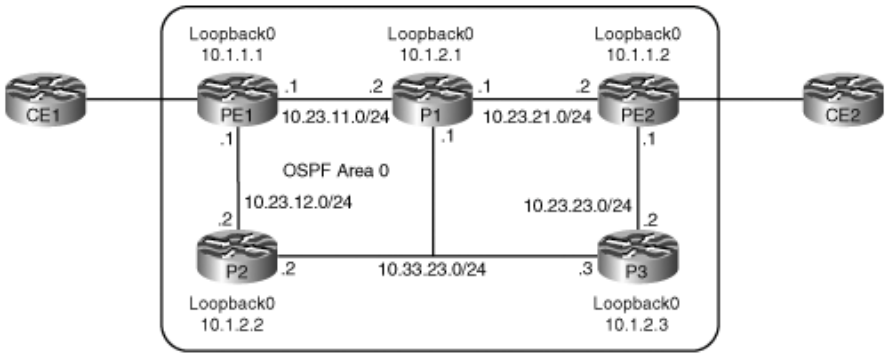
If PE2 needs to send pseudowire traffic to PE1 over the same preferred path, repeat Steps 1 through 8 with appropriate parameters on PE2.

Case Study 9-4: Configuring a Preferred Path Using MPLS Traffic Engineering Tunnels

In "Case Study 9-3: Configuring Preferred Path Using IP Routing," PE1 can select an output interface for a pseudowire using IP routing, but it cannot control how P1, P2, and P3 route the pseudowire traffic. For example, if P1 also has a link connecting to P2 and P3, as illustrated in Figure 9-3, using IP routing to select the preferred path on PE1 cannot guarantee the pseudowire traffic always flows from P1 to PE2. It is entirely possible that the traffic goes through P1 and P3 and then finally arrives at PE2.

Figure 9-3. ***Preferred Path with MPLS Traffic Engineering***

[View full size image]



The dynamic nature of IGP routing protocols makes it difficult to engineer explicit paths in a meshed network, and it is not always feasible to configure static routes on all the routers along the path. Explicit paths are useful when network operators know the traffic pattern of their networks and want to direct certain traffic through predetermined paths. It takes the guesswork out of predicting how the traffic traverses across the network. An MPLS traffic engineering tunnel with an explicit path option fulfills this objective precisely.

When real-time traffic is encapsulated inside pseudowires, pseudowire traffic must be able to reserve and maintain the bandwidth needed to guarantee the service quality at a reasonable level. When network operators care more about reducing jitter and congestion for real-time traffic than directing traffic through a predetermined path, an MPLS traffic engineering tunnel with dynamic path option is more appropriate; this means that as long as a path satisfies the specified bandwidth requirement, it is considered a feasible path.

The **preferred-path interface** *tunnel* command configures a preferred path using the MPLS traffic engineering tunnel interface, as shown in Example 9-17.

Example 9-17. *Configuring Preferred Path Using Traffic Engineering Tunnel*

```
PE1(config)#pseudowire-class PE1-P1-PE2
PE1(config-pw-class)#encapsulation mpls
PE1(config-pw-class)#preferred-path interface Tunnel1 ?
  disable-fallback disable fall back to alternative route
```

Note

The tunnel interface that is specified for the preferred path has to be an MPLS traffic engineering tunnel. It cannot be any other type of tunnel, such as GRE over IP. "Case Study 9-6: Configuring AToM Pseudowire over GRE Tunnel," describes how to configure AToM pseudowires over GRE tunnels.

The **disable-fallback** option works the same way as in the previous case study, which disables the use of the default forwarding path when the preferred path is unavailable.

Assume that the initial configuration is identical to Example 9-1. In the following configuration steps, two pseudowires take the preferred paths set up by MPLS traffic engineering tunnels on PE1. Pseudowire with VC ID 200 goes through the traffic engineering tunnel with an explicit path through P1 and PE2, and pseudowire 300 goes through the traffic engineering tunnel with 5-Mbps guaranteed bandwidth:

Step 1. Enable MPLS traffic engineering globally on PE1.

```
PE1(config)#mpls traffic-eng tunnels
```

Step 2. Configure MPLS-enabled interfaces to support RSVP traffic engineering signaling. For interface Ethernet1/0, reserve 8000-Kbps bandwidth. For interface Serial3/0, reserve 1200-Kbps bandwidth.

```
PE1(config)#interface Ethernet1/0
PE1(config-if)#mpls traffic-eng tunnels
PE1(config-if)#ip rsvp bandwidth 8000
PE1(config-if)#exit
PE1(config)#interface Serial3/0
PE1(config-if)#mpls traffic-eng tunnels
PE1(config-if)#ip rsvp bandwidth 1200
```

Step 3. Configure OSPF for MPLS traffic engineering.

```
PE1(config)#router ospf 1
PE1(config-router)#mpls traffic-eng router-id Loopback0
PE1(config-router)#mpls traffic-eng area 0
```

Step 4. Repeat Steps 1 through 3 on P1, P2, P3, and PE2, and in Step 2, substitute the interface and bandwidth parameters accordingly. Configure RSVP bandwidth reservation on all MPLS interfaces that the traffic engineering tunnels might traverse.

Step 5. On PE1, configure an MPLS traffic engineering tunnel with an explicit path through P1 and PE2, of which the addresses are 10.23.11.2 and 10.23.21.2, respectively. The bandwidth requirement for this traffic engineering tunnel is 1000 Kbps.

```
PE1(config)#ip explicit-path name P1-PE2 enable
PE1(cfg-ip-expl-path)#next-address 10.23.11.2
Explicit Path name P1-PE2:
    1: next-address 10.23.11.2
PE1(cfg-ip-expl-path)#next-address 10.23.21.2
Explicit Path name P1-PE2:
    1: next-address 10.23.11.2
    2: next-address 10.23.21.2
PE1(cfg-ip-expl-path)#exit
PE1(config)#interface Tunnel1
PE1(config-if)#ip unnumbered Loopback0
PE1(config-if)#tunnel destination 10.1.1.2
PE1(config-if)#tunnel mode mpls traffic-eng
PE1(config-if)#tunnel mpls traffic-eng priority 7 7
PE1(config-if)#tunnel mpls traffic-eng bandwidth  1000
PE1(config-if)#tunnel mpls traffic-eng path-option 1 explicit name P1-PE2
```

Step 6. Verify the status of the MPLS traffic engineering tunnel with the explicit path using the **show mpls traffic-eng tunnels** command.

```
PE1#show mpls traffic-eng tunnels Tunnel1

Name: PE1_t1                                (Tunnel1) Destination: 10.1.1.2
  Status:
    Admin: up         Oper: up    Path: valid        Signalling: connected

    path option 1, type explicit P1-PE2 (Basis for Setup, path weight 128)

  Config Parameters:
    Bandwidth: 1000    kbps (Global) Priority: 7 7    Affinity: 0x0/0xFFFF
    Metric Type: TE (default)
    AutoRoute: disabled   LockDown: disabled Loadshare: 1000 bw-based
    auto-bw: disabled
  Active Path Option Parameters:
    State: explicit path option 1 is active
    BandwidthOverride: disabled  LockDown: disabled Verbatim: disabled


  InLabel  : -
  OutLabel : Serial3/0, 16
  RSVP Signalling Info:
       Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 1, Tun_Instance 9
    RSVP Path Info:
      My Address: 10.1.1.1
      Explicit Route: 10.23.11.2 10.23.21.2 10.1.1.2
      Record   Route:   NONE
      Tspec: ave rate=1000 kbits, burst=1000 bytes, peak rate=1000 kbits
    RSVP Resv Info:
      Record  Route:    NONE
      Fspec: ave rate=1000 kbits, burst=1000 bytes, peak rate=1000 kbits
    Shortest Unconstrained Path Info:
      Path Weight: 30 (TE)
      Explicit Route: 10.23.12.1 10.23.12.2 10.33.23.2 10.33.23.3
                      10.23.23.2 10.23.23.1 10.1.1.2
    History:
      Tunnel:
        Time since created: 14 minutes, 20 seconds
        Time since path change: 11 minutes, 9 seconds
      Current LSP:
        Uptime: 11 minutes, 9 seconds
```

From the output of the **show mpls traffic-eng tunnels** command, the MPLS traffic tunnel uses the interface Serial3/0 as the output interface and takes the explicit path to P1 and PE2. The tunnel takes the shortest path through P2 and P3 if no path constraint is imposed on the tunnel.

Step 7. Verify that the intermediate router P1 sets up the traffic engineering tunnel properly.

```
P1#show mpls traffic-eng tunnels

LSP Tunnel PE1_t1 is signalled, connection is up
  InLabel  : Serial2/0, 16
```

```
  OutLabel : Serial3/0, implicit-null
  RSVP Signalling Info:
       Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 1, Tun_Instance 8
    RSVP Path Info:
      My Address: 10.23.11.2
      Explicit Route: 10.23.21.2 10.1.1.2
      Record   Route:   NONE
      Tspec: ave rate=1000 kbits, burst=1000 bytes, peak rate=1000 kbits
    RSVP Resv Info:
      Record   Route:   NONE
      Fspec: ave rate=1000 kbits, burst=1000 bytes, peak rate=1000 kbits
```

Notice that the incoming label for the tunnel is 16 on Serial2/0. This should be the tunnel imposition label that PE1 uses for the pseudowires that are going through this traffic engineering tunnel. The outgoing label is implicit-null on Serial3/0, which means that the interface is connected directly to the tunnel tailend PE2.

Step 8. On PE1, configure an MPLS traffic engineering tunnel with 5-Mbps guaranteed bandwidth.

```
PE1(config)#interface Tunnel2
PE1(config-if)#ip unnumbered Loopback0
PE1(config-if)#tunnel destination 10.1.1.2
PE1(config-if)#tunnel mode mpls traffic-eng
PE1(config-if)#tunnel mpls traffic-eng priority 7 7
PE1(config-if)#tunnel mpls traffic-eng bandwidth  5000
PE1(config-if)#tunnel mpls traffic-eng path-option 1 dynamic
```

Step 9. Verify the status of the MPLS traffic engineering tunnel by using the **show mpls traffic-eng tunnels** command.

```
PE1#show mpls traffic-eng tunnels Tunnel2

Name: PE1_t2                              (Tunnel2) Destination: 10.1.1.2
  Status:
    Admin: up         Oper: up     Path: valid       Signalling: connected

    path option 1, type dynamic (Basis for Setup, path weight 30)

  Config Parameters:
    Bandwidth: 5000    kbps (Global) Priority:  7 7   Affinity: 0x0/0xFFFF
    Metric Type: TE (default)
    AutoRoute:  disabled  LockDown: disabled  Loadshare: 5000     bw-based
    auto-bw: disabled
  Active Path Option Parameters:
    State: dynamic path option 1 is active
    BandwidthOverride: disabled  LockDown: disabled  Verbatim: disabled

  InLabel : -
  OutLabel : Ethernet1/0, 22
  RSVP Signalling Info:
       Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 2, Tun_Instance 16
    RSVP Path Info:
      My Address: 10.23.12.1
      Explicit Route: 10.23.12.2 10.33.23.2 10.33.23.3 10.23.23.2
```

```
                    10.23.23.1 10.1.1.2
      Record   Route:   NONE
      Tspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
    RSVP Resv Info:
      Record   Route:   NONE
      Fspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
  Shortest Unconstrained Path Info:
    Path Weight: 30 (TE)
    Explicit Route: 10.23.12.1 10.23.12.2 10.33.23.2 10.33.23.3
                    10.23.23.2 10.23.23.1 10.1.1.2
  History:
    Tunnel:
      Time since created: 9 minutes, 50 seconds
      Time since path change: 6 minutes, 27 seconds
    Current LSP:
      Uptime: 6 minutes, 27 seconds
```

Step 10. Verify that the intermediate routers P2 and P3 set up the traffic engineering tunnel properly.

```
P2#show mpls traffic-eng tunnels

LSP Tunnel PE1_t2 is signalled, connection is up
  InLabel  : Ethernet1/0, 22
  OutLabel : Ethernet0/0, 22
  RSVP Signalling Info:
       Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 2, Tun_Instance 16
    RSVP Path Info:
      My Address: 10.33.23.2
      Explicit Route: 10.33.23.3 10.23.23.2 10.23.23.1 10.1.1.2
      Record   Route:   NONE
      Tspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
    RSVP Resv Info:
      Record   Route:   NONE
      Fspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits

P3#show mpls traffic-eng tunnels

LSP Tunnel PE1_t2 is signalled, connection is up
  InLabel : Ethernet0/0, 22
  OutLabel : Ethernet1/0, implicit-null
  RSVP Signalling Info:
       Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 2, Tun_Instance 16
    RSVP Path Info:
      My Address: 10.23.23.2
      Explicit Route: 10.23.23.1 10.1.1.2
      Record   Route:   NONE
      Tspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
    RSVP Resv Info:
      Record   Route:   NONE
      Fspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
```

Notice that on P2, the incoming label for the tunnel is 22 on Ethernet1/0. This should be the tunnel imposition label that PE1 uses for the pseudowires that are going through this traffic engineering tunnel. On P3, the outgoing label is implicit-null on Ethernet1/0, which means that the interface is connected directly to the tunnel tailend PE2.

Step 11. Verify that the PE2 sets up both traffic engineering tunnels correctly as the tailend.

```
PE2#show mpls traffic-eng tunnels

LSP Tunnel PE1_t1 is signalled, connection is up
InLabel  : Serial3/0, implicit-null
OutLabel :  -
RSVP Signalling Info:
     Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 1, Tun_Instance 8
  RSVP Path Info:
    My Address: 10.1.1.2
    Explicit Route:  NONE
    Record   Route:   NONE
    Tspec: ave rate=1000 kbits, burst=1000 bytes, peak rate=1000 kbits
  RSVP Resv Info:
    Record   Route:   NONE
    Fspec: ave rate=1000 kbits, burst=1000 bytes, peak rate=1000 kbits

LSP Tunnel PE1_t2 is signalled, connection is up
  InLabel  : Ethernet1/0, implicit-null
  OutLabel :  -
  RSVP Signalling Info:
       Src 10.1.1.1, Dst 10.1.1.2, Tun_Id 2, Tun_Instance 16
    RSVP Path Info:
      My Address: 10.1.1.2
      Explicit Route:   NONE
      Record   Route:    NONE
      Tspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
    RSVP Resv Info:
      Record   Route:    NONE
      Fspec: ave rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
```

Step 12. Configure a pseudowire class with a preferred path going through the explicit path.

```
PE1(config)#pseudowire-class PE1-P1-PE2
PE1(config-pw-class)#encapsulation mpls
PE1(config-pw-class)#preferred-path interface Tunnel1 disable-fallback
```

Notice that the **disable-fallback** option is enabled to prevent the traffic from taking the default route when the traffic engineering tunnel becomes unavailable.

Step 13. Configure a pseudowire class that prefers a high-bandwidth path.

```
PE1(config-pw-class)#pseudowire-class High_Bandwidth
PE1(config-pw-class)#encapsulation mpls
PE1(config-pw-class)#preferred-path interface Tunnel2
```

Because the disable-fallback option is not present, the traffic takes the default route when the high-bandwidth traffic engineering tunnel becomes unavailable.

Step 14. Provision the pseudowire of VC ID 200 with the explicit path and pseudowire 300 with the high-bandwidth path.

```
PE1(config)#interface Ethernet0/0.2
PE1(config-subif)#xconnect 10.1.1.2 200 pw-class PE1-P1-PE2
PE1(config-subif)#exit
PE1(config)#interface Ethernet0/0.3
PE1(config-subif)#xconnect 10.1.1.2 300 pw-class High_Bandwidth
```

Step 15. Verify that pseudowires are active and taking the specified path by using the **show mpls l2transport vc detail** command.

```
PE1#show  mpls l2transport vc detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 100 up
  Destination address: 10.1.1.2, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 23, next hop 10.23.12.2
    Output interface: Et1/0, imposed label stack {23 23}
  Create time: 00:33:08, last status change time: 00:32:40
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 16, remote 23
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 33, send 32
    byte totals:   receive 12375, send 12000
    packet drops:  receive 0, send 0

Local interface: Et0/0.2 up, line protocol up, Eth VLAN 200 up
  Destination address: 10.1.1.2, VC ID: 200, VC status: up
    Preferred path: Tunnel1, active
    Default path: disabled
    Tunnel label: 3, next hop point2point
    Output interface: Tu1, imposed label stack {16 24}
  Create time: 00:33:15, last status change time: 00:32:47
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 17, remote 24
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 32, send 30
    byte totals:   receive 12000, send 11250
    packet drops:  receive 0, send 3

Local interface: Et0/0.3 up, line protocol up, Eth VLAN 300 up
  Destination address: 10.1.1.2, VC ID: 300, VC status: up
    Preferred path: Tunnel2,  active
    Default path: ready
    Tunnel label: 3, next hop point2point
    Output interface: Tu2, imposed label stack {22 25}
  Create time: 00:33:15, last status change time: 00:32:47
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 18, remote 25
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
```

```
Sequencing: receive disabled, send disabled
VC statistics:
  packet totals: receive 32, send 32
  byte totals:   receive 12000, send 12000
  packet drops:  receive 0, send 0
```

Note that the default path for pseudowires 100 and 300 is established through LDP, not Resource Reservation Protocol (RSVP) Traffic Engineering. The default path is disabled for pseudowire 200 by using the **disable-fallback** option.

The traffic engineering tunnel labels for pseudowires 200 and 300 are 16 and 22, respectively, which match the traffic engineering labels that P1 and P2 assign. Because the traffic engineering tunnels are from PE1 to PE2, the forwarding process on PE1 perceives PE2 as if it were directly connected through the tunnel interfaces. Therefore, the tunnel label fields have a label value of 3, which is the implicit-null label.

Upon completion of these steps, the network has two MPLS traffic engineering tunnels established from PE1 to PE2. Traffic engineering tunnels are always unidirectional. If you want the same forwarding properties for pseudowire traffic from PE2 to PE1, PE2 needs to configure its own traffic engineering tunnels toward PE1 by repeating Steps 5 through 15 with appropriate parameters.

Caution

When you are using an MPLS traffic engineering tunnel as a preferred path for pseudowires, you need to make sure that the tunnel endpoints (headend and tailend) are on the PE routers that provision these pseudowires.

After you complete these steps, the configuration on PE1 is shown in Example 9-18.

Example 9-18. *Configuration for Preferred Path Using MPLS Traffic Engineering Tunnel*

```
hostname PE1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
mpls traffic-eng tunnels
pseudowire-class PE1-P1-PE2
 encapsulation mpls
 preferred-path interface Tunnel1 disable-fallback
!
pseudowire-class High_Bandwidth
 encapsulation mpls
 preferred-path interface Tunnel2
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
!
interface Tunnel1
```

```
 ip unnumbered Loopback0
 tunnel destination 10.1.1.2
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng priority 7 7
 tunnel mpls traffic-eng bandwidth  1000
 tunnel mpls traffic-eng path-option 1 explicit name P1-PE2
!
interface Tunnel2
 ip unnumbered Loopback0
 tunnel destination 10.1.1.2
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng priority 7 7
 tunnel mpls traffic-eng bandwidth  5000
 tunnel mpls traffic-eng path-option 1 dynamic
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.2 100 encapsulation mpls
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 pw-class PE1-P1-PE2
!
interface Ethernet0/0.3
 encapsulation dot1Q 300
 xconnect 10.1.1.2 300 pw-class High_Bandwidth
!
interface Ethernet1/0
 ip address 10.23.12.1 255.255.255.0
 mpls ip
 mpls traffic-eng tunnels
 ip rsvp bandwidth 8000
!
interface Serial3/0
 ip address 10.23.11.1 255.255.255.0
 mpls ip
 mpls traffic-eng tunnels
 ip rsvp bandwidth 1200
!
router ospf 1
 mpls traffic-eng router-id Loopback0
 mpls traffic-eng area 0
 network 10.1.1.1 0.0.0.0 area 0
 network 10.23.11.0 0.0.0.255 area 0
 network 10.23.12.0 0.0.0.255 area 0
!
ip explicit-path name P1-PE2 enable
 next-address 10.23.11.2
 next-address 10.23.21.2
```

Besides using the **preferred-path interface** command, you can also direct pseudowire traffic to an MPLS traffic engineering tunnel by using the **preferred-path peer** command. The net effect is similar to using IP routing for the preferred path. The only difference is that the specified /32 host route has a traffic engineering tunnel interface as the output interface instead of a physical interface in the forwarding table.

When the traffic engineering tunnel is configured with the **autoroute** option, IGP can learn the host route through the traffic engineering tunnel interface. As a result, IGP forwards both IP and pseudowire packets through the traffic engineering tunnel for the routing prefixes it learns through the tunnel. To enable the autoroute option, configure the **tunnel mpls traffic-eng autoroute announce** command under the tunnel interface configuration mode.

Case Study 9-5: Protecting AToM Pseudowires with MPLS Traffic Engineering Fast Reroute

MPLS traffic engineering automatically establishes and maintains LSPs across the MPLS core network using RSVP. Such LSPs are created based on the resource constraints that are configured and available network resources, such as bandwidth. IGP routing protocols such as IS-IS or OSPF announce available network resources using traffic engineering protocol extensions along with link state advertisements throughout the network.

In any network, links, routers, or both can fail because of unexpected events. Network operators include this factor their network planning by having redundant links and routers at the physical or logical locations where the failures are most likely to happen. When such failure conditions occur, routers within the network might temporarily have inconsistent routing information. They might need to exchange routing updates and come up with a new, consistent view of the network. This process is known as *network convergence*. During network convergence, routing loops and black holes can cause packet loss. The longer the convergence takes, the larger the amount of packet loss.

The convergence time includes the amount of time for an adjacent router to detect the link (or router) failure. It also includes the amount of time for this router to distribute the information to all other routers and for all other routers to recalculate routes in the forwarding tables. Detecting a link failure requires physical and link layerspecific mechanisms. MPLS traffic engineering does not have a way to reduce the amount of time to detect failures. However, it can reduce the time required to distribute the failure information and update the forwarding tables by using MPLS traffic engineering fast rerouting capability.

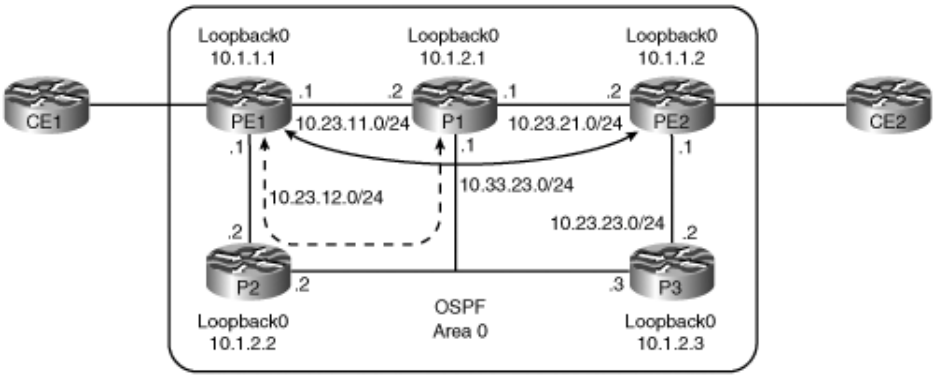
Prior to a failure, fast reroute calculates and establishes a protection traffic engineering tunnel around the link or node that is deemed vulnerable. Upon detecting such a failure, the backup tunnel takes over packet forwarding immediately. Rerouting typically takes less than 50 ms upon failure detection, and packet loss is kept minimal.

Before you enable fast reroute for an AToM pseudowire, you need to configure an MPLS traffic engineering tunnel as the preferred path, as shown in the previous case study. Then at the ingress PE where the traffic engineering tunnel headend is, you can use fast reroute options to configure a backup traffic engineering tunnel to protect the primary traffic engineering tunnel.

In Figure 9-4, a pseudowire takes the explicit path from PE1 to PE2 through P1. Suppose that the link between PE1 and P1 is considered vulnerable. PE1 provisions a fast reroute traffic engineering tunnel through P2 and P1 to circumvent the possible failing link. To configure the primary traffic engineering tunnel with the explicit path, refer to "Case Study 9-4: Configuring a Preferred Path Using MPLS Traffic Engineering Tunnels."

Figure 9-4. *Protect AToM Pseudowire with Fast Reroute*

[View full size image]



Assume that the pseudowire has been provisioned with a preferred path that uses MPLS traffic engineering's explicit path, as shown in Case Study 9-4. The following steps describe how to enable fast reroute on the primary traffic engineering tunnel.

Step 1. Add an explicit path on PE1 that originates from the PE, traverses through P2, and ends at P1.

```
PE1(config)#ip explicit-path name P2-P1 enable
PE1(cfg-ip-expl-path)#next-address 10.23.12.2
Explicit Path name P2-P1:
    1: next-address 10.23.12.2
PE1(cfg-ip-expl-path)#next-address 10.33.23.1
Explicit Path name P2-P1:
    1: next-address 10.23.12.2
    2: next-address 10.33.23.1
```

Step 2. Provision a backup traffic engineering tunnel with the explicit path configured in Step 1. Note that the tailend of this backup tunnel is P1, and its IP address is 10.1.2.1.

```
PE1(config)#interface Tunnel100
PE1(config-if)#ip unnumbered Loopback0
PE1(config-if)#tunnel destination 10.1.2.1
PE1(config-if)#tunnel mode mpls traffic-eng
PE1(config-if)#tunnel mpls traffic-eng priority 7 7
PE1(config-if)#tunnel mpls traffic-eng bandwidth  1000
PE1(config-if)#tunnel mpls traffic-eng path-option 1 explicit name P2-P1
```

Step 3. Configure the primary traffic engineering tunnel with fast reroute protection. The initial tunnel interface configuration is as follows:

```
PE1#show running-config interface Tunnel1
Building configuration...

Current configuration : 274 bytes
!
```

```
interface Tunnel1
 ip unnumbered Loopback0
 no ip directed-broadcast
 tunnel destination 10.1.1.2
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng priority 7 7
 tunnel mpls traffic-eng bandwidth  1000
 tunnel mpls traffic-eng path-option 1 explicit name P1-PE2
end

PE1#config t
Enter configuration commands, one per line. End with CNTL/Z.
PE1(config)#interface Tunnel1
PE1(config-if)#tunnel mpls traffic-eng fast-reroute
```

Step 4. Configure the protected link to use the backup tunnel. The interface that connects to the protected link on PE1 is Serial3/0.

```
PE1(config)#interface Serial3/0
PE1(config-if)#mpls traffic-eng backup-path Tunnel100
```

Step 5. Verify that the primary tunnel is protected by fast reroute and the backup tunnel is ready under normal conditions. Use the **show mpls traffic-eng tunnels protection** and **show mpls interfaces** commands.

```
PE1#show mpls traffic-eng tunnels protection
PE1_t1
  LSP Head, Tunnel1, Admin: up, Oper: up
  Src 10.1.1.1, Dest 10.1.1.2, Instance 31
  Fast Reroute Protection: Requested
    Outbound: FRR Ready
      Backup Tu100 to LSP nhop
        Tu100: out i/f: Et1/0, label: 16
      LSP signalling info:
        Original: out i/f: Se3/0, label: 16, nhop: 10.23.11.2
        With FRR: out i/f: Tu100, label: 16
      LSP bw: 1000 kbps, Backup level: any-unlim, type: any pool
PE1_t2
  LSP Head, Tunnel2, Admin: up, Oper: up
  Src 10.1.1.1, Dest 10.1.1.2, Instance 18
  Fast Reroute Protection: None
PE1_t100
  LSP Head, Tunnel100, Admin: up, Oper: up
  Src 10.1.1.1, Dest 10.1.2.1, Instance 18
  Fast Reroute Protection: None

PE1#show mpls interfaces Tunnel1 detail
Interface Tunnel1:
        MPLS TE Tunnel Head
        IP labeling not enabled
        LSP Tunnel labeling not enabled
        BGP labeling not enabled
        MPLS not operational
```

```
        Fast Switching Vectors:
          IP to MPLS Fast Switching Vector
          MPLS Disabled
        MTU = 1496
Tun hd Untagged    0          Tu1        point2point
        MAC/Encaps=4/8, MRU=1500, Tag Stack{16}, via Se3/0
        0F008847 00010000
        No output feature configured
        Fast Reroute Protection via {Tu100, outgoing label 16}
```

Notice that the fast reroute status for the primary tunnel is ready. This means that the backup tunnel is operational and ready to protect the primary tunnel.

Step 6. Verify the status of AToM pseudowire with VC ID 200, which traverses the primary tunnel under normal conditions. Label 16 is the traffic engineering tunnel label.

```
PE1#show mpls l2transport vc 200 detail
Local interface: Et0/0.2 up, line protocol up, Eth VLAN 200 up
  Destination address: 10.1.1.2, VC ID: 200, VC status: up
    Preferred path: Tunnel1,  active
    Default path: disabled
    Tunnel label: 3, next hop point2point
    Output interface: Tu1, imposed label stack {16 24}
  Create time: 01:14:59, last status change time: 01:11:17
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 17, remote 24
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 101, send 101
    byte totals:   receive 31270, send 29960
    packet drops:  receive 0, send 5
```

Step 7. To verify the effectiveness of the fast reroute capability, introduce a link failure and use the **show mpls traffic-eng tunnels protection** and **show mpls l2transport vc** commands to examine the fast reroute status and pseudowire information.

```
PE1#show mpls traffic-eng tunnels protection
PE1_t1
  LSP Head, Tunnel1, Admin: up, Oper: up
  Src 10.1.1.1, Dest 10.1.1.2, Instance 124
  Fast Reroute Protection: Requested
    Outbound: FRR Active
      Backup Tu100 to LSP nhop
        Tu100: out i/f: Et1/0, label: 16
      LSP signalling info:
        Original: out i/f: Se3/0, label: 16, nhop: 10.1.2.1
        With FRR: out i/f: Tu100, label: 16
      LSP bw: 1000 kbps, Backup level: any-unlim, type: any pool
PE1_t2
  LSP Head, Tunnel2, Admin: up, Oper: up
```

```
    Src 10.1.1.1, Dest 10.1.1.2, Instance 18
    Fast Reroute Protection: None
  PE1_t100
    LSP Head, Tunnel100, Admin: up, Oper: up
    Src 10.1.1.1, Dest 10.1.2.1, Instance 19
    Fast Reroute Protection: None

PE1#show mpls l2transport vc 200 detail
Local interface: Et0/0.2 up, line protocol up, Eth VLAN 200 up
  Destination address: 10.1.1.2, VC ID: 200, VC status: up
    Preferred path: Tunnel1,  active
    Default path: disabled
    Tunnel label: 16, next hop point2point
    Output interface: Tu100, imposed label stack {16 16 24}
  Create time: 01:17:49, last status change time: 01:14:07
  Signaling protocol: LDP, peer 10.1.1.2:0 up
    MPLS VC labels: local 17, remote 24
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 111, send 114
    byte totals:   receive 33316, send 32384
    packet drops:  receive 0, send 5
```

Notice that the fast reroute status has changed from ready to active. The output interface for the pseudowire has switched from Tunnel1 to Tunnel100, and the label stack has become {16 16 24}. The top label 16 is the backup tunnel label so that pseudowire packets can be forwarded to the tailend router P1 through the backup traffic engineering tunnel. The second label 16 is the primary tunnel label that P1 assigns. The last label 24 is the VC label for the pseudowire.

The configuration on PE1 after finishing these steps is shown in Example 9-19.

Example 9-19. *Configuration for MPLS Fast RerouteProtected Pseudowire*

```
hostname PE1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
mpls traffic-eng tunnels
pseudowire-class PE1-P1-PE2
 encapsulation mpls
 preferred-path interface Tunnel1 disable-fallback
!
pseudowire-class High_Bandwidth
 encapsulation mpls
 preferred-path interface Tunnel2
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
```

```
!
interface Tunnel1
 ip unnumbered Loopback0
 tunnel destination 10.1.1.2
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng priority 7 7
 tunnel mpls traffic-eng bandwidth  1000
 tunnel mpls traffic-eng path-option 1 explicit name P1-PE2
 tunnel mpls traffic-eng fast-reroute
!
interface Tunnel2
 ip unnumbered Loopback0
 tunnel destination 10.1.1.2
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng priority 7 7
 tunnel mpls traffic-eng bandwidth  5000
 tunnel mpls traffic-eng path-option 1 dynamic
!
interface Tunnel100
 ip unnumbered Loopback0
 no ip directed-broadcast
 tunnel destination 10.1.2.1
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng priority 7 7
 tunnel mpls traffic-eng bandwidth  1000
 tunnel mpls traffic-eng path-option 1 explicit name P2-P1
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.2 100 encapsulation mpls
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 pw-class PE1-P1-PE2
!
interface Ethernet0/0.3
 encapsulation dot1Q 300
 xconnect 10.1.1.2 300 pw-class High_Bandwidth
!
interface Ethernet1/0
 ip address 10.23.12.1 255.255.255.0
 mpls ip
 mpls traffic-eng tunnels
 ip rsvp bandwidth 8000
!
interface Serial3/0
 ip address 10.23.11.1 255.255.255.0
 mpls ip
 mpls traffic-eng tunnels
 mpls traffic-eng backup-path Tunnel100
 ip rsvp bandwidth 1200
!
router ospf 1
 mpls traffic-eng router-id Loopback0
```

```
 mpls traffic-eng area 0
 network 10.1.1.1 0.0.0.0 area 0
 network 10.23.11.0 0.0.0.255 area 0
 network 10.23.12.0 0.0.0.255 area 0
!
ip explicit-path name P1-PE2 enable
 next-address 10.23.11.2
 next-address 10.23.21.2
!
ip explicit-path name P2-P1 enable
 next-address 10.23.12.2
 next-address 10.33.23.1
```

Case Study 9-6: Configuring AToM Pseudowire over GRE Tunnel

Typically, when AToM pseudowire packets traverse an MPLS network, they carry label stacks that have more than one label. As described in Chapter 3, "Layer 2 VPN Architectures," each label represents an LSP. The top label is responsible for delivering pseudowire packets from one PE router to another through a tunnel LSP; therefore, it is known as the *tunnel label*. The tunnel label serves as an encapsulation header for the rest of the packet, which has little dependency on the tunnel label. Analogically, the relationship is somewhat like the IP header of an IP packet to the payload it carries. When an IP header or GRE/IP header replaces the tunnel label, the tunnel label has little impact on the pseudowire emulation functionality.

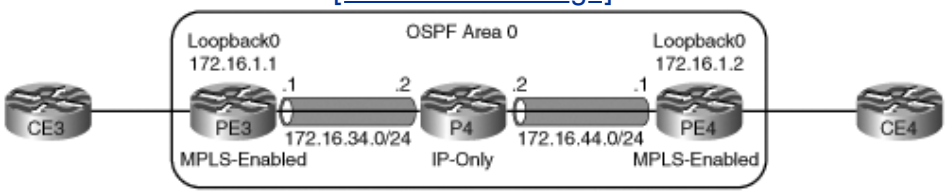
This case study explores the deployment model of transporting AToM pseudowire packets over GRE tunnels. Although this model enables you to deploy AToM pseudowires in any IP or MPLS network, they are most advantageous and efficient in networks that do not have MPLS forwarding. For example, the pseudowire endpoints are located in MPLS edge routers with a plain IP core network or two separate MPLS networks connected by a transit network with plain IP forwarding. With pseudowire emulation in MPLS networks, you should choose the native MPLS tunnel label to reduce encapsulation overhead and leverage advanced features, such as MPLS traffic engineering and fast reroute.

Forwarding pseudowire traffic over a GRE tunnel is quite similar to that over an MPLS traffic engineering tunnel with the autoroute option, where both IP and pseudowire packets can go through the same tunnel. You cannot use the **preferred-path interface** command with a GRE tunnel interface.

As illustrated in Figure 9-5, the PE routers are enabled with MPLS services, but the core network runs plain IP forwarding only.

Figure 9-5. ***AToM Pseudowire over GRE Tunnel***

[View full size image]



The following steps configure an AToM pseudowire to traverse the IP network through a GRE tunnel:

Step 1. Enable MPLS forwarding and set the MPLS label protocol to LDP in the global configuration mode on PE3.

```
PE3(config)#ip cef
PE3(config)#mpls ip
PE3(config)#mpls label protocol ldp
```

Step 2. Create a loopback interface with a host IP address.

```
PE3(config)#interface Loopback0
PE3(config-if)#ip address 172.16.1.1 255.255.255.255
```

Step 3. Configure a GRE tunnel interface on PE3, and set the tunnel source and destination addresses to be a routable address on PE3 and PE4, respectively.

```
PE3(config-if)#interface Tunnel1
PE3(config-if)#ip unnumbered Loopback0
PE3(config-if)#tunnel source Serial2/0
PE3(config-if)#tunnel destination 172.16.44.1
```

Step 4. To avoid recursive routing loops, make sure the tunnel destination address does not use the tunnel interface as the outgoing interface. It can accomplish this by using static route or dynamic routing protocols. Here, OSPF runs on the core facing network interface.

```
PE3(config)#router ospf 1
PE3(config-router)#network 172.16.34.0 0.0.0.255 area 0
```

Step 5. Enable MPLS forwarding on the GRE tunnel interface. This step is necessary so that MPLS applications see the tunnel interface as a feasible outgoing interface for MPLS traffic.

```
PE3(config-if)#interface Tunnel1
PE3(config-if)#mpls ip
```

Step 6. Add a static route to redirect pseudowire traffic into the tunnel interface.

```
PE3(config)#ip route 172.16.1.2 255.255.255.255 Tunnel1
```

Step 7. Provision an AToM pseudowire on the CE-facing interface.

```
PE3(config)#interface Ethernet0/0.1
PE3(config-subif)#encapsulation dot1Q 100
PE3(config-subif)#xconnect 172.16.1.2 100 encapsulation mpls
```

Step 8. Repeat Steps 1 through 7 on PE4 with appropriate parameters.

Step Verify the tunnel interface status and encapsulation using the **show interface** and

9. **show adjacency** commands.

```
PE3#show interface Tunnel1
Tunnel1 is up, line protocol is up
  Hardware is Tunnel
  Interface is unnumbered. Using address of Loopback0 (172.16.1.1)
  MTU 1514 bytes, BW 9 Kbit, DLY 500000 usec, rely 255/255, load 1/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 172.16.34.1 (Serial2/0), destination 172.16.44.1
  Tunnel protocol/transport GRE/IP, sequencing disabled
  Tunnel TTL 255
  Key disabled
  Checksumming of packets disabled, fast tunneling enabled
  Last input 00:00:00, output 00:00:00, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     25314 packets input, 2578630 bytes, 0 no buffer
     Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     26389 packets output, 2905870 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 output buffer failures, 0 output buffers swapped out

PE3#show adjacency Tunnel1 detail
Protocol  Interface                 Address
TAG       Tunnel1                   point2point(5)
                                    2148 packets, 856752 bytes
                                    4500000000000000FF2F15ACAC102201
                                    AC102C0100008847
                                    TFIB       never
                                    Epoch: 0
IP        Tunnel1                   point2point(7)
                                    0 packets, 0 bytes
                                    4500000000000000FF2F15ACAC102201
                                    AC102C0100000800
                                    CEF   expires: 00:02:16
                                          refresh: 00:00:16
                                    Epoch: 0
```

The output of the show adjacency command contains two adjacency entries. The tag adjacency is for MPLS traffic, such as the AToM pseudowire traffic, and the IP adjacency is for IP traffic. Notice that the GRE tunnel encapsulation for switching MPLS traffic consists of an IP header and a GRE header. The IP header contains an IP protocol type 47 (0x2F) indicating a payload GRE packet, and the tunnel source and destination addresses are in hex format (0xAC102201, 0xAC102C01). The GRE header has a protocol type 0x8847 for MPLS unicast traffic.

Step 1. Verify the pseudowire status by using the **show mpls l2transport vc detail** command.

```
PE3#show mpls l2transport vc detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 100 up
  Destination address: 172.16.1.2, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: imp-null, next hop point2point
    Output interface: Tu1, imposed label stack {16}
  Create time: 17:47:08, last status change time: 17:46:36
  Signaling protocol: LDP, peer 172.16.1.2:0 up
    MPLS VC labels: local 16, remote 16
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 1070, send 1070
    byte totals:   receive 398956, send 398956
    packet drops:  receive 0, send 0
```

Notice that the output interface of the pseudowire is the GRE tunnel interface. The tunnel label is the implicit null label, as if the two PE routers are connected directly.

After you complete these steps, view the configuration on PE3, as shown in Example 9-20.

Example 9-20. *PE3 Configuration for AToM Pseudowire over GRE Tunnel*

```
hostname PE3
!
ip cef
mpls label protocol ldp
!
interface Loopback0
 ip address 172.16.1.1 255.255.255.255
!
interface Tunnel1
 ip unnumbered Loopback0
 mpls ip
 tunnel source Serial2/0
 tunnel destination 172.16.44.1
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 172.16.1.2 100 encapsulation mpls
!
interface Serial2/0
 ip address 172.16.34.1 255.255.255.0
!
```

```
router ospf 1
 network 172.16.34.0 0.0.0.255 area 0
!
ip route 172.16.1.2 255.255.255.255 Tunnel1
```

The PE4 configuration is shown in Example 9-21.

Example 9-21. *PE4 Configuration for AToM Pseudowire over GRE Tunnel*

```
hostname PE4
!
ip cef
mpls label protocol ldp
!
interface Loopback0
 ip address 172.16.1.2 255.255.255.255
!
interface Tunnel1
 ip unnumbered Loopback0
 mpls ip
 tunnel source Serial2/0
 tunnel destination 172.16.34.1
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 172.16.1.1 100 encapsulation mpls
!
interface Serial2/0
 ip address 172.16.44.1 255.255.255.0
!
router ospf 1
 network 172.16.44.0 0.0.0.255 area 0
!
ip route 172.16.1.1 255.255.255.255 Tunnel1
```

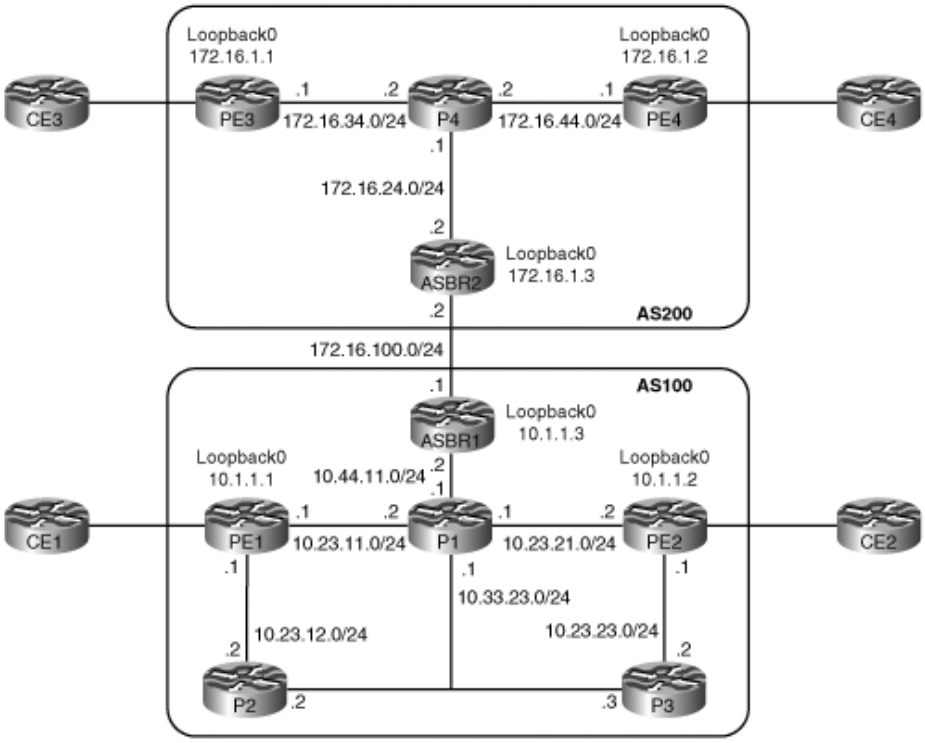
Pseudowire Emulation in Multi-AS Networks

When two attachment circuits (ACs) should be connected by a pseudowire but the attached PE routers are in different autonomous systems, the PE routers cannot exchange IGP routes with each other. In multi-AS networks, autonomous system border routers (ASBRs) establish Border Gateway Protocol (BGP) connections with each other to exchange routes between different autonomous systems.

Figure 9-6 illustrates an example of connecting ACs across the autonomous system boundary. Suppose that the AC of Ethernet VLAN 100 on CE1 in AS100 needs to be connected to the one on CE4 in AS200, and the AC of Ethernet VLAN 200 on CE2 in AS100 needs to be connected to the one on CE3 in AS200. The following case studies present three different solutions to accomplish the goal. Each solution has its own merits and applicable deployment scenarios.

Figure 9-6. ***Pseudowire Emulation in Multi-AS Networks***

[View full size image]



Case Study 9-7: Interconnecting Pseudowires with Dedicated Circuits

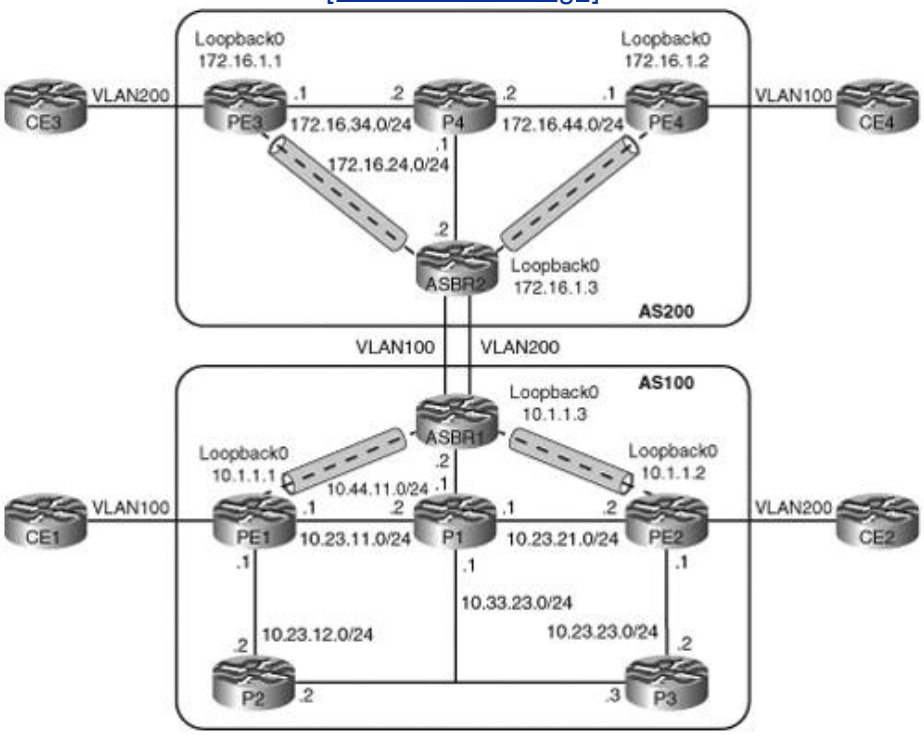
The configuration and routing that are necessary for forwarding IP traffic in a multi-AS environment are complex, and adding pseudowire emulation services does not make network operation much easier. However, the solution discussed in this case study takes a simplistic approach.

Instead of building end-to-end pseudowires across the autonomous system boundary, each ASBR acts as a PE router and provides pseudowire emulation services. Essentially, each ASBR treats the peering ASBR in a different domain as a CE router and the links between the ASBRs as ACs. In the pseudowire emulation architecture, the connectivity between CE and PE devices is at Layer 2. In theory, Layer 3 connectivity is not required between ASBRs if only pseudowire emulation services are required. However, ASBRs typically provide interdomain routing services, too, so Layer 3 connectivity is configured in the example. MPLS forwarding is not required between ASBRs in this deployment model.

As shown in Figure 9-7, a pseudowire with VC ID 100 is provisioned between PE1 and ASBR1 in AS100. A pseudowire with VC ID 100 is also provisioned between PE4 and ASBR2 in AS200. To have end-to-end connectivity between CE1 and CE4, ASBR1 and ASBR2 allocate a dedicated Ethernet VLAN 100 and use it as the common AC for both pseudowires. It is not mandatory for both pseudowires to have the same VC ID, but it is a good self-documenting practice. Similarly, CE2 and CE3 are connected by concatenating a pseudowire to a dedicated Ethernet VLAN 200 and then to another pseudowire. BGP is configured between ASBRs to exchange interdomain routing information, but it is not essential for inter-AS pseudowire emulation in this particular case.

Figure 9-7. ***Inter-AS Pseudowire Emulation with Dedicated Circuits***

[View full size image]



The following configuration examples give you some ideas of how to configure the PE and ASBR routers to use dedicated circuits between autonomous systems to provide inter-AS pseudowire connectivity.

On PE1, configure the pseudowire with VC ID 100 on the AC that connects to CE1, as shown in Example 9-22.

Example 9-22. *PE1 Pseudowire Configuration*

```
hostname PE1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.3 100 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.23.12.1 255.255.255.0
 mpls ip
!
interface Serial3/0
 ip address 10.23.11.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.1 0.0.0.0 area 0
 network 10.23.11.0 0.0.0.255 area 0
 network 10.23.12.0 0.0.0.255 area 0
```

On PE2, configure the pseudowire with VC ID 200 on the AC that connects to CE2, as shown in Example 9-23.

Example 9-23. *PE2 Pseudowire Configuration*

```
hostname PE2
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.2 255.255.255.255
!
```

```
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.3 200 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.23.23.1 255.255.255.0
 mpls ip
!
interface Serial3/0
 ip address 10.23.21.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.2 0.0.0.0 area 0
 network 10.23.21.0 0.0.0.255 area 0
 network 10.23.23.0 0.0.0.255 area 0
```

On PE3, configure the pseudowire with VC ID 200 on the AC that connects to CE3, as shown in Example 9-24.

Example 9-24. *PE3 Pseudowire Configuration*

```
hostname PE3
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.1 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 172.16.1.3 200 encapsulation mpls
!
interface Serial2/0
 ip address 172.16.34.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 172.16.1.1 0.0.0.0 area 0
 network 172.16.34.0 0.0.0.255 area 0
```

On PE4, configure the pseudowire with VC ID 100 on the AC that connects to CE4, as shown in Example 9-25.

Example 9-25. *PE4 Pseudowire Configuration*

```
hostname PE4
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.2 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 172.16.1.3 100 encapsulation mpls
!
interface Serial2/0
 ip address 172.16.44.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 172.16.1.2 0.0.0.0 area 0
 network 172.16.44.0 0.0.0.255 area 0
```

When you are using dedicated circuits between ASBRs, ensure that the encapsulation of these circuits matches that of the ACs between CE and PE routers, because the ASBRs effectively act as PE routers. In this example, the connection between ASBR1 and ASBR2 is Ethernet.

On ASBR1, configure the pseudowire with VC ID 100 and the pseudowire with VC ID 200 on the corresponding dedicated circuits, as shown in Example 9-26.

Example 9-26. *ASBR1 Pseudowire Configuration*

```
hostname ASBR1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.3 255.255.255.255
!
interface Ethernet0/0
 description Connect to ASBR2 in AS200
 ip address 172.16.100.1 255.255.255.0
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.1 100 encapsulation mpls
!
```

```
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.43.11.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.3 0.0.0.0 area 0
 network 10.43.11.0 0.0.0.255 area 0
!
router bgp 100
 no synchronization
 neighbor 172.16.100.2 remote-as 200
 no auto-summary
```

On ASBR2, configure the pseudowire with VC ID 100 and the pseudowire with VC ID 200 on the corresponding dedicated circuits, as shown in Example 9-27.

Example 9-27. *ASBR2 Pseudowire Configuration*

```
hostname ASBR2
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.3 255.255.255.255
!
interface Ethernet0/0
 description Connect to ASBR1 in AS100
 ip address 172.16.100.2 255.255.255.0
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 172.16.1.2 100 encapsulation mpls
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 172.16.1.1 200 encapsulation mpls
!
interface Ethernet1/0
 ip address 172.16.24.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 172.16.1.3 0.0.0.0 area 0
 network 172.16.24.0 0.0.0.255 area 0
!
router bgp 200
 no synchronization
```

```
neighbor 172.16.100.1 remote-as 100
no auto-summary
```

When considering this deployment model for inter-AS pseudowire emulation services, you need to evaluate the following restrictions that are associated with it:

- To be interconnected, the Layer 2 encapsulation of the links between ASBRs must be identical to that of the ACs.
- The number of dedicated circuits or virtual circuits between ASBRs can be limited. For example, ASBR1 and ASBR2 are connected through an Ethernet connection that can support up to 4096 802.1q VLANs, which is 4096 dedicated circuits at most.

In the future, when you can replace dedicated circuits with pseudowires for inter-AS pseudowire emulation services, these restrictions should be eliminated.

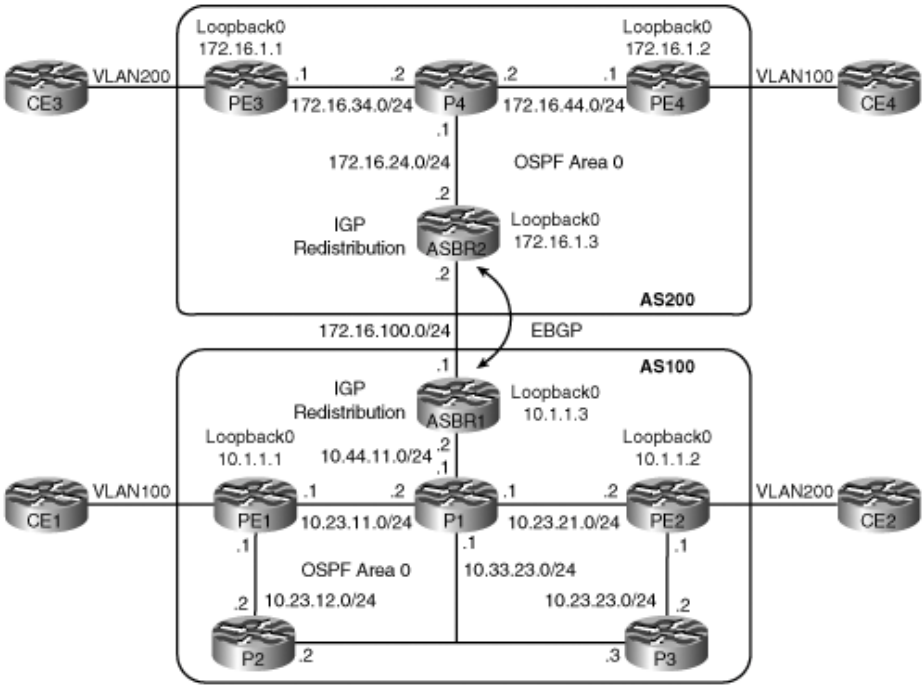
> **Note**
>
> A new pseudowire emulation solution is being developed at press time to replace the dedicated circuits between ASBRs with pseudowires. In other words, disjointed pseudowires of different autonomous systems can be interconnected through another set of pseudowires that is established between ASBRs. You can imagine that the end-to-end connectivity is provided by "stitching" several pseudowires together. By replacing dedicated circuits with pseudowires, pseudowire emulation in multi-AS networks achieves better flexibility and scalability.

Case Study 9-8: BGP IPv4 Label Distribution with IGP Redistribution

To provide edge-to-edge network connectivity for pseudowires within a single autonomous system, you need to have the /32 host routes of PE routers and the corresponding labels, which you learn through IGP and LDP (or RSVP if you are using MPLS traffic engineering). One solution for obtaining the same level of connectivity in a multi-AS environment is by using external BGP (eBGP) to exchange the /32 host routes of PE devices and the corresponding labels across the autonomous system boundaries and then redistributing the /32 host routes learned through EBGP into IGP. From the point of view of the PE routers, this is similar to the single autonomous system scenario except that the /32 host routes appear to be of IGP external types in the routing tables. Instead of using LDP, ASBRs piggyback IPv4 label information along with the host route advertisements in BGP update messages so that ASBRs can set up LSPs between one another for these host routes. Figure 9-8 illustrates such an example.

Figure 9-8. ***Inter-AS Pseudowire Emulation with IGP Redistribution***

[View full size image]



To provide the same end-to-end inter-AS pseudowire emulation services, you configure PE1, PE2, PE3, and PE4 identically to "Case Study 9-7: Interconnecting Pseudowires with Dedicated Circuits," except the pseudowire endpoint addresses. On ASBR1 and ASBR2, BGP is configured to announce /32 host routes of PE1, PE2, PE3, and PE4 to the remote autonomous system. To distribute IPv4 labels for these routes, ASBR1 and ASBR2 specify the **send-label** keyword in the BGP neighbor command.

The following configuration gives you some examples of how to configure the PE and ASBR routers to use BGP IPv4 label distribution with IGP redistribution to provide inter-AS pseudowire connectivity.

On PE1, configure the router ID and the pseudowire with VC ID 100 on the AC that connects to CE1, as shown in Example 9-28.

Example 9-28. *PE1 Pseudowire Configuration*

```
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 172.16.1.2 100 encapsulation mpls
```

On PE2, configure the router ID and the pseudowire with VC ID 200 on the AC that connects to CE2, as shown in Example 9-29.

Example 9-29. *PE2 Pseudowire Configuration*

```
interface Loopback0
 ip address 10.1.1.2 255.255.255.255
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 172.16.1.1 200 encapsulation mpls
```

On PE3, configure the router ID and the pseudowire with VC ID 200 on the AC that connects to CE3, as shown in Example 9-30.

Example 9-30. *PE3 Pseudowire Configuration*

```
interface Loopback0
 ip address 172.16.1.1 255.255.255.255
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 encapsulation mpls
```

On PE4, configure the router ID and the pseudowire with VC ID 100 on the AC that connects to CE4, as shown in Example 9-31.

Example 9-31. *PE4 Pseudowire Configuration*

```
interface Loopback0
 ip address 172.16.1.2 255.255.255.255
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 10.1.1.1 100 encapsulation mpls
```

On ASBR1, configure BGP IPv4 label distribution and redistribute BGP routes into OSPF, as shown in Example 9-32.

Example 9-32. *ASBR1 BGP and OSPF Configuration*

```
hostname ASBR1
!
ip cef
mpls label protocol ldp
!
interface Loopback0
 ip address 10.1.1.3 255.255.255.255
!
interface Ethernet0/0
 description Connect to ASBR2 in AS200
 ip address 172.16.100.1 255.255.255.0
!
interface Ethernet1/0
```

```
 ip address 10.43.11.2 255.255.255.0
 mpls ip
!
router ospf 1
 redistribute bgp 100 subnets
 network 10.1.1.3 0.0.0.0 area 0
 network 10.43.11.0 0.0.0.255 area 0
 default-metric 20
!
router bgp 100
 neighbor 172.16.100.2 remote-as 200
 !
 address-family ipv4
 neighbor 172.16.100.2 activate
 neighbor 172.16.100.2 send-label
 no auto-summary
 no synchronization
 network 10.1.1.1 mask 255.255.255.255
 network 10.1.1.2 mask 255.255.255.255
 exit-address-family
```

On ASBR2, configure BGP IPv4 label distribution and redistribute BGP routes into OSPF, as shown in Example 9-33.

Example 9-33. *ASBR2 BGP and OSPF Configuration*

```
hostname ASBR2
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.3 255.255.255.255
!
interface Ethernet0/0
 description Connect to ASBR1 in AS100
 ip address 172.16.100.2 255.255.255.0
!
interface Ethernet1/0
 ip address 172.16.24.2 255.255.255.0
 mpls ip
!
router ospf 1
 redistribute bgp 200 subnets
 network 172.16.1.3 0.0.0.0 area 0
 network 172.16.24.0 0.0.0.255 area 0
 default-metric 20
!
router bgp 200
 neighbor 172.16.100.1 remote-as 100
 !
 address-family ipv4
 neighbor 172.16.100.1 activate
```

```
 neighbor 172.16.100.1 send-label
 no auto-summary
 no synchronization
 network 172.16.1.1 mask 255.255.255.255
 network 172.16.1.2 mask 255.255.255.255
 exit-address-family
```

On ASBR1 and ASBR2, the /32 addresses of the PE routers appear as BGP routes, as shown in Example 9-34.

Example 9-34. *Host Routes of PE Routers on ASBR1 and ASBR2*

```
ASBR1#show ip route bgp
     172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
B       172.16.1.1/32 [20/75] via 172.16.100.2, 03:29:45
B       172.16.1.2/32 [20/75] via 172.16.100.2, 03:29:45

ASBR2#show ip route bgp
     10.0.0.0/32 is subnetted, 2 subnets
B       10.1.1.2 [20/31] via 172.16.100.1, 03:31:51
B       10.1.1.1 [20/31] via 172.16.100.1, 03:31:51
```

These routes are redistributed into OSPF and appear as OSPF External type 2 routes in the routing tables of the PE devices. Example 9-35 shows the routing table entries for PE1 in AS100 and PE3 in AS200.

Example 9-35. *Redistributed BGP Routes in OSPF Routing Table*

```
PE1#show ip route ospf
     172.16.0.0/32 is subnetted, 2 subnets
O E2    172.16.1.1 [110/20] via 10.23.12.2, 03:16:11, Ethernet1/0
O E2    172.16.1.2 [110/20] via 10.23.12.2, 03:16:11, Ethernet1/0
     10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O       10.23.21.0/24 [110/84] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.1.2.1/32 [110/21] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.1.1.2/32 [110/31] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.1.1.3/32 [110/31] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.23.23.0/24 [110/30] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.1.2.3/32 [110/21] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.1.2.2/32 [110/11] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.43.11.0/24 [110/30] via 10.23.12.2, 03:16:11, Ethernet1/0
O       10.33.23.0/24 [110/20] via 10.23.12.2, 03:16:11, Ethernet1/0

PE3#show ip route ospf
     172.16.0.0/16 is variably subnetted, 6 subnets, 2 masks
O       172.16.44.0/24 [110/128] via 172.16.34.2, 03:16:07, Serial2/0
O       172.16.24.0/24 [110/74] via 172.16.34.2, 03:16:07, Serial2/0
O       172.16.1.3/32 [110/75] via 172.16.34.2, 03:16:07, Serial2/0
O       172.16.1.2/32 [110/129] via 172.16.34.2, 03:16:07, Serial2/0
     10.0.0.0/32 is subnetted, 2 subnets
```

```
O E2     10.1.1.2 [110/20] via 172.16.34.2, 03:16:07, Serial2/0
O E2     10.1.1.1 [110/20] via 172.16.34.2, 03:16:07, Serial2/0
```

One caveat for this deployment model is that /32 host routes are injected into the IGP routing domain through redistributions; therefore, every transit router in the same IGP routing domain installs these host routes. As shown in Example 9-36, transit routers P1, P2, and P3 all see the host routes for PE3 and PE4 in their routing tables, and P4 has host routes for PE1 and PE2.

Example 9-36. *Transit Routers Learn Host Routes for PE1 and PE2*

```
P1#show ip route ospf
     172.16.0.0/32 is subnetted, 2 subnets
O E2     172.16.1.1 [110/20] via 10.43.11.2, 00:01:31, Ethernet1/0
O E2     172.16.1.2 [110/20] via 10.43.11.2, 00:01:31, Ethernet1/0
     10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O        10.1.1.2/32 [110/21] via 10.33.23.3, 08:34:35, Ethernet0/0
O        10.1.1.3/32 [110/11] via 10.43.11.2, 08:34:35, Ethernet1/0
O        10.23.23.0/24 [110/20] via 10.33.23.3, 08:34:35, Ethernet0/0
O        10.1.2.3/32 [110/11] via 10.33.23.3, 08:34:35, Ethernet0/0
O        10.1.2.2/32 [110/11] via 10.33.23.2, 08:34:35, Ethernet0/0
O        10.1.1.1/32 [110/21] via 10.33.23.2, 08:34:35, Ethernet0/0
O        10.23.12.0/24 [110/20] via 10.33.23.2, 08:34:35, Ethernet0/0
```

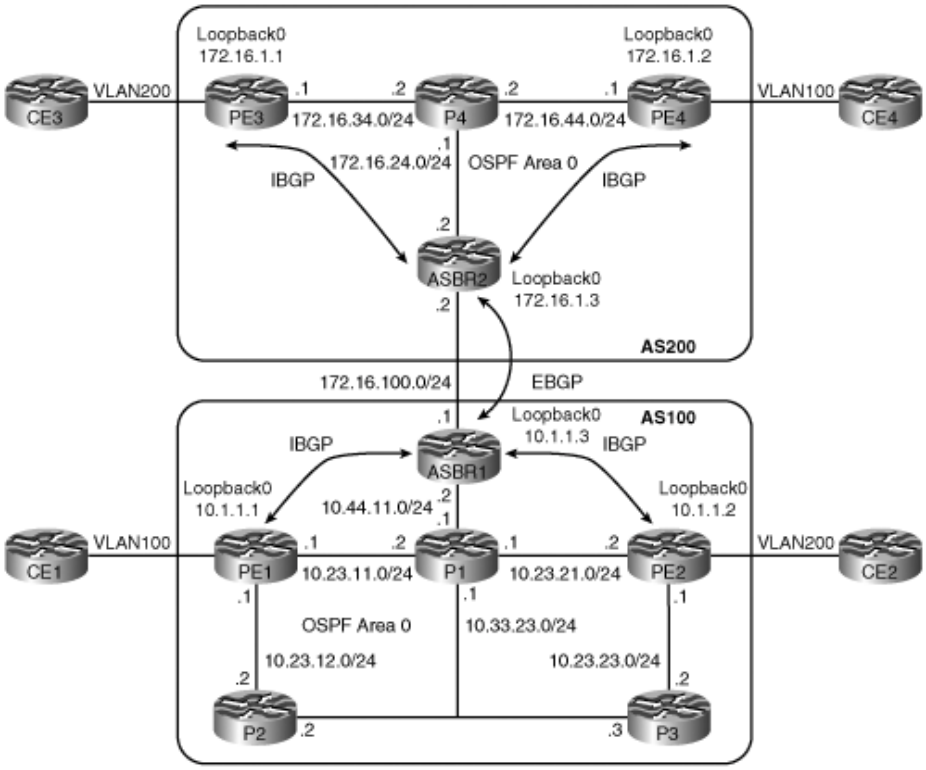
Case Study 9-9: BGP IPv4 Label Distribution with IBGP Peering

Using IGP redistribution of BGP routes, all transit routers in the IGP routing domain install the routes to their routing tables. If the BGP routing database also contains Internet routes, the number of entries that is redistributed into IGP is enormous. Applying route maps that only allow PE host routes to be redistributed at ASBRs can mitigate the IGP routing table explosion. However, when the number of host routes to be filtered increases, the configuration task becomes quite tedious and the routing table size of the transit routers still grows. To solve this problem, PE routers can establish internal BGP (IBGP) sessions with ASBRs within the same autonomous system so that external routes are distributed via IBGP sessions. This confines the external routes within the BGP routing domain, and the transit routers that do not participate in BGP routing never see these external routes.

Figure 9-9 shows an example of inter-AS pseudowire emulation that uses IBGP peering. PE1, PE2, and ASBR1 belong to the same autonomous system. PE1 and PE2 are configured with IBGP peering to ASBR1 to learn host routes for PE3 and PE4. Similarly, PE3 and PE4 are configured with IBGP peering to ASBR2 to learn host routes for PE1 and PE2.

Figure 9-9. *Inter-AS Pseudowire Emulation with IBGP Peering*

[View full size image]



After the IBGP sessions are established, these host routes are still missing in the routing tables on PE devices. For example, the routing table on PE1 does not have the entry for PE4 (see Example 9-37).

Example 9-37. *Host Route for PE4 Not in IP Routing Table on PE1*

```
PE1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O       10.23.21.0/24 [110/84] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.1.2.1/32 [110/21] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.1.1.2/32 [110/31] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.1.1.3/32 [110/31] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.23.23.0/24 [110/30] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.1.2.3/32 [110/21] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.1.2.2/32 [110/11] via 10.23.12.2, 02:59:03, Ethernet1/0
C       10.1.1.1/32 is directly connected, Loopback0
C       10.23.12.0/24 is directly connected, Ethernet1/0
```

```
C       10.23.11.0/24 is directly connected, Serial3/0
O       10.43.11.0/24 [110/30] via 10.23.12.2, 02:59:03, Ethernet1/0
O       10.33.23.0/24 [110/20] via 10.23.12.2, 02:59:03, Ethernet1/0
```

A closer examination of the BGP routing table on PE1 reveals the problem. The host route entry 172.16.1.2 for PE4 exists in the BGP routing table, but its next-hop address, 172.16.100.2, is inaccessible from PE1 (see Example 9-38).

Example 9-38. *Host Route for PE4 in BGP Routing Table on PE1*

```
PE1#show ip bgp 172.16.1.2
BGP routing table entry for 172.16.1.2/32, version 68
Paths: (1 available, no best path)
  Not advertised to any peer
  200
    172.16.100.2 (inaccessible) from 10.1.1.3 (10.1.1.3)
      Origin IGP, metric 75, localpref 100, valid, internal
```

The address 172.16.100.2 is of the interface Ethernet0/0 on ASBR2, which announces the host route 172.16.1.2 with the next-hop address set to 172.16.100.2. When ASBR1 relays this host route to PE1 through IBGP, the next-hop address is kept intact by default. The interface Ethernet0/0 on ASBR1 is directly connected to the interface Ethernet0/0 on ASBR2. Typically, IGP routing is not enabled on these interfaces, which means the interfaces are not reachable from the PE routers through IGP routing. You can fix this problem in several ways. For example, you can enable IGP routing on these interfaces and set them as passive interfaces, or you can configure the ASBRs as the next hop in the IBGP peering. For the sake of simplicity, the ASBRs are configured with the **next-hop-self** keyword in the BGP neighbor command for the IBGP peers in this case study.

PE1 sees that the host route of PE4 is reachable through ASBR1 in its routing table, and transit routers such as P2 do not have these host routes in their routing table (see Example 9-39).

Example 9-39. *Host Route for PE4 in IP Routing Table on PE1, But Not on P2*

```
PE1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR

Gateway of last resort is not set

     172.16.0.0/32 is subnetted, 2 subnets
B       172.16.1.1 [200/75] via 10.1.1.3, 00:37:57
```

```
B       172.16.1.2 [200/75] via 10.1.1.3, 00:37:57
     10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O       10.23.21.0/24 [110/84] via 10.23.12.2, 04:25:50, Ethernet1/0
O       10.1.2.1/32 [110/21] via 10.23.12.2, 04:25:50, Ethernet1/0
O       10.1.1.2/32 [110/31] via 10.23.12.2, 04:25:50, Ethernet1/0
O       10.1.1.3/32 [110/31] via 10.23.12.2, 04:25:50, Ethernet1/0
O       10.23.23.0/24 [110/30] via 10.23.12.2, 04:25:50, Ethernet1/0
O       10.1.2.3/32 [110/21] via 10.23.12.2, 04:25:50, Ethernet1/0
O       10.1.2.2/32 [110/11] via 10.23.12.2, 04:25:50, Ethernet1/0
C       10.1.1.1/32 is directly connected, Loopback0
C       10.23.12.0/24 is directly connected, Ethernet1/0
C       10.23.11.0/24 is directly connected, Serial3/0
O       10.43.11.0/24 [110/30] via 10.23.12.2, 04:25:51, Ethernet1/0
O       10.33.23.0/24 [110/20] via 10.23.12.2, 04:25:51, Ethernet1/0

P2#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O       10.23.21.0/24 [110/74] via 10.33.23.1, 12:06:10, Ethernet0/0
O       10.1.2.1/32 [110/11] via 10.33.23.1, 12:06:10, Ethernet0/0
O       10.1.1.2/32 [110/21] via 10.33.23.3, 12:06:10, Ethernet0/0
O       10.1.1.3/32 [110/21] via 10.33.23.1, 12:06:10, Ethernet0/0
O       10.23.23.0/24 [110/20] via 10.33.23.3, 12:06:10, Ethernet0/0
O       10.1.2.3/32 [110/11] via 10.33.23.3, 12:06:10, Ethernet0/0
C       10.1.2.2/32 is directly connected, Loopback0
O       10.1.1.1/32 [110/11] via 10.23.12.1, 12:06:10, Ethernet1/0
C       10.23.12.0/24 is directly connected, Ethernet1/0
O       10.23.11.0/24 [110/74] via 10.33.23.1, 12:06:10, Ethernet0/0
                      [110/74] via 10.23.12.1, 12:06:10, Ethernet1/0
O       10.43.11.0/24 [110/20] via 10.33.23.1, 12:06:10, Ethernet0/0
C       10.33.23.0/24 is directly connected, Ethernet0/0
```

In previous case studies, in which host routes of PE routers are exchanged through IGP, having /32 host routes and the corresponding labels is sufficient for establishing pseudowires. In this example, PE1 has a /32 route and the corresponding label to PE4. If the AToM pseudowire is configured and its status is up, it gives the impression that the pseudowire functions fully. Based on the output of the **show ip cef** and **show mpls l2transport vc** commands in Example 9-40, the pseudowire with VC ID 100 has a label stack of {27 16}. Label 27 is the tunnel label, and label 16 is the VC label.

Example 9-40. *Reachability Information for PE4 and Pseudowire Status for VC ID 100*

```
PE1#show ip cef 172.16.1.2
172.16.1.2/32, version 83, epoch 0, cached adjacency 10.23.12.2
0 packets, 0 bytes
  tag information from 10.1.1.3/32, shared, all rewrites owned
    local tag: 28
    fast tag rewrite with Et1/0, 10.23.12.2, tags imposed {27}
  via 10.1.1.3, 0 dependencies, recursive
    next hop 10.23.12.2, Ethernet1/0 via 10.1.1.3/32 (Default)
    valid cached adjacency
    tag rewrite with Et1/0, 10.23.12.2, tags imposed {27}

PE1#show mpls l2transport vc 100 detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 100 up
  Destination address: 172.16.1.2, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 27, next hop 10.23.12.2
    Output interface: Et1/0, imposed label stack {27 16}
  Create time: 00:00:35, last status change time: 00:00:30
  Signaling protocol: LDP, peer 172.16.1.2:0 up
    MPLS VC labels: local 21, remote 16
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  VC statistics:
    packet totals: receive 0, send 6
    byte totals: receive 0, send 728
    packet drops: receive 0, send 0
```

When you send packets from CE1 to CE4, CE4 never receives anything even though the counters for packets sent increase on PE1. Apparently, IP connectivity is fine between PE1 and PE4, because the LDP session that is used for pseudowire signaling functions properly. With that in mind, the problem might be related to how the pseudowire packets are forwarded.

Tracking down the pseudowire packet forwarding path from PE1 through the intermediate routers and all the way to ASBR1, use the **show mpls forwarding-table** command to examine the MPLS label forwarding actions. On Router P2, label 27 gets swapped with label 24, which results in a new label stack {24 16} for the pseudowire packets (see Example 9-41).

Example 9-41. *Label Operation for Label 27 on P2 Router*

```
P2#show mpls forwarding-table labels 27
Local  Outgoing    Prefix           Bytes tag  Outgoing  Next Hop
tag    tag or VC   or Tunnel Id     switched   interface
27     24          10.1.1.3/32      640453     Et0/0     10.33.23.1
```

On Router P3, the top label 24 is removed, which leaves the label stack at {16} (see Example 9-42). Note that label 16 is the VC label for the pseudowire packets.

Example 9-42. *Label Operation for Label 24 on P3 Router*

```
P3#show mpls forwarding-table labels 24
Local  Outgoing    Prefix           Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id     switched   interface
24     Pop tag     10.43.11.0/24    0          Et0/0       10.33.23.1
```

When labeled pseudowire packets arrive at ASBR1, the last label 16 in the label stack is removed according to the MPLS forwarding table, which leaves the pseudowire packets unlabeled (see Example 9-43).

Example 9-43. *Label Operation for Label 16 on ASBR1 Router*

```
ASBR1#show mpls forwarding-table labels 16
Local  Outgoing    Prefix           Bytes tag  Outgoing    Next Hop
tag    tag or VC   or Tunnel Id     switched   interface
16     Pop tag     10.33.23.0/24    0          Et1/0       10.43.11.1
```

Under normal operations, PE routers that are provisioned with pseudowires assign the VC labels. Therefore, they should be the label switching routers that remove the VC labels and process the rest of the pseudowire packets. In other words, intermediate label switching routers should not remove VC labels from the label stack. ASBR1 does not act as a PE router in this case study, so removing VC labels on ASBR1 is a mistake.

The cause of the problem lies in the interaction between the routing process and the label distribution process. In all previous case studies, a router that advertises a given route through a routing protocol also advertises the label that is associated with that route through a label distribution protocol, but that is not the case here. PE1 learns the host route 172.16.1.2 from ASBR1 through IBGP with the next-hop address set to ASBR1, but PE1 does not learn the label that is associated with the host route from ASBR1. You can observe this behavior by using the **show ip bgp labels** command, as shown in Example 9-44.

Example 9-44. *Labels Learned Through BGP on PE1*

```
PE1#show ip bgp labels
   Network          Next Hop      In label/Out label
   172.16.1.1/32    10.1.1.3        nolabel/nolabel
   172.16.1.2/32    10.1.1.3        nolabel/nolabel
```

The absence of the label for 172.16.1.2 does not impose a problem for IP packets because IP packets can always be forwarded unlabeled in an MPLS network. This fact explains how PE1 and PE4 can establish a targeted LDP session and exchange VC labels for the pseudowire. On the other hand, PE1 does not have an end-to-end contiguous LSP to reach PE4, which means the MPLS packets cannot reach PE4. The partial LSP is between PE1 and ASBR1 because PE1 does have an LSP to reach the next-hop address 10.1.1.3.

To eliminate the MPLS connectivity problem and create an end-to-end contiguous LSP, ASBR1 needs to send the corresponding label and the host route for 172.16.1.2 to PE1. In other words, besides enabling IPv4 label distribution over the EBGP session between ASBR1 and ASBR2, as described in "Case Study 9-8: BGP IPv4 Label Distribution with IGP Redistribution," the IBGP sessions between PE and ASBR also need to distribute IPv4 labels for the host routes that are being advertised. After adding the **send-label** keyword to the BGP neighbor command for the IBGP peers on PE and ASBR, PE1 obtains label 32 for the host route 172.16.1.2 from IBGP (see Example 9-45).

Example 9-45. *Labels Learned Through BGP on PE1 After Enabling IPv4 Label Distribution in IBGP*

```
PE1#show ip bgp label
   Network          Next Hop        In label/Out label
   172.16.1.1/32    10.1.1.3          nolabel/31
   172.16.1.2/32    10.1.1.3          nolabel/32
```

From the **show ip cef** and **show mpls l2transport vc** commands, the pseudowire with VC ID 100 now has a label stack of {27 32 16}. Label 16 is still the VC label, but to reach PE4, it requires two labels: Label 27 is the IGP label to reach ASBR1 that has the address 10.1.1.3, and label 32 is the BGP IPv4 label assigned by ASBR1 to reach PE4 that has the address 172.16.1.2 (see Example 9-46).

Example 9-46. *Reachability Information for PE4 and Pseudowire Status for VC ID 100 After Fixing the MPLS Connectivity Problem*

```
PE1#show ip cef 172.16.1.2
172.16.1.2/32, version 85, epoch 0, cached adjacency 10.23.12.2
0 packets, 0 bytes
  tag information set, all rewrites owned
    local tag: BGP route head
    fast tag rewrite with Et1/0, 10.23.12.2, tags imposed {27 32}
  via 10.1.1.3, 0 dependencies, recursive
    next hop 10.23.12.2, Ethernet1/0 via 10.1.1.3/32 (Default)
    valid cached adjacency
    tag rewrite with Et1/0, 10.23.12.2, tags imposed {27 32}

PE1#show mpls l2transport vc 100 detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 100 up
  Destination address: 172.16.1.2, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 32, next hop 10.23.12.2
    Output interface: Et1/0, imposed label stack {27 32 16}
  Create time: 07:38:14, last status change time: 07:38:09
  Signaling protocol: LDP, peer 172.16.1.2:0 up
    MPLS VC labels: local 21, remote 16
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
```

```
  VC statistics:
    packet totals: receive 29, send 465
    byte totals:   receive 8308, send 176142
    packet drops:  receive 0, send 8
```

When sending packets from CE1 to CE4 again, they arrive at ASBR1 with a label stack {32 16} this time. The **show mpls forwarding-table** command on ASBR1 further confirms that pseudowire packets are properly forwarded to ASBR2 through an LSP, as shown in Example 9-47.

Example 9-47. *Label Operation for Label 32 on ASBR1 Router*

```
ASBR1#show mpls forwarding-table labels 32
Local  Outgoing    Prefix           Bytes tag   Outgoing    Next Hop
tag    tag or VC   or Tunnel Id     switched    interface
32     17          172.16.1.2/32    36750       Et0/0       172.16.100.2
```

After you complete the configuration on all PEs and ASBRs, you can accomplish the end-to-end inter-AS pseudowire connectivity by using BGP IPv4 label distribution with IBGP peering.

The following configuration gives you some examples of how to configure the PE and ASBR routers to use BGP IPv4 label distribution with IBGP peering to provide inter-AS pseudowire connectivity.

On PE1, configure BGP IPv4 label distribution with IBGP peering and the pseudowire with VC ID 100, as shown in Example 9-48.

Example 9-48. *PE1 BGP and Pseudowire Configuration*

```
hostname PE1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.1 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100
 xconnect 172.16.1.2 100 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.23.12.1 255.255.255.0
 mpls ip
!
```

```
interface Serial3/0
 ip address 10.23.11.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.1 0.0.0.0 area 0
 network 10.23.11.0 0.0.0.255 area 0
 network 10.23.12.0 0.0.0.255 area 0
!
router bgp 100
 neighbor 10.1.1.3 remote-as 100
 neighbor 10.1.1.3 update-source Loopback0
 !
 address-family ipv4
 neighbor 10.1.1.3 activate
 neighbor 10.1.1.3 send-label
 no auto-summary
 no synchronization
 exit-address-family
```

On PE2, configure BGP IPv4 label distribution with IBGP peering and the pseudowire with VC ID 200, as shown in Example 9-49.

Example 9-49. *PE2 BGP and Pseudowire Configuration*

```
hostname PE2
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 10.1.1.2 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 172.16.1.1 200 encapsulation mpls
!
interface Ethernet1/0
 ip address 10.23.23.1 255.255.255.0
 mpls ip
!
interface Serial3/0
 ip address 10.23.21.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.2 0.0.0.0 area 0
 network 10.23.21.0 0.0.0.255 area 0
 network 10.23.23.0 0.0.0.255 area 0
!
```

```
router bgp 100
 neighbor 10.1.1.3 remote-as 100
 neighbor 10.1.1.3 update-source Loopback0
 !
 address-family ipv4
 neighbor 10.1.1.3 activate
 neighbor 10.1.1.3 send-label
 no auto-summary
 no synchronization
 exit-address-family
```

On PE3, configure BGP IPv4 label distribution with IBGP peering and the pseudowire with VC ID 200, as shown in Example 9-50.

Example 9-50. *PE3 BGP and Pseudowire Configuration*

```
hostname PE3
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.1 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.2
 encapsulation dot1Q 200
 xconnect 10.1.1.2 200 encapsulation mpls
!
interface Serial2/0
 ip address 172.16.34.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 172.16.1.1 0.0.0.0 area 0
 network 172.16.34.0 0.0.0.255 area 0
!
router bgp 200
 neighbor 172.16.1.3 remote-as 200
 neighbor 172.16.1.3 update-source Loopback0
 !
 address-family ipv4
 neighbor 172.16.1.3 activate
 neighbor 172.16.1.3 send-label
 no auto-summary
 no synchronization
 exit-address-family
```

On PE4, configure BGP IPv4 label distribution with IBGP peering and the pseudowire with VC ID 100, as shown in Example 9-51.

Example 9-51. *PE4 BGP and Pseudowire Configuration*

```
hostname PE4
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.2 255.255.255.255
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 100

 xconnect 10.1.1.1 100 encapsulation mpls
!
interface Serial2/0
 ip address 172.16.44.1 255.255.255.0
 mpls ip
!
router ospf 1
 network 172.16.1.2 0.0.0.0 area 0
 network 172.16.44.0 0.0.0.255 area 0
!
 router bgp 200
 neighbor 172.16.1.3 remote-as 200
 neighbor 172.16.1.3 update-source Loopback0
 !
 address-family ipv4
 neighbor 172.16.1.3 activate
 neighbor 172.16.1.3 send-label
 no auto-summary
 no synchronization
 exit-address-family
```

On ASBR1, configure BGP IPv4 label distribution with both EBGP and IBGP peers, as shown in Example 9-52.

Example 9-52. *ASBR1 BGP Configuration*

```
hostname ASBR1
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
```

```
interface Loopback0
 ip address 10.1.1.3 255.255.255.255
!
interface Ethernet0/0
 description Connect to ASBR2 in AS200
 ip address 172.16.100.1 255.255.255.0
!
interface Ethernet1/0
 ip address 10.43.11.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 10.1.1.3 0.0.0.0 area 0
 network 10.43.11.0 0.0.0.255 area 0
!
router bgp 100
 neighbor 10.1.1.1 remote-as 100
 neighbor 10.1.1.1 update-source Loopback0
 neighbor 10.1.1.2 remote-as 100
 neighbor 10.1.1.2 update-source Loopback0
 neighbor 172.16.100.2 remote-as 200
 !
 address-family ipv4
 neighbor 10.1.1.1 activate
 neighbor 10.1.1.1 next-hop-self
 neighbor 10.1.1.1 send-label
 neighbor 10.1.1.2 activate
 neighbor 10.1.1.2 next-hop-self
 neighbor 10.1.1.2 send-label
 neighbor 172.16.100.2 activate
 neighbor 172.16.100.2 send-label
 no auto-summary
 no synchronization
 network 10.1.1.1 mask 255.255.255.255
 network 10.1.1.2 mask 255.255.255.255
 exit-address-family
```

On ASBR2, configure BGP IPv4 label distribution with both EBGP and IBGP peers, as shown in Example 9-53.

Example 9-53. *ASBR2 BGP Configuration*

```
hostname ASBR2
!
ip cef
mpls label protocol ldp
mpls ldp router-id Loopback0
!
interface Loopback0
 ip address 172.16.1.3 255.255.255.255
!
interface Ethernet0/0
 description Connect to ASBR1 in AS100
 ip address 172.16.100.2 255.255.255.0
```

```
!
interface Ethernet1/0
 ip address 172.16.24.2 255.255.255.0
 mpls ip
!
router ospf 1
 network 172.16.1.3 0.0.0.0 area 0
 network 172.16.24.0 0.0.0.255 area 0
!
router bgp 200
 neighbor 172.16.1.1 remote-as 200
 neighbor 172.16.1.1 update-source Loopback0
 neighbor 172.16.1.2 remote-as 200
 neighbor 172.16.1.2 update-source Loopback0
 neighbor 172.16.100.1 remote-as 100
 !
 address-family ipv4
 neighbor 172.16.1.1 activate
 neighbor 172.16.1.1 next-hop-self
 neighbor 172.16.1.1 send-label
 neighbor 172.16.1.2 activate
 neighbor 172.16.1.2 next-hop-self
 neighbor 172.16.1.2 send-label
 neighbor 172.16.100.1 activate
 neighbor 172.16.100.1 send-label
 no auto-summary
 no synchronization
 network 172.16.1.1 mask 255.255.255.255
 network 172.16.1.2 mask 255.255.255.255
 exit-address-family
```

Case Study 9-10: Configuring LDP Authentication for Pseudowire Signaling

In an MPLS network, where the trust relationship is assumed within the network boundary, authentication for pseudowire signaling is usually absent. However, Cisco IOS still provides LDP authentication when network operators consider it necessary. Like other MPLS applications that use LDP, AToM can also enable LDP authentication for pseudowire signaling.

LDP performs authentication through the TCP MD5 Signature Option, which is essentially a message digest checksum to validate the integrity of the message. The checksum is calculated based on the content being transmitted and a shared password.

To configure LDP authentication for pseudowire signaling, use the **mpls ldp neighbor password** command under the global configuration mode. For example, PE1 and PE2 need to configure LDP authentication and have a shared password l2vpn, as shown in Example 9-54.

Example 9-54. *Configuring LDP Authentication*

```
PE1(config)#mpls ldp neighbor 10.1.1.2 password ?
  LINE  The password
  <0-7>  Encryption type (0 to disable encryption, 7 for proprietary)

PE1(config)#mpls ldp neighbor 10.1.1.2 password l2vpn

PE2#config t
Enter configuration commands, one per line.  End with CNTL/Z.
PE2(config)#mpls ldp neighbor 10.1.1.1 password l2vpn
```

To verify that the LDP session is enabled with MD5 authentication, use the **show mpls ldp neighbor** detail command, as shown in Example 9-55.

Example 9-55. *Verify That LDP Authentication Is Enabled*

```
PE1#show mpls ldp neighbor 10.1.1.2 detail
    Peer LDP Ident: 10.1.1.2:0; Local LDP Ident 10.1.1.1:0
        TCP connection: 10.1.1.2.11035 - 10.1.1.1.646; MD5 on
        State: Oper; Msgs sent/rcvd: 26/26; Downstream; Last TIB rev sent 22
        Up time: 00:08:10; UID: 5; Peer Id 2;
        LDP discovery sources:
          Targeted Hello 10.1.1.1 -> 10.1.1.2, active, passive;
            holdtime: infinite, hello interval: 10000 ms
        Addresses bound to peer LDP Ident:
          10.23.23.1      10.1.1.2        10.23.21.2
        Peer holdtime: 180000 ms; KA interval: 60000 ms; Peer state: estab
        Clients: Dir Adj Client
```

If a PE router has a password configured for a peer PE router, but the peer PE router does not have the password configured, a message such as the following appears on the console of the PE router:

```
00:53:41: %TCP-6-BADAUTH: No MD5 digest from 10.1.1.2(11037) to 10.1.1.1(646)
```

If two PE routers have different passwords configured, a message such as the following appears on the console:

```
00:55:57: %TCP-6-BADAUTH: Invalid MD5 digest from 10.1.1.2(11041) to 10.1.1.1(646)
```

When the password is missing from one PE router or the passwords that are configured on two PE routers do not match, the LDP session is not established.

Verifying Pseudowire Data Connectivity

Fault detection, isolation, and verification techniques are critical for the deployment of MPLS applications, including pseudowire emulation. The ability to detect faults in the data plane or forwarding path for pseudowire services in a packet-switched network is critical for network operators. This section explores virtual circuit connectivity verification (VCCV), which provides an answer to pseudowire fault detection.

The connectivity verification model for pseudowires consists mainly of two distinctive building blocks that are specified in two different Internet drafts:

- Advertising the VCCV capability
- Verifying data plane connectivity

Case Studies 9-11 and 9-12 describe both building blocks in detail.

You can verify the pseudowire dataplane connectivity by creating a control channel within the pseudowire. This control channel is associated with the pseudowire, and data connectivity packets flow in this control channel. The control channel has two requirements:

- To follow the pseudowire data path as closely as possible
- To divert data connectivity verification packets so that they are processed by the receiving PE device as opposed to being forwarded out to the CE devices

As you will see in "Case Study 9-11: Advertising the VCCV Capability," three control channel types (CC types) provide the preceding two requirements.

After you define the control channel, you need to specify the connectivity verification packets and protocols that will use the control channel. You can use multiple protocols over the control channel, which have different data connectivity verification types (CV types). The three currently defined CV types are IP-based protocols.

Case Study 9-11: Advertising the VCCV Capability

The capability of VCCV is advertised as part of the MPLS Label Mapping message in the Pseudowire ID FEC as an interface parameter. Example 9-56 shows a decoding example of the VCCV interface parameter taken with Ethereal software.

Example 9-56. *VCCV Interface Parameter*

```
Interface Parameter: VCCV
ID: VCCV (0x0c)
Length: 4
CC Type
    .... ...1 = PWE3 Control Word: True
    .... ..1. = MPLS Router Alert: True
    .... .0.. = MPLS Inner Label TTL = 1: False
CV Type
    .... ...0 = ICMP Ping: False
    .... ..1. = LSP Ping: True
   .... .0.. = BFD: False
```

The ID value 0x0C indicates that this is a VCCV interface parameter. It consists of two fields that have various options:

- **Control Channel (CC) type** Defines a bitmask that indicates the types of control channel that can be used to receive CC traffic to verify connectivity. If more than one is specified, the router agrees to accept control traffic at any time over any control channel:

  PWE3 Control Word (type 1) The control channel traffic is carried inband with data traffic on the pseudowire being monitored using the same label stack. When you use this control channel, a special format of the AToM control word instructs the PE router to inspect the control channel traffic.

  MPLS Router Alert Label (type 2) The control channel is created out-of-band from the pseudowire, and it utilizes the reserved Router Alert (RA) label. The notion of "out-of-band" comes from the fact that the connectivity verification packet has a slightly different MPLS label stack than the actual pseudowire data packet.

  MPLS Inner Label TTL = 1 (type 3) It is also known as TTL Expiry that sets the TTL of the VC label to 1, which forces the control packet to be processed by the receiving PE router.

- **Connectivity Verification (CV) type** Defines a bitmask that indicates the types of CV packets and protocols that can be sent on the specified control channel:

  Internet Control Message Protocol (ICMP) Ping ICMP-based Echo Request and Reply.

  LSP Ping MPLS-based Echo Request and Reply.

  BFD Bidirectional Forwarding Detection provides a continuous monitoring and forward and backward defect indication and propagation.

Table 9-1 compares the three control channel types.

Table 9-1. *Comparing VCCV Control Channel Types*

Control Channel Type	Channel Type	Pros	Cons and Limitations
Type 1PWE Control Word	Inband	VCCV traffic follows the same path as pseudowire data traffic.	Pseudowire must use the control word.
Type 2MPLS RA Label	Out-of-band	Available even if the control word is not present or cannot be inspected.	VCCV traffic might take a different path than the pseudowire data traffic.

Control Channel Type	Channel Type	Pros	Cons and Limitations
Type 3MPLS VC Label TTL = 1	Inband	VCCV traffic follows the same path as pseudowire data traffic, and no control word is necessary.	Might not work if the penultimate hop overwrites the TTL.

When you create an inband control channel of a pseudowire, the data flow and the control flow are effectively multiplexed over the same forwarding path, which is the most accurate picture of the data connectivity. This is why inband methods are preferred.

In contrast, the out-of-band control flow might follow a different forwarding path from the actual data flow because of the ECMP load sharing forwarding behavior described earlier in this chapter. There is no impact, however, if the pseudowire path is free of ECMPs, although that is not a realistic assumption. The out-of-band channel is created by using the reserved RA label. The RA label means that every router must examine the packet. With an RA label, all packets are punted to the route processor (RP) for processing; therefore, you can use this method to detect inconsistencies between the linecard and the RP. In an intermediate router, after the packet that contains the RA label is processed, if the packet needs to be forwarded further, the RA label is pushed back onto the label stack before forwarding.

Currently, Cisco routers advertise CC types 1 and 2, but the Cisco router prefers to use the control word CC type because of its inband capabilities to traverse the same path as the pseudowire data plane. The only CV type that is currently supported is LSP Ping.

You can display the VCCV capability advertisement by using the **show mpls l2transport binding** command. Example 9-57 provides output of this command.

Example 9-57. show mpls l2transport binding *Command Output*

```
PE1#show mpls l2transport binding 300
  Destination Address: 10.0.0.203, VC ID: 300
    Local Label: 18
        Cbit: 1,    VC Type: ATM VCC CELL,    GroupID: 5
        MTU: n/a,   Interface Desc: *** Packed Cell VC AToM to CE1 ***
        Max Concatenated ATM Cells: 10
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 18
        Cbit: 1,    VC Type: ATM VCC CELL,    GroupID: 2
        MTU: n/a,   Interface Desc: *** Packed Cell VC AToM to CE2 ***
        Max Concatenated ATM Cells: 10
        VCCV Capabilities: Type 1, Type 2
```

The VCCV CC types that are displayed with the **show mpls l2transport binding** command are displayed as type 1 for the control word and type 2 for the MPLS RA label.

To understand the ECMP implications of using CC type 1 versus CC type 2, you need to be familiar with the ECMP procedures. To load share traffic between multiple paths with equal cost in traditional IP networks, routers use a hashing algorithm performed on the source and destination IP addresses

in the IPv4 or IPv6 packet header to choose the outgoing path within the multiple paths. This minimizes misordering of packets by sending IP flows on a single path.

MPLS networks adapted the same technique by inspecting the payload of MPLS packets. However, because LDP is a stateful protocol and the MPLS header does not have an upper layer protocol identification field, it uses a heuristic method to determine the payload type by inspecting the first nibble of the MPLS payload. The first field in an IP packet is the IP version. Therefore, if the value of the first nibble is 4, it is assumed that the payload is IPv4. If the first nibble is 6, however, it is assumed that the payload is IPv6. Because pseudowire packets do not carry raw IP traffic and do not guarantee that the first nibble is either 4 or 6, this can lead to undesired results, in which case pseudowire packets from the same flow are sent in different paths.

To avoid mistreatment for pseudowire data packets, the first nibble in the control word is reserved and set to 0. For VCCV traffic with control channel type 1, the control word is required. Its first nibble is set to 1 to avoid aliasing the payload with an IPv4 or IPv6 packet. However, for VCCV label; therefore, VCCV traffic can take a different path than pseudowire data traffic.

Case Study 9-12: Verifying Data Plane Connectivity

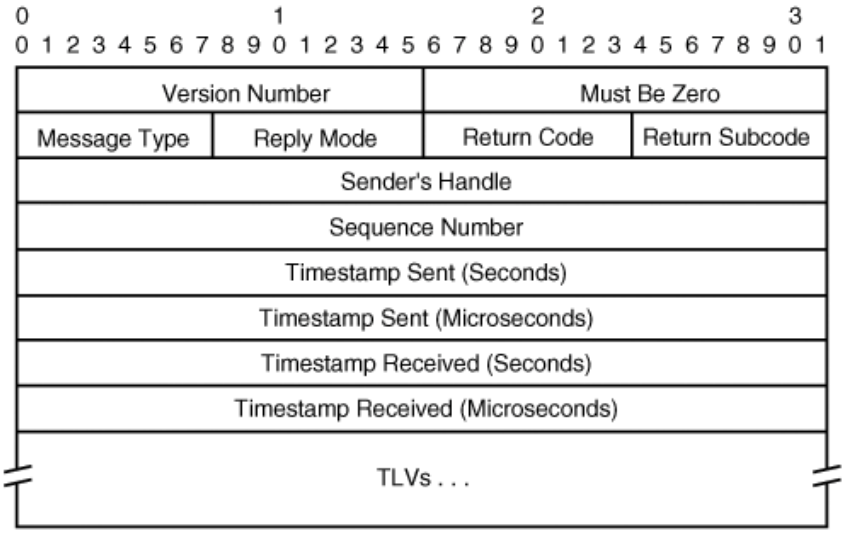
After the VCCV capability has been exchanged, each control channel distinguishes data and VCCV packets as follows:

- For CC type 1, a special control word is used. The first nibble is set to 1 to indicate VCCV packets. The first nibble of the control word is set to 0 for all data packets.
- For CC type 2, the RA label is placed immediately above the pseudowire label for VCCV packets, and data packets do not have the RA label in the MPLS label stack.

The special control word in CC type 1 also includes a protocol type field to indicate the protocol that is being carried. The protocol type field that is used is the Internet Assigned Numbers Authority (IANA) PPP Data Link Layer (DLL) Protocol Number.

LSP Ping is currently the only supported CV type, where MPLS Echo packets are IPv4 or IPv6 User Datagram Protocol (UDP) packets using the IANA assigned well-known UDP port of 3503. These UDP packets are possibly MPLS labeled. In an MPLS Echo Request, the source IP address is the originating router's outgoing interface address as expected, but the destination IP address is within the reserved range of internal host loopback addresses of 127.0.0.0/8. The IP TTL of the MPLS Echo Request packet is set to 1 so that when all of the MPLS labels are popped, the underlying LSP Ping IP packet is not forwarded, and the RA option is set in the IP header. The format of an LSP Echo packet is shown in Figure 9-10.

Figure 9-10. *MPLS Echo Packet Format*



The message type is either 1 for MPLS Echo Request or 2 for MPLS Echo Reply. The reply mode can specify no reply, reply via IPv4/IPv6 with or without RA option, or reply via application-level control channel. The ability to specify the reply mode gives great flexibility to LSP Ping. You can use the option with no reply to verify one-way connectivity by checking the Sequence Number field, or you can gather SLA statistics by checking the TimeStamp Sent field. The mode of reply via the application level control channel is currently not further defined. You can choose between the remaining two reply modes of IP with and without RA option when issuing LSP Ping packets from the Cisco IOS command line. The difference and applicability between these two reply modes are covered at the end of this case study.

Currently, the five Type Length Values (TLV) defined are as follows:

- Target FEC Stack
- Downstream Mapping
- Pad
- Error Code
- Vendor Enterprise Code

In a pseudowire ping, you will use the Target FEC Stack TLV with a pseudowire sub-TLV to identify the pseudowire. Optionally, you will use the Pad TLV and the Vendor Enterprise Code TLV with a Cisco SMI enterprise number of 9.

Note

You can use the MPLS Echo procedures for many different FEC types by specifying a different sub-TLV in Target FEC stack TLV. Besides connectivity verification for pseudowires, you can use MPLS Echo to test the following FEC types: LDP signaled IPv4 and IPv6 prefix FECs, RSVP-TE signaled IPv4 and IPv6 session FECs, and VPN-IPv4 and IPv6 prefix FECs. LSP Ping is used to check connectivity, not only in ping mode, but also in traceroute mode. This section concentrates on pseudowire connectivity testing.

Example 9-58 shows an MPLS Echo Request decoding for a pseudowire.

Example 9-58. *MPLS Echo Request Decoding*

```
Ethernet II, Src: xx:xx:xx:xx:xx:xx, Dst: yy:yy:yy:yy:yy:yy
    Destination: yy:yy:yy:yy:yy:yy (yy:yy:yy:yy:yy:yy)
    Source: xx:xx:xx:xx:xx:xx (xx:xx:xx:xx:xx:xx)
    Type: MPLS label switched packet (0x8847)
MultiProtocol Label Switching Header
    MPLS Label: Unknown (17)
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 0
    MPLS TTL: 255
MultiProtocol Label Switching Header
    MPLS Label: Unknown (22)
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 2
MPLS PW Control Channel Header
    Control Channel: 0x1
    Reserved: 0x000
    PPP DLL Protocol Number: IP (0x0021)
Internet Protocol, Src Addr: 10.0.0.201 (10.0.0.201), Dst Addr: localhost (127.0.0.1)
    Version: 4
    Header length: 24 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 100
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 1
    Protocol: UDP (0x11)
    Header checksum: 0x5cba (correct)
    Source: 10.0.0.201 (10.0.0.201)
    Destination: localhost (127.0.0.1)
    Options: (4 bytes)
        Router Alert: Every router examines packet
User Datagram Protocol, Src Port: 3503 (3503), Dst Port: 3503 (3503)
    Source port: 3503 (3503)
    Destination port: 3503 (3503)
    Length: 76
    Checksum: 0x4f8f (correct)
Multiprotocol Label Switching Echo
    Version: 1
    MBZ: 0
    Message Type: MPLS Echo Request (1)
    Reply Mode: Reply via an IPv4/IPv6 UDP packet with Router Alert (3)
    Return Code: No return code (0)
    Return Subcode: 0
    Sender's Handle: 0xc8000033
    Sequence Number: 1
    Timestamp Sent: 2004-05-03 15:32:22.5040 UTC
    Timestamp Received: NULL
    Target FEC Stack
        Type: Target FEC Stack (1)
        Length: 20
        FEC Element 1: L2 circuit ID
            Type: L2 cirtuit ID (9)
            Length: 16
```

```
            Sender's PE Address: 10.0.0.203 (10.0.0.203)
            Remote PE Address: 10.0.0.201 (10.0.0.201)
            VC ID: 50
            Encapsulation: HDLC (6)
            MBZ: 0x0000
    Pad
        Type: Pad (3)
        Length: 8
        Pad Action: Drop Pad TLV from reply (1)
        Padding: ABCDABCDABCDAB
```

The highlighted lines in Example 9-58 show how an MPLS Echo packet is encapsulated in IP/UDP with the RA option in the IP header, and in turn MPLS-labeled. You can also see that a Pseudowire Control Channel Header is included when using CC type 1.

To verify data connectivity using LSP Ping on Cisco routers, you can execute the **ping mpls** command with the **pseudowire** keyword in the EXEC mode. Other available keywords are **ipv4** for an LDP IPv4 FEC and **traffic-eng** for an RSVP-TE Tunnel FEC (see Example 9-59).

Example 9-59. *ping mpls pseudowire* Command Output

```
PE1#ping mpls pseudowire 10.0.0.201 100
Sending 5, 100-byte MPLS Echos to 10.0.0.201/0,
      timeout is 2 seconds, send interval is 0 msec:

Codes: '!' - success, 'Q' - request not transmitted,
       '.' - timeout, 'U' - unreachable,
       'R' - downstream router but not target

Type escape sequence to abort.
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
PE1#
PE1#ping mpls pseudowire 10.0.0.201 200
Sending 5, 100-byte MPLS Echos to 10.0.0.201/0,
      timeout is 2 seconds, send interval is 0 msec:

Codes: '!' - success, 'Q' - request not transmitted,
       '.' - timeout, 'U' - unreachable,
       'R' - downstream router but not target

Type escape sequence to abort.
QQQQQ
Success rate is 0 percent (0/5)
PE1#
PE1#ping mpls pseudowire 10.0.0.201 200 reply mode ?
  ipv4          Send reply via IPv4
  router-alert  Send reply via IPv4 UDP with router alert
```

The first test for the pseudowire with VC ID 100 is successful, and the result is "!!!!!". The second test for the pseudowire with VC ID 200 has a result of "QQQQQ", meaning "request not transmitted." The following are the most common reasons for not transmitting an MPLS Echo Request:

- The VC is down.

- The peer does not advertise VCCV capabilities.

The **ping mpls** command allows two different reply modes discussed before: **ipv4** and **router-alert**. The default is **ipv4**, which is the option normally used. However, if an LSP Ping is unsuccessful and times out (resulting "....."), the failure might occur on the return path. In this case, retry the **router-alert** reply mode. This mode instructs all intermediate routers in the return path to process the packet. This option is most useful for isolating MPLS switching path problems.

If you do not get a reply to a **ping mpls pseudowire** using the default IPv4 reply mode, but you do get a successful reply using the RA reply mode, you can infer that a switching path problem exists in the return path, most likely a CEF inconsistency between the linecard and the RP card in an intermediate node. You can reach this conclusion because with RA reply mode, all MPLS Echo Reply packets are punted to the RP to be processed switched.

Quality of Service in AToM

This final section of the chapter covers concepts and configuration on quality of service (QoS). You review the common QoS techniques that are applicable to all Layer 2 protocols over MPLS and then explore the protocol-specific aspects. This section tries to be as platform-independent as possible. However, you might find hardware-specific conditions that preclude support of some QoS features.

The QoS model for AToM follows the Differentiated Services (DiffServ) QoS architecture in Cisco IOS that uses the Modular QoS CLI (MQC). DiffServ defines a scalable QoS architecture that relies on the separation of complex edge versus simple core behaviors. The edge behaviors are summarized in a small number of classes defined in the DiffServ code point (DSCP). MPLS support for DiffServ is defined in RFC 3270. It uses the Experimental bits in the MPLS header, also referred to as class of service (CoS) bits for the few classes that DiffServ uses to which LSPs are mapped.

The MQC model can be summarized as follows:

1. Interesting traffic is defined and classified as one or more classes using the **class-map** command.
2. Policies pertaining to these classes are defined using the **policy-map** command.
3. The policies are applied to either the input or output direction of the traffic flow using the **service-policy** command.

Case Study 9-13: Traffic Marking

The first of the QoS building blocks is the marking of traffic by setting the MPLS Experimental (Exp) bits. You apply the Exp bit setting to both the pseudowire and tunnel labels because of the possibility of PHP, which removes the tunnel label at the penultimate hop. This traffic marking based on the Exp bits is meaningful if the core network performs differentiated treatment of different classes, such as by queuing highest class traffic in a strict priority queue.

Example 9-60 shows how to set the Exp bits for an ATM AAL5 SDU and Cell Relay VC Mode pseudowires on PE1 shown in Figure 9-1.

Example 9-60. *Setting Exp Bits*

```
hostname PE1
!
class-map match-any all_traffic
  match any
!
```

```
policy-map exp3
  class all_traffic
   set mpls experimental 3
!
policy-map exp5
  class all_traffic
   set mpls experimental 5
!
interface ATM4/0.1 point-to-point
 description *** AAL5 SDU AToM to CE1 ***
 pvc 0/100 l2transport
  encapsulation aal5
  xconnect 10.1.1.2 100 encapsulation mpls
  service-policy input exp3
!
interface ATM4/0.2 point-to-point
 description *** Cell VC AToM to CE1 ***
 pvc 0/200 l2transport
  encapsulation aal0
  xconnect 10.1.1.2 200 encapsulation mpls
  service-policy input exp5
```

The **class-map all_traffic** matches all traffic in which the corresponding service-policy is applied. Instead of defined class, you could have used the built-in **class-default** to obtain the same results. The **policy-map** exp3 sets the Exp bits to 3 for all the classified traffic (that is, all traffic). This policy is applied as an input **service-policy** to the ATM AAL5-SDU mode AC. Similarly, the **policy-map** exp5 sets the Exp bits to 5 and is applied to the ATM Cell Relay VC AC. Therefore, all traffic that is incoming into ATM PVC 0/100 and 0/200 is encapsulated with the MPLS Exp bits set to 3 or 5, respectively. You can also perform traffic marking using policing.

You can see this service policy working in Example 9-61 by enabling the **debug mpls packets** in PE2 and sending 5 default size (100 Bytes) PING packets from CE1 to CE2 in each PVC. Do not enable the **debug mpls packets** command in production networks.

Example 9-61. *QoS Traffic Marking Verification*

```
PE2#
*Jun 2 11:26:17.733: MPLS: Fa0/0: recvd: CoS=3, TTL=2, Label(s)=19
*Jun 2 11:26:17.737: MPLS: Fa0/0: recvd: CoS=3, TTL=2, Label(s)=19
*Jun 2 11:26:17.737: MPLS: Fa0/0: recvd: CoS=3, TTL=2, Label(s)=19
*Jun 2 11:26:17.737: MPLS: Fa0/0: recvd: CoS=3, TTL=2, Label(s)=19
*Jun 2 11:26:17.741: MPLS: Fa0/0: recvd: CoS=3, TTL=2, Label(s)=19
PE2#
*Jun 2 11:26:26.793: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.793: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.793: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.793: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
```

```
*Jun 2 11:26:26.793: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.793: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
*Jun 2 11:26:26.797: MPLS: Fa0/0: recvd: CoS=5, TTL=2, Label(s)=21
```

You can see from Example 9-61 that all the MPLS packets are arriving with the VC label only because of PHP, and the TTL of the VC label is 2. The first five MPLS packets have the Exp bits set to 3. These are the five 100-byte PING packets encapsulated in ATM AAL5-SDU mode, and the Exp bits are set by the **policy-map exp3** on PE1. The output of the debug command shows the Exp value as CoS. After these five packets, you see 15 packets with the Exp set to 5. These are the five 100-byte PING packets that are encapsulated in ATM Cell Relay VC mode without cell packing. Each packet is broken into three ATM cells, and therefore three corresponding AToM packets. The Exp bits are set to 5 as defined in the **policy-map exp5** on PE1.

Note

For ATM Cell Relay VP mode with QoS configuration, configure each ATM permanent virtual path (PVP) into its own multipoint ATM subinterface, and apply the service policy to the subinterface. This allows you to apply various service policies with unique policy actions such as marking or policing to the different ATM PVPs. In contrast to ATM PVC configuration, the **atm pvp** command-line interface (CLI) command does not enable a configuration submode.

The case for other Layer 2 transports is analogous, applying the service-policy in the main interface for ATM CRoMPLS Port mode, High-Level Data Link Control over MPLS (HDLCoMPLS), PPP over MPLS (PPPoMPLS), Ethernet over MPLS (EoMPLS), and in the subinterface for EoMPLS VLAN mode and ATM cell relay over MPLS (CRoMPLS) VP mode. The case for Frame Relay over MPLS (FRoMPLS) is slightly different. It is covered in "Case Study 9-17: ===Layer 2-Specific Matching and Setting."

Case Study 9-14: Traffic Policing

Policing CE traffic is similar to marking. The difference is the policy action taken with the classified traffic. The following two modes support policing actions for Frame

Relay, ATM, and Ethernet:

- **Single-rate policer** Policed traffic is checked against a single committed information rate (CIR).
- **Dual-rate policer** Policed traffic is checked against two rates: CIR and peak information rate (PIR). This policer for IP networks is modeled after the Frame Relay policer.

The two policing modes can have color-awareness enabled or disabled:

- **Color-blind** All the policed traffic is treated equally and policed against the same rate or rates.
- **Color-aware** A user-defined criteria preclassifies policed traffic and checks it against different rates depending on the preclassification result. To this extent, you can use the **conform-action** and **exceed-action** commands under the **police** configuration mode to color traffic to be policed. Packets that are not classified under either the **conform-action** or **exceed-action** class belong to the violate-action class.

Example 9-62 shows a single bucket color-blind policing action.

Example 9-62. *Single Bucket, Color-Blind Policing*

```
hostname PE1
!
class-map match-any all_traffic
  match any
!
policy-map policing
  class all_traffic
   police cir 128000
     conform-action set-mpls-exp-transmit 5
     exceed-action drop
!
interface ATM4/0.2 point-to-point
 description *** Cell VC AToM to CE1 ***
 pvc 0/200 l2transport
  encapsulation aal0
  xconnect 10.0.0.203 200 encapsulation mpls
  service-policy in policing
```

A dual rate color-aware policer configuration is included in Example 9-66 in Case Study 9-17. Cisco IOS implements the single rate three-color policer based on RFC 2697 and the dual rate three-color policer based on RFC 2698.

Case Study 9-15: Queuing and Shaping

In general, the following features are supported for queuing and shaping actions:

- **Low-latency queuing (LLQ), also called priority queuing (PQ)** The LLQ is a strict priority first-in, first-out (FIFO) queue. Strict priority queuing allows delay-sensitive data to receive a preferential queuing treatment by being dequeued and serviced before any other queues.

- **Class-based weighted fair queuing (CBWFQ)** CBWFQ provides fair queuing based on defined classes with no strict priority. The weight for a packet that belongs to a specific class is given from the bandwidth that you assigned to the class.

- **Byte-based weighted random early detection (WRED)** WRED drops packets selectively based on IP precedence. The higher the IP precedence, the less likely packets are to be dropped.

You can see egress queuing policies to provide CIR guarantees in Example 9-63.

Example 9-63. *Queuing Configuration for CIR Guarantees in Frame Relay Pseudowires*

```
!
hostname PE1
!
class-map match-all CustomerA
  match fr-dlci 100
class-map match-all CustomerB
  match fr-dlci 200
!
policy-map CIR_guarantee
  class CustomerA
    bandwidth 128
  class CustomerB
    bandwidth 256
!
interface Serial3/1
 no ip address
 service-policy output CIR_guarantee
 encapsulation frame-relay
 frame-relay intf-type dce
!
```

In this example, customers use two seperate DLCIs in the same Frame Relay interface. Using a different class-map for each DLCI allows you to apply CBWFQ with

the **bandwidth** command to each class for each DLCI. In addition, FRoMPLS supports traffic shaping and ATMoMPLS supports class-based shaping on ATM VCs.

You can accomplish per-class traffic shaping for ATM PVC and PVP ACs with the ATM PVC and PVP service type configuration using the following commands:

- **cbr** {**PCR**}
- **ubr** {**PCR**}
- **vbr-rt** {**PCR**} {**SCR**} [**MBS**]
- **vbr-nrt** {**PCR**} {**SCR**} [**MBS**]

> **Note**
>
> The distributed forms of these features are supported in distributed switching platforms, such as the Cisco 7500 series.

All queuing and shaping features are applied in the outbound direction. Marking and policing are input policies.

Case Study 9-16: Intermediate Markings

In this case study, you learn how to apply QoS actions on an egress interface based on matching criteria used at the ingress interface. This can be useful, for example, to match traffic based on MPLS Exp bits from the MPLS network and perform a policy action on packets going out of the egress interface.

Two internal markings called qos-group and discard-class preserve the classification that happened before the MPLS header popping operation. This classification would otherwise be lost when applying an output service-policy on the AC. The QoS group ID identifies an internal class, and the Discard Class identifies an internal precedence. These two intermediate markings "remember" the classification from the MPLS network. You can use the intermediate step to mark traffic from the MPLS network with a qos-group ID and use this qos-group ID to apply policy actions on the egress interface. Example 9-64 shows you how to set the ATM CLP bit in cells going toward the CE device based on the MPLS Exp bits received from the P router.

Example 9-64. *Intermediate Marking*

```
!
hostname PE1
```

```
!
class-map match-all exp3
  match mpls experimental 3
class-map match-all qosg_class
  match qos-group 1
!
policy-map clp1
  class qosg_class
   set atm-clp
policy-map qosg
  class exp3
   set qos-group 1
!
interface Serial4/0
 ip unnumbered Loopback0
 mpls ip
 service-policy input qosg
!
interface ATM5/0
 no ip address
 pvc 0/100 l2transport
  encapsulation aal5
  xconnect 10.1.1.2 100 encapsulation mpls
  service-policy out clp1
 !
!
```

You can see from Example 9-64 that the service-policy qosg is applied to traffic coming into the PE device from the P router on interface Serial4/0. With this service-policy, MPLS packets with Exp = 3 (from the class exp3) are marked with the qos-group of 1. On AC PVC 0/100 in ATM5/0, the outbound service-policy clp1 is applied. This service policy sets the ATM CLP bit for cells that were previously marked with a qos-group of 1. With this internal qos-group ID marking, a classification is conveyed from one interface to another.

Case Study 9-17: Layer 2Specific Matching and Setting

Different Layer 2 protocols comprise different characteristics and sometimes have an impact on the QoS configuration. This case study discusses the protocol-specific QoS characteristics and configuration. Table 9-2 outlines the different matching and setting criteria based on Layer 2 protocol.

Table 9-2. *Layer 2-Specific Matching and Marking Criteria*

Layer 2 Protocol	Matching	Setting
Ethernet	**match cos** **match vlan**	**set cos**
Frame Relay	**match fr-de** **match fr-dlci**	**set fr-de** **set fr-fecn-becn**
ATM	**match atm clp**	**set atm-clp**

Ethernet over MPLS QoS

Ethernet frames that use IEEE 802.1q encapsulation contain the 802.1p CoS bits, which you can use for traffic classification (see Example 9-65).

Example 9-65. *Traffic Classification*

```
hostname PE1
!
class-map match-any cos2
  match cos 2
!
policy-map eompls3
  class cos2
   set mpls experimental 3
  class class-default
   set mpls experimental 0
!
interface Ethernet0/0.10
 description *** To CE1 ***
 encapsulation dot1Q 10
 xconnect 10.1.1.2 10 encapsulation mpls
 service-policy input eompls3
```

In Example 9-65, the Exp bits are set to 3 for traffic matching a CoS value of 2, and the rest of the traffic always matches the default class, which sets the Exp bits to 0. Besides traffic marking, policing is also supported.

In EoMPLS, the service policy is applied on the main interface for port mode EoMPLS and in the subinterface for VLAN mode EoMPLS.

Some platforms support a **match vlan** classification directive for a VLAN range, as follows:

```
class-map match-any ethernet
 match vlan 3-5
```

However, no platforms support a **set vlan** policy or include a **set vlan** command. The VLAN rewrite configuration was covered in detail in Chapter 7, "LAN Protocols over MPLS Case Studies."

Frame Relay over MPLS QoS

With FRoMPLS, you can match traffic using Frame Relay specific fields. The following QoS directives are specific to Frame Relay:

Matching:

match fr-de

match fr-dlci

match fr-dlci range

Setting:

set fr-de

set fr-fecn-becn

Example 9-66 shows a dual-rate color-aware policer using Frame Relay-specific fields.

Example 9-66. *Dual-Rate Color-Aware Policer*

```
hostname PE1
!
class-map match-any FR_DLCI_100
  match fr-dlci 100
class-map match-any FR_DE0
  match not fr-de
!
policy-map FR_Policing
  class FR_DLCI_100
   police cir 64000 pir 128000
     conform-color FR_DE0
```

```
    conform-action set-mpls-exp-transmit 5
    exceed-action set-mpls-exp-transmit 2
    violate-action drop
  class class-default
   set mpls experimental 0
```

In the FR_DE0 class, the **not** qualifier matches traffic that does not have the DE bit set. The FR_DE0 class is used for the color.

This policer allows policing traffic according to the color classification of whether the discard eligible (DE) bit is set in incoming Frame Relay frames. With this policy, only packets that do not have DE set are policed against CIR and PIR. Packets that do have the DE bit set are not treated as conforming. They are policed against PIR to determine whether they are exceeding or violating.

To apply this policy, you need to create a subinterface effectively to map to the Frame Relay PVC. This is accomplished with the command **switched-dlci** in Cisco 12000 series router platforms (see Example 9-67).

Example 9-67. *Mapping a Subinterface to the Frame Relay PVC*

```
interface POS4/0
encapsulation frame-relay cisco
!
interface POS4/0.1 point-to-point
 switched-dlci 100
 service-policy input FR_Policing
!
connect frompls101 POS4/0 100 l2transport
 xconnect 10.0.0.203 70 encapsulation mpls
```

Applying the FR_Policing policy to the point-to-point subinterface POS4/0.1 effectively applies the policy to the local AC that is defined with the **connect** command.

ATM over MPLS QoS

Currently, the only ATM-specific field for matching or setting is the cell loss priority (CLP) bit in the ATM Cell header. For ATM over MPLS, you can apply a service policy under an interface, a subinterface, or a PVC.

You can use the commands **match atm clp** and **set atm-clp** to match and set the ATM CLP bit, respectively.

Example 9-68 demonstrates how to use these two commands.

Example 9-68. *Matching and Setting ATM CLP*

```
!
hostname PE1
!
class-map match-all not-clp
  match not atm clp
policy-map exp-4
  class not-clp
   set mpls experimental 4
policy-map atm-clp
  class class-default
   set atm-clp
!
interface ATM5/0
pvc 0/100 l2transport
  encapsulation aal5
  xconnect 10.1.1.2 100 encapsulation mpls
  service-policy input exp-4
  service-policy output atm-clp
 !
!
```

You can see in Example 9-68 that because of the qualifier **not**, the class-map **not-clp** matches on all incoming ATM cells in PVC 0/100 that have the CLP bit clear. All AToM packets that encapsulate these matched cells have the MPLS Exp bits set to a value of 4. In addition, the service policy **atm-clp** that is applied in the same PVC in the outbound direction is setting the ATM CLP bit for all cells out of the PVC, because the **set atm-clp** directive is applied to the **class-default** for all outbound ATM cells.

Summary

Deploying pseudowire emulation services in MPLS networks can be a rather sophisticated task when you take factors such as routing, network resource utilization, and path protection into account. This chapter discussed some of the most common but complex deployment scenarios you might encounter when offering pseudowire emulation services, as follows:

- Load share across multiple equal-cost paths.
- Forward pseudowire traffic through a preferred path using the IP Routing protocol.
- Direct pseudowire traffic through an explicit path using an MPLS traffic engineering tunnel.
- Dynamically route pseudowire traffic through an MPLS traffic engineering tunnel with a specific bandwidth requirement.
- Protect pseudowire traffic from pocket loss with the MPLS traffic engineering fast reroute capability.
- Provide inter-AS pseudowire connectivity through dedicated circuits.
- Provide inter-AS pseudowire connectivity using BGP IPv4 label distribution and IGP redistribution.
- Provide inter-AS pseudowire connectivity and improve scalability with BGP IPv4 label distribution and IBGP peering.
- Authenticate for pseudowire signaling.
- Verify pseudowire data connectivity and provide troubleshooting.
- Configure general and protocol-specific pseudowire QoS.

This is not intended to be an exhaustive list of all possible deployment scenarios for AToM. Rather, it serves as a building block for more complex deployment of a large scale.

Part IV: Layer 2 Tunneling Protocol Version 3

Chapter 10. Understanding L2TPv3

This chapter covers the following topics:

- Universal Transport Interface
- L2TPv3

As mentioned in Chapter 3, "Layer 2 VPN Architectures," Layer 2 Tunnel Protocol Version 3 (L2TPv3) is an IP-based solution in the Cisco Unified VPN Suite that provides pseudowire emulation for a variety of Layer 2 protocols, including Ethernet, High-Level Data Link Control (HDLC), PPP, Frame Relay, and ATM. The base L2TPv3 protocol, which includes the control protocol and data encapsulation, is defined in the Layer 2 Tunnel Protocol Extensions (l2tpext) working group. At the time of this writing, the L2TPv3 base draft had not reached RFC status. Supplemental specifications that are particular to the data link protocols such as ATM are defined in separate drafts.

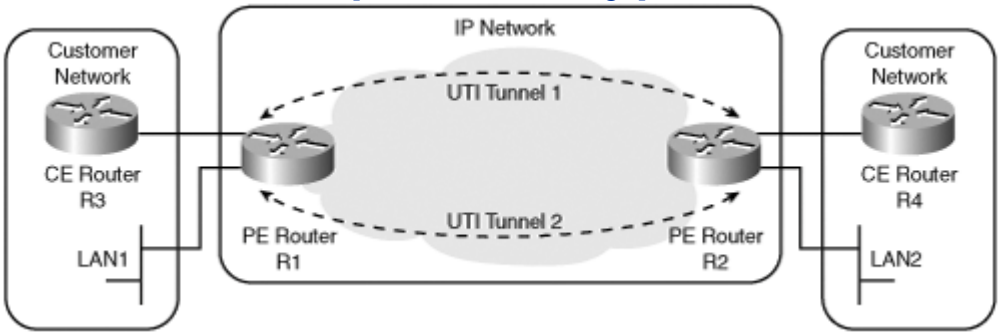
This chapter examines the base L2TPv3 protocol by first reviewing the history of its development from its prestandard beginnings. This exploration into the protocol's evolution is then followed by an examination of L2TPv3's data encapsulation and control channel signaling.

Universal Transport Interface: L2TPv3's Predecessor

The prestandard predecessor for L2TPv3 was a Cisco proprietary protocol known as Universal Transport Interface (UTI). UTI's goal was to provide a high-performance IP-based tunneling mechanism for circuit-like Layer 2 connectivity (that is, pseudowire) over a packet-based core. UTI has no inherent signaling mechanism. Layer 2 frames from the attachment circuits are encapsulated with the necessary UTI formatting and are forwarded towards the remote endpoint. After the received frame is validated, the original data-link payload is forwarded out the appropriate attachment circuit. Figure 10-1 illustrates this connectivity model.

Figure 10-1. *UTI Connectivity Model*

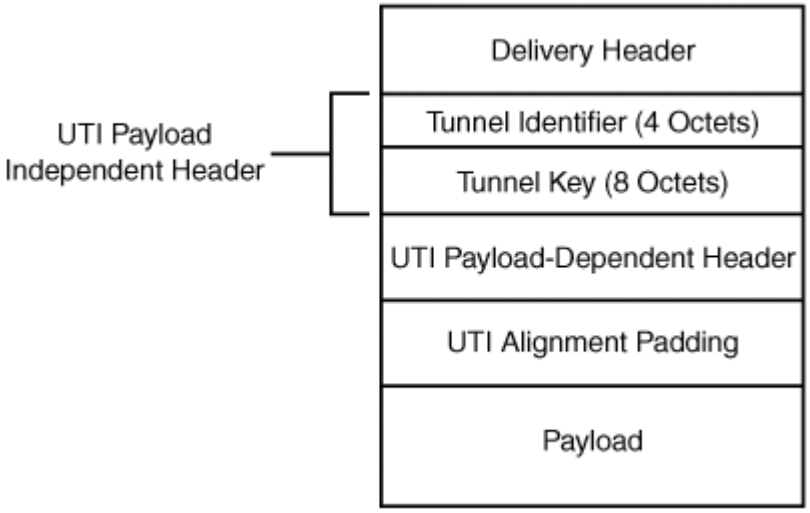
[View full size image]



R1 and R2 are provider edge (PE) routers with connectivity to each other through an IP core. These PE routers provide pseudowire connectivity via two separate UTI tunnels: Tunnel 1 for the serial line connectivity between the customer edge (CE) routers, R3 and R4, and Tunnel 2 for Ethernet connectivity between LAN 1 and LAN 2. Assuming the serial lines are using Frame Relay encapsulation, a Frame Relay frame from R3 is encapsulated with a UTI header for UTI Tunnel 1 and an IP header with the destination address of R2. After R2 verifies the UTI header contents, it de-encapsulates the original Layer 2 payload and sends it to R4. Likewise, LAN 1 and LAN 2 are essentially bridged across UTI Tunnel 2 in a similar fashion.

UTI also attempts to optimize performance by avoiding suboptimal tunnel identification and parsing schemes that are present in other tunneling protocols. For example, generic routing encapsulation (GRE) tunnel identification requires a lookup on a combination of the source and destination address pair or tunnel key depending on RFC implementation: RFC 2784, RFC 1701, or RFC 2890. UTI's encapsulation shown in Figure 10-2 is designed to avoid some of the overhead that is required in tunnel identification and parsing by means of a tunnel ID that identifies the tunnel context on the de-encapsulating system.

Figure 10-2. *UTI Encapsulation*



The UTI encapsulation consists of the following fields:

- **Delivery Header** The Delivery Header is the header that carries the UTI packet across the packet core. Although this header can be an IPv4 or IPv6 header, the initial Cisco implementation supports only an IPv4 header without IPv4 options and an IP protocol number of 120. Fragmentation is not supported, so the IPv4 Don't Fragment (DF) bit is set. Therefore, the IP MTU of any intermediate links along the tunnel path should be sufficiently large to carry the largest Layer 2 packet.

- **UTI Payload Independent Header** The Payload Independent Header is composed of the following two subcomponents:

  Tunnel Identifier The Tunnel Identifier, sometimes referred to as a Session Identifier, is a 4-octet value that distinguishes the tunnel at the de-encapsulating endpoint. The Tunnel Identifier represents a unidirectional session. A bidirectional tunnel has two identifiers: a local and remote value. If the tunnel identifier does not match the tunnel value on the de-encapsulating endpoint, the packet is discarded. The UTI specification reserves value 0x00000000 and limits the user-defined tunnel identifier to the first 10 bits, leaving 1023 available values.

  Tunnel Key The Tunnel Key is an 8-octet field used to avoid misconfiguration or malicious attempts that lead to inserting unwanted traffic into the Layer 2 stream. The tunnel key value must match on both ends of the de-encapsulating endpoint; otherwise, the packet is discarded. The tunnel key is configured using a high key (the most significant 4 bytes) and low key value (the least significant 4 bytes).

- **UTI Payload-Dependent Header** The Payload-Dependent Header contains any payload information that is essential for the egress PE to properly forward the original Layer 2 frame toward the CE. The Cisco implementation does not define this header value and is not used.

- **UTI Alignment Padding** Alignment Padding ensures that the payload is aligned to a byte boundary that might assist implementations to more efficiently parse the payload. Although this field is defined in the UTI specification, Alignment Padding was never used in the initial Cisco implementation.

- **Payload** Payload is the original data link layer frame transported by UTI. This can be a Frame Relay, Ethernet, HDLC, or PPP frame.

Although UTI fulfilled its original goal of providing pseudowire connectivity, it had some limitations. As mentioned earlier in this section, one of UTI's restrictions is that it does not support IP fragmentation; therefore, the end-to-end packet-switched core MTU must be greater than the size of the UTI encapsulated packet.

Furthermore, although UTI eventually added support for an optional keepalive mechanism, this only detected whether the remote endpoint was no longer reachable and had no granularity at a pseudowire level. Because no inherent signaling method was available, UTI could not signal an individual pseudowire state. For example, in Figures 10-1, if R3's Frame Relay permanent virtual circuit (PVC) failed, the PE routers would not have a way to signal that information to each other so that R2 could signal that information via Local Management Interface (LMI) to R4. Instead, R4 and R2 would consider the Frame Relay PVC to be active, and R2 would continue sending Frame Relay traffic to the opposing PE router.

Another UTI limitation was the lack of a signaling protocol, which required that Tunnel Identifiers and Tunnel Keys be manually configured and preprovisioned on each PE router for each pseudowire. Although some providers might prefer the simplicity of manual provisioning, this can be operationally infeasible for large deployments.

Finally, UTI was a Cisco proprietary protocol. As such, it prohibited multivendor interoperable implementations. An open standards-based IP pseudowire solution that overcame these limitations was required.

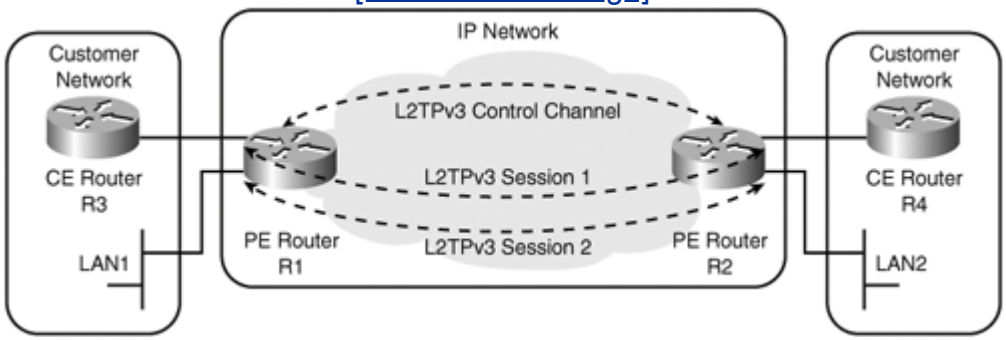
Introducing L2TPv3

L2TPv3 is the IETF standard's track successor to UTI for Layer 2 Tunneling. To overcome some of the limitations that UTI possessed, L2TPv3 built upon UTI's encapsulation format and coupled it with an optional signaling mechanism that heavily borrowed from L2TPv2's control plane to provide pseudowire connectivity (L2TPv2 is described in RFC 2661, "Layer 2 Tunneling Protocol 'L2TP'").

Figure 10-3 shows the L2TPv3 connectivity model. L2TPv3's control messages are sent inband using the same packet core path as the data traffic. Each pseudowire is maintained through separate L2TPv3 data sessions similar to UTI tunnels: one for the Frame Relay PVC between R3 and R4, and a separate session for connectivity between LAN 1 and LAN 2.

Figure 10-3. *L2TPv3 Connectivity Model*

[View full size image]



L2TPv3's signaling protocol is optional. Therefore, it can operate in the same way as UTI: manually defined static sessions with or without a keepalive mechanism for dead peer detection. However, with its signaling protocol enabled, L2TPv3 can signal individual attachment circuit states per pseudowire and dynamically negotiate values for Session Identifiers and Key values without having predefined values on each PE router. The base L2TPv3 protocol essentially accomplishes this by extending L2TPv2's control channel signaling by supporting additional attributes that are passed in message formats referred to as Attribute-Value Pairs (AVPs). The next two sections examine L2TPv3's data encapsulation and control plane in more detail.

L2TPv3 Data Encapsulation

As mentioned in Chapter 2, "Pseudowire Emulation Framework and Standards," the IETF Pseudowire Emulation Edge to Edge (PWE3) group laid some of the framework and specified requirements for a pseudowire emulation protocol. One of the architecture aspects explored in the PWE3 architecture draft was the Pseudowire

Protocol Layering Model. To understand L2TPv3's frame encapsulation, this section describes each of the encapsulation components of L2TPv3 and, where applicable, relates it to the Pseudowire Emulation Protocol Layer subset shown in Figure 10-4.

Figure 10-4. *Pseudowire Emulation Protocol Layers*

Packet-Switched Network (PSN Layer)
Demultiplexing Sublayer
Encapsulation Sublayer (Optional)
Payload

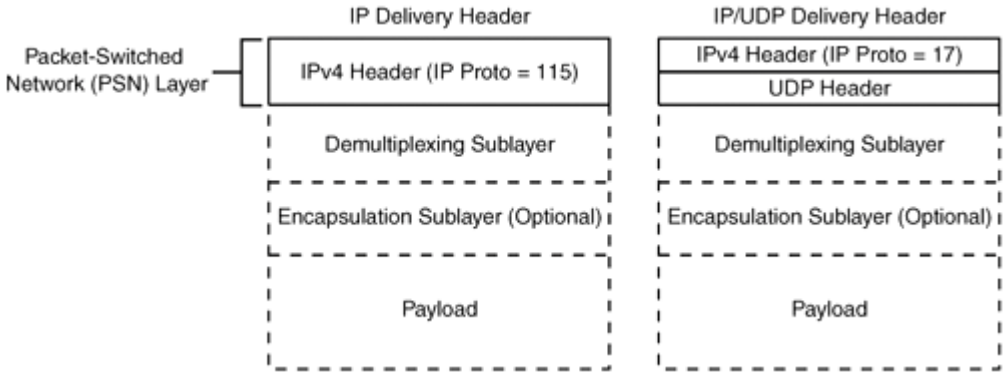
Packet-Switched Network Layer

Unlike Any Transport over MPLS (AToM), which uses an outer MPLS label-to-label switch traffic to the far-end PE, L2TPv3 expects an IP-based packet core (IPv4 or IPv6). The L2TPv3 draft specifies two alternative delivery header encapsulations, illustrated in Figure 10-5:

- Plain IP
- IP/UDP

Figure 10-5. *L2TPv3 Packet-Switched Network Layer*

[View full size image]



L2TPv3 with IPv4 encapsulation uses a standard 20-byte IPv4 header without options, using an IP protocol ID of 115. Unlike UTI, however, L2TPv3 does support fragmentation utilizing a Path Maximum Transmission Unit (PMTU) discovery mechanism. This mechanism is discussed in Chapter 13, "Advanced L2TPv3 Case Studies."

L2TPv3 with an IPv4/UDP encapsulation uses a standard 20-byte IPv4 header without options and contains an IP protocol value of 17 to signify a UDP payload. The IP header is then followed by a UDP header with the requirement that the UDP destination port be 1701 for initial control channel signaling. The IP or IP/UDP header in L2TPv3 satisfies the packet-switched network (PSN) layer mentioned in Chapter 2.

Compared to IPv4 encapsulation, one of the advantages of using IPv4/UDP encapsulation is that it is friendlier to applications such as Network Address Translation (NAT). Furthermore, IPv4 encapsulation provides only a header checksum, whereas UDP also offers a checksum that verifies payload integrity. This could be an issue especially when you are dealing with and confirming the reliability of L2TPv3 control messages, which are covered in the section "L2TPv3 Control Connection," later in this chapter.

Note

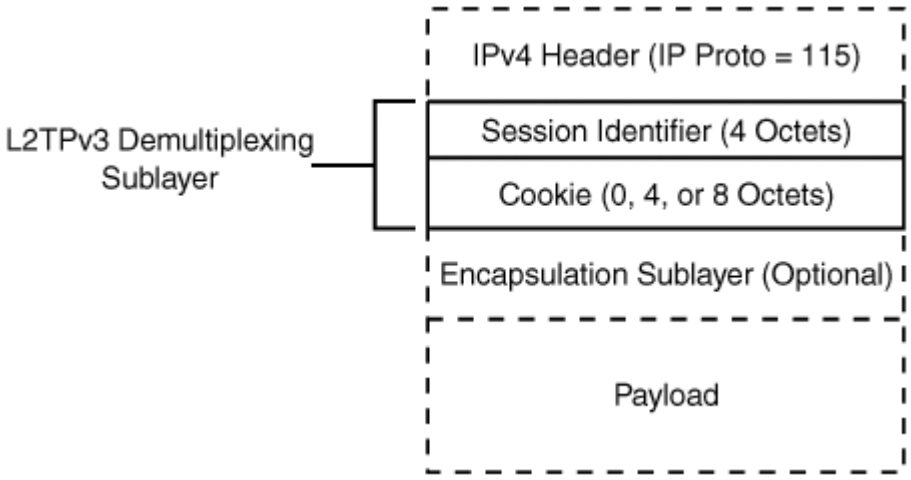
The Cisco implementation of L2TPv3 only supports IPv4 header encapsulation for the L2TPv3 Delivery Header. As such, the remainder of this chapter focuses on the IPv4 L2TPv3 implementation.

Demultiplexing Sublayer

The L2TPv3 Demultiplexing Sublayer field allows the IPv4 tunnel (an IPv4 source and destination pair) to carry and demultiplex multiple pseudowires. This field is the

equivalent of the Demultiplexing Sublayer described in Chapter 2. L2TPv3 supports demultiplexing through a combination of a Session Identifier and Cookie values shown in Figure 10-6.

Figure 10-6. *L2TPv3 Demultiplexer Field*



Note

Figure 10-6 illustrates the L2TPv3 Demultiplexer field for an IP implementation. The L2TPv3 Demultiplexer field in an IP/UDP implementation contains fields in addition to the Session Identifier and Cookie to differentiate and coexist with other Layer 2 tunneling protocols such as L2TPv2 and Layer 2 Forwarding (L2F).

The Session Identifier is a 4-byte field with a nonzero value that identifies a specific L2TPv3 session between two tunnel endpoints. A Session ID value of 0 is reserved for control channel communication. Like the Tunnel Identifier in UTI, the Session Identifier is locally significant; therefore, it utilizes a local and remote value to represent a bidirectional session.

The Cookie field fulfills the same role as the UTI Tunnel Key. It is an optional layer containing a variable length field (maximum of 64 bits) that protects against inadvertent insertion of Layer 2 frames into the tunnel through either misconfiguration or malicious blind attacks. When the Cookie field is negotiated through the control channel, it is consistent throughout the duration of the session.

From a security perspective, the Cookie provides a lightweight security option on the L2TPv3 payload; therefore, it covers just a small subset of attacks known as

blind insertion attacks. The blind insertion attack assumes that the attacker can inject data into the core but cannot sniff out data within the PSN core. Assuming that the Session Identifier is predictable, the only barrier restricting the attacker from injecting traffic into a Layer 2 stream is to guess this random Cookie value. The maximum field length is 64 bits because such a value makes it infeasible from a resource perspective to perform a brute force attack. For example, a brute force attack to insert a 40-byte spoofed packet into a tunnelassuming the attacker can inject data at an OC-192 ratewould require approximately 18,000 years.

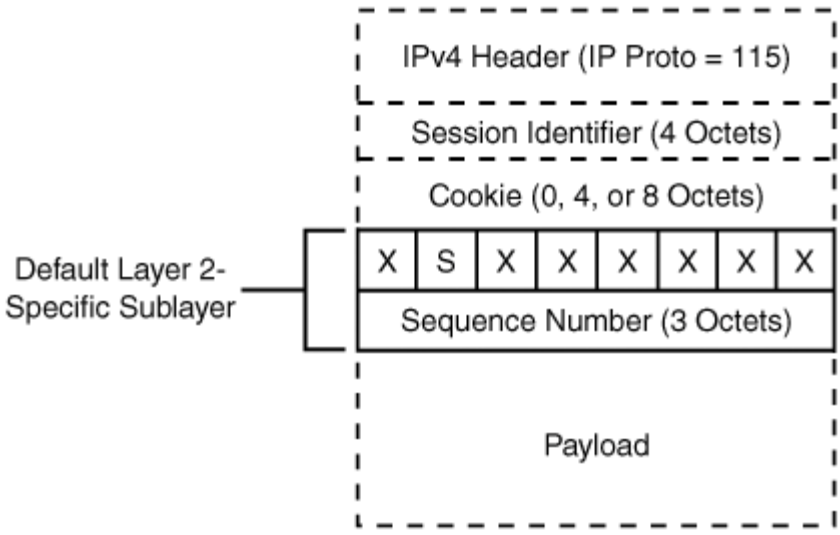
The Cisco implementation allows for the Session Identifier and Cookie to be either manually predefined on each tunnel endpoint or negotiated over the L2TPv3 control channel. The Cookie field can be negotiated to a 0-, 4-, or 8-byte field size, depending on the platform restrictions.

Encapsulation Sublayer

L2TPv3 uses an optional field, referred to as an Layer 2-Specific Sublayer, to convey information that is not carried in the Layer 2 Payload but that is required for the tunnel de-encapsulating endpoint to properly reconstruct the Layer 2 Payload and send the frame to the CE device. This field is the equivalent of the optional Encapsulation Sublayer that is defined in the Pseudowire Emulation Protocol Layers.

The L2TPv3 base draft specifies a default Layer 2-Specific Sublayer illustrated in Figure 10-7 that you use if it meets the Layer 2 Payload requirements. Otherwise, you can define alternate Layer 2-Specific Sublayers and use them as negotiated through an Layer 2-Specific Sublayer Type AVP control message. A Data Sequencing AVP is signaled during session negotiation to determine whether sequencing is required or what type of traffic needs to be sequenced.

Figure 10-7. *L2TPv3 Default Layer 2-Specific Sublayer*



A Sequence bit (S-bit) set to 1 indicates that the 24-bit Sequence Number field contains a valid value. When the S-bit is not set, the de-encapsulating endpoint must ignore the Sequence Number field. The Sequence Number in the remainder of the default Layer 2-Specific Sublayer is a 24-bit field containing a free-running counter that starts at 0.

If sequencing is enabled, the current expected sequence number on the receiving device is equal to the previous sequence number of the last in-order packet plus 1. Sequenced L2TPv3 data is accepted if the stored sequence number is equal to or greater than the current expected sequence number. Any other packets that do not fit this description are either out-of-order or duplicate packets and are discarded. Because of the finite range of sequence numbers, you must take the wrapping of the field into account by tracking a window of sequence numbers greater than the current expected value. The recommended default range is equal to half of the available sequence number space ($2^{24}/2=8388608$). For example, assuming a sequence number field of 24 bits, the window range of "new" sequence numbers for the current sequence number of 10,040,243 is 10,040,244 through 1,677,216 and 0 through 3,303,269.

The Sequence Field allows you to detect lost, duplicate, or out-of-order packets for an individual session. However, the criticality of preserving the correct ordering depends on the sensitivity of the encapsulated Layer 2 traffic. If the Layer 3 traffic in the Layer 2 tunneled frame is IP, the upper layer protocol might handle out-of-sequence packets. Therefore, the aforementioned Data Sequencing AVP supports the following three options:

- No sequencing.
- Non-IP data requires sequencing.
- All data packets require sequencing.

The Cisco sequencing implementation only drops out-of-order frames and does not attempt to reorder out-of-sequence packets. You should understand the implications of this behavior prior to enabling sequencing relative to the Layer 2 protocol.

L2TPv3 Control Connection

L2TPv3 supports an optional control connection mechanism, which handles peer capability negotiation and detection in addition to pseudowire creation, maintenance, and teardown. This section explores L2TPv3's control connection by examining how control messages are encapsulated and what the different negotiation phases are for control channel initialization and session negotiation.

Unlike AToM, which uses link Layer Distribution Protocol (LDP) for LSP tunneling (PSN tunnel signaling) and directed LDP for virtual circuit (VC) label distribution

(pseudowire/PE maintenance), L2TPv3 utilizes a single reliable, inband control plane for both purposes. This control plane setup phase begins with an L2TP Control Connection (sometimes referred to in L2TP terminology as the L2TP tunnel) establishment phase for advertising and negotiating capabilities between peers. After the L2TP Control Connection is established, individual pseudowire sessions (referred to in L2TP terminology as L2TP sessions) are set up in the Session Negotiation phase as needed based on the attachment circuit state.

Note

There are three variations on the control plane implementation of L2TPv3. L2TPv3 in its simplest mode of operation, known as *Manual Mode*, obviates the need for a control plane protocol and simply requires predefined session IDs and cookies. The second variation, called *Manual Mode with Keepalive*, negotiates the Control Connection phase but not the Session Negotiation phase. This offers a simple dead-peer detection mechanism that is similar to what is available in UTI with keepalives. *Dynamic Mode* negotiates both the Control Connection phase as well as the Session Negotiation phase for each pseudowire session.

As mentioned earlier in the section "Introducing L2TPv3," L2TPv3 essentially borrowed from L2TPv2's control channel signaling and expanded upon it by defining additional AVPs. These AVPs are used as an extensible mechanism to identify message types within the control channel. In addition to identifying the nature of the PW session or control channel, the control messages can indicate or define the operational state of the attachment circuits. The next sections describe control message encapsulation and control channel signaling in more detail.

Control Message Encapsulation

Because the L2TPv3 control channel is sent inband with L2TPv3 data packets, it is necessary to have some method of differentiating control channel messages from data packets. To understand how this is accomplished, you must examine the control message formatting. Figure 10-6 and 10-7 examined the encapsulation for L2TPv3 data messages only. The formatting of L2TPv3 control messages over IP differs slightly, as shown in Figure 10-8.

Figure 10-8. *L2TPv3 Control Channel Encapsulation over IP*

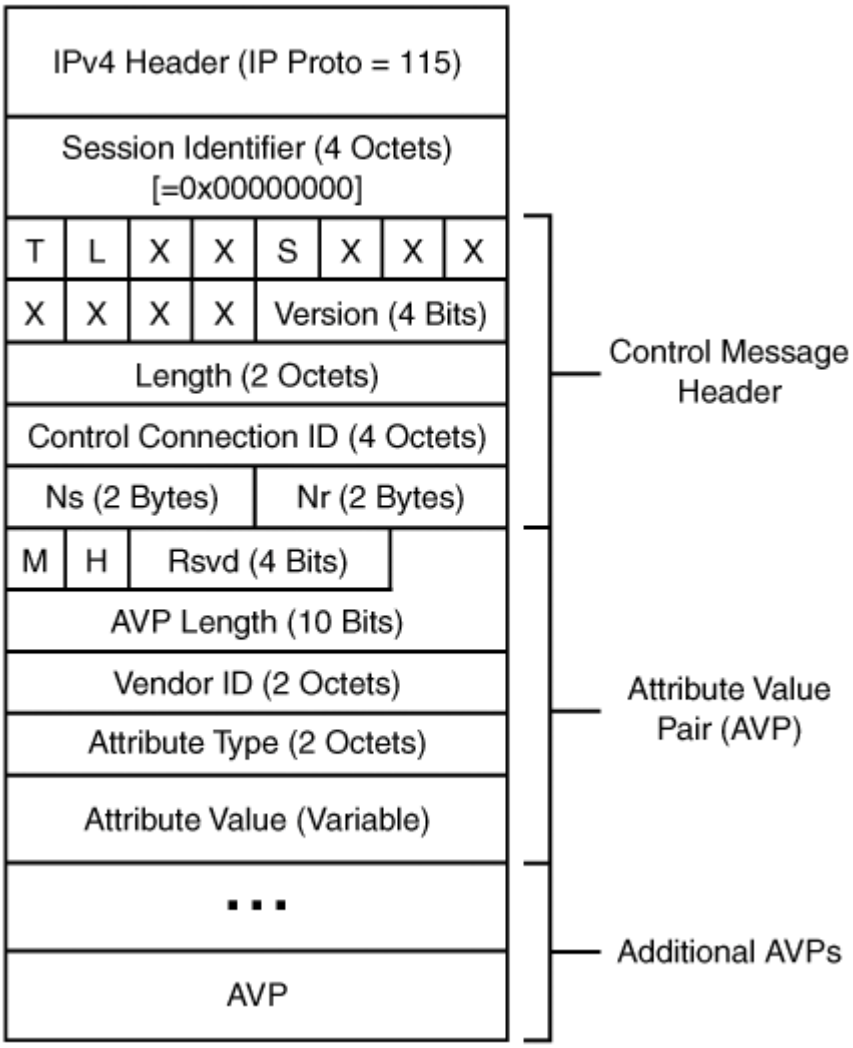


Figure 10-8 illustrates the encapsulation for L2TPv3 Control Messages, assuming that an IPv4 header is used. As described in the earlier "Demultiplexing Sublayer" section, L2TPv3 data packets utilize a nonzero Session Identifier. The Session Identifier value of 0 is reserved for control channel messages and distinguishes data packets from control messages. The remaining fields are unique to the control message encapsulation and are examined individually:

- **T, L, S** You must set the Type bit (T-bit) to 1 to indicate that this is a control message. You must also set the Length bit (L-bit) and Sequence bit (S-bit) to 1 to indicate that length and sequence numbers are present in the L2TPv3 control message header. Do not confuse the sequence numbers with the Sequence Number field in the Layer 2-Specific Sublayer. The former are sequence numbers that are used in the control message header for reliable delivery.

- **Version** The Version field indicates which version of L2TP is in use. Set this value to 3 to indicate L2TPv3.

- **Length** The Length field indicates the total size of the control message calculated from the beginning of the message starting with the T-bit.

- **Control Connection ID** The Control Connection Identifier contains a locally significant field to represent the control channel (L2TPv3 tunnel). The nonzero Control Connection IDs are exchanged during the L2TP Control Channel phase by using the Assigned Control Connection ID AVP.

- **Ns** Ns, or the sequence number sent, indicates the sequence number for this control message. This field begins at 0 and increments by 1 for each control message that is sent to the peer.

- **Nr** Nr is the sequence number expected to be received in the next control message. Ns and Nr provide a simple sliding window mechanism to handle control message transmission, retransmission, and detection of lost or duplicate control message packets.

Following the control message header are one or more AVPs. Each AVP follows a consistent format that contains the following fields:

- **M** When the Mandatory bit (M-bit) is set, it indicates that the associated Control Connection or PW Session must be shut down if the recipient does not recognize this AVP. If a Control Connection occurs, a Stop Control Connection (STOPCCN) message is sent. If a PW Session occurs, a Call Disconnect Notification (CDN) message is sent.

- **H** The Hidden bit (H-bit) indicates to the recipient whether the AVP content is passed in clear text or obfuscated in some manner to hide sensitive information. For this AVP encryption to occur, a shared secret must be defined on both endpoints, Control Message Authentication must be enabled, and a Random Vector AVP must be sent.

- **AVP Length** The AVP Length field indicates the length of the entire AVP, as highlighted in Figure 10-8.

- **Vendor ID** The Vendor ID is a 2-byte field that follows Internet Assigned Numbers Authority (IANA) assigned values that are defined in RFC 1700 in the "SMI Network Management Private Enterprise Codes" section. This allows vendors to define private Attribute Types. A Vendor ID field of 0 represents that this Attribute Type is an Internet Engineering Task Force (IETF) adopted attribute value that is defined in the L2TPv3 base draft.

- **Attribute Type** The Attribute Type contains a 2-byte field representing the Attribute Message. You must interpret this field's value relative to the Vendor ID field.

- **Attribute Value** The Attribute Value contains the actual content of the defined Vendor ID Attribute Type. The length of this field is the Attribute Length minus 6 bytes for Attribute Header fields.

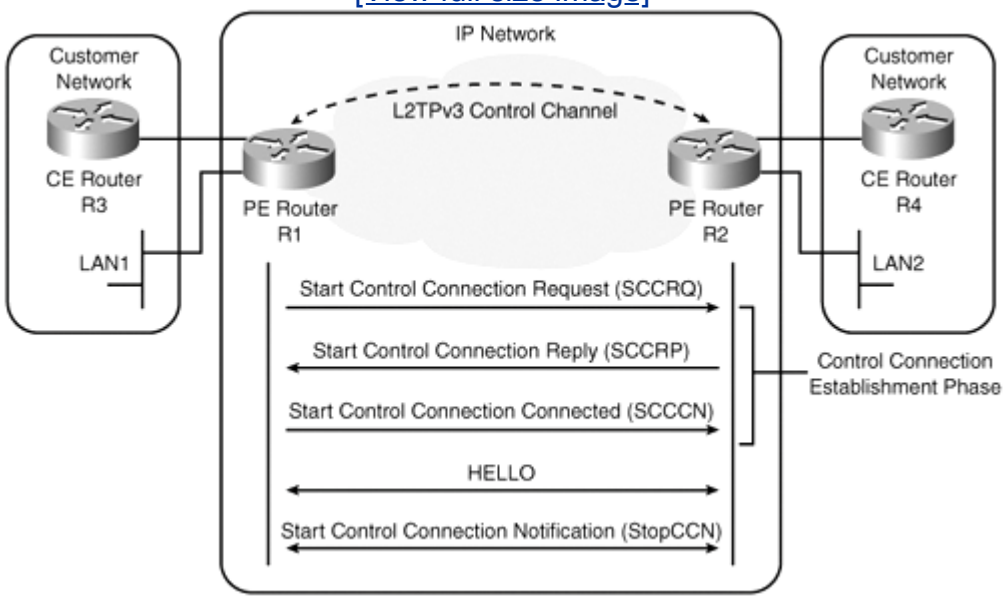
L2TPv3 Control Channel Signaling

Control channel signaling operates in two phases: control connection establishment followed by session establishment (optional).

Figure 10-9 builds upon the network layout in Figure 10-3 and shows the control connection establishment that is required to build the control channel between the two L2TPv3 endpoints, R1 and R2, and subsequent control channel messages.

Figure 10-9. *L2TPv3 Control Connection Phase*

[View full size image]



The Control Connection establishment begins with a three-way handshake:

1. After the L2TPv3 peer is defined on the PE router, the Start-Control-Connection-Request (SCCRQ) is sent to initiate the control channel to the peer PE device and to advertise the capabilities that the local PE can support.

   Several AVPs must be sent with this control message. Two noteworthy AVPs are Assigned Control Connection ID, which defines the locally significant Control Connection ID value shown in Figure 10-8, and Pseudowire Capabilities List, which defines the pseudowire types that the local PE router can support. The Pseudowire Capabilities AVP values are drawn from the MPLS-based Layer

2 pseudowire type described in Chapter 6, "Understanding Any Transport over MPLS." Table 6-1 illustrates the associated values that are also used for the L2TPv3 Pseudowire Capabilities List AVP.

2. The peer PE router responds with a Start-Control-Connection-Reply (SCCRP) advertising its own capability set. The SCCRP message is sent in response to an accepted SCCRQ message and indicates that the Control Connection establishment can continue. A similar set of AVPs to those sent in SCCRQ are sent in the SCCRP message. One mandatory AVP is Assigned Control Connection ID, which identifies the Control Connection ID value that the peer PE router has selected.

3. Finally, the Start-Control-Connection-Connected (SCCCN) is sent in reply to the SCCRP. The SCCCN message acknowledges that the SCCRP was accepted and that the Control Connection establishment phase is complete.

After the Control Connection is established, both peers send hello messages as keepalive mechanisms during regular intervals to detect dead peers. If these maintenance messages are not received within a hold time period, the PE router can consider the peer unreachable and send a teardown message for the Control Channel. Because the hello message is representative of the Control Channel, the Session Identifier value in the encapsulation of the Control Message is set to 0.

Whether because of hello timer expiration or some other critical error (such as unrecognized Mandatory AVP), you use a Stop-Control-Connection-Notification (StopCCN) to tear down the Control Channel. The StopCCN message must contain the Assigned Control Connection ID if you send the teardown after an SCCRQ or SCCRP message. Including the Control Channel ID explicitly defines the Control Channel that you need to disable. If you send a StopCCN message, not only must you tear down the Control Channel, but you also must implicitly clear all the associated active sessions that you might have subsequently negotiated.

One of the optional features negotiated during Control Channel establishment is a lightweight security option known as *Control Message Authentication*. This authentication provides peer authentication and integrity checking against all control messages. Control Message Authentication performs a one-way hash against the header and body of the control message (with L2TPv3 over IP, this begins after the Session Identifier of 0), a shared secret preconfigured on both PE routers, and a local and remote nonce value that is passed via the Control Message Authentication Nonce AVP during Control Connection establishment.

Note

Earlier versions of the L2TPv3 base draft described a form of Control Connection Authentication using a Challenge Handshake Authentication Protocol (CHAP)-like mechanism. The differences between this mechanism and the newer form of Control Message Authentication described earlier will be explored in more detail in Chapter 11.

Note

When *nonce* is used in relation to cryptography, it refers to a random value generated for onetime use to protect against replay attacks.

The result of this hash is passed via a Message Digest AVP, which contains the digest type specifying the hashing mechanism and a Message Digest field, containing the resultant hash output.

Upon receipt of the Message Digest AVP, the PE router must perform the same hashing mechanism and compare the locally computed value to the Message Digest field value it obtains from the remote PE. If the locally computed value does not match, the PE router must discard the entire control message. Prior to utilizing or reacting to any of the information from the control messages, the PE router must validate the Message Digest AVP.

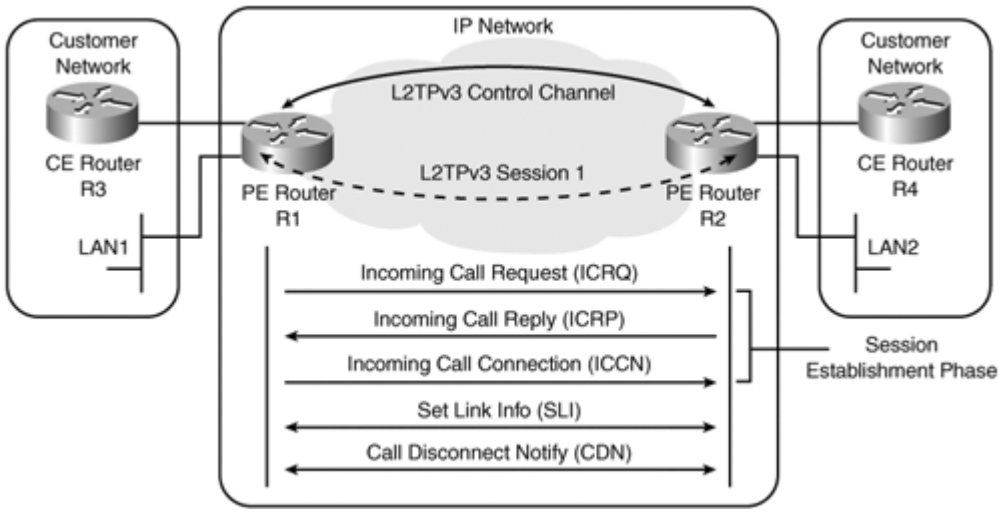
To perform peer authentication, you must configure the shared secret between the PE routers. However, if a shared secret is undefined, you can still perform control message integrity checking by using the hash against an empty shared secret.

This Control Message Authentication scheme for L2TPv3 is similar to the mechanism you use for Tunnel Authentication, which is defined in L2TPv2. However, the primary difference is that unlike L2TPv2, you perform the hashing mechanism against the entire control message. This provides the added benefit of providing a checksum-like functionality for control messages in the case of L2TPv3 over IP, where the IP Checksum is only computed for the IP header.

The second phase of control connection signaling is an optional Session establishment phase, which dynamically establishes pseudowire sessions. The Session establishment phase can involve incoming call requests (that is, receiving a call) and outgoing call requests (that is, asking to place an outbound call). The Cisco implementation supports only incoming call messages for pseudowire session establishment. Figure 10-10 illustrates the various messages that are used in session negotiation, maintenance, and teardown. Similar to the Control Connection establishment phase, the Session negotiation establishment uses a three-way handshake mechanism.

Figure 10-10. *L2TPv3 Session Negotiation*

[View full size image]



Following is the three-way handshake process:

1. When the attachment circuit transitions to an active state, the PE router exchanges parameter information about the session by sending an Incoming-Call-Request (ICRQ) to the remote PE.

   Some of the noteworthy required AVPs passed in this request include Local Session ID, which defines the locally significant Session Identifier value, and Serial Number, which is a unique identifier of the attachment circuit.

   A few of the optional AVPs that you can pass include Assigned Cookie, which determines the value used in the Cookie field, and L2-Specific Sublayer, which defines whether an Layer 2-Specific Sublayer field is used in the L2TPv3 encapsulation. Refer to Figure 10-7 to see the Cookie field and Layer 2-Specific Sublayer.

2. After the remote PE receives the ICRQ message, it sends an Incoming-Call-Reply (ICRP) to indicate that the ICRQ was accepted. Similar AVPs are sent in the ICRP reply to pass relevant properties with regard to the L2TPv3 session.

3. An Incoming-Call-Connected (ICCN) message is sent in reply to the received ICRQ to indicate that the pseudowire session is fully established.

This three-way session negotiation mechanism occurs for each pseudowire that needs to be dynamically built. To signal an individual session state, any PE can send Set-Link-Info (SLI) messages to indicate attachment circuit status changes. For example, if a Frame Relay PVC changes to down, a PE can send an SLI message to the remote PE to indicate this change in state. The remote PE can use this information to inform the end devices via Frame Relay LMI that the PVC is no longer usable.

A peer can also tear down individual pseudowire sessions by using Circuit-Disconnect-Notify (CDN) messages. When the peer receives this message, it must

silently tear down this session and its associated resources.

Summary

This chapter explored how L2TPv3 evolved into a pseudowire emulation protocol by examining its evolution from its prestandard implementation, its Data Plane encapsulation, and its Control Plane Signaling.

Following are several key aspects to take away from this chapter:

- L2TPv3 borrowed heavily from UTI's encapsulation format and L2TPv2's control plane to provide pseudowire emulation.
- L2TPv3 supports IP encapsulation using an IP protocol value of 115, whereas UTI uses an IP protocol value of 120.
- Although the base L2TPv3 draft supports both IP and IP/UDP encapsulation, the Cisco initial implementation supports only IP encapsulation.
- The Cisco L2TPv3 data packet encapsulation essentially is composed of an IP header, Session ID, cookie, an optional Layer 2-Specific Sublayer, and the Layer 2 payload.
- The Cisco L2TPv3 control packet encapsulation is composed of an IP header, Session ID, Control Message Header, and AVPs if necessary. The Control Message header includes a 12-octet field containing T-, L-, and S-bits; Version field; Length field; Control Connection ID; and Sequence Number sent and received fields.
- L2TPv3's control channel is inband along the data path, as opposed to AToM.
- Control Connection IDs are locally significant values to identify a specific Control Channel. One Control Channel usually exists between a pair of PE routers.
- Session IDs are locally significant values that identify a specific pseudowire session.
- AVPs are an extensible method of defining individual parameters in each of the control messages.
- When you have Control Plane signaling enabled, you must first build the Control Channel between the PE devices using SCCRQ/SCCRP/SCCCN messages. You negotiate any subsequent pseudowire sessions that you need to build through a similar three-way handshake using ICRQ/ICRP/ICCN messages.

- Although L2TPv3 has a defined control plane, the signaling is entirely optional. You can reduce it to just Control Channel negotiation or Control and Session Negotiation.

Chapter 11. LAN Protocols over L2TPv3 Case Studies

This chapter covers the following topics:

- Introducing the L2TPv3 configuration syntax
- LAN protocols over L2TPv3 case studies

The previous chapter examined the Layer 2 Tunneling Protocol Version 3 (L2TPv3) data and control planes. You learned the details of the L2TPv3 data plane and how the Layer 2 data link payload is encapsulated and uniquely identified for proper session demultiplexing. You can statically define or maintain these L2TPv3 sessions through the use of an optional L2TPv3 control plane protocol.

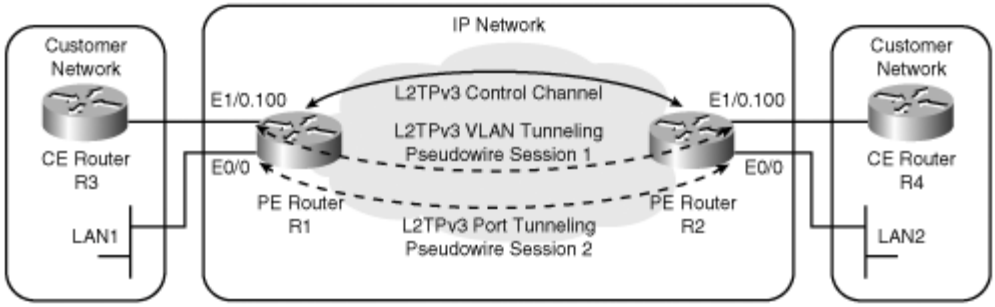
Now that you understand the operation of the protocol, this chapter focuses on LAN transport using L2TPv3, beginning with an introduction to the configuration syntax and how it relates to the pseudowire connection model. Following this are case studies for the two types of point-to-point LAN transport that L2TPv3 supports: Ethernet port emulation and VLAN emulation.

Introducing the L2TPv3 Configuration Syntax

One of the design goals of the Layer 2 pseudowire command-line interface (CLI) is to provide a consistent and flexible syntax for configuring pseudowires. Because all pseudowire realizationsincluding AToM and L2TPv3provide the same functionality, it was natural for these protocols to share a uniform configuration model. Before delving into the CLI, review Figure 11-1, which shows the L2TPv3 connectivity model that was introduced in Chapter 10, "Understanding L2TPv3." The original model has been modified slightly to illustrate two specific L2TPv3 Ethernet pseudowire types: port tunneling and VLAN tunneling.

Figure 11-1. ***L2TPv3 Connectivity Model***

[View full size image]



In Figure 11-1, the two provider edge (PE) routers, R1 and R2, are L2TPv3 endpoints providing Layer 2 connectivity between each pair of like-to-like attachment circuits via separate L2TPv3 pseudowire sessions. Ethernet 802.1q trunks exist between R3 and R1 and between R2 and R4. Although each 802.1q trunk is capable of transporting multiple VLANs, VLAN 100 is used in Figure 11-1 to demonstrate a VLAN tunneling pseudowire. The VLAN 100 attachment circuit on R3's E1/0.100 subinterface is attached to R4's E1/0.100 subinterface via L2TPv3 pseudowire session 1. When you are per-forming VLAN tunneling, only 802.1q Ethernet frames with VLAN IDs that match the VLAN ID value defined by the local attachment circuit are transported onto the pseudowire. When the VLAN IDs for the local and remote attachment circuits differ, the far end PE device is responsible for rewriting the VLAN tag on the outgoing Ethernet frame.

In contrast, L2TPv3 pseudowire session 2 is an example of a port tunneling pseudowire that stitches together R3's E0/0 Ethernet interface on LAN1 to R4's E0/0 Ethernet interface on LAN2. Unlike VLAN tunneling, any Ethernet frame that is received on Ethernet interface E0/0 is transported on the pseudowire and replicated on the far-end attachment circuit. Therefore, regardless of whether the frame is untagged, has an 802.1q tag, or has a QinQ tag, the frame is replicated transparently on the outgoing attachment circuit.

All LAN and WAN (covered in Chapter 12, "WAN Protocols over L2TPv3 Case Studies") L2TPv3 pseudowire sessions can be defined statically or have an optional control channel exist between the L2TPv3 endpoints to negotiate session details (such as session IDs and sequencing) and endpoint information (that is, control channel authentication and hidden Attribute-Value Pairs [AVP]).

From a configuration perspective, you need a method to tie the attachment circuit to the pseudowire session. For scalability purposes, the CLI should also allow multiple sessions to share the same session characteristic's template (that is, sequencing and source IP address) and multiple dynamic sessions to share the same control channel parameters. Essentially, the L2TPv3 configuration syntax fulfills these requirements through the use of the **xconnect**, **pseudowire-class**, and **l2tp-class** command syntax and configuration modes.

xconnect Command Syntax

In the case of Ethernet transport, L2TPv3 supports two types of attachment circuits, as described in the previous section:

- Port tunneling on Ethernet interface
- VLAN tunneling on Ethernet VLAN subinterface

The **xconnect** command that is configured under these interface types locally binds the attachment circuit to the pseudowire session and employs the following syntax:

```
xconnect peer-ip-address vcid pseudowire-parameters [sequencing
  {transmit | receive | both}]
```

The following list explains the arguments in the syntax:

- *peer-ip-address* The *peer-ip-address* argument identifies the remote PE router where the AC resides. This IP address should reference a virtual interface such as a loopback interface, whose reachability depends solely on its administrative state.
- *vcid* The 32-bit virtual circuit identifier, *vcid,* acts as a unique per-peer-address identifier of the pseudowire. You should configure the matching VC ID on the remote L2TPv3 endpoint's attachment circuit to associate the pseudowire session to the attachment circuit.
- *pseudowire-parameters* This placeholder syntax represents the encapsulation and pseudowire session parameters when defining the **xconnect** command. As briefly described in Chapter 10, you can configure L2TPv3 in three modes:

Manual mode Manual mode requires all session characteristics to be configured on each end of the L2TPv3 endpoint. In this setting, the attachment circuit state cannot be signaled to the remote end, and reachability to the remote L2TPv3 endpoint is not monitored.

Manual mode with keepalive Manual mode with keepalive operates in the same manner as manual mode but enables a simple peer keepalive mechanism for dead peer detection.

Dynamic mode Dynamic mode utilizes the control channel for peer capability and pseudowire session negotiation so that manual preconfiguration is unnecessary.

Because of these variations, the **xconnect** syntax must handle both manually defined and dynamically negotiated sessions. As described earlier, the *pseudowire-parameters* field is merely a placeholder for an expanded set of command options. *pseudowire-parameters* takes the following form:

```
encapsulation {l2tpv3 [manual] ¦ mpls} pw-class
{pseudowire-class-name}
```

The following list explains the syntax:

encapsulation {**l2tpv3** [**manual**] | **mpls**} The **encapsulation** command defines the tunneling method used. The **manual** keyword indicates that session negotiation is nonexistent. Configuring the session into manual mode forces the user into a config-if-xconn configuration submode for manual definition of session parameters, such as session cookies and session IDs. The case studies in this chapter discuss the various modes in more detail.

pw-class {*pseudowire-class-name*} The **pw-class** option references a pseudowire-class template that defines whether a control channel is used in addition to other shared session characteristics, which are explored later in this section. The **pw-class** command is a mandatory argument when L2TPv3 manual mode is selected as the encapsulation method.

sequencing {**transmit** | **receive** | **both**} The **sequencing** syntax is an optional argument that is used primarily when configuring L2TPv3 in manual mode. The **transmit** and **receive** options configure sequencing of L2TPv3 data packets sent and received over the pseudowire, respectively. Selecting **both** enables transmit and receive sequencing. Packets received from the pseudowire session that are considered out of order are dropped.

pseudowire-class Command Syntax

The **pseudowire-class** command defines a named template containing a series of session characteristics. The pseudowire adopts these session characteristics when the **xconnect pw-class** *pseudowire-class-name* argument refers to the respective template. The syntax has the following format:

```
pseudowire-class [ pseudowire-class-name]
```

pseudowire-class-name is a locally significant, unique identifier of the template. When you enter this argument, the CLI enters into config-pw-class configuration submode, and the following options are available:

- **encapsulation** {**l2tpv3** | **mpls**} The encapsulation option defines the tunneling method that is used. After you define the pseudowire-class template and enter the config-pw-class submode, this is the only command that is initially available to the user because the remaining options depend on the encapsulation that you choose.

- **ip local interface** *interface-name* The **ip local interface** command defines the source address of the L2TPv3 control and data packets. The **encapsulation** and **ip local interface** definitions are minimum arguments for a complete L2TPv3 pseudowire-class.

- **protocol** {**l2tpv3** | **none**}[*l2tp-class-name*] The **protocol** syntax defines whether the L2TPv3 signaling protocol is used for session negotiation. If you prefer dynamic session negotiation, configure **protocol l2tpv3** and optionally reference an l2tp-class template so that multiple sessions can share the same control channel characteristics. If no session negotiation is required, select **protocol none**. If no definition is made, the default assumes that **protocol l2tpv3** is configured and that dynamic session negotiation will occur.

- **sequencing** {**transmit** | **receive** | **both**} The sequencing configuration follows the same format as previously defined in the **xconnect** syntax.

- **ip dfbit set** By enabling this option, the don't fragment (DF) bit is set on the IP packet header of the L2TPv3 packets.

- **ip pmtu** L2TPv3 supports the discovery of path maximum transmission unit (MTU) to reach the remote L2TPv3 endpoint. This topic is explored in more detail in Chapter 13, "Advanced L2TPv3 Case Studies."

- **ip tos** {**value** *value* | **reflect**} When enabled, the **ip tos** option uses the configured type of service (ToS) value in the IP header of the L2TPv3 packet. If the payload of the Layer 2 frame is IP, the **reflect** option reflects the ToS value that is stored in the inner IP header to the outer IP header. If **ip tos value** and **ip tos reflect** are configured simultaneously, the configured ToS **value** is used on the outer IP header when the Layer 2 frame payload is not IP, while reflection would occur when the payload is IP.

- **ip ttl** *value* The IP Time to Live (TTL) of the outer IP packet is configured with the defined TTL value that is configured in this command.

- **ip protocol** {**l2tp** | **uti**} To allow for interoperability with Universal Transport Interface (UTI), you can adjust the IP protocol field to identify the IP packet as either L2TPv3 using IP protocol 115 or UTI using IP protocol 120.

Note

It is highly recommended as a best practice that every **xconnect** command reference a pseudowire-class template so that the source address of the L2TPv3 packets is defined to a virtual interface, such as a loopback interface. If you do not define a source address using the **ip local** interface command, the source address is not deterministic; it uses the PE's egress interface address that is closest to the destination, which might change depending on the network topology.

l2tp-class Command Syntax

Similar to the **pseudowire-class** command, which acts as a template of pseudowire session characteristics, the **l2tp-class** command defines a named template containing a series of control channel characteristics, such as control channel authentication and hidden AVPs. You can reference the **l2tp-class** command in the **pseudowire-class** definition via the **protocol l2tpv3** syntax for dynamic session negotiation or via the config-if-xconn configuration submode when defining a manual L2TPv3 session with keepalive support. This chapter examines the latter case in more detail in "Case Study 11-2: Ethernet Port-to-Port Manual Session with Keepalive" and "Case Study 11-3: Ethernet Port-to-Port Dynamic Session." The following syntax is used when defining the **l2tp-class**:

```
l2tp-class [ l2tp-class-name]
```

l2tp-class-name is the locally unique name for this template. Configuring this places the user in a config-l2tp-class configuration submode. When the user is in this mode, several control channel parameters are available, falling into four categories, as follows:

- Local cookie size

- Control channel timing

- Control channel authentication and integrity checking

- Control channel maintenance

The optional L2TPv3 local cookie size contains a single command and has the following form:

- **`cookie size [4 | 8]`** [*size*] The **cookie size** defines the size of the locally unique cookie for each dynamically negotiated pseudowire session that shares this l2tp-class template. Only two options are offered: a 4- or 8-byte cookie value. The default assumes a 0-byte cookie (that is, no local cookie is defined). As such, the local peer does not pass a Cookie AVP to the remote peer.

L2TPv3 control channel timing parameters include the following options:

- **receive-window** [*size*] The L2TPv3 control channel utilizes a sliding window implementation using Ns, the sequence number found in the L2TPv3 control message that was sent, and Nr, the sequence number expected in the next L2TPv3 control message to be received. The receive window value determines the number of outstanding messages that the remote device can send before receiving an acknowledgement from the local device.

- **retransmit** {**initial retries** *initial-retries* | **retries** *retries* | **timeout** {**max** | **min**} *timeout*} The **retransmit retries** *retries* interval defines the number of retransmission attempts before declaring the remote end as unresponsive. More specifically, **initial retries** *initial-retries* defines the number of Start-Control-Connection Request (SCCRQ) attempts made when trying to initialize the control channel. The first retransmission is sent at the **timeout min** *timeout* value after the first unacknowledged request. The time between each subsequent retransmission increases exponentially until it reaches the value specified in the **timeout max** *timeout* configuration.

- **timeout setup** [*seconds*] This specifies the maximum amount of time permitted to set up the control channel.

L2TPv3 control channel authentication and integrity checking parameters implement two forms of control channel authentication. The old implementation utilized a simple Challenge Handshake Authentication Protocol (CHAP) mechanism borrowed from L2TPv2, which utilizes challenge and challenge response messages for peer authentication. Cisco later extended L2TPv3 to implement a new authentication method that utilizes a control message hash. This new authentication system follows the method described in Chapter 10's "L2TPv3 Control Channel Signaling" section. One advantage of the new authentication mechanism is that this feature performs the cryptographic hash against the entire L2TPv3 control message, whereas the old mechanism performed the hash only against specific AVPs. Another benefit is that the control message hashing feature includes the resultant hash output referred to as the message digest in all L2TPv3 control messages. The old mechanism only exchanged challenge and challenge responses against SCCRQ and Start-Control-Connection Reply (SCCRP) messages.

The L2TPv3 control connection authentication and integrity checking parameters contain the following options:

- **authentication** This option enables the old CHAP-like lightweight control channel authentication between peers.

- **hostname** [*host name*] The **hostname** option explicitly defines the host name to identify the local device in the old CHAP-like control channel authentication. If you do not explicitly use the **hostname** command, the host name of the router is used.

- **password** {**encryption-type**}[*password*] The **password** definition establishes the predefined shared secret between peers used in the old CHAP-like control channel authentication mechanism. This value along with several other fields is hashed, and the resultant value is passed to the peer for control message authentication. If this value is not specified, the password value is taken from the globally configured **username** [*username*] **password** [*password*] value, where *username* is the host name of the local device.

- **digest** [**secret** [**0** | **7**] *password*] [**hash** {**md5** | **sha**}] The **digest secret** defines the shared secret and hashing mechanism used in the new control message hashing authentication mechanism. The [**0** | **7**] input type option defines the format of the *password* that is defined. A **0** indicates that the subsequent *password* is entered in plaintext, whereas a **7** indicates that the *password* is encrypted. The **hash** {**md5** | **sha**} option defines the hashing mechanism that calculates the message digest. The default assumes a **0** input type option and **hash md5**. Both peers should use the same shared digest secret and hashing mechanism for the control message.

- **digest check** The **digest check** enables validation of the contained message digest. By default, this option is enabled. You can disable it only when **digest secret** is not configured to obtain a slight performance improvement by obviating the checking for a message digest.

- **hidden** When you define the **hidden** keyword in a **l2tp-class**, AVPs are encoded to hide sensitive information. If the **hidden** keyword is not configured, AVPs are sent in the clear. The Cookie AVP that protects against blind insertion attacks is an example of an AVP that would be hidden when the **hidden** keyword is configured along with a **digest secret** command. The length of the hidden AVP is different from the original AVP because of potential padding and additional overhead in the process of hiding the AVP.

  ### Note

  When you use a **digest secret** option, you perform control connection authentication of the remote peer. In this case, the message digest is calculated against the L2TPv3 control message content along with the configured **digest secret** *password* and the local and remote Control Message Authentication Nonce.

However, you can configure the **digest** keyword by itself to simply perform a control connection integrity check. In this scenario, the message digest is calculated against the L2TPv3 control message content and provides a unidirectional integrity check.

L2TPv3 control channel maintenance parameters involve the following option:

- **hello** [*interval*] The **hello** *interval* defines the interval in seconds between Hello messages after the control channel is initialized. The hello mechanism provides a simple dead peer detection mechanism and defaults to 60 seconds if it is not configured explicitly.

LAN Protocols over L2TPv3 Case Studies

Now that you understand some of the CLI concepts, the subsequent sections examine how to use these CLI components to provide an L2TPv3 Layer 2 VPN service focusing on two specific types of Ethernet pseudowire transport:

- Ethernet port-to-port emulation
- Ethernet VLAN-to-VLAN emulation

Some optional L2TPv3 concepts will be introduced in various case studies to reinforce your understanding of their functionality.

To help explain some of the concepts in an L2TPv3 LAN environment, each case study in this chapter is based on a single IP lab topology shown in Figure 11-2. The service provider has chosen a pure IP-based packet switched network (PSN) to provide Layer 2 services to its customer. The IP PSN consists of two PE routers called SanFran and NewYork and a P router called Denver. The PE and P routers are connected to each other via Cisco-HDLC (C-HDLC) Serial links addressed with a /30 prefix. Each core device also has a /32 loopback configured.

Figure 11-2. ***L2TPv3 Ethernet Port-to-Port Emulation Case Study***

[View full size image]

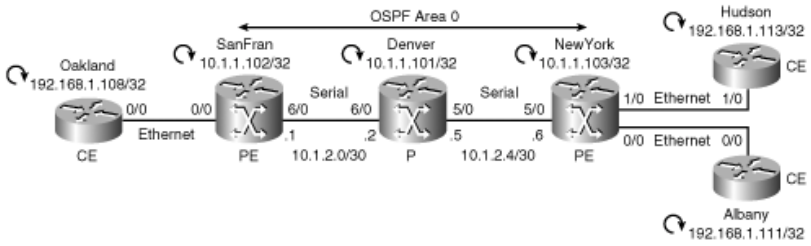


Table 11-1 lists the relevant addressing for the case study.

Table 11-1. ***Address Space for IP Case Study Topology***

Device	Site	Subnet
PE	SanFran (Loopback 0)	10.1.1.102/32
P	Denver (Loopback 0)	10.1.1.101/32
PE	NewYork (Loopback 0)	10.1.1.103/32
PE-P links	Point-to-point links in IP PSN core	/30s out of 10.1.2.0/24 block

Device	Site	Subnet
CE	Oakland (Loopback 0)	192.168.1.108/32
CE	Albany (Loopback 0)	192.168.1.111/32
CE	Hudson (Loopback 0)	192.168.1.113/32
CE-CE pseudowire links	Point-to-point links between CE devices	/30s out of 192.168.2.0/24 block

The following list describes the required steps in establishing the IP PSN core:

Step 1. Create a loopback interface and assign a /32 IP address to it.

Step 2. Enable IP CEF globally. Depending on the platform type, configure distributed CEF (dCEF) to further improve switching performance.

Step 3. Assign IP addresses to all physical links that connect the core routers. In this chapter, /30 subnets are allocated for each core serial link.

Step 4. Enable an Interior Gateway Protocol (IGP) among the core devices. In this chapter, OSPF is configured as a single area 0.

Example 11-1 includes the base configuration of the SanFran PE router. Denver and NewYork are configured equivalently to the SanFran router, with adjustments to the interface IP addressing.

Example 11-1. *SanFran Required Preconfiguration*

```
hostname SanFran
!
ip cef
!
interface Loopback0
 ip address 10.1.1.102 255.255.255.255
!
interface Serial6/0
 ip address 10.1.2.1 255.255.255.252
!
router ospf 1
 log-adjacency-changes
 network 10.1.1.0 0.0.0.255 area 0
 network 10.1.2.0 0.0.0.255 area 0
!
```

The highlighted lines indicate the relevant IP addressing specific to the SanFran router that would change in the respective configuration in the Denver and NewYork devices.

To confirm that the proper routes are being distributed, Example 11-2 captures the IP routing table of the IP PSN network.

Example 11-2. ***SanFran Preconfiguration Verification***

```
SanFran#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR

Gateway of last resort is not set
       10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
C         10.1.2.0/30 is directly connected, Serial6/0
O         10.1.2.4/30 [110/96] via 10.1.2.2, 00:07:50, Serial6/0
C         10.1.1.102/32 is directly connected, Loopback0
O         10.1.1.103/32 [110/97] via 10.1.2.2, 00:07:50, Serial6/0
O         10.1.1.101/32 [110/49] via 10.1.2.2, 00:07:50, Serial6/0
```

The highlighted routes are NewYork and Denver loopback addresses that are learned through Open Shortest Path First (OSPF). Both addresses are reachable from SanFran via the Serial 6/0 interface.

The next sections present the following case studies:

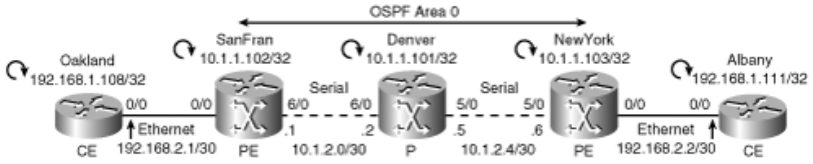
- Case Study 11-1: Ethernet Port-to-Port Manual Session
- Case Study 11-2: Ethernet Port-to-Port Manual Session with Keepalive
- Case Study 11-3: Ethernet Port-to-Port Dynamic Session
- Case Study 11-4: Ethernet VLAN-to-VLAN Dynamic Session

Case Study 11-1: Ethernet Port-to-Port Manual Session

In this case study, the customer has requested that the service provider supply transparent Layer 2 Ethernet port-to-port connectivity between the Oakland and Albany routers, as shown in Figure 11-3. Because the service provider has chosen an IP-based core, L2TPv3 is used to meet the customer's requirements. In the section "**xconnect** Command Syntax," you learned about manual, manual with keepalive, and dynamic modes. These three modes are relevant and applicable to all L2TPv3 sessions emulating LAN or WAN protocols. However, because this section is the introduction to configuring these sessions, it examines each mode using Ethernet port-to-port emulation as an example starting with the simplest case: a manual L2TPv3 Ethernet port-to-port session, as illustrated in Figure 11-3.

Figure 11-3. ***CE Ethernet Port-to-Port Connectivity***

[View full size image]



The goal in this case study is to provide Layer 2 Ethernet port-to-port connectivity over an IP-based PSN between Oakland's CE router E0/0 interface and Albany's CE router E0/0 interface. The subsequent sections examine the necessary configuration, verification, and data plane details of this environment.

Ethernet Port-to-Port Manual Configuration

To provision an Ethernet port-to-port manual pseudowire, the SanFran and NewYork PE devices will configure the following to act as L2TPv3 endpoints for the customer:

1. Define a **pseudowire-class** template.
2. Define an **xconnect** on the appropriate interface with the necessary preconfigured manual attributes.

Example 11-3 shows the relevant configuration for the SanFran PE device.

Example 11-3. *Ethernet Port-to-Port Manual Session Configuration on SanFran*

```
hostname SanFran
!
pseudowire-class pw-manual
 encapsulation l2tpv3
 protocol none
 ip local interface Loopback0
!
interface Loopback0
 ip address 10.1.1.102 255.255.255.255
!
interface Ethernet0/0
 no ip address
 no cdp enable
 xconnect 10.1.1.103 33 encapsulation l2tpv3 manual pw-class pw-manual
  l2tp id 245 329
  l2tp cookie local 8 957344 9379092
  l2tp cookie remote 8 76429 945
```

Example 11-4 shows the equivalent configuration for NewYork.

Example 11-4. *Ethernet Port-to-Port Manual Session Configuration on NewYork*

```
hostname NewYork
ip cef
!
pseudowire-class pw-manual
 encapsulation l2tpv3
 protocol none
 ip local interface Loopback0
!
interface Loopback0
 ip address 10.1.1.103 255.255.255.255
!
interface Ethernet0/0
 no ip address
 no cdp enable
 xconnect 10.1.1.102 33 encapsulation l2tpv3 manual pw-class pw-manual
  l2tp id 329 245
  l2tp cookie local 8 76429 945
  l2tp cookie remote 8 957344 9379092
```

As mentioned earlier in the section, both PE routers should configure a pseudowire-class template first. San Francisco's and New York's pseudowire-class template is called pw-manual, as highlighted in the example. Although defining a pseudowire-class is an optional step, it is a highly recommended best practice at least to define the L2TPv3 local interface to a loopback interface. This forces the source address for the L2TPv3 session to use a virtual interface that otherwise would never go down unless it was shut administratively.

In the case of the pw-manual template, both PE routers tie the local interface to their respective Loopback 0. Because you are configuring a manual L2TPv3 session, the template also defines **protocol none** to disable L2TPv3 from initiating a control channel connection to the peer. Keep in mind that this pw-manual template can be referenced from multiple **xconnect** statements. This example happens to have only one **xconnect** defined.

The second major configuration task is defining the **xconnect** statement. In SanFran's configuration, the **xconnect** command is defined underneath the relevant attachment circuit, which is the Ethernet major interface, Ethernet 0/0, because this example uses Ethernet port-to-port emulation. The **xconnect** statement defines the peer address as NewYork's loopback address of 10.1.1.103 and defines a virtual circuit (VC) ID of 33. The VC ID uniquely identifies the attachment circuit on a per-peer basis. Equivalently, NewYork's **xconnect** statement defines the peer address to be SanFran's loopback address of 10.1.1.102. You must use the VC ID value of 33 on NewYork so that the pseudowire session between the PEs can identify which AC to bind to.

As highlighted in the **xconnect** configuration line in both PE router examples, the **manual** keyword after the **encapsulation l2tpv3** syntax defines this pseudowire as a manually defined session and references the **pw-manual** pseudowire-class template via the **pw-class pw-manual** configuration. After you enter the **xconnect** command with the **manual** keyword, the configuration is placed into config-if-xconn submode configuration, which requires the user to manually define the necessary attributes of the session. Table 11-2 summarizes the manually defined session and cookie values in decimal and hexadecimal format from SanFran's perspective. These hexadecimal values are referenced later in the verification and data plane examination of this case study to assist in decoding captured output.

Table 11-2. *L2TPv3 Manual Configuration Values for Ethernet Port-to-Port Manual Session*

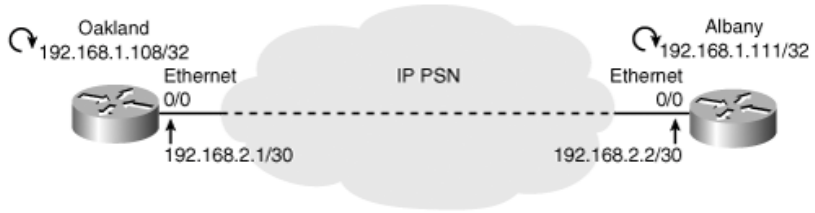
	Local	Remote
Session ID	245 (0x000000F5)	329 (0x00000149)
Cookie Size	8	8
Cookie Value (Low)	957344 (0x000E9BA0)	76429 (0x00012A8D)
Cookie Value (High)	9379092 (0x008F1D14)	945 (0x000003B1)

Because every session requires a Session ID, the SanFran router defines the local Session ID as 245 and the remote Session ID as 329. Optionally, this example uses L2TPv3 cookies to protect the pseudowire session from blind insertion attacks. Preconfigure the Session ID and optionally the cookie definition and use agreed upon values; otherwise, the far-end peer will not recognize the L2TPv3 encapsulation. The NewYork configuration, as highlighted in Example 11-4 underneath the **xconnect** command, configures the identical values in the reverse position to appropriately match the local and remote values from NewYork's perspective.

The two CE routers, Oakland and Albany, are configured without knowledge that the Ethernet port connectivity is provided over an L2TPv3 service. Figure 11-4 illustrates that the Layer 3 CE-to-CE connectivity exists between Oakland and Albany as if their Ethernet ports were connected back to back.

Figure 11-4. ***CE Routers Oakland and Albany Layer 3 Connectivity***

[View full size image]



Verifying Ethernet Port-to-Port Manual Session

Several **show** commands are useful for determining the state of the manual Ethernet port-to-port L2TPv3 session. This section describes the following:

- **show l2tun**
- **show l2tun session all**
- **show sss circuits**

The **show l2tun** command shows summarized information regarding all defined tunnels and sessions on the local device, as demonstrated in Example 11-5. As mentioned in Chapter 10, the term *L2TP tunnel* refers to the L2TPv3 control connection. As highlighted in Example 11-5, you can see the term *tunnel* in this context. Because this case study configures a manual session, no L2TPv3 control connection is established, and the total tunnels is 0. However, one session does exist, which happens to be the Ethernet port-to-port manual session defined between Oakland and Albany.

Example 11-5. *SanFran* show l2tun *Output*

```
SanFran#show l2tun
 Tunnel and Session Information Total tunnels 0 sessions 1
 Tunnel control packets dropped due to failed digest 0

LocID      RemID       TunID       Username, Intf/            State
                                   Vcid, Circuit
245        329         0           33, Et0/0                  est
```

The final line of the Example 11-5 output provides summarized information regarding each of the sessions. The output includes the local and remote session IDs, which in SanFran's case are 245 and 329, respectively. The defined VC ID is 33, and the attachment circuit is Ethernet 0/0. Also notice that the tunnel ID in this case is 0. The tunnel ID is essentially the control connection ID defined in Chapter 10. Normally, this would be a negotiated value; however, because the pseudowire session is in manual mode, no control channel is used and no control connection ID is necessary. Finally, the state of the session shows established (est), indicating that this pseudowire session is active.

You can glean additional session attributes from the **show l2tun session all** output shown in Example 11-6.

Example 11-6. *SanFran* show l2tun *all Output*

```
SanFran#show l2tun session all
 Session Information Total tunnels 0 sessions 1
 Tunnel control packets dropped due to failed digest 0


Session id 245 is up, tunnel id 0
Call serial number is 0
Remote tunnel name is
  Internet address is 10.1.1.103
  Session is manually signalled
  Session state is established, time since change 00:53:09
    692 Packets sent, 693 received
    66992 Bytes sent, 66981 received
    Receive packets dropped:
      out-of-order:              0
      total:                     0
    Send packets dropped:
      exceeded session MTU:      0
      total:                     0
  Session vcid is 33
  Session Layer 2 circuit, type is Ethernet, name is Ethernet0/0
  Circuit state is UP
    Remote session id is 329, remote tunnel id 0
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 8 bytes, value 00 8F 1D 14 00 0E 9B A0
    remote cookie, size 8 bytes, value 00 00 03 B1 00 01 2A 8D
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

Sequencing is off
```

The **show l2tun session all** command displays more detailed information for every session that exists on the PE device. Like the **show l2tun session** command, the capture from Example 11-6 shows the local and remote session ID as 245 and 329 respectively, the VCID of 33, a local TunID of 0, the attachment circuit interface of E0/0, and the session state of established.

The **show l2tun session all** command also displays the peering address as New York's loopback address of 10.1.1.103, the session type of manually signaled, and timers since the previous session state change. The session state timer is followed by counters for packets and bytes sent and received. The sent packets/bytes are from the perspective of attachment circuit frames that are encapsulated with an L2TPv3 header and sent onto the pseudowire session. Conversely, the received packets/bytes are from the perspective of received L2TPv3 packets that are from the session to be sent toward the attachment circuit. The size and hexadecimal value of local and remote cookies is also displayed for the session.

Finally, the local L2TPv3 encapsulation overhead required to transport this Ethernet frame is displayed as encap size = 32 bytes. As described in Chapter 10's "L2TPv3 Data Encapsulation" section, the L2TPv3 overhead is composed of a 20-byte IP header, a 4-byte session ID, an optional and variable-length cookie, and an optional L2-Specific Sublayer. In this case study, an 8-byte cookie is defined on both PE routers, and no L2-Specific Sublayer is used. (Sequencing is not enabled.) Therefore, the 32-byte encapsulation is derived from the 20-byte IP header, the 4-byte remote session ID of 329, and the 8-byte remote cookie.

The encapsulation details are further highlighted in the **show sss circuits** output listed in Example 11-7.

Example 11-7. *SanFran* show sss circuits *Output*

```
SanFran#show sss circuits

Current SSS Circuit Information: Total number of circuits 1

Common Circuit ID 0             Serial Num 5      Switch ID 22074136
------------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  0
   Y  AES  32  45000000 00000000 FF73A4BC 0A010166 0A010167
                  00000149 000003B1 00012A8D
```

The first 20 bytes are the IP packet header with a source address of SanFran's loopback of 10.1.1.102 (0x0A010166) and a destination address of 10.1.1.103 (0x0A010167). The IP header is then followed by the remote session ID of 329 (0x00000149) and the 8-byte remote cookie (cookie value high = 0x000003B1, cookie value low = 00012A8D).

You can perform a final verification by passing traffic between the CE routers to the next-hop interface address. As shown in Example 11-8, a ping is executed from the Oakland CE router to the E0/0 IP address on the Albany CE router of 192.168.2.2, indicating successful connectivity.

Example 11-8. *Ethernet Port-to-Port Manual Session CE Verification*

```
Oakland#ping 192.168.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/34/48 ms
```

Ethernet Port-to-Port L2TPv3 Data Plane Details

Chapter 10 discussed at length the L2TPv3 encapsulation format for encapsulating data. In this case study, correlating the encapsulated packet with the preconfigured details for the pseudowire session should be fairly simple. An L2TPv3 frame that is captured in the IP PSN core is shown in Example 11-9.

Example 11-9. *Ethereal Decode and Capture of Oakland to Albany ICMP Ping*

```
Cisco HDLC
    Address: Unicast (0x0f)
    Protocol: IP (0x0800)
Internet Protocol, Src Addr: 10.1.1.102 (10.1.1.102), Dst Addr: 10.1.1.103 (10.1.1.103)
```

```
    Version: 4
    Header length: 20 bytes
    ! IP header DSCP, Flags detail, Fragment offset and TTL omitted for brevity
    Protocol: Layer 2 Tunneling (0x73)
    Header checksum: 0x6858 (correct)
    Source: 10.1.1.102 (10.1.1.102)
    Destination: 10.1.1.103 (10.1.1.103)
Layer 2 Tunneling Protocol version 3
    Session ID: 329
    Cookie: 000003B100012A8D
Ethernet II, Src: 00:00:0c:00:6c:00, Dst: 00:00:0c:00:6f:00
    Destination: 00:00:0c:00:6f:00 (00:00:0c:00:6f:00)
    Source: 00:00:0c:00:6c:00 (00:00:0c:00:6c:00)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.2.1 (192.168.2.1), Dst Addr: 192.168.2.2
(192.168.2.2)
    Version: 4
    Header length: 20 bytes
    ! IP header DSCP, Flags detail, Fragment offset and TTL omitted for brevity
    Protocol: ICMP (0x01)
    Header checksum: 0x31d7 (correct)
    Source: 192.168.2.1 (192.168.2.1)
    Destination: 192.168.2.2 (192.168.2.2)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x9d07 (correct)
    Identifier: 0x0009
    Sequence number: 0x0000
    Data (72 bytes)

0000  0f 00 08 00 45 00 00 92 3b d2 00 00 ff 73 68 58
       ^^^^^^^^^^  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      Cisco HDLC IPv4 Delivery Header (IP Protocol L2TPv3)
0010  0a 01 01 66 0a 01 01 67 00 00 01 49 00 00 03 b1
       ^^^^^^^^^^^^^^^^^^^^^   ^^^^^^^^^^^^^^^^^^^^^^^
      IPv4 Delivery Header         L2TPv3 Header
0020  00 01 2a 8d 00 00 0c 00 6f 00 00 00 0c 00 6c 00
       ^^^^^^^^^^  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    L2TPv3 Header L2TPv3 Payload (Ethernet II Frame)
0030  08 00 45 00 00 64 04 6e 00 00 ff 01 31 d7 c0 a8
       ^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
   Ethertype          IPv4 Hdr (ICMP packet)

0040  02 01 c0 a8 02 02 08 00 9d 07 00 09 00 00 00 00
       ^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
   IPv4 Hdr (ICMP packet)         ICMP packet
!remainder omitted for brevity
```

Note

Note that in Examples 11-9, the offline hand decoding of the packets is shown in bold font.

Example 11-9 includes the full Ethereal decode of the packet followed by the hexadecimal capture of the frame. Each highlighted section of the Ethereal decode corresponds to a highlighted hexadecimal field. The decode captures a Cisco-HDLC frame between the SanFran PE router and the Denver P router that contains an L2TPv3 packet. This L2TPv3 packet payload contains an Ethernet frame from the previous CE verification ICMP ping executed from Oakland's E0/0 interface to Albany's E0/0 interface.

Notice that the capture has two Layer 2 frame headers. The first Layer 2 frame is the Cisco-HDLC frame between the SanFran and Denver routers. This is followed by the IP delivery header that is part of the L2TPv3 packet. Because the L2TPv3 endpoints are the PE routers, the source and destination IP addresses in the outer IP delivery header are SanFran and NewYork's Loopback 0 addresses, respectively. An IP protocol type of 0x73, 115, indicates that the IP payload is an L2TPv3 packet. The session ID is 0x00000149, which in decimal is 329 and is equal to the remote session ID configured on SanFran that uniquely identifies the session on NewYork. Following the session ID is the 8-byte remote cookie value, 0x000003B100012A8D, which was configured on SanFran as the remote cookie value.

The L2TPv3 payload follows the L2TPv3 header and is the second Layer 2 frame header. In this case, the L2TPv3 payload is a standard Ethernet Version II untagged frame minus the frame check sequence (FCS) sent to Albany's Ethernet port, MAC address 0000.0c00.6f00, from Oakland's Ethernet port, MAC address 0000.0c00.6c00. Further inspection shows that this Ethernet frame is carrying an IP payload that is sourced from Oakland's E0/0 interface, whose IP address is 192.168.2.1 (0xc0a80201), to Albany's E0/0 interface, whose IP address is 192.168.2.2 (0xc0a80202).

When the NewYork router receives this L2TPv3 frame, the outer IP header and L2TPv3 header are stripped off. The session ID and cookie value in the L2TPv3 header provide the NewYork router with enough information to properly demultiplex the frame and associate it with the appropriate egress attachment circuit. In this case, the Ethernet frame is forwarded out of NewYork's Ethernet 0/0 interface and destined to Albany's Ethernet port.

The nature of the Ethernet port-to-port session is that any valid Ethernet frame minus the FCS is transported across the pseudowire to be replayed on the far-end attachment circuit. Because of this, the Ethernet port-to-port session is oblivious to the fact that tagged or untagged frames might be received on Oakland's E0/0 interface. In an alternate scenario in which the Oakland and Albany routers are sending 802.1q tagged frames, the Ethernet port-to-port emulation would be oblivious to the 802.1q tag. It would transport the tagged Ethernet II frame across the pseudowire transparently, leaving the 802.1q tag untouched.

Case Study 11-2: Ethernet Port-to-Port Manual Session with Keepalive

Although a manually defined Ethernet port-to-port session fulfills the original goal for Layer 2 connectivity, one of the disadvantages is that the PE devices cannot detect whether the remote peer is responding. If connectivity is lost to the NewYork PE, the SanFran router would not tear down the pseudowire; it would continue sending data packets on the session. In fact, the pseudowire session state for a manual session always shows established, as shown in the **show l2tun session** output, regardless of the state of the peer PE router.

This case study explores the configuration, verification, and control plane details for an Ethernet port-to-port manual session with keepalives. Because the addition of the keepalives affects only the control plane, the data plane details are the same as in Case Study 11-1 and are not reexamined.

Ethernet Port-to-Port Manual Session with Keepalive Configuration

Recognizing the drawback of no peer detection, the service provider has decided to adjust the original configuration to support a simple keepalive mechanism. This keepalive mechanism is essentially an L2TPv3 control channel maintained between the PE devices. This case study adjusts the control connection and management timers so that you can understand how they function. To provision this service, Example 11-10 contains the new configuration applied to the SanFran PE router.

Example 11-10. *Ethernet Port-to-Port Manual Session with Keepalive Configuration on SanFran*

```
hostname SanFran
!
l2tp-class l2-keepalive
 hello 30
 retransmit retries 5
```

```
 retransmit timeout max 4
 retransmit timeout min 2
 retransmit initial retries 3
 retransmit initial timeout max 7
 retransmit initial timeout min 2
!
pseudowire-class pw-manual
 encapsulation l2tpv3
 protocol none
 ip local interface Loopback0
!
interface Ethernet0/0
 no ip address
 no cdp enable
xconnect 10.1.1.103 33 encapsulation l2tpv3 manual pw-class pw-manual
  l2tp id 245 329
  l2tp cookie local 8 957344 9379092
  l2tp cookie remote 8 76429 945
  l2tp hello l2-keepalive
```

Example 11-11 contains the new configuration applied to the SanFran PE router.

Example 11-11. *Ethernet Port-to-Port Manual Session with Keepalive Configuration on NewYork*

```
hostname NewYork
!
l2tp-class l2-keepalive
 hello 30
 retransmit retries 5
 retransmit timeout max 4
 retransmit timeout min 2
 retransmit initial retries 3
 retransmit initial timeout max 7
 retransmit initial timeout min 2
!

pseudowire-class pw-manual
 encapsulation l2tpv3
 protocol none
 ip local interface Loopback0
!
interface Ethernet0/0
 no ip address
 no cdp enable
 xconnect 10.1.1.102 33 encapsulation l2tpv3 manual pw-class pw-manual
  l2tp id 329 245
  l2tp cookie local 8 76429 945
  l2tp cookie remote 8 957344 9379092
  l2tp hello l2-keepalive
```

Note

Although the **ip cef** and **interface Loopback 0** definitions are not shown, these commands are still required in the configuration. They were removed from the example to reduce the redundancy in the example configurations.

Following are the primary steps involved in defining an L2TPv3 manual session with keepalives:

Step 1. Define an **l2tp-class** template. Optionally modify L2TPv3 control channel timers.

Step 2. Define a **pseudowire-class** template.

Step 3. Define an **xconnect** on the appropriate interface with the necessary preconfigured manual attributes. In the config-if-xconn submode, reference the **l2tp-class** template using the **l2tp hello** syntax.

The first step to enabling a manual session with keepalives involves defining an l2tp-class. In this example, an l2tp-class was defined named *l2-keepalive* that consists of a series of timer modifications. The Hello message keepalive timer was modified from the default 60 seconds to 30 seconds via the **hello 30** syntax. If a Hello message is not acknowledged, the control channel attempts five retries per the **retransmit retries** configuration. The **retransmit retries min** configuration defines the first retry interval at 2 seconds and doubles per interval up to the **retransmit retries max** value. After the pseudowire session fails, the L2TPv3 endpoints try to restore the control channel by sending the initial SCCRQ message. In the same manner, the number of SCCRQ retries and time between retry attempts are dictated by the **retransmit initial retries**, **retransmit initial min**, and **retransmit initial max configured values**.

The **pseudowire-class pw-manual** definition is reused from the previous "Ethernet Port-to-Port Manual Session" section of Case Study 11-1. As mentioned in that case study, defining a **pseudowire-class** template is a highly recommended best practice to make the source address of L2TPv3 packets deterministic.

The **xconnect** configuration then references this **l2tp-class** by name in the config-if-xconn submode as a template to use for its keepalive mechanism via the **l2tp hello l2-keepalive** command. The control channel negotiation and keepalive are examined in more detail in the subsequent "Ethernet Port-to-Port Manual Session with Keepalive Control Plane Details" section.

Ethernet Port-to-Port Manual Session with Keepalive Verification

Because of the use of the control channel in this case study, additional output is available from several **show** commands used to monitor the health of the L2TPv3 control channel and sessions. They include the following commands:

- **show l2tun**
- **show l2tun tunnel all**

Example 11-12 lists the output for the **show l2tun** command**.**

Example 11-12. *SanFran* show l2tun *Output*

```
SanFran#show l2tun
 Tunnel and Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0

LocID RemID Remote Name State Remote Address  Port  Sessions L2TPclass
37528 27854 NewYork     est   10.1.1.103      0     0        l2-keepalive

LocID      RemID      TunID  Username, Intf/                      State
                             Vcid, Circuit
245        329        37528  33, Et0/0                            est
```

The **show l2tun tunnel** command displays three sections of output:

- Total tunnel and session information
- Tunnel summary information
- Session summary information

When comparing the Example 11-12 output to the Example 11-5 output in "Case Study 11-1: Ethernet Port-to-Port Manual Session," you notice that highlighted fields contain additional details that were not previously shown in manually defined sessions. In the first section of output, referred to as the total tunnel and session information section, the total tunnels value is now 1. Configuring a manual session with keepalive initiates an L2TPv3 tunnel, also referred to as an L2TPv3 control channel, which explains the one tunnel count.

In the second portion of the **show l2tun** output, described as the tunnel summary section, the L2TP class field refers to the l2-keepalive template for this L2TPv3 tunnel. Also note that the tunnel ID, also referred to as the control connection ID, in the session summary section is now a nonzero value of 37528 that was negotiated as part of the control channel establishment. The method by which the control channel is established and the way the tunnel ID is negotiated are explored in more detail in the "Ethernet Port-to-Port Manual Session with Keepalive Control Plane Details" section of this case study.

> **Note**
>
> In the first section of output from the **show l2tun tunnel** command, the total tunnels value is 1, as is the total sessions count. However, note that in tunnel summary information, the Sessions field is listed as 0 for the one tunnel in the case study with local tunnel ID 37528. The reason for this apparent discrepancy is that the first section of output lists the global session count, which consists of both manually and dynamically negotiated sessions. The Sessions field counts only dynamic sessions negotiated against that specific L2TPv3 tunnel. Because this case study examines a manually defined session, the Sessions field does not register it against the tunnel.

The **show l2tun tunnel all** output shown in Example 11-13 captures additional details for the L2TPv3 control channel.

Example 11-13. *SanFran* show l2tun tunnel all *Output*

```
SanFran#show l2tun tunnel all
 Tunnel Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0

Tunnel id 37528 is up, remote id is 27854, 0 active sessions
  Tunnel state is established, time since change 00:52:10
  Tunnel transport is IP (115)
  Remote tunnel name is NewYork
    Internet Address 10.1.1.103, port 0
  Local tunnel name is SanFran
    Internet Address 10.1.1.102, port 0
  Tunnel domain is
  VPDN group for tunnel is -
  L2TP class for tunnel is l2-keepalive
  0 packets sent, 0 received
  0 bytes sent, 0 received
  Control Ns 105, Nr 106
```

```
  Local RWS 1024 (default), Remote RWS 1024 (max)
  Tunnel PMTU checking disabled
  Retransmission time 1, max 2 seconds
  Unsent queuesize 0, max 0
  Resend queuesize 0, max 1
  Total resends 0, ZLB ACKs sent 105
  Current nosession queue check 0 of 5
  Retransmit time distribution: 0 0 0 0 0 0 0 0 0
  Sessions disconnected due to lack of resources 0
  Control message authentication is disabled
```

Example 11-13 displays both the local and remote tunnel IDs that were negotiated during control channel establishment and the current tunnel state and tunnel timer since the last state change. Because L2TPv3 is used instead of UTI, the IP protocol tunnel transport type is set to 115. Also, as a part of the control channel establishment, the remote and local tunnel names and negotiated receive window sizes (RWS) are displayed. The RWS is the maximum number of control messages that can be sent before the L2TPv3 control connection must wait for an acknowledgement. The negotiated window size allows for a sliding window mechanism for control channel messages, as described in Chapter 10.

> **Note**
>
> The **show l2tun session all** command output is essentially the same as in "Case Study 11-1: Ethernet Port-to-Port Manual Session." Therefore, it is not reviewed in this case study.

Ethernet Port-to-Port Manual Session with Keepalive Control Plane Details

As mentioned earlier, the use of keepalives for dead peer detection initiates a control connection between the L2TPv3 PE endpoints. Following are two debug commands that you can use to display the control connection establishment:

- **debug vpdn l2x-events**
- **debug vpdn l2x-packets**

debug vpdn l2x-events displays general L2TP protocol events, whereas **debug vpdn l2xpackets** displays parsed L2TP control packets. Example 11-14 captures the SanFran PE router debug output for the L2TPv3 control connection to NewYork.

Example 11-14. *SanFran* debug vpdn l2x-events *Output on Control Connection Initialization*

```
*Nov 17 11:27:42.460: L2X: Parse AVP 0, len 8, flag 0x8000 (M)
*Nov 17 11:27:42.460: L2X: Parse SCCRQ
! AVP 2 Protocol Version, AVP 6 Firmware Version, Cisco AVP 8 Vendor Name, and
  Cisco AVP 10 Vendor AVP version omitted for brevity
*Nov 17 11:27:42.460: L2X: Parse AVP 7, len 13, flag 0x8000 (M)
*Nov 17 11:27:42.460: L2X: Hostname NewYork
*Nov 17 11:27:42.460: L2X: Parse AVP 10, len 8, flag 0x8000 (M)
*Nov 17 11:27:42.460: L2X: Rx Window Size 1024
*Nov 17 11:27:42.460: L2X: Parse Cisco AVP 1, len 10, flag 0x8000 (M)
*Nov 17 11:27:42.460: L2X: Assigned Control Connection ID 27854
*Nov 17 11:27:42.460: L2X: Parse Cisco AVP 2, len 22, flag 0x8000 (M)
*Nov 17 11:27:42.460: L2X: Pseudo Wire Capabilities List:
*Nov 17 11:27:42.460: L2X:   FR-DLCI [0001], ATM-AAL5 [0002], ATM-Cell [0003],
```

```
*Nov 17 11:27:42.460: L2X:   Ether-Vlan [0004], Ether [0005], HDLC [0006],
*Nov 17 11:27:42.460: L2X:    PPP [0007], ATM-VCC-Cell [0009],
*Nov 17 11:27:42.460: L2X:    ATM-VPC-Cell [000A], IP [000B]
*Nov 17 11:27:42.460: L2X: No missing AVPs in SCCRQ
*Nov 17 11:27:42.460: L2X: I SCCRQ, flg TLS, ver 3, len 122, tnl 0, ns 0, nr 0
contiguous pak, size 122
*Nov 17 11:27:42.460: L2TP: I SCCRQ from NewYork tnl 27854
*Nov 17 11:27:42.460: Tnl37528 L2TP: Control connection authentication skipped/
  passed.
*Nov 17 11:27:42.460: Tnl37528 L2TP: New tunnel created for remote NewYork,
  address 10.1.1.103
*Nov 17 11:27:42.460: Tnl37528 L2TP: O SCCRP  to NewYork tnlid 27854
*Nov 17 11:27:42.460: Tnl37528 L2TP: O SCCRP, flg TLS, ver 3, len 122, tnl 27854,
  ns 0, nr 1
*Nov 17 11:27:42.460: Tnl37528 L2TP: Control channel retransmit delay set to 2
  seconds
*Nov 17 11:27:42.460: Tnl37528 L2TP: Tunnel state change from idle to wait-ctl-
  reply
*Nov 17 11:27:42.536: Tnl37528 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Nov 17 11:27:42.536: Tnl37528 L2TP: Parse SCCCN
*Nov 17 11:27:42.536: Tnl37528 L2TP: No missing AVPs in SCCCN
*Nov 17 11:27:42.536: Tnl37528 L2TP: I SCCCN, flg TLS, ver 3, len 20, tnl 37528,
  ns 1, nr 1 contiguous pak, size 20
*Nov 17 11:27:42.536: Tnl37528 L2TP: I SCCCN from NewYork tnl 27854
*Nov 17 11:27:42.536: Tnl37528 L2TP: Control connection authentication skipped/
  passed.
*Nov 17 11:27:42.536: Tnl37528 L2TP: Tunnel state change from wait-ctl-reply to
established
*Nov 17 11:27:42.536: Tnl37528 L2TP: O ZLB ctrl ack, flg TLS, ver 3, len 12, tnl
  27854, ns 1, nr 2
*Nov 17 11:27:42.536: Tnl37528 L2TP: SM State established
                      ! Debug output omitted for brevity
*Nov 17 11:28:12.552: Tnl37528 L2TP: O Hello to NewYork tnlid 27854
*Nov 17 11:28:12.552: Tnl37528 L2TP: O Hello, flg TLS, ver 3, len 20, tnl 27854,
  ns 1, nr 3
                      ! Debug output omitted for brevity
*Nov 17 11:28:12.600: Tnl37528 L2TP: I ZLB ctrl ack, flg TLS, ver 3, len 12, tnl
  37528, ns 3, nr 2
                      ! Debug output omitted for brevity
*Nov 17 11:28:42.568: Tnl37528 L2TP: O Hello  to NewYork tnlid 27854
*Nov 17 11:28:42.568: Tnl37528 L2TP: O Hello, flg TLS, ver 3, len 20, tnl 27854,
  ns 2, nr 3
```

Note

The hexadecimal output for each control message from the **debug vpdn l2x-packets** command has been removed from Example 11-14 and any subsequent **debug vpdn l2x-packets** captures in future examples.

The debug output shows the typical three-way handshake, SCCRQ/SCCRP/SCCCN (Start-Control-Connection Connected), for control channel initialization, as described in Chapter 10. The SanFran router receives an inbound SCCRQ (I SCCRQ) from NewYork with multiple AVPs, such as Hostname and RWS. The control connection ID that NewYork allocates is 27854, which corresponds to the remote tunnel ID from the **show l2tun tunnel all** output demonstrated in the "Ethernet Port-to-Port Manual Session with Keepalive Verification" section of this case study. Also notice that the Pseudowire Capabilities List AVP includes Ethernet VLAN and Ethernet port pseudowires. Upon receipt of the SCCRQ, SanFran creates a new tunnel that has a local control connection ID of 37528 and replies with an SCCRP message. NewYork completes the negotiation of the control connection by sending an SCCCN message, which SanFran (I SCCCN) receives. SanFran acknowledges the SCCCN message with a Zero Length Body (ZLB) message.

After the control connection is established, Hello messages are sent periodically from each L2TPv3 endpoint and serve as a keepalive mechanism for dead peer detection. The Hello messages are acknowledged through the use of ZLB messages. Note that the time between the first Outbound Hello message (sent at 11:28:12) and the second Hello message (11:28:42) is 30 seconds. The 30-second interval coincides with the **hello 30** configuration line defined in the SanFran PE router l2-keepalive template.

The purpose of configuring keepalives in this case study is to provide a means of detecting L2TPv3 peer loss. Whereas Example 11-14 illustrates control connection initialization, the next example demonstrates a control connection teardown for a different L2TPv3 tunnel with a local tunnel ID of 40786 and remote tunnel ID of 22379. More specifically, Example 11-15 shows the **debug vpdn l2x-events** output from the SanFran router during a core link failure, where it loses connectivity to its L2TPv3 peer, NewYork.

Example 11-15. *SanFran* debug vpdn l2x-events *Output Control Connection Teardown*

```
SanFran#
*Nov 20 15:11:55.979: Tnl40786 L2TP: O Hello to NewYork tnlid 22379
*Nov 20 15:11:55.979: Tnl40786 L2TP: Control channel retransmit delay set to 2
  seconds
*Nov 20 15:11:57.999: Tnl40786 L2TP: O Resend Hello, flg TLS, ver 3, len 20, tnl
  22379, ns 97, nr 98
*Nov 20 15:11:57.999: Tnl40786 L2TP: Control channel retransmit delay set to 4
  seconds
*Nov 20 15:12:01.999: Tnl40786 L2TP: O Resend Hello, flg TLS, ver 3, len 20, tnl
  22379, ns 97, nr 98
*Nov 20 15:12:05.999: Tnl40786 L2TP: O Resend Hello, flg TLS, ver 3, len 20, tnl
  22379, ns 97, nr 98
*Nov 20 15:12:10.019: Tnl40786 L2TP: O Resend Hello, flg TLS, ver 3, len 20, tnl
  22379, ns 97, nr 98
*Nov 20 15:12:13.999: Tnl40786 L2TP: O Resend Hello, flg TLS, ver 3, len 20, tnl
  22379, ns 97, nr 98
*Nov 20 15:12:17.999: Tnl40786 L2TP: O StopCCN to NewYork tnlid 22379
*Nov 20 15:12:17.999: Tnl40786 L2TP: Tunnel state change from established to
  shutting-down
*Nov 20 15:12:23.019: Tnl40786 L2TP: Shutdown tunnel
*Nov 20 15:12:23.019: Tnl/Sn0/245 L2TP: Destroying session
! Debug output omitted for brevity
```

Note

In Example 11-15, SanFran's Hello messages are unacknowledged via ZLB messages because of the loss of connectivity to NewYork. To display ZLB acknowledgements or the lack thereof, enable **debug vpdn l2x-packets** in Example 11-15.

Initially, SanFran sends a Hello at 15:11:55.979. Unfortunately, because of the link failure, the Hello message is not acknowledged via a ZLB message. As configured in **retransmit retries 5**, five retransmissions are sent at 15:11:57.999, 15:12.01.999, 15:12:05.999, 15:12:10.019, and 15:12:13.999. In SanFran's l2-keepalive template configuration, the retransmission intervals are bounded by the **retransmit retries min 2** and **retransmit retries max 4** values. Any subsequent retries use the max value when it is reached. After the final retransmission, the control channel is torn down and a Stop-Control-Connection-Notification (STOPCCN) message is sent at 15:12:17.999. Because the loss of connectivity to NewYork is detected, not only is the tunnel shut down, but its associated sessions are, too.

Example 11-16 captures the **debug vpdn l2x-events** output for SanFran's attempt to reinitialize the L2TPv3 control connection after it detects peer loss.

Example 11-16. *SanFran* debug vpdn l2x-events *Output on Control Connection Reinitialization*

```
*Nov 20 15:12:33.099: Tnl/Sn0/245 L2TP: Create session
*Nov 20 15:12:33.099: Tnl52029 L2TP: SM State idle
*Nov 20 15:12:33.099: Tnl52029 L2TP: O SCCRQ
*Nov 20 15:12:33.099: Tnl52029 L2TP: Control channel retransmit delay set to 2 seconds
*Nov 20 15:12:33.099: Tnl52029 L2TP: Tunnel state change from idle to wait-ctl-reply
*Nov 20 15:12:33.099: Tnl52029 L2TP: SM State wait-ctl-reply
*Nov 20 15:12:33.099: Tnl/Sn0/245 L2TP: L2TP: INSTALL: Manually configured
  session using tunnel 52029 for keepalive support
*Nov 20 15:12:33.099: Tnl/Sn0/245 L2TP: Session state change from idle to wait-
  for-tunnel
*Nov 20 15:12:35.119: Tnl52029 L2TP: O Resend SCCRQ, flg TLS, ver 3, len 122,
  tnl 0, ns 0, nr 0
*Nov 20 15:12:35.119: Tnl52029 L2TP: Control channel retransmit delay set to 4 seconds
*Nov 20 15:12:39.119: Tnl52029 L2TP: O Resend SCCRQ, flg TLS, ver 3, len 122, tnl
  0, ns 0, nr 0
*Nov 20 15:12:39.119: Tnl52029 L2TP: Control channel retransmit delay set to 7 seconds
*Nov 20 15:12:46.119: Tnl52029 L2TP: O Resend SCCRQ, flg TLS, ver 3, len 122,
  tnl 0, ns 0, nr 0
*Nov 20 15:12:53.119: Tnl52029 L2TP: O StopCCN
*Nov 20 15:12:53.119: Tnl52029 L2TP: Tunnel state change from wait-ctl-reply to
  shutting-down
*Nov 20 15:12:58.139: Tnl52029 L2TP: Shutdown tunnel
*Nov 20 15:12:58.139: Tnl/Sn0/245 L2TP: Destroying session
```

In Example 11-16, SanFran attempts to reinitialize the control channel by sending an SCCRQ request at 15:12:33.099. Although this is the continuation of the debug output from Example 11-15 where SanFran's local tunnel ID is 40786, the SanFran PE router is attempting to initialize a new control connection and has allocated a new local tunnel ID of 52029. Unfortunately, because connectivity to NewYork has not been restored yet, the SCCRQ is not acknowledged, and three SCCRQ attempts are sent at 15:12:35.119, 15:12:39.119, and 15:12:46.119 based on the **retransmit retries initial 3 configuration**. These retransmissions begin with the **retransmit retries initial min 2** and double per retransmission up to the **retransmit retries initial max 7**. Because all three attempts fail, a STOPCCN is sent to tear down the control channel at 15:12:53.119.

Case Study 11-3: Ethernet Port-to-Port Dynamic Session

Although a manually defined session with keepalive provides some added benefit, this method still has some drawbacks. From a management perspective, manual sessions require the administrator to predefine the session IDs and cookies on each peer, whereas dynamic sessions automatically negotiate them. Furthermore, dynamic sessions can signal individual pseudowire session states.

This case study covers the configuration, verification, and control plane details of an Ethernet port-to-port dynamic session. Because the change from a manual to dynamic session primarily affects the control plane, the data plane details are the same as in Case Study 11-1 and are not reexamined.

> **Note**
>
> Although L2TPv3 dynamic sessions could signal the session state, the attachment circuit must have some management mechanism to indicate the health of its circuit to the CE device. Unfortunately, Ethernet presently does not have a management mechanism to signal individual Ethernet VLAN failures or Ethernet port failures outside of disabling the port, unlike Frame Relay or ATM. Therefore, although a pseudowire might have failed with dynamically negotiated sessions, the attachment circuits would not fail because Ethernet inherently does not support this capability.

Ethernet Port-to-Port Dynamic Configuration

This case study modifies the L2TPv3 configurations to dynamically negotiate the pseudowire sessions. The SanFran router configuration is shown in Example 11-17. A similar configuration exists on NewYork.

Example 11-17. *SanFran Configuration*

```
hostname SanFran
!
l2tp-class l2-dyn
 authentication
 password i8spr42
 cookie size 8
!
pseudowire-class pw-dynamic
 encapsulation l2tpv3
 protocol l2tpv3 l2-dyn
 ip local interface Loopback0
!
interface Ethernet0/0
 no ip address
 no ip directed-broadcast
 no cdp enable
 xconnect 10.1.1.103 33 pw-class pw-dynamic
```

The general steps to provisioning a dynamic L2TPv3 session are similar to those in Ethernet port-to-port manual session with keepalives. However, a few differences are worth highlighting. Because the session is no longer manual, config-if-xconn submode configuration is not required. Instead, the **xconnect** command references a pseudowire-class called pw-dynamic in this example. Whereas previously the protocol type was none, it is now configured as protocol l2tpv3, which indicates that an L2TPv3 control channel is requested with dynamic session negotiation. Optionally, the **protocol l2tpv3** command can also reference an l2tp-class template. In this case, because the service provider wants to implement an 8-byte L2TPv3 cookie, an l2tp-class called l2-dyn is defined with those characteristics. To revert to the default control channel timers, the explicit timer commands for Hello messages and control message retransmits have been removed. Another difference with the l2-dyn template is the use of authentication. To provide control channel authentication, this case study employs a shared secret by enabling the **authentication** keyword and defining a shared secret via the **password** *shared-secret* configuration.

Ethernet Port-to-Port Dynamic Session Verification

The same **show** commands that were used in the verification of manual session case studies are reused in this example. The **show** commands include the following:

- **show l2tun**
- **show l2tun tunnel all**
- **show l2tun session all**

In fact, the majority of the output of the **show l2tun** and **show l2tun tunnel all** commands is similar, with only a few minor differences. Example 11-18 captures the output from the **show l2tun** command.

Example 11-18. *SanFran* show l2tun *Output*

```
SanFran#show l2tun
 Tunnel and Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0

LocID RemID Remote Name   State  Remote Address Port Sessions L2TPclass
33819 41993 NewYork       est    10.1.1.103     0    1        l2-dyn

LocID       RemID     TunID       Username, Intf/                      State
                                  Vcid, Circuit
23878       36820     33819       33, Et0/0                            est
```

One of the differences with the previous case is the session count. As in Example 11-12, both the total tunnels and sessions values are equal to 1 in the first line of output. However, in the tunnel summary information, the Sessions field is also 1. That is because this pseudowire session is negotiated dynamically against the L2TPv3 tunnel. Also notice that in the session summary information, the local LocID and RemID fields represent the local and remote session IDs. Unlike the manual session case studies, in which session IDs were configured explicitly under the **xconnect** command in the config-if-xconn submode, these values were negotiated dynamically via the control channel. The dynamic creation of the session and the negotiation of the session ID are examined in detail in the control plane negotiation.

Example 11-19 captures the output from the **show l2tun tunnel all** command.

Example 11-19. *SanFran* show l2tun all *Output*

```
SanFran#show l2tun tunnel all
 Tunnel Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0

Tunnel id 33819 is up, remote id is 41993, 1 active sessions
  Tunnel state is established, time since change 00:02:05
  Tunnel transport is IP (115)
  Remote tunnel name is NewYork
    Internet Address 10.1.1.103, port 0
  Local tunnel name is SanFran
    Internet Address 10.1.1.102, port 0
  Tunnel domain is
  VPDN group for tunnel is -
  L2TP class for tunnel is l2-dyn
  38 packets sent, 37 received
  3720 bytes sent, 3556 received
  Control Ns 6, Nr 4
  Local RWS 1024 (default), Remote RWS 1024 (max)
  Tunnel PMTU checking disabled
  Retransmission time 1, max 1 seconds
  Unsent queuesize 0, max 0
  Resend queuesize 0, max 2
  Total resends 0, ZLB ACKs sent 2
  Current nosession queue check 0 of 5
  Retransmit time distribution: 0 0 0 0 0 0 0 0 0
  Sessions disconnected due to lack of resources 0
  Control message authentication is disabled
```

As highlighted in Example 11-19, because the pseudowire session is negotiated dynamically in the tunnel, one session is listed as active. The last line indicates that control message authentication is disabled. As described in the section "**l2tp-class** Command Syntax," the two methods of control channel authentication are an old style using a CHAP-like mechanism and a new style using message digests. The last line of highlighted output refers to the new style. Because the case study implements the old style of control channel authentication, the message shows the authentication as disabled.

The major differences in output between the manual and dynamic session case studies are reflected in the **show l2tun session all** command. Example 11-20 captures the output from the **show l2tun session all** command on the SanFran PE router.

Example 11-20. *SanFran* show l2tun session all *Output*

```
SanFran#show l2tun session all
 Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0


Session id 23878 is up, tunnel id 33819
Call serial number is 2931100006
Remote tunnel name is NewYork
  Internet address is 10.1.1.103
  Session is L2TP signalled
  Session state is established, time since change 00:02:11
    39 Packets sent, 38 received
    3780 Bytes sent, 3616 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 33
  Session Layer 2 circuit, type is Ethernet, name is Ethernet0/0
  Circuit state is UP
    Remote session id is 36820, remote tunnel id 41993
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 8 bytes, value 34 D4 9E 24 6B AF AF CE
    remote cookie, size 8 bytes, value F5 6A 1F 7F 1A 7B 12 BF
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

  Sequencing is off
```

The local session ID of 23878 and the remote session ID of 36820 is shown in the output. The second highlighted line in Example 11-20 lists the call serial number of 2931100006, which is a shared value between the L2TPv3 endpoints that uniquely references this pseudowire session. Finally, you can see that 8-byte session cookies are negotiated per the **cookie size 8** command in the l2-dyn template. The session IDs, call serial number, and cookie values are also negotiated dynamically during session negotiation.

Ethernet Port-to-Port Dynamic Session Control Plane Details

Dynamic negotiation of the control plane takes place in two phases:

- Control connection (tunnel) establishment
- Session (pseudowire) establishment

Although the **debug vpdn l2x-events** command displays these events, you can obtain further detail with the **debug vpdn l2x-packets** command, which displays the AVPs contained in each control message. The debug output from the control connection establishment phase is captured on the SanFran router and displayed in Example 11-21.

Example 11-21. *SanFran* debug vpdn l2x-events *and* debug vpdn l2x-packet *Output for Control Connection Establishment Phase*

```
SanFran#
*Nov 22 07:34:46.445: Tnl33819 L2TP: O SCCRQ
*Nov 22 07:34:46.445: Tnl33819 L2TP: O SCCRQ, flg TLS, ver 3, len 144, tnl 0,
  ns 0, nr 0
*Nov 22 07:34:46.445: Tnl33819 L2TP: Control channel retransmit delay set to
  1 seconds
*Nov 22 07:34:46.445: Tnl33819 L2TP: Tunnel state change from idle to wait-ctl-
  reply
*Nov 22 07:34:46.445: Tnl33819 L2TP: SM State wait-ctl-reply
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse SCCRP
! AVP 2 Protocol Version, AVP 6 Firmware Version, AVP 10 Rx Window Size, Cisco AVP
  8 Vendor Name, and Cisco AVP 10 Vendor AVP version omitted for brevity
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse AVP 7, len 13, flag 0x8000 (M)
*Nov 22 07:34:46.525: Tnl33819 L2TP: Hostname NewYork
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse AVP 11, len 22, flag 0x8000 (M)
*Nov 22 07:34:46.525: Tnl33819 L2TP: Chlng
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse AVP 13, len 22, flag 0x8000 (M)
*Nov 22 07:34:46.525: Tnl33819 L2TP: Chlng Resp
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse Cisco AVP 1, len 10, flag 0x8000 (M)
*Nov 22 07:34:46.525: Tnl33819 L2TP: Assigned Control Connection ID 41993
*Nov 22 07:34:46.525: Tnl33819 L2TP: Parse Cisco AVP 2, len 22, flag 0x8000 (M)
*Nov 22 07:34:46.525: Tnl33819 L2TP: Pseudo Wire Capabilities List:
! Pseudo Wire Capabilities List omitted for brevity
*Nov 22 07:34:46.525: Tnl33819 L2TP: No missing AVPs in SCCRP
*Nov 22 07:34:46.525: Tnl33819 L2TP: I SCCRP, flg TLS, ver 3, len 166, tnl 33819,
  ns 0, nr 1 contiguous pak, size 166
*Nov 22 07:34:46.525: Tnl33819 L2TP: I SCCRP from NewYork
*Nov 22 07:34:46.525: Tnl33819 L2TP: Got a challenge in SCCRP, NewYork
*Nov 22 07:34:46.525: Tnl33819 L2TP: Got a response in SCCRP, from remote peer
  NewYork
*Nov 22 07:34:46.525: Tnl33819 L2TP: Tunnel Authentication success
*Nov 22 07:34:46.525: Tnl33819 L2TP: Control connection authentication skipped/
  passed.
*Nov 22 07:34:46.525: Tnl33819 L2TP: Tunnel state change from wait-ctl-reply to
established
*Nov 22 07:34:46.525: Tnl33819 L2TP: O SCCCN to NewYork tnlid 41993
*Nov 22 07:34:46.525: Tnl33819 L2TP: O SCCCN, flg TLS, ver 3, len 42, tnl 41993,
  ns 1, nr 1
*Nov 22 07:34:46.525: Tnl33819 L2TP: Control channel retransmit delay set to
  1 seconds
*Nov 22 07:34:46.525: Tnl33819 L2TP: SM State established
```

In this case, SanFran initiates the control channel request by sending out an SCCRQ. SanFran's debug output shows the outbound SCCRQ request in addition to the dynamically chosen tunnel ID of 33819. Although it is not shown in this outbound message, the SCCRQ contains SanFran's Challenge AVP sent to NewYork. The Challenge AVP contains a random value used in the CHAP-like control channel authentication mechanism. In response to the SCCRQ, NewYork sends an SCCRP message. SanFran receives this message, and the debug output lists the various AVPs contained in the message. Two notable AVPs include NewYork's Challenge AVP and NewYork's Challenge Response AVP. In this case, NewYork's Challenge Response AVP contains the output of the hashing function performed against SanFran's Challenge AVP value and the shared secret. For the control channel message to be validated properly, the SanFran router performs the same hashing mechanism and compares the output to the received Challenge Response AVP value. If the shared secret from both peers is equal, the two values are equivalent. Another AVP that is sent in the SCCRP is NewYork's Control Connection ID value of 41993, which matches the output from the **show l2tun tunnel** command. To complete the three-way handshake, SanFran responds with an SCCCN. The SCCCN contains a Challenge Response AVP that is not shown in outbound debugs. The SCCCN Challenge Response AVP is sent in reply to NewYork's Challenge AVP sent in the SCCRP.

Example 11-22 contains the debug output from the second part of the negotiation: the session establishment phase.

Example 11-22. *SanFran* debug vpdn l2x-events *and* debug vpdn l2x-packet *Output for Session Establishment Phase*

```
SanFran#
*Nov 22 07:34:46.525:  Tnl/Sn33819/23878 L2TP: O  ICRQ to NewYork 41993/0
*Nov 22 07:34:46.525: Tnl/Sn33819/23878 L2TP: O ICRQ, flg TLS, ver 3, len 94, tnl 41993,
lsid 23878, rsid 0, ns  2, nr 1
*Nov 22 07:34:46.525: Tnl/Sn33819/23878 L2TP: Session state change from wait-for-tunnel
to wait-reply
*Nov 22 07:34:46.573:  Tnl33819 L2TP: Early authen passing ZLB
*Nov 22 07:34:46.573: Tnl33819 L2TP: I ZLB ctrl ack, flg TLS, ver 3, len 12, tnl 33819,
ns 1, nr  3
*Nov 22 07:34:46.665: Tnl33819 L2TP: Perform early message digest validation for ICRP
*Nov 22 07:34:46.665: Tnl33819 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Nov 22 07:34:46.665: Tnl33819 L2TP: Local Session ID 36820
*Nov 22 07:34:46.665: Tnl33819 L2TP: Control connection authentication skipped/passed.
*Nov 22 07:34:46.665: Tnl33819 L2TP: Parse AVP 0, len 8,  flag 0x8000 (M)
*Nov 22 07:34:46.665: Tnl33819 L2TP: Parse ICRP
*Nov 22 07:34:46.665: Tnl33819 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Nov 22 07:34:46.665: Tnl33819 L2TP: Local Session ID 36820
*Nov 22 07:34:46.665: Tnl33819 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
*Nov 22 07:34:46.665: Tnl33819 L2TP: Remote Session ID 23878
*Nov 22 07:34:46.665: Tnl33819 L2TP: Parse Cisco AVP 5, len 14, flag 0x8000 (M)
*Nov 22 07:34:46.665: Tnl33819 L2TP: Assigned Cookie
         F5 6A 1F 7F  1A 7B 12 BF
*Nov 22 07:34:46.665:  Tnl33819 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Nov 22 07:34:46.665:  Tnl33819 L2TP: Pseudo Wire Type 5
*Nov 22 07:34:46.665:  Tnl33819 L2TP: No missing AVPs in ICRP
*Nov 22 07:34:46.665: Tnl/Sn33819/23878 L2TP: I ICRP, flg TLS, ver 3, len 62, tnl 33819,
lsid 23878, rsid 0, ns  1, nr 3 contiguous pak, size  62
*Nov 22 07:34:46.665:  Tnl/Sn33819/23878 L2TP: O ICCN to NewYork 41993/36820
*Nov 22 07:34:46.665: Tnl/Sn33819/23878 L2TP: O ICCN, flg TLS, ver 3, len 50, tnl 41993,
lsid 23878, rsid  36820, ns 3, nr 2
*Nov 22 07:34:46.665: Tnl33819 L2TP: Control channel retransmit delay set to 1 seconds
*Nov 22 07:34:46.665: Tnl/Sn33819/23878 L2TP: Session state change from wait-reply to
established
```

After successfully initiating a control channel, the session negotiation begins. SanFran sends an Incoming-Call Request (ICRQ), which contains several AVPs, including a dynamically assigned Session ID of 23878, Cookie, End Identifier, and Serial Number AVP that is not shown in the outbound debug output. The End Identifier AVP equals the VC ID that is configured on the **xconnect** command, 33. This AVP allows the pseudowire to associate itself to the necessary attachment circuit. The Serial Number is equivalent to the Call Serial Number field identified in the **show l2tunn sess all** output. It serves as an identifier for the session that is consistent on both peers.

In response to the ICRQ, NewYork sends an Incoming-Call Reply (ICRP) message with several AVPs. The ICRP contains a Local Session ID AVP of 36820, a Remote Session ID AVP of 23878, an Assigned Cookie AVP of 0xF56A1F7F1A7B12BF, and a Pseudowire Type AVP. A Pseudowire Type AVP is included to identify the type of pseudowire that is being negotiated, which in this case is 0x0005 for type Ethernet. Keep in mind that the Local and Remote Session ID values and the Assigned Cookies in the ICRP message are from NewYork's perspective. When you compare them to SanFran's **show l2tunn sess all** output in Example 11-20, the values correspond to appropriate SanFran fields (that is, SanFran's Local Session ID equals NewYork's Remote Session ID). Finally, SanFran completes this three-way handshake with the Incoming-Call Connected (ICCN) message. After the ICCN message is sent, the pseudowire session is fully established.

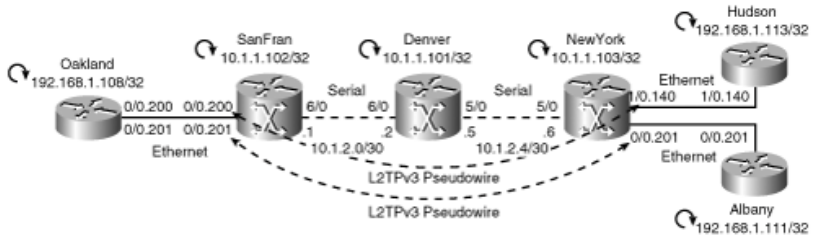
Case Study 11-4: Ethernet VLAN-to-VLAN Dynamic Session

In this case study, the customer has modified his design. The Oakland hub site now needs to have connectivity to branch offices in Albany and Hudson. The customer would like to use a dot1Q trunk from the Oakland router to connect a separate VLAN to each branch office site.

To meet the customer requirements, the service provider has decided to utilize L2TPv3 to transport the necessary VLANs. You can see an illustration of the new design in Figure 11-5. Oakland's VLAN 201 attachment circuit is connected via a pseudowire session to VLAN 201 at Albany. Similarly, Oakland's VLAN 200 attachment circuit is connected to VLAN 140 at Hudson. In the latter case, the NewYork router must rewrite the original VLAN header tag with the value of 140 so that the Ethernet frames are properly understood at Hudson.

Figure 11-5. *L2TPv3 Ethernet VLAN-to-VLAN Session Case Study*

[View full size image]



This case study explores the configuration, verification, control plane details, and data plane details for an Ethernet VLAN-to-VLAN dynamic session.

Ethernet VLAN-to-VLAN Dynamic Configuration

Similar to "Case Study 11-3: Ethernet Port-to-Port Dynamic Session," the SanFran and NewYork PE routers use dynamically negotiated VLAN L2TPv3 sessions with 8-byte cookies. To introduce a new concept in this case study, the PE routers are configured to use the new control channel authentication by utilizing message digests.

Example 11-23 shows the relevant configuration in the SanFran routers.

Example 11-23. *SanFran VLAN-to-VLAN Dynamic Configuration*

```
hostname SanFran
!
l2tp-class l2-dyn
 digest secret p7jd8ge
 cookie size 8
!
pseudowire-class pw-dynamic
 encapsulation l2tpv3
 protocol l2tpv3 l2-dyn
 ip local interface Loopback0

interface Ethernet0/0
 no ip address
 no ip directed-broadcast
 no cdp enable

interface Ethernet0/0.200
 encapsulation dot1Q 200
 no ip directed-broadcast
 no cdp enable
```

```
 xconnect 10.1.1.103 33 pw-class pw-dynamic
!
interface Ethernet0/0.201
 encapsulation dot1Q 201
 no ip directed-broadcast
 no cdp enable
 xconnect 10.1.1.103 34 pw-class pw-dynamic
```

Example 11-24 contains the configuration for the NewYork router.

Example 11-24. *NewYork VLAN-to-VLAN Dynamic Configuration*

```
hostname NewYork
!
l2tp-class l2-dyn
 digest secret p7jd8ge
 cookie size 8
!
pseudowire-class pw-dynamic
 encapsulation l2tpv3
 protocol l2tpv3 l2-dyn
 ip local interface Loopback0
!
interface Ethernet0/0
 no ip address
 no ip directed-broadcast
 no cdp enable
!
interface Ethernet0/0.201
 encapsulation dot1Q 201
 no ip directed-broadcast
 no cdp enable
 xconnect 10.1.1.102 34 pw-class pw-dynamic
!
interface Ethernet1/0
 no ip address
 no ip directed-broadcast
 no cdp enable
!
interface Ethernet1/0.140
 encapsulation dot1Q 140
 no ip directed-broadcast
 no cdp enable
 xconnect 10.1.1.102 33 pw-class pw-dynamic
!
```

The general steps to provisioning a dynamic Ethernet VLAN-to-VLAN session are similar to those in the Ethernet port-to-port dynamic session. The major difference is that the attachment circuit in this case study is the Ethernet VLAN. Therefore, the **xconnect** statements are configured under the appropriately tagged Ethernet subinterfaces, as highlighted in Example 11-23 and 11-24.

Because this case study introduces the new form of control channel authentication, the **digest secret** *password* configuration is applied underneath the **l2tp-class** command.

Ethernet VLAN-to-VLAN Dynamic Session Verification

The **show l2tun tunnel** and **show l2tun session** output is similar to the previous dynamic port-to-port session. Example 11-25 contains the **show l2tun tunnel** output detail for SanFran.

Example 11-25. *SanFran VLAN-to-VLAN Dynamic Session* show l2tun tunnel *Output*

```
SanFran#show l2tun tunnel
 Tunnel Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0

LocID RemID Remote Name   State  Remote Address  Port  Sessions L2TPclass
41796 8769  NewYork       est    10.1.1.103      0     2        l2-dyn
```

Notice in Example 11-25 that the number of sessions is two because of the two pseudowire sessions for the two VLANs in this case study. The two sessions are negotiated against the same tunnel; therefore, they are listed against the L2TPv3 tunnel with a local ID of 41796.

Example 11-26 captures the **show l2tun session** and **show l2tunnel session all** detail for VCID 34.

Example 11-26. *SanFran VLAN-to-VLAN Dynamic Session* show l2tun session *Output*

```
SanFran#show l2tun session
 Session Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0

LocID      RemID      TunID      Username,  Intf/                        State
                                 Vcid,  Circuit
23944      36877      41796      34, Et0/0.201:201                       est
23945      36878      41796      33, Et0/0.200:200                       est

SanFran#show l2tun session all vcid 34
 Session Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 5

Session id 23944 is up, tunnel id 41796
Call serial number is 2931100013
Remote tunnel name is NewYork
  Internet address is 10.1.1.103
  Session is L2TP signalled
  Session state is established, time since change 00:13:11
    93 Packets sent, 91 received
    9306 Bytes sent, 8950 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 34
  Session Layer 2 circuit, type is Ethernet Vlan, name is Ethernet0/0.201:201
  Circuit state is UP
    Remote session id is 36877, remote tunnel id 8769
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 8 bytes, value E9 B8 78 B2 C2 6C 8E 16
    remote cookie, size 8 bytes, value 88 9E DD 75 03 A0 39 75
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

  Sequencing is off
```

The primary differences in the output when compared to Case Study 11-3 are related to the attachment circuit type. In the **show l2tun session** output in Example 11-26, notice the two session lines for the two VCIDs 34 and 33 and their corresponding attachment circuits Et0/0.201 and Et0/0.200, respectively.

The **show l2tun session all vcid 34** output includes more specific details about that particular attachment circuit. Unlike the manual case studies, Example 11-26 shows the session as L2TP signaled, indicating that the pseudowire was negotiated dynamically. Also, as highlighted midway in the **show l2tun session all vcid 34** output, the type of pseudowire session is Ethernet VLAN, whereas in previous Ethernet port-to-port case studies, the pseudowire type was just Ethernet.

Ethernet VLAN-to-VLAN Dynamic Session Control Plane Details

As with any dynamically negotiated L2TPv3 session, the control channel setup and session negotiation can be monitored via **debug vpdn l2x-events** and **debug vpdn l2x-packets**. Example 11-27 captures this debug output for SanFran's control channel establishment phase.

Example 11-27. *SanFran* debug vpdn l2x-events *and* debug vpdn l2x-packets *on Control Connection Initialization*

```
SanFran#
*Nov 22 13:19:04.312: Tnl/Sn41796/23944 L2TP: Create session
*Nov 22 13:19:04.312: Tnl41796 L2TP: SM State idle
*Nov 22 13:19:04.312: Tnl41796 L2TP: O SCCRQ
*Nov 22 13:19:04.312: Tnl41796 L2TP: O SCCRQ, flg TLS, ver 3, len 167, tnl 0,
  ns 0, nr 0
*Nov 22 13:19:04.312: Tnl41796 L2TP: Control channel retransmit delay set to
  1 seconds
*Nov 22 13:19:04.312: Tnl41796 L2TP: Tunnel state change from idle to
  wait-ctl-reply
*Nov 22 13:19:04.312: Tnl41796 L2TP: SM State wait-ctl-reply
*Nov 22 13:19:04.312: L2X: L2TP: Received L2TUN message <Connect>
*Nov 22 13:19:04.312: Tnl/Sn41796/23945 L2TP: Session state change from idle to
  wait-for-tunnel
*Nov 22 13:19:04.312: Tnl/Sn41796/23945 L2TP: Create session
*Nov 22 13:19:04.312: Tnl41796 L2TP: SM State wait-ctl-reply
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse SCCRP
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse Cisco AVP 12, len 23, flag 0x8000 (M)
*Nov 22 13:19:04.372: Tnl41796 L2TP: Message Digest
! Message Digest hex output omitted for brevity
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse Cisco AVP 13, len 22, flag 0x8000 (M)
*Nov 22 13:19:04.372: Tnl41796 L2TP: CC Auth Nonce
        D7 6E BB ED 3D 01 33 31 0E 45 1A E7 67 24 4E A1
! AVP 2 Protocol Version ,AVP 6 Firmware Version, AVP 10 Rx Window Size, Cisco
  AVP 8 Vendor Name, and Cisco AVP 10 Vendor AVP version omitted for brevity
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse AVP 7, len 13, flag 0x8000 (M)
*Nov 22 13:19:04.372: Tnl41796 L2TP: Hostname NewYork
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse Cisco AVP 1, len 10, flag 0x8000 (M)
*Nov 22 13:19:04.372: Tnl41796 L2TP: Assigned Control Connection ID 8769
*Nov 22 13:19:04.372: Tnl41796 L2TP: Parse Cisco AVP 2, len 22, flag 0x8000 (M)
*Nov 22 13:19:04.372: Tnl41796 L2TP: Pseudo Wire Capabilities List:
! Pseudo Wire Capabilities List omitted for brevity
*Nov 22 13:19:04.372: Tnl41796 L2TP: No missing AVPs in SCCRP
*Nov 22 13:19:04.372: Tnl41796 L2TP: I SCCRP, flg TLS, ver 3, len 167, tnl 41796,
  ns 0, nr 1 contiguous pak, size 167
*Nov 22 13:19:04.372: Tnl41796 L2TP: I SCCRP from NewYork
*Nov 22 13:19:04.372: Tnl41796 L2TP: Message digest match performed, passed.
*Nov 22 13:19:04.372: Tnl41796 L2TP: Control connection authentication skipped/
  passed.
*Nov 22 13:19:04.372: Tnl41796 L2TP: Tunnel state change from wait-ctl-reply to
established
*Nov 22 13:19:04.372: Tnl41796 L2TP: O SCCCN to NewYork tnlid 8769
```

```
*Nov 22 13:19:04.372: Tnl41796 L2TP: O SCCCN, flg TLS, ver 3, len 43, tnl 8769,
  ns 1, nr 1
*Nov 22 13:19:04.372: Tnl41796 L2TP: Control channel retransmit delay set to
  1 seconds
*Nov 22 13:19:04.372: Tnl41796 L2TP: SM State established
```

One notable difference in the control channel establishment is that a few additional AVPs are used during the SCCRQ/SCCRP/SCCCN three-way handshake. As is typical, the three-way handshake begins with SanFran sending an SCCRQ message. The SCCRQ contains a Message Digest and a Control Message Authentication Nonce AVP. As described in Chapter 10, these AVPs are used in control channel authentication. More specifically, they are used in the newer form of control channel authentication, unlike Case Study 11-3, which uses the CHAP-like mechanism.

The second step in the three-way handshake involves SanFran sending an SCCRP reply. This SCCRP message, like the SCCRQ message, contains a Message Digest and a Control Message Authentication Nonce AVP, as highlighted in Example 11-27. All subsequent control channel messages will contain the Message Digest AVP.

The third and final step in the control channel initialization is an SCCCN message sent from SanFran.

Example 11-28 captures this debug output for SanFran's session establishment phase.

Example 11-28. *SanFran* debug vpdn l2x-events *and* debug vpdn l2x-packets *on Session Initialization*

```
SanFran#
*Nov 22 13:19:04.372: Tnl/Sn41796/23944 L2TP: O ICRQ to NewYork 8769/0
*Nov 22 13:19:04.372: Tnl/Sn41796/23944 L2TP: O ICRQ, flg TLS, ver 3, len 117,
  tnl 8769, lsid 23944, rsid 0, ns 2, nr 1
*Nov 22 13:19:04.372: Tnl/Sn41796/23944 L2TP: Session state change from
  wait-for-tunnel to wait-reply
*Nov 22 13:19:04.420: Tnl41796 L2TP: Perform early message digest validation
  for ACK
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse Cisco AVP 12, len 23, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Message Digest
! Message Digest hex output omitted for brevity
*Nov 22 13:19:04.420: Tnl41796 L2TP: Message digest match performed, passed.
*Nov 22 13:19:04.420: Tnl41796 L2TP: Control connection authentication skipped/
  passed.
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse ICRP
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse Cisco AVP 12, len 23, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Local Session ID 36877
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Remote Session ID 23944
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse Cisco AVP 5, len 14, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Assigned Cookie
         88 9E DD 75 03 A0 39 75
*Nov 22 13:19:04.420: Tnl41796 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Nov 22 13:19:04.420: Tnl41796 L2TP: Pseudo Wire Type 4
*Nov 22 13:19:04.420: Tnl41796 L2TP: No missing AVPs in ICRP
*Nov 22 13:19:04.420: Tnl/Sn41796/23944 L2TP: I ICRP, flg TLS, ver 3, len 85,
  tnl 41796, lsid 23944, rsid 0, ns 1, nr 3 contiguous pak, size 85
*Nov 22 13:19:04.420: Tnl/Sn41796/23944 L2TP: O ICCN to NewYork 8769/36877
*Nov 22 13:19:04.420: Tnl/Sn41796/23944 L2TP: O ICCN, flg TLS, ver 3, len 73,
  tnl 8769, lsid 23944, rsid 36877, ns 4, nr 2
*Nov 22 13:19:04.420: Tnl/Sn41796/23944 L2TP: Session state change from
  wait-reply to established
```

After the control channel is authenticated, the session negotiation occurs. Because this case study utilizes two pseudowires, two three-way ICRQ/ICRP/ICCN phases are initiated. For the sake of brevity, Example 11-28 only captures the three-way session negotiation for VCID 34. As highlighted in the output, SanFran sends an ICRQ message for local session ID 23944. In response, SanFran receives an ICRP message from NewYork. The ICRP message contains NewYork's local session ID of 36877. The pseudowire type is type 4 for Ethernet VLAN. Also notice that the Message Digest AVP is contained in the ICRP message. In fact, the Message Digest AVP is also contained in the ICRQ and ICCN messages, but the debug output does not show this level of detail for outbound messages. Finally, SanFran sends an ICCN to NewYork to fully establish the pseudowire session.

Ethernet VLAN-to-VLAN Frame Encapsulation

The L2TPv3 data frame encapsulation for a VLAN-emulated session is comparable to the port-emulated session except that the Ethernet frame carries the Ethernet frames that are specific to the attachment circuit VLAN ID. Example 11-29 captures an Ethereal decode and the hexadecimal capture of an L2TPv3 frame from the Oakland Ethernet subinterface E0/0.201 to Albany E0/0.201.

Example 11-29. *Ethereal Decode and Capture of Oakland to Albany ICMP Ping*

```
Cisco HDLC
    Address: Unicast (0x0f)
    Protocol: IP (0x0800)
Internet Protocol, Src Addr: 10.1.1.102 (10.1.1.102), Dst Addr: 10.1.1.103 (10.1.1.103)
    Version: 4
    Header length: 20 bytes
    ! IP header DSCP, Flags detail, Fragment offset and TTL omitted for brevity
    Protocol: Layer 2 Tunneling (0x73)
    Header checksum: 0x3a3e (correct)
    Source: 10.1.1.102 (10.1.1.102)
    Destination: 10.1.1.103 (10.1.1.103)
Layer 2 Tunneling Protocol version 3
    Session ID: 36877
    Cookie: 889EDD7503A03975
Ethernet II, Src: 00:00:0c:00:6c:00, Dst: 00:00:0c:00:6f:00
    Destination: 00:00:0c:00:6f:00 (00:00:0c:00:6f:00)
    Source: 00:00:0c:00:6c:00 (00:00:0c:00:6c:00)
    Type: 802.1Q Virtual LAN (0x8100)
802.1q Virtual LAN
    000. .... .... .... = Priority: 0
    ...0 .... .... .... = CFI: 0
    .... 0000 1100 1001 = ID: 201
    Type: IP (0x0800)
Internet Protocol, Src Addr: 192.168.2.1 (192.168.2.1), Dst Addr: 192.168.2.2
(192.168.2.2)
    Version: 4
    Header length: 20 bytes
    ! IP header DSCP, Flags detail, Fragment offset and TTL omitted for brevity
    Protocol: ICMP (0x01)
    Header checksum: 0x31be (correct)
    Source: 192.168.2.1 (192.168.2.1)
    Destination: 192.168.2.2 (192.168.2.2)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xb7fa (correct)
    Identifier: 0x000e
    Sequence number: 0x0000
    Data (72 bytes)
0000  0f 00 08 00 45 00 00 96 69 e8 00 00 ff 73 3a 3e
      ^^^^^^^^^^  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      Cisco HDLC  IPv4  Delivery  Header (IP Protocol L2TPv3)
0010  0a 01 01 66 0a 01 01 67 00 00 90 0d 88 9e dd 75
```

```
      ^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^
      IPv4 Delivery Header          L2TPv3 Header
0020  03 a0 39 75 00 00 0c 00 6f 00 00 00 0c 00 6c 00
      ^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    L2TPv3  Header L2TPv3 Payload  (Ethernet II Frame)
0030  81 00 00 c9 08 00 45 00 00 64 04 87 00 00 ff 01
      ^^^^^  ^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  Ethertype .1q  VLANID     IPv4 Hdr  (ICMP packet)
0040  31 be c0 a8 02 01 c0 a8 02 02 08 00 b7 fa 00 0e
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^
        IPv4 Hdr  (ICMP packet)         ICMP packet
!remainder omitted for brevity
```

Because the Ethereal capture was performed between the SanFran and Denver routers, the first Layer 2 header is the HDLC frame. The outer IP Delivery header is sourced from SanFran's loopback address of 10.1.1.102 (0x0a010166), destined to NewYork's address of 10.1.1.103 (0x0a010167). The L2TPv3 header consists of NewYork's local session ID of 36877 (0x0000900d) and a cookie value of 0x889edd7503a03975. Following the L2TPv3 header is the Ethernet II frame destined to Albany's MAC address of 0x00000c006c00 and sourced from Oakland's Ethernet port with MAC address 0x00000c006f00. In this particular case, the Ethernet frame is an 802.1q tagged frame that contains the VLAN tag protocol ID of 0x8100, a 3-bit VLAN CoS field of 0, a 1-bit VLAN canonical format indicator of 0, and the VLAN ID tag of 201. Because the far-end router's attachment circuit also uses a dot1Q tag of 201, NewYork does not change the original VLAN ID tag before sending it to the CE device. However, in the case of an Ethernet frame from Oakland to Hudson, the original VLAN ID tag of 200 would have to be rewritten to the far-end attachment circuit VLAN ID value of 140.

Case Study 11-2 focused on providing VLAN-to-VLAN emulation. It examined scenarios in which the VLAN header was transported unmodified (that is, between Oakland and Albany) and in which the VLAN header was rewritten (that is, between Oakland and Hudson). You need to consider several design issues when deploying such a solution.

In this VLAN-to-VLAN case study, the service provider dictates the VLAN values. From a customer perspective, this might be a heavy restriction. In such a scenario, the customer can use QinQ to alleviate this requirement. In essence, the customer can use the inner 802.1q tag in QinQ to represent the customer VLANs and then set the outer 802.1q tag to equal the value that the service provider requires. This allows for flexibility on the customer's behalf to use any VLAN rewrite.

Another consideration when dealing with VLAN-to-VLAN transport revolves around spanning tree. In the Oakland-to-Hudson scenario, the respective L2TPv3 endpoints rewrite the VLAN header as needed. Although the VLAN header is rewritten, the BPDU payload also contains a field known as the Per VLAN ID (PVID), which is not rewritten. If spanning tree is enabled in such a VLAN rewrite scenario, the BPDUs show a mismatch and the ports are blocked. The only solution is to avoid this rewrite scenario so that the BPDU payload matches the expected VLAN ID value on either end.

Summary

This chapter examined how L2TPv3 emulates LAN protocols. The first section began with an exploration of the **l2tp-class**, **pseudowire-class**, and **xconnect** syntax. This CLI introduction served as the basis for examining the two modes of Ethernet emulation L2TPv3 supports: port-to-port emulation and VLAN-to-VLAN emulation.

Following are several key aspects to take away from this chapter:

- The **l2tp-class** definition serves as a template for the L2TP control channel. **pseudowire-class** templates contain session attributes. The **xconnect** statements bind attachment circuits to the necessary pseudowire.

- You can configure L2TPv3 pseudowires for manual mode, manual mode with keepalive, and dynamic mode. Although these modes were shown in the Ethernet port-to-port model only, they are relevant for any L2TPv3 LAN and WAN session type.

- An Ethernet port-to-port session emulates any frame that is received on the respective PE ports and transports it to the far-end device transparently. The payload could be tagged or untagged depending on the CE devices.

- An Ethernet VLAN-to-VLAN session emulates any frame that is received on the respective VLAN to the far-end PE. If the VLAN ID values on the attachment circuit do not match, the remote PE has the responsibilty of performing VLAN header rewrite.

Chapter 12. WAN Protocols over L2TPv3 Case Studies

This chapter covers the following topics:

- WAN Protocols over L2TPv3 Technology Overview
- Configuring WAN Protocols over L2TPv3 Case Studies
- Verifying control and data planes for WAN protocols over L2TP3

In this chapter, you learn the functional aspects and configuration of the transport and tunneling of WAN protocols over Layer 2 Tunnel Protocol Version 3 (L2TPv3). Building on Chapter 5, "WAN Data-Link Protocols," and Chapter 10, "Understanding L2TPv3," this chapter presents the configuration, verification, and troubleshooting of High-Level Data Link Control (HDLC), PPP, Frame Relay, and ATM protocols over L2TPv3. This chapter also presents multiple case studies describing the different L2TPv3 configurations for the multiple WAN protocols that are transported.

WAN Protocols over L2TPv3 Technology Overview

Chapter 11, "LAN Protocols over L2TPv3 Case Studies," presented the configuration steps for LAN protocols over L2TPv3. In the broadest sense, the configuration, transport, and tunneling of WAN protocols over L2TPv3 are analogous to what was explored in Chapter 11; however, differences and special cases exist. When you transport WAN protocols using L2TPv3, the fundamental control plane does not change. Different virtual circuit (VC) types indicate the specific attachment circuit technology. This section presents some opening ideas about the transport of WAN protocols using L2TPv3.

Control Plane

All the control plane concepts covered in Chapter 10 are applicable to the transport and tunneling of WAN protocols. On session negotiation, the type of attachment circuit is indicated in the Pseudowire Type AVP (currently Cisco AVPusing a Structure of Management Information [SMI] enterprise code of 9Type 7) part of the Session Management AVPs. The values that the Pseudowire Type AVP can take for WAN protocols are enumerated in Table 12-1.

Table 12-1. *Pseudowire Type AVP Values Used in WAN Transport*

Pseudowire Type	Description	Usage
0x0001	Frame Relay DLCI[1]	FRoL2TPv3[2] DLCI mode
0x0002	ATM AAL5 SDU[3] VCC[4]	ATMoL2TPv3[5] AAL5 SDU mode
0x0003	ATM Transparent Cell	ATMoL2TPv3 Cell Port mode
0x0006	HDLC	HDLCoL2TPv3[6]
0x0007	PPP	PPPoL2TPv3[7]
0x0009	ATM n-to-one VCC cell	ATMoL2TPv3 Cell VC mode

Pseudowire Type	Description	Usage
0x000A	ATM n-to-one VPC[8] cell	ATMoL2TPv3 Cell VP mode

[1] DLCI = data-link connection identifier

[2] FRoL2TPv3 = Frame Relay over L2TPv3

[3] SDU = service data unit

[4] VCC = virtual channel connection

[5] ATMoL2TPv3 = ATM over L2TPv3

[6] HDLCoL2TPv3 = HDLC over L2TPv3

[7] PPPoL2TPv3 = PPP over L2TPv3

[8] VPC = virtual path connection

Note

Although the values for the pseudowire type AVP from Table 12-1 are numerically the same as the ones used in Any Transport over MPLS (AToM) for the pseudowire type forward error correction (FEC) field, they belong to different registries. You can find the registry for "L2TPv3 Pseudowire Types" at the Internet Assigned Numbers Authority (IANA) at http://www.iana.org/assignments/l2tp-parameters.

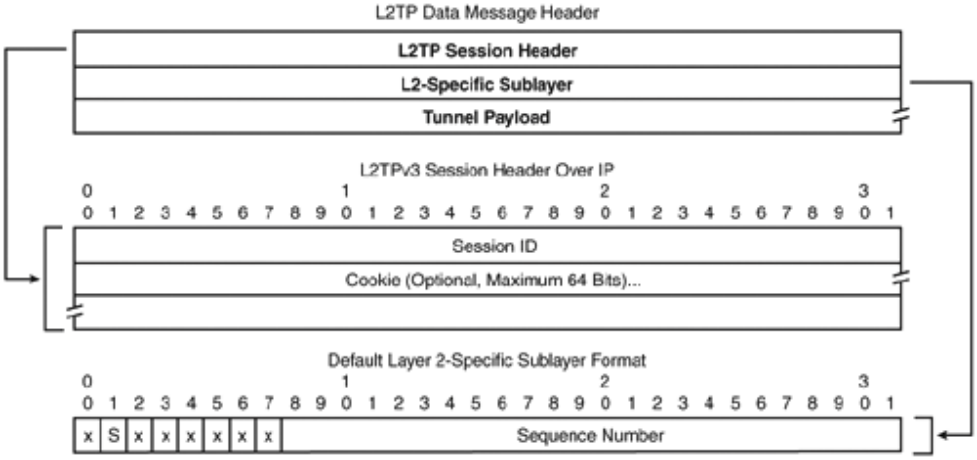
These pseudowire types are also included in the Pseudowire Capabilities List AVP part of the Control Connection Management AVPs to indicate the Layer 2 payload types that a sender can support.

Data Plane

The transport of WAN protocols over L2TPv3 follows the base specification Internet document for L2TPv3, plus the additional companion documents for each WAN technology. Cisco implemented L2TPv3 directly over IP using IP protocol number 115. Figure 12-1 shows the data plane encapsulation for the transport of WAN protocols over L2TPv3.

Figure 12-1. ***Encapsulation of WAN Protocols over L2TPv3 over IP***

[View full size image]

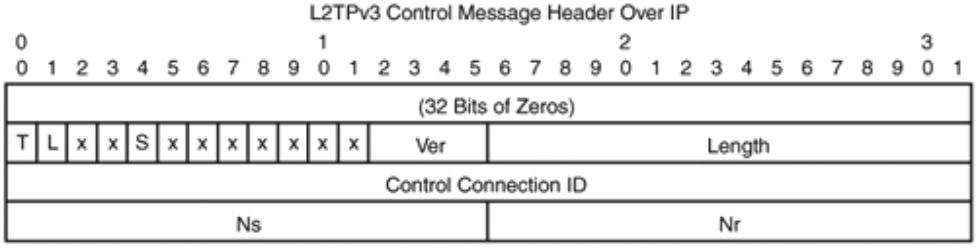


In Figure 12-1, you can see the data plane packet that goes directly encapsulated in IP. The L2TP session header consists of a 32-bit session ID and an optional cookie. In addition, an optional 32-bit L2-specific sublayer exists, referred to as Pseudowire Control Encapsulation. Figure 12-1 shows the default Layer 2-Specific Sublayer header. The presence of the Layer 2-Specific Sublayer is indicated in the Layer 2-Specific Sublayer AVP part of the session management AVPs.

A NULL session ID identifies L2TPv3 control messages over IP, as shown in Figure 12-2.

Figure 12-2. ***L2TPv3 Control Message Header Over IP***

[View full size image]



The fields from Figure 12-2 are as follows:

- **T-Bit** A field setting of 1 indicates that it is a control message.
- **L-Bit** A field setting of 1 indicates that the length field is present.

- **S-Bit** A field setting of 1 indicates that sequence numbers are present.
- **Ver** The version is set to 3 for L2TPv3.
- **Length** This is the total length of the message in octets.
- **Control Connection ID** This contains an identifier for the "tunnel" or control connection.
- **Ns** This contains the sequence number for this control message.
- **Nr** This indicates the sequence number that is expected in the next control message to be received.

Using the Layer 2-Specific Sublayer

The Layer 2-Specific Sublayer is equivalent to the control word in AToM that was discussed in Chapter 8, "WAN Protocols over MPLS Case Studies," and carries control and payload-specific fields. During session establishment, the use of an Layer 2-Specific Sublayer is negotiated by means of the Layer 2-Specific Sublayer AVP part of the Session Management AVPs. The Layer 2-Specific Sublayer AVP type is an unsigned 16-bit integer that can take the following values:

- **0** No Layer 2-Specific Sublayer is present.
- **1** The default Layer 2-Specific Sublayer is used.
- **2** The ATM Layer 2-Specific Sublayer is used.

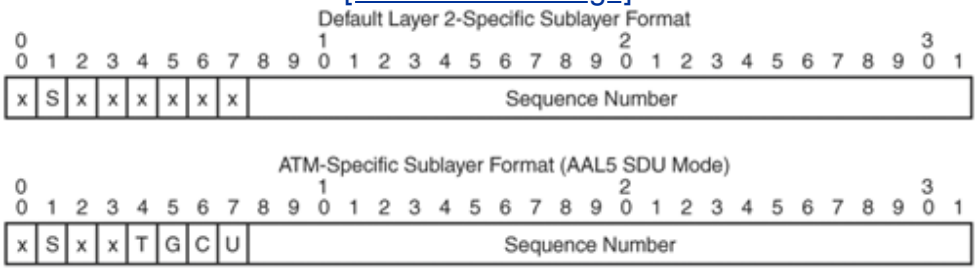
Note

The first two values are defined and assigned in the base "Layer Two Tunneling Protocol (Version 3)" IETF document, whereas the third value is defined in the "ATM Pseudo-Wire Extensions for L2TP" IETF document. ATM AAL5 transport needs an ATM-Specific Sublayer to transport ATM cell header fields that would otherwise be lost; other transported protocols, however, rely on the default Layer 2-Specific Sublayer.

If the usage of the Layer 2-Specific Sublayer header has been negotiated, the L2TP Control Connection Endpoint (LCCE) must include the specified Layer 2-Specific Sublayer in all outgoing data messages. Figure 12-3 details the two currently defined Layer 2-Specific Sublayer formats: default and ATM-Specific.

Figure 12-3. *Layer 2-Specific Sublayer Usage for WAN Protocols*

[View full size image]



The fields shown in Figure 12-3 are defined as follows:

- All Layer 2-Specific Sublayers:

  S-Bit The Sequence bit is set to indicate that the Sequence Number field contains a valid sequence number for this sequenced frame, and it is cleared otherwise. When the field is cleared, you must ignore the contents of the Sequence Number.

  Sequence Number The Sequence Number field contains a free-running counter of 2^{24} sequence numbers. As opposed to AToM, the sequence number begins at 0, which is a valid sequence number.

  X-Bits Set the Reserved bits to 0 on transmission and ignore them on reception.

- ATM-Specific Sublayer:

  T-Bit The Transport bit indicates whether the L2TPv3 packet contains an ATM admin cell (when T is set) or an AAL5 payload (when T is cleared). OAM cells are examples of admin cells.

  G-Bit The Explicit Forward Congestion Indication (EFCI) bit indicates congestion. The LCCE sets the G bit if the EFCI bit of the final cell of the incoming AAL5 payload or the (EFCI) in the single ATM cell is set to 1.

  C-Bit The cell loss priority (CLP) bit in the ATM cell header indicates cell loss priority. The LCCE sets the C bit if any of the CLP bits of any of the incoming ATM cells of the AAL5 payload or of the single ATM cell is set to 1.

  U-Bit The U-bit carries the Command/Response (C/R) bit, which is used with FRF.8.1 "Frame Relay/ATM PVC Service Interworking."

Note

Bits 2 and 3 in both Layer 2-Specific Sublayers indicate fragmentation as negated Beginning and End of fragment bits.

Different transported Layer 2 WAN protocols have different requirements for the Layer 2-Specific Sublayer. Two situations require the use of the Layer 2-Specific Sublayer:

- **AAL5** The transport and tunneling of ATM AAL5 CPCS-SDU require the usage of an ATM Specific Sublayer that carries the EFCI, CLP, and C/R and identifies AAL5 CPCSSDU versus ATM Cell. Otherwise, those fields would be lost because the cell header is not transported.

- **Sequencing** Sequencing for all Layer 2 protocols transported requires an Layer 2-Specific Sublayer that carries the Sequence Number.

For all other cases, an Layer 2-Specific Sublayer is optional. In contrast to the transport of Frame Relay over MPLS, which requires the control word, FRoL2TPv3 does not require the Layer 2-Specific Sublayer, which is equivalent to the control word. The difference lies in the fact that Frame Relay over MPLS (FRoMPLS) does not transport the Q.922 header, and the only way to transport control bits is by piggybacking them in the control word. In FRoL2TPv3, the Q.922 header is transported; therefore, the Layer 2-Specific Sublayer header is not needed.

MTU Considerations

When you tunnel a Layer 2 protocol data unit (PDU) by means of encapsulation, you need to factor the additional overheads associated with this tunneling scheme into packet sizes and maximum transmission unit (MTU). When encapsulating the Layer 2 PDU to be transported using L2TPv3 across an IP packet-switched network (PSN), you need to take into account a series of overheads that are added. This section details all the associated overheads.

You can categorize these overheads as follows:

- **Transport Overhead** The overhead that is associated with the specific Layer 2 being transported. Table 12-2 lists this overhead for the transport and tunneling of different WAN protocols.

Table 12-2. *Transport Overhead for Different WAN Protocols over L2TPv3*

Transport Type	Transport Header Size	Transport Header Reason [Bytes]
Frame Relay DLCI, Cisco encapsulation	4 bytes	Q.922 Header [2] + Ethertype [2]
Frame Relay DLCI, IETF[1] encapsulation	10 bytes	Q.922 Header [2] + SNAP[2] => Control [1] + Pad [1] + NLPID[3] [1] + OUI[4] [3] + Ethertype [2]
Cisco HDLC	4 bytes	Address [1] + Control [1] + Ethertype [2]
PPP	2 bytes	PPP DLL[5] Protocol [2]
AAL5	0-32 bytes	Header

[1] IETF = Internet Engineering Task Force

[2] SNAP = Subnetwork Access Protocol

[3] NLPID = Network Layer Protocol Identifier

[4] OUI = Organizationally Unique Identifier

[5] DLL = Data Link Layer

- **L2TPv3 Overhead** The tunneling overhead that is associated with the L2TP data message headers. It can be further subdivided into the following:
    - **L2TP Session Overhead** The overhead that is associated to the L2TP Session Header:

      Session ID The 4-byte overhead that is always present

      Cookie Optional overhead that can be NULL, 4 bytes, or 8 bytes
    - **L2-Specific Overhead** An optional overhead that is associated with the Layer 2-Specific Sublayer. It can be either NULL or 4 bytes, depending on whether the field is present.

- **Delivery (IPv4) Overhead** The overhead that is associated with the outer IP header without options identifying protocol type 115 for L2TPv3. It is always 20 bytes.

You can see the transport overhead for all WAN protocols over L2TPv3 in Table 12-2. ATM Cell transport is deliberately left out of Table 12-2. In ATM cell relay over L2TPv3 (CRoL2TPv3), the packets transported are of a fixed length of 52 bytes. You can concatenate them up to a maximum number of packed cells, making MTU calculation different from all other Layer 2 transports.

> **Note**
>
> You can compare transport overheads for L2TPv3 in Table 12-2 with the AToM transport overheads and draw the conclusion that the only different overhead is for transporting Frame Relay DLCI mode. In L2TPv3, the Q.922 header is transported but is not in AToM. A 2-byte Q.922 header without extended addressing is assumed.

From the different overheads presented, you can calculate the total overhead and infer the MTU in the provider edge (PE) and provider (P) routers toward the PSN (Core MTU) from the MTU in the PE attachment circuit interface (Edge MTU). The following equations calculateg the core MTU for different WAN protocols:

```
Core MTU ≥ Edge MTU + Transport Header + L2TPv3 Header + IPv4 Header
Where
L2TPv3 Header = L2TP Session Header + Layer 2-Specific Sublayer Header
L2TP Session Header = Session ID (4 bytes) + Cookie (0, 4 or 8 bytes)
Layer 2-Specific Sublayer Header = 0 or 4 bytes
```

In addition to the transport overhead, the maximum overhead that IP and L2TPv3oIP add is 36 bytes (20 bytes from the IP header, 4 bytes of the L2TPv3 Session ID, 8 bytes of Cookie, and 4 bytes of the Layer 2-Specific Sublayer Header). The minimum overhead is 24 bytes, skipping the Cookie and Layer 2-Specific Sublayer fields. This minimum overhead is the default for the transport of WAN protocols over L2TPv3. By default, cookies and sequencing are nonexistent, except for AAL5 SDU transport, in which the ATM-Specific Sublayer is required.

HDLC and PPP over L2TPv3

You can transport HDLC pseudowire that is defined in an L2TPv3 companion Internet document using L2TPv3 by including all HDLC data and control fields (address, control, and protocol fields) and stripping the flag and frame check sequence (FCS) fields. From Chapter 5, you know that Cisco routers use a proprietary version of HDLC referred to as Cisco HDLC. It differs from standard HDLC in that the higher layer protocol identification is performed using the Ethernet type.

Because the behavior of an HDLC pseudowire is to function in a port-mode fashion, removing the flag and FCS during imposition and transporting the complete packet over the pseudowire without inspecting it, Cisco HDLC is also transported over an HDLC pseudowire. In fact, therein is one of the most important facets of the HDLC pseudowire: it can transport transparently in an interface-to-interface mode all protocols that contain HDLC-like framing (meaning 0x7E flag and FCS). This includes but is not limited to PPP, Frame Relay, X.25, Synchronous Data Link Control (SDLC), and so on.

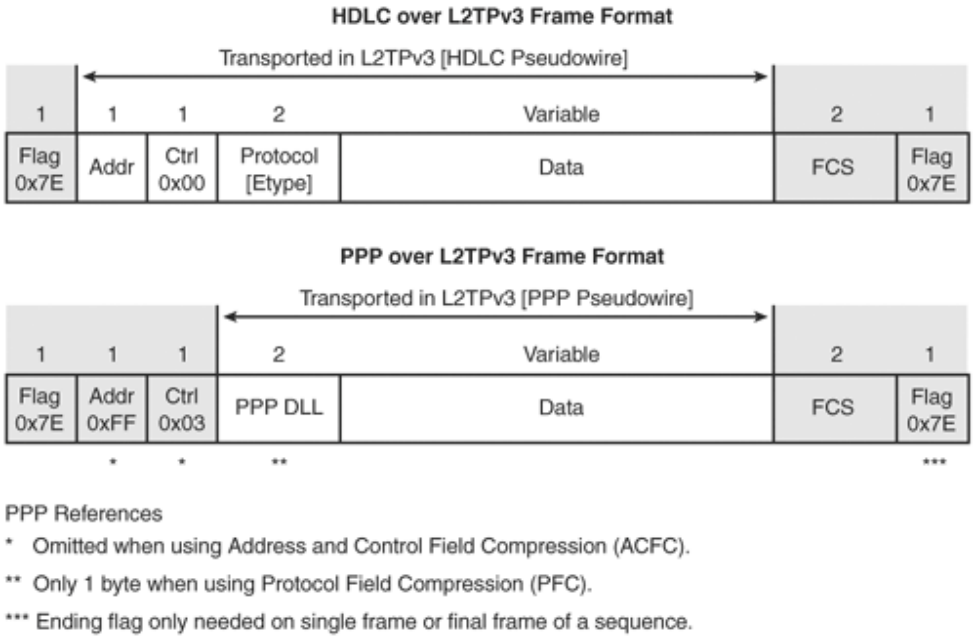
The transport of PPP frames over L2TPv3 pseudowires is quite similar to the transport of HDLC frames. This coincides with the fact that PPP was modeled after HDLC with the addition of protocol fields to transport multiprotocol datagrams over point-to-point links.

Differences and optimizations exist, however, traceable to the fact that PPPoL2TPv3 has some Layer 2 packet inspection. At imposition, the Address (0xFF) and Control (0x03) fields of the PPP frame are removed, leaving the first field transported as the PPP DLL Protocol Number. The IANA assigns these PPP DLL Protocol Numbers. You can check them at http://www.iana.org/assignments/ppp-numbers.

Figure 12-4 shows the encapsulation and packet formats for HDLCoL2TPv3 and PPPoL2TPv3.

Figure 12-4. ***HDLC Pseudowire and PPP Pseudowire over L2TPv3 Packet Formats***

[View full size image]



PPP References

* Omitted when using Address and Control Field Compression (ACFC).

** Only 1 byte when using Protocol Field Compression (PFC).

*** Ending flag only needed on single frame or final frame of a sequence.

Because the Address and Control fields are stripped at imposition and not transported over L2TPv3, FCS Alternatives (specify different FCS formats or no FCS at all by means of the LCP configuration option) and Address and Control Field Compression (ACFC) do not work. In contrast, the protocol field is transported so that Protocol Field Compression (PFC) works.

Frame Relay over L2TPv3

At this point, you already know that you can transport and tunnel Frame Relay in a port-mode fashion using an HDLC pseudowire because Frame Relay frames employ HDLC-like framing. The PE configuration for Frame Relay Port mode does not specify the encapsulation as Frame Relay, but it leaves the default of HDLC. In this case, however, Local Management Interface (LMI) is also tunneled over the pseudowire; therefore, you need to properly configure the customer edge (CE) devices for LMI:

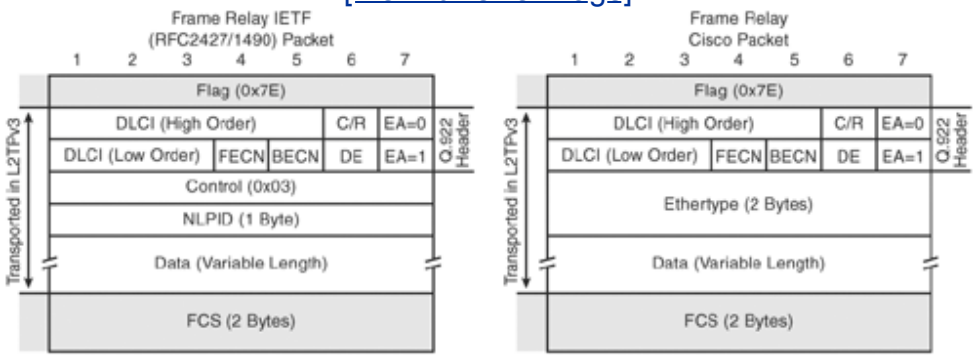
- If the CE devices are Frame Relay switches, use Frame Relay Network-to-Network Interface (NNI) LMI in both ends.
- If the CE devices are Frame Relay routers, use Frame Relay data terminal equipment (DTE) LMI in one end and Frame Relay DCE LMI in the other end. Alternatively, configure Frame Relay NNI LMI to run between routers to better indicate the DLCI status between CE and CE.

Another method of transporting Frame Relay frames at the DLCI level uses the pseudowire type of 0x0001. This method to provide Frame Relay pseudowires is specified in an L2TPv3 IETF companion document.

When you use Frame Relay pseudowire DLCI mode, the Frame Relay PDU is transported in its entirety. Only the opening and closing HDLC flags of 0x7E and the FCS field are stripped at imposition, and the complete Frame Relay frameincluding the Q.922 headeris transported. When you are using different DLCIs at both ends, the DLCI value is rewritten at the disposition (egress) PE (see Figure 12-5).

Figure 12-5. ***Frame Relay Pseudowire over L2TPv3 Packet Formats***

[View full size image]



From Figure 12-5, you can see both Frame Relay IETF encapsulation and Frame Relay Cisco encapsulation. They differ in the upper-layer protocol identification. You can configure Cisco routers for either type of encapsulation.

Because the complete Q.922 header is transported, you do not need to transport the Command/Response (C/R), forward explicit congestion notification (FECN), backward explicit congestion notification (BECN), and discard eligibility (DE) bits separately, as was the case with FRoMPLS. However, the DLCIs can be different at both ends of the Frame Relay pseudowire, so you must rewrite the Frame Relay DLCI at disposition.

ATM over L2TPv3

The tunneling and transport of ATM over L2TPv3 presents two operational modes:

- **ATM AAL5-SDU Mode** The ingress LCCE performs reassembly and the AAL5 CPCSSDU is transported over the ATM pseudowire. The EFCI, CLP, and C/R bits are transported in the required ATM-Specific Sublayer. The egress LCCE performs segmentation.

- **ATM Cell Mode** No reassembly occurs at the ingress LCCE, and ATM-Layer ATM cells are transported over the ATM pseudowire. The ATM-Specific Sublayer is optional. Two operational submodes are also supported:

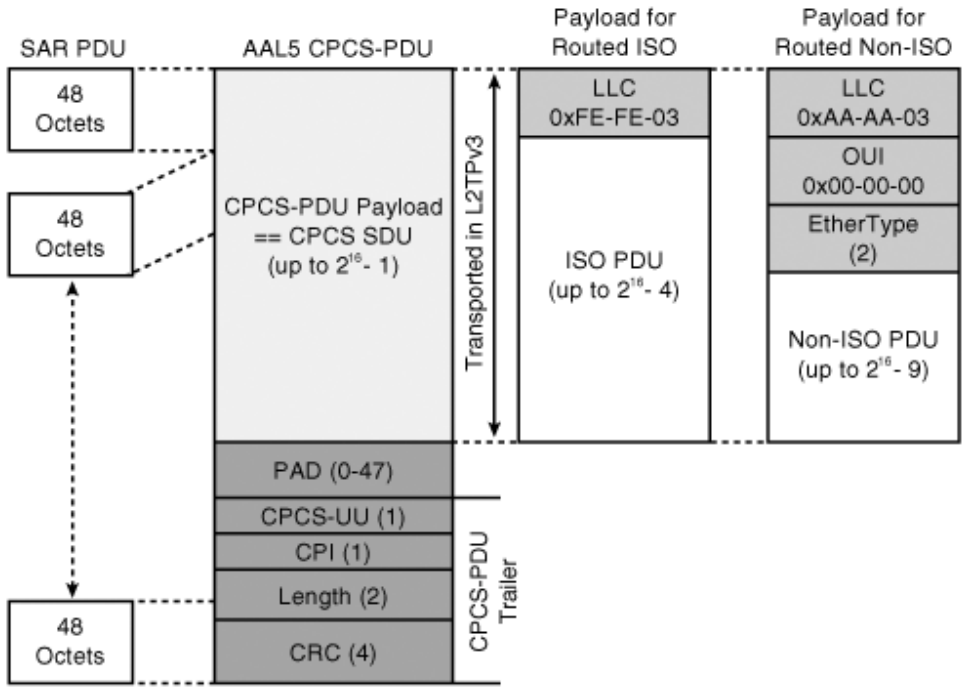
    - Single cell relay

    - Cell concatenation

Figure 12-6 shows a graphical definition of the AAL5 CPCS-SDU transported over L2TPv3. The payload that is transported in AAL5 CPCS-SDUs over L2TPv3 is the same as the one transported in AAL5 CPCS-SDUs over MPLS. From Figure 12-6, you can see that the ATM cell headers are not transported, which is why it is necessary to add the ATM-Specific Sublayer.

Figure 12-6. ***AAL5 CPCS-SDU over L2TPv3 Packet Formats***

[View full size image]



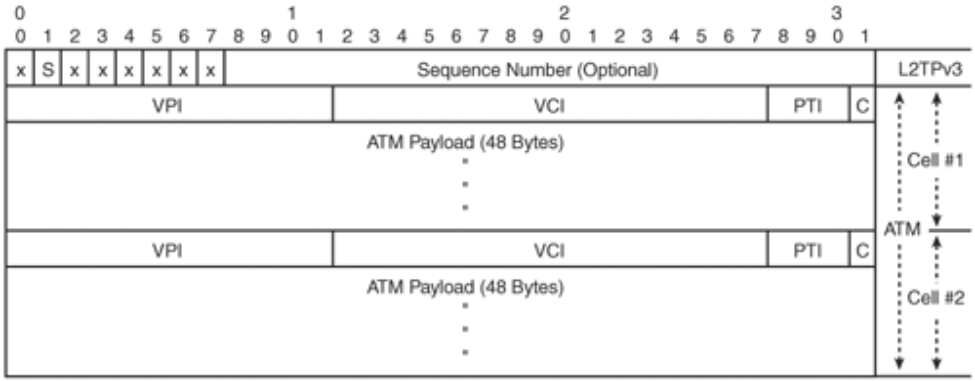
ATM cell mode over L2TPv3 also allows multiple granularities with three different services:

- ATM VCC Cell-Relay Service
- ATM VPC Cell-Relay Service
- ATM Port Cell-Relay Service

These three modes exist for both single-cell relay mode and cell concatenation mode. Figure 12-7 shows the packet format for the transport of two packed ATM cells over L2TPv3.

Figure 12-7. *Cell Relay over L2TPv3 Packet Formats*

[View full size image]



Note

In ATM cell mode, ATM layer cells are transported. This translates to a 4-byte ATM cell header plus a 48-byte cell payload. The fifth byte in the ATM cell header contains the header error control (HEC) and is appended by the Transmission Convergence (TC) sublayer in the ATM physical layer. The HEC byte is not transported over an L2TPv3 ATM pseudowire.

Because the transport of ATM over L2TPv3 has unique characteristics compared to other protocols that are transported, the control plane needs to signal these attributes that only pertain to ATM attachment circuits. To that effect, the following are two new AVPs defined for transport of ATM over L2TPv3 that are not present for other protocols:

- **ATM Maximum Concatenated Cells AVP** This AVP only applies to ATM cell relay pseudowire types. It consists of a 16-bit value that indicates the maximum number of concatenated or packed ATM cells that the sending LCCE can process at disposition. Using cell concatenation increases the bandwidth efficiency given that multiple cells share the same L2TPv3 tunneling overhead; the expense is additional latency incurred while waiting for cells to be concatenated in a single L2TPv3 packet.

- **OAM Emulation Required AVP** This AVP can be used in AAL5 CPCS-SDU mode to request OAM emulation. This is helpful when the ATM pseudowire does not support the transport of OAM cells (by setting the T-bit) in an AAL5 ATM pseudowire; therefore, you can terminate OAM cells in the LCCE. For it to work, you must use OAM emulation in both ends simultaneously. This AVP has a NULL value. The mere presence of this AVP indicates that OAM emulation is required.

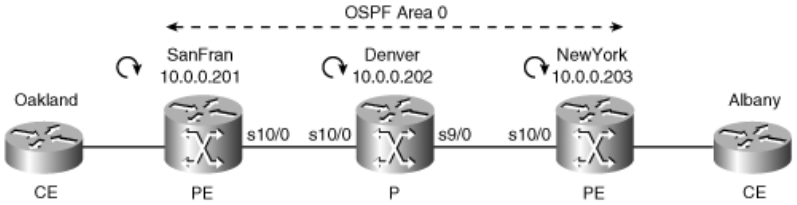
Configuring WAN Protocols over L2TPv3 Case Studies

This section discusses the configuration steps required to enable Layer 2 tunneling transport of WAN protocols using L2TPv3. Several case studies cover configuration and verification examples for HDLCoL2TPv3, PPPoL2TPv3, FRoL2TPv3, and ATMoL2TPv3.

Every one of the case studies uses the same IP PSN, shown in Figure 12-8. The goal is to demonstrate the configuration steps, verification, and troubleshooting stages required to set up Layer 2 connectivity between CE devices over the IP network.

Figure 12-8. *WAN Protocols over IP Case Study Topology*

[View full size image]



The first step is to set up the common underlying IP network. The IP PSN consists of a PE router with host name SanFran connected to a P router named Denver which, in turn, is connected to a second PE router NewYork. The three core devices have a /32 loopback configured and advertised through an Interior Gateway Protocol (IGP). (These case studies use Open Shortest Path First [OSPF].) In these examples, the PE-P links are point-to-point Cisco HDLC (C-HDLC) links with IP addresses unnumbered to the loopback interfaces' IP addresses. This effectively results in just one IP address per core device. The following list describes the required steps:

- Create a loopback interface and assign a /32 IP address to it.
- Enable IP CEF globally.
- Assign IP addresses (unnumbered to the loopbacks) to all physical links that connect the core routers.
- Enable an IGP among the core routers. These case studies use OSPF with a single area 0.

Example 12-1 shows the required configuration for the SanFran PE router. The configuration for the other two core routers is equivalent to this one.

Example 12-1. *Required Preconfiguration*

```
!
hostname SanFran
!
ip cef
!
interface Loopback0
 ip address 10.0.0.201 255.255.255.255
!
interface Serial10/0
 ip unnumbered Loopback0
!
router ospf 1
 log-adjacency-changes
```

```
 network 10.0.0.201 0.0.0.0 area 0
!
```

The highlighted lines show how you are using only one /32 IP address in the SanFran PE. You can now verify that IP routes are being distributed.

The routes highlighted are learned through OSPF. You can see in Example 12-2 that the 10.0.0.202/32 prefix with cost 65 (64 of 1544 kbps link + 1 of loopback) and the 10.0.0.203/32 prefix with cost 129 (2 * 64 of 2 * 1544 kbps links + 1 of loopback) are reachable from SanFran through Serial 10/0.

Example 12-2. *Preconfiguration Verification*

```
SanFran#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR

Gateway of last resort is not set

     10.0.0.0/32 is subnetted, 3 subnets
O       10.0.0.202 [110/65] via 10.0.0.202, 6d22h, Serial10/0
O       10.0.0.203 [110/129] via 10.0.0.202, 6d22h, Serial10/0
C       10.0.0.201 is directly connected, Loopback0
SanFran#
SanFran#traceroute 10.0.0.203

Type escape sequence to abort.
Tracing the route to 10.0.0.203

  1 10.0.0.202 20 msec 20 msec 28 msec
  2 10.0.0.203 56 msec 20 msec 20 msec
SanFran#
```

The next subsections present the following case studies:

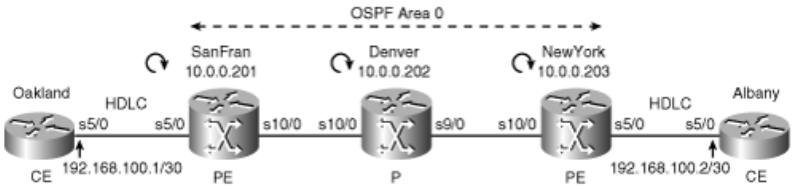
- Case Study 12-1: HDLC over L2TPv3 with Static Session
- Case Study 12-2: PPP over L2TPv3 with Dynamic Session
- Case Study 12-3: Frame Relay DLCI over L2TPv3 with Dynamic Session
- Case Study 12-4: AAL5 SDU over L2TPv3 with Dynamic Session
- Case Study 12-5: ATM Cell Relay over L2TPv3 with Dynamic Session

Case Study 12-1: HDLC over L2TPv3 with Static Session

In this section, you learn how to configure HDLCoL2TPv3 according to the topology shown in Figure 12-9.

Figure 12-9. *HDLCoL2Tv3 Static Session Case Study Topology*

[View full size image]



The IP PSN provides the transport of HDLC connecting Serial 5/0 interface in the Oakland and Albany CE routers.

Configuring HDLCoL2TPv3

In Chapter 11, you learned the configuration required for static L2TPv3 sessions for Ethernet pseudowires. To recap, the creation of a pseudowire class is required because it must include the **protocol none** statement for a static session. A static session has no signaling protocol.

The pillar of configuring pseudowires is the **xconnect** command, and this case is no exception. For HDLCoL2TPv3, the **xconnect** command is used under the attachment circuit, which is the Serial 5/0 interface on the PE routers. It specifies the IP address and pseudowire ID of the peer PE. This case study uses a pseudowire ID (also known as the VC ID or remote end ID) of 50. The VCID binds L2TP sessions to a given attachment circuit (virtual circuit, interface, or interface bundle). For a static L2TPv3 session, enter the **manual** keyword after the **encapsulation l2tpv3** statement. You need to follow it with the previously defined pseudowire class.

These steps are shown for the SanFran endpoint in Example 12-3.

Example 12-3. *HDLCoL2TPv3 Static Configuration in SanFran*

```
!
hostname SanFran
!
pseudowire-class hdlc-v3-manual
 encapsulation l2tpv3
 protocol none
 ip local interface Loopback0
!
interface Serial5/0
 no ip address
 xconnect 10.0.0.203 50 encapsulation l2tpv3 manual pw-class hdlc-v3-manual
  l2tp id 221 238
  l2tp cookie local 4 286331153
  l2tp cookie remote 8 572662306 572662306
!
```

From Example 12-3, you can see the specification of **protocol none** for a static session under the **hdlc-v3-manual pseudowire-class**. You must set the **ip local interface** directive to a loopback interface. For dynamic sessions, the protocol is specified as **l2tpv3** followed by an optional l2tp-class. The **encapsulation manual** directive in the **xconnect** command instructs the LCCE that no signaling is to be used in the L2TPv3 control channel (or to only use the control channel for keepalives) and enters the xconnect configuration submode to configure the L2TPv3 static session parameters.

By entering the **xconnect** command specifying the encapsulation as **l2tpv3** with the **manual** keyword, you are taken into the config-if-xconn configuration mode. In this new lower-level configuration mode, you specify L2TP manual configuration commands using the **l2tp** keyword, such as local and remote session ID, local and remote cookie size and value, and hello control messages.

Table 12-3 summarizes the values chosen from the SanFran perspective and configured in Example 12-3. Note that the values are simple in hexadecimal to facilitate the decoding. Table 12-3 also shows the hexadecimal values.

Table 12-3. *L2TPv3 Manual Configuration Values for HDLCoL2TPv3*

	Local	Remote
Session ID	221 (0x000000DD)	238 (0x000000EE)
Cookie Size	4	8
Cookie Value (Low)	286331153 (0x11111111)	572662306 (0x22222222)
Cookie Value (High)	N/A	572662306 (0x22222222)

From Table 12-3, you can see that the local cookie value does not have a high-order part, because it is only 4 bytes. Example 12-4 shows the configuration in the NewYork PE router.

Example 12-4. *HDLCoL2TPv3 Static Configuration in NewYork*

```
!
hostname NewYork
!
pseudowire-class hdlc-v3-manual
 encapsulation l2tpv3
 protocol none
 ip local interface Loopback0
!
interface Serial5/0
 no ip address
 no ip directed-broadcast
 xconnect 10.0.0.201 50 encapsulation l2tpv3 manual pw-class hdlc-v3-manual
  l2tp id 238 221
  l2tp cookie local 8 572662306 572662306
  l2tp cookie remote 4 286331153
!
```

Because in a static session no protocol is involved to signal the L2TPv3 parameters such as session ID, cookie size, and cookie value, you must manually configure these values for the local and remote session endpoints. For dynamic sessions, only the local cookie size is configured. The session ID and cookie value are dynamically assigned at the local LCCE. For dynamic sessions, the remote values are signaled in L2TPv3 AVPs.

By comparing Examples 12-3 and 12-4, you can see that the manually configured local and remote session IDs, cookie sizes, and cookie values mirror each other. The local session ID, cookie size, and cookie value that are configured in SanFran are the remote ones in NewYork and vice versa. To emphasize, this case study also shows that although the local cookie size that is configured in an LCCE needs to match the

remote cookie size that is configured in the peer LCCE, the cookie sizes do not need to match in both endpoints. That is because the local cookie size in SanFran is 32 bits, whereas the local cookie size in NewYork is 64 bits.

Verifying HDLCoL2TPv3

The first verification step is to issue the command **show l2tun** (see Example 12-5).

Example 12-5. *Verifying the HDLCoL2TPv3 PE*

```
SanFran#show l2tun
 Tunnel and Session Information Total tunnels 0 sessions 1
 Tunnel control packets dropped due to failed digest 0


LocID RemID Remote Name State Remote Address Port Sessions L2TPclass

LocID       RemID       TunID   Username, Intf/                       State
                                Vcid, Circuit
221         238         0       50, Se5/0                             est
SanFran#
```

> **Note**
>
> The term *tunnel* in this chapter refers to the optional L2TPv3 control connection. L2TPv3 negotiates a session for each pseudowire. Usually, two L2TPv3 speaking peers have a single tunnel and multiple sessions.

From Example 12-5, you can see one session and no tunnels, because you only have an HDLCoL2TPv3 static session. You can also see the local and remote session ID values that you configured and a Null Tunnel ID because no tunnel is available for a static session without keepalives. The state is established. This is expected because control plane signaling does not exist, and the state is established even if the remote PE goes down. You can view more details about the L2TPv3 session using the command **show l2tun session all**, as in Example 12-6.

Example 12-6. *HDLCoL2TPv3 Session Details*

```
SanFran#show l2tun session all
 Session Information Total tunnels 0 sessions 1
 Tunnel control packets dropped due to failed digest 0


Session id 221 is up, tunnel id 0
Call serial number is 0
Remote tunnel name is
  Internet address is 10.0.0.203
  Session is manually signalled
  Session state is established, time since change 00:25:39
    180 Packets sent, 180 received
    12666 Bytes sent, 12640 received
    Receive packets dropped:
      out-of-order:              0
      total:                     0
    Send packets dropped:
      exceeded session MTU:      0
      total:                     0
```

```
  Session vcid is 50
  Session Layer 2 circuit, type is HDLC, name is Serial5/0
  Circuit state is UP
    Remote session id is 238, remote tunnel id 0
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 4 bytes, value 11 11 11 11
    remote cookie, size 8 bytes, value 22 22 22 22 22 22 22 22
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

  Sequencing is off
SanFran#
```

From Example 12-6, you can see all the session parameters locally and statically configured. The local session ID is 221, and the remote session ID is 238. Both the local and remote tunnel IDs are shown as 0, because no tunnel or tunnel ID exists. You can also see that the session is manually signaled, and the VC ID that would be advertised in the End Identifier AVP for dynamic sessions is 50. The Type is HDLC using Pseudowire Type 0x0006 from Table 12-1.

The **show l2tun session all** command displays local and remote cookie information. You can see a 32-bit local cookie of 0x11111111 and a 64-bit remote cookie of 0x2222222222222222. Finally, the local encapsulation size (added to HDLC frames tunneled over L2TPv3 toward the NewYork PE) is 32 bytes. This is 20 bytes of the IP header, 4 bytes of the remote session ID of 238 (0x000000EE), and 8 bytes of the remote cookie of 16 hexadecimal number 2s. Example 12-7 depicts this overhead using the command **show sss circuits**.

Example 12-7. *HDLCoL2TPv3 Encapsulation Details from SanFran*

```
SanFran#show sss circuits

Current SSS Circuit Information: Total number of circuits 1

Common Circuit ID 0             Serial Num 2          Switch ID 18797112
------------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  0
   Y  AES  32  45000000 00000000 FF73A5F7 0A0000C9 0A0000CB
                  000000EE 22222222 22222222

SanFran#
```

The SanFran 32-byte encapsulation consists of the following:

- **Delivery (IPv4) Header** This is the 20-byte IP header indicating IP protocol 115 (0x73) for L2TPv3.
- **L2TPv3 Session Header** This includes the remote session ID (4 bytes) and optional cookie (8 bytes).

Similarly, the NewYork side shows a 28-byte encapsulation consisting of the following:

- **Delivery (IPv4) Header** This is a 20-byte IP header indicating IP protocol 115 (0x73) for L2TPv3.
- **L2TPv3 Session Header** This includes a 4-byte remote session ID of 221 (0x000000DD) and a 4-byte remote cookie of 0x11111111.

Example 12-8 shows the NewYork HDLCoL2TPv3 encapsulation details using the command **show l2tun session all**.

Example 12-8. *HDLCoL2TPv3 Encapsulation Details from NewYork*

```
NewYork#show l2tun session all
 Session Information Total tunnels 0 sessions 1
 Tunnel control packets dropped due to failed digest 0


Session id 238 is up, tunnel id 0
Call serial number is 0
Remote tunnel name is
  Internet address is 10.0.0.201
  Session is manually signalled
  Session state is established, time since change 1w0d
    70767 Packets sent, 70757 received
    4940396 Bytes sent, 4949086 received
    Receive packets dropped:

      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 50
  Session Layer 2 circuit, type is HDLC, name is Serial5/0
  Circuit state is UP
    Remote session id is 221, remote tunnel id 0
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 8 bytes, value 22 22 22 22 22 22 22 22
    remote cookie, size 4 bytes, value 11 11 11 11
  FS cached header information:
    encap size = 28 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000
  Sequencing is off
NewYork#show sss circuits

Current SSS Circuit Information: Total number of circuits 1

Common Circuit ID 0             Serial Num 1          Switch ID 18785464
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y AES   0
   Y AES   28 45000000 00000000 FF73A5F7 0A0000CB 0A0000C9
                 000000DD 11111111
NewYork#
```

Note

As a reminder, the data message format consists of the following:

- **Delivery Header** The IPv4 header that transports the L2TPv3 packets across the IP backbone network.
- **L2TPv3 Session Header** The header that uniquely identifies tunneled traffic among multiple L2TP data sessions. It is further subdivided into the following:

  *Session ID*4 bytes.

*Cookie*0, 4, or 8 bytes.

- **L2-Specific Sublayer** The control fields that facilitate tunneling of each frame (that is, sequencing, flags).
- **L2 Payload** The Data Link Layer payload to be transported over L2TPv3.

The ultimate verification involves checking connectivity between CE devices, as shown in Example 12-9. Successful pings are highlighted.

Example 12-9. *Verifying the HDLCoL2TPv3 CE*

```
Oakland#ping 192.168.100.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.100.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/22/32 ms
Oakland#
```

Data Plane Details

Because this is a static session, it is not possible to show control plane information exchange. However, you can capture and decode the data plane packets of the ping in Example 12-9 from the SanFran PE using the commands **debug vpdn packet** and **debug vpdn packet detail**. See Example 12-10.

Example 12-10. *Capturing and Decoding HDLCoL2TPv3 Packets*

```
SanFran#
SanFran#debug vpdn packet
VPDN packet debugging is on
SanFran#debug vpdn packet detail
VPDN packet details debugging is on
SanFran#
02:01:13: L2TP:(Tnl0:Sn221):FS/CEF Into tunnel (SSS): Sending pak
02:01:13: L2TP:(Tnl0:Sn221):FS/CEF Into tunnel: Sending 136 byte pak
contiguous pak, size 136
         45 00 00 88 06 4E 00 00 FF 73 9F 21 0A 00 00 C9
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         IPv4 Delivery Header (IP protocol L2TPv3)

         0A 00 00 CB 00 00 00 EE 22 22 22 22 22 22 22 22
      ...^^^^^^^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^
IPv4 Delivery Header Session Id Cookie (Remote)
                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                     L2TP Session Header
         0F 00 08 00 45 00 00 64 00 2D 00 00 FF 01 72 17
         ^^ ^^ ^^^^^ ^^^^^...
         |  |  |     Begins IP Packet
         |  |  etype = IPv4
         |  Control
         Address = Unicast Frame

         C0 A8 64 01 C0 A8 64 02 08 00 FC 51 00 09 00 00
         00 00 00 00 24 3F 5D B0 ...
```

```
02:01:13: L2TP:(Tnl0:Sn221):CEF Into tunnel (SSS): Pak send successful
02:01:13: L2X:CEF From tunnel: Received 136 byte pak
contiguous pak, size 136

        0F 00 08 00 45 00 00 84 05 72 00 00 FD 73 A2 01
        ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
        HDLC L2 IPv4 Delivery Header (IP protocol L2TPv3)

        0A 00 00 CB 0A 00 00 C9 00 00 00 DD 11 11 11 11
     ...^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^^
           IPv4 Delivery Header Session Id Cookie (Local)
                                ^^^^^^^^^^^^^^^^^^^^^^^^
                                L2TP Session Header
        0F 00 08 00 45 00 00 64 00 2D 00 00 FF 01 72 17
        ^^ ^^ ^^^^^ ^^^^^...
        |  |  |     Begins IP Packet
                  |  |  etype = IPv4
                   |  Control
                 Address = Unicast Frame

        C0 A8 64 02 C0 A8 64 01 00 00 04 52 00 09 00 00
        00 00 00 00 24 3F 5D B0 ...
02:01:13: L2TP:(Tnl0:Sn221):CEF From tunnel: Pak send successful
SanFran#
```

Note

Note that in Example 12-10 and several of the following examples that deal with packet decoding, the offline hand decoding of the packets is shown in bold.

Example 12-10 shows two packets captured in the SanFran PE. The highlighted portion indicates the overhead added to the HDLC frames that are transported. The first packet labeled "Into tunnel" is an ICMP Echo received from Oakland and forwarded to the L2TPv3 tunnel. The second one labeled "From tunnel" is the ICMP Echo Reply received from Denver P and forwarded to the Oakland CE. It is worth noting that the imposition packets (that is, the "Into tunnel" packets) display the IPv4 and L2TPv3 headers in addition to the HDLC payload, whereas the disposition packets (that is, the "From Tunnel" packets) also include the data-link layer header (C-HDLC) between the PE and P routers.

The L2TPv3 portion of the first packet contains the following fields:

- Session ID: 238
- Cookie: 2222222222222222

Note

Similar to AToM, L2TPv3 is a stateful protocol in which the PE device stores the state of interaction after connection initialization. This implies that it is impossible to perform a nonheuristic decode by mere inspection of a single packet out of context and lacking the state information.

Example 12-10 shows this behavior. Without the state information, you do not know the cookie size (0, 4, or 8), the presence of an Layer 2-Specific Sublayer, or the encapsulated Layer 2 protocol. For AToM, you do not know the presence of the control word or the Layer 2 protocol that is being tunneled.

You can see that the HDLC frames are transported in their entirety, including the following:

- **Address** 0x0F for unicast frame
- **Control** 0x00
- **Ethertype** 0x0800 for IPv4
- **IPv4 packet** The HDLC payload is the IPv4 packet of the CE. It contains the ICMP echo request, which is often referred to as IP-framed because it is IP transported over a Layer 2 frame over L2TPv3.

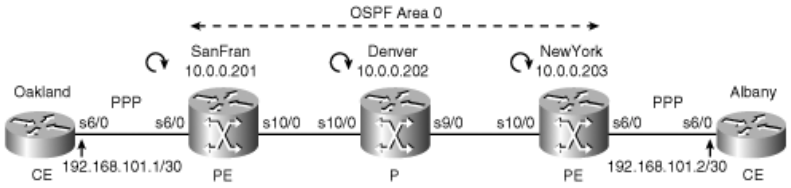
The 0x7E flags and FCS are stripped at imposition and regenerated at disposition.

Case Study 12-2: PPP over L2TPv3 with Dynamic Session

Both static and manual L2TPv3 sessions are limited in that they are prone to configuration errors and do not allow for dynamic pseudowire status notifications. You can use them, however, in small deployments or when a peer does not support dynamic sessions. Static sessions with keepalives are an intermediate stage between static and dynamic sessions that was explored in Chapter 11. The real scalability and manageability advantages of L2TPv3 occur in dynamic sessions. This section presents a case study of PPPoL2TPv3 with dynamic sessions using the topology shown in Figure 12-10.

Figure 12-10. ***PPPoL2TPv3 Dynamic Session Case Study Topology***

[View full size image]



Configuring PPPoL2TPv3

You can divide the configuration for the PE routers of SanFran and NewYork into four separate yet related steps:

Step 1. Using the **l2tp-class** command, create an L2TP class to serve as a template for L2TPv3 sessions, and configure the required parameters. This case study specifies authentication and a cookie size of 4 bytes. This step is optional.

Step 2. Using the **pseudowire-class** command, create a pseudowire class that specifies L2TPv3 encapsulation. In this pseudowire class, specify the protocol as l2tpv3 for a dynamic session referencing the l2tp-class template from Step 1.

Step 3. Configure the attachment circuit. In this case, you are limited to configuring PPP encapsulation using the **encapsulation ppp** command under the Serial interface and disabling CDP.

Step 4. Apply an **xconnect** statement under the attachment circuit (serial interface) from the Step 3 interface. The statement should specify the remote peer, VC ID, and pseudowire class from Step 2.

In this example, you also enable sequencing processing in both directions under the **xconnect** command in Step 4. You can also configure sequencing under the pseudowire class. See Example 12-11 for the required configuration in the SanFran PE.

Example 12-11. *Configuring PPPoL2TPv3*

```
!
hostname SanFran
!
l2tp-class l2tpv3-wan
 authentication
 password 0 cisco
 cookie size 4
!
pseudowire-class wan-l2tpv3-pw
 encapsulation l2tpv3
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
!
interface Serial6/0
 no ip address
 encapsulation ppp
 no cdp enable
 xconnect 10.0.0.203 60 pw-class wan-l2tpv3-pw sequencing both
!
```

It is interesting to observe in Example 12-11 under the l2tp-class l2tpv3-wan that the password controls not only Challenge Handshake Authentication Protocol (CHAP) authentication but also the AVP hiding. You can configure the **hidden** command under the l2tp-class to hide AVPs in control messages by encrypting AVP values with a shared secret between LCCEs that derive a unique shared key via an HMAC-MD5 keyed hash. You can also specify the host name used for authentication. The router host name is the default.

It is important to note that PPP runs transparently between CE devices, and the PEs do not participate in PPP negotiation. After you enter the **xconnect** command in the PE interface, the PPP state machine goes into a closed state. Example 12-12 was captured using the **debug ppp negotiation** command.

Example 12-12. *PPP Negotiation in a PE Device*

```
SanFran(config-if)#xconnect 10.0.0.203 60 pw-class wan-l2tpv3-pw sequencing both
SanFran(config-if)#
00:03:53: Se6/0 LCP: O TERMREQ [Open] id 4 len 4
00:03:53: Se6/0 LCP: State is Closed
00:03:53: Se6/0 PPP: Phase is DOWN
SanFran(config-if)#
```

Verifying PPPoL2TPv3

The L2TPv3 Layer 2 transport and tunneling feature includes multiple commands that present different information about L2TPv3 tunnels and sessions. The first command you can check is **show l2tun**. It displays tunnel and session summary information (see Example 12-13).

Example 12-13. *Displaying the L2TPv3 Tunnel and Session Summary*

```
SanFran#show l2tun
 Tunnel and Session Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0
```

```
LocID RemID Remote Name   State  Remote Address  Port  Sessions L2TPclass
61936 64821 NewYork       est    10.0.0.203      0     1        l2tpv3-wan

LocID       RemID       TunID       Username, Intf/                    State
                                    Vcid, Circuit
54459       51837       61936       60, Se6/0                          est
221         238         0           50, Se5/0                          est
SanFran#
```

You can see that, as opposed to Case Study 12-1, a tunnel now exists (because you have a dynamic session), in addition to a new session.

You can divide the output of the **show l2tun** command into three areas:

- Tunnel and session summary information
- Tunnel (control connection) summary information
- Session summary information

In the tunnel summary information in Example 12-13, you can see that the local tunnel ID is 61936. This number will become significant in the next section, "Control Plane Negotiation," when you analyze debug command output. The tunnel state is established, and it is using the l2tpv3-wan L2TP class as configured. You can also see that one session is negotiated in this Control Connection because the first session is static. Finally, the remote port is as 0 because the current implementation supports L2TPv3 directly over IP, which means there is no port.

You can see two sessions in the session summary information. The session that uses VC ID 50 indicates a Tunnel ID of 0 because it is static (HDLCoL2TPv3). The session with VC ID 60 uses tunnel ID 61936 (control connection) with local and remote session IDs of 54459 and 51837, respectively. The attachment circuits are serial interfaces, because in both sessions you configured a port-to-port type of tunneling service.

Note

Normally, a single L2TPv3 Control Connection (tunnel) exists between two peer LCCEs or PE routers. It advertises and negotiates capabilities and sessions.

To have multiple tunnels between PE routers, you can set up multiple loopback addresses. Multiple tunnels between different PE routers using multiple loopback interfaces (also referred to as multiple tunnel loopbacks in this context) provide multipath load sharing in the IP core network.

You can even configure two loopback addresses (loopback 1 and loopback 2) in a single router and create two L2TPv3 endpoints in two attachment circuits in the same router. To achieve this, you can build an **xconnect** toward loopback 2 and VCID 100 using loopback 1 as **ip local interface** in the **pseudowire-class** template 1, and the other endpoint **xconnect** toward loopback 1 using loopback 2 as **ip local interface** in the **pseudowire-class** template 2. By using two loopback IP addresses, you can have two local L2TPv3 tunnels (mirror to each other) with one hairpinning or local switching (linking two attachment circuits in the same router) connection. This hairpinning configuration is unique to L2TPv3. It is not allowed in AToM.

You can find the remaining commands that provide more detailed information or different group summaries hanging off the **show l2tun** exec parser tree by adding different keywords. In particular, you can choose between tunnel or session information. In either case, the **all** keyword displays all details. Example 12-14 shows detailed tunnel information.

Example 12-14. ***Displaying L2TPv3 Control Connection Information***

```
SanFran#show l2tun tunnel all
 Tunnel Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0


Tunnel id 61936 is up, remote id is 64821, 1 active sessions
  Tunnel state is established, time since change 00:04:37
  Tunnel transport is IP (115)

  Remote tunnel name is NewYork
    Internet Address 10.0.0.203, port 0
  Local tunnel name is SanFran
    Internet Address 10.0.0.201, port 0
  Tunnel domain is
  VPDN group for tunnel is -
  L2TP class for tunnel is l2tpv3-wan
  69 packets sent, 70 received
  3306 bytes sent, 3644 received
  Control Ns 6, Nr 8
  Local RWS 1024 (default), Remote RWS 1024 (max)
  Tunnel PMTU checking disabled
  Retransmission time 1, max 1 seconds
  Unsent queuesize 0, max 0
  Resend queuesize 0, max 1
  Total resends 0, ZLB ACKs sent 7
  Current nosession queue check 0 of 5
  Retransmit time distribution: 0 0 0 0 0 0 0 0 0
  Sessions disconnected due to lack of resources 0
SanFran#
```

Example 12-14 shows the local and remote tunnel IDs, the encapsulation of L2TPv3oIPv4 (with IPv4 protocol number 115) that the tunnel is using, the remote and local tunnel names (the name equals the router host name by default) and IP addresses, the L2TP class used, and the control sequence numbers.

Following are the control sequence numbers:

- **Ns** Sequence number sent (my sequence number). This is the sequence number for the particular control message. It is incremented by 1 for each message sent.
- **Nr** Sequence number received (your sequence number seen plus 1). This is the sequence number expected to be received in the next control message. It is set to the Ns of the last message received in order plus 1.

  Note

  The control message sequence numbers Ns and Nr in the control message header are included for all control messages and ensure reliable delivery of control messages. Do not mistake these sequence numbers with the optional data plane sequencing in the L2-Specific Sublayer in data packets that is configured by using the **sequencing** command.

See Example 12-15 for the PPPoL2TPv3 session details.

Example 12-15. ***Displaying L2TPv3 Session Information***

```
SanFran#show l2tun session all vcid 60
 Session Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0


Session id 54459 is up, tunnel id 61936
Call serial number is 3084400000
Remote tunnel name is NewYork
  Internet address is 10.0.0.203
  Session is L2TP signalled
  Session state is established, time since change 00:05:42
    83 Packets sent, 84 received
    3840 Bytes sent, 4177 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 60
  Session Layer 2 circuit, type is PPP, name is Serial6/0
  Circuit state is UP
    Remote session id is 51837, remote tunnel id 64821
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 4 bytes, value 5B AD 54 4D
    remote cookie, size 4 bytes, value 9B 16 16 5E
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

  Sequencing is on
    Ns 83, Nr 84, 0 out of order packets received
SanFran#
```

From Example 12-15, you can see that the output of the **show l2tun session all vcid 60** command is similar to a static session. However, many of the values displayed have been dynamically negotiated. You can see that the session is L2TP signaled. The output shows the local and remote session and tunnel IDs, local and remote cookie sizes, and values. The session (pseudowire) state is established and the circuit (attachment circuit) state is UP. The end of the command output displays sequencing information.

As usual, the definitive verification is to test connectivity between CE devices. See Example 12-16 for a ping from the Oakland CE highlighting successful pings.

Example 12-16. *Connectivity Verification from the CEs*

```
Oakland#ping 192.168.101.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.101.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/24/32 ms
Oakland#
```

Control Plane Negotiation

This section demonstrates the following two L2TPv3 control plane negotiations from the SanFran PE router debug output:

- Control connection (tunnel) establishment
- Session (pseudowire) establishment

The debugs that are enabled are **debug vpdn l2x-events** and **debug vpdn l2x-packets**. They display L2TP protocol events and packets, including AVP parsing.

Example 12-17 shows the debug output for the control connection establishment, highlighting the L2TPv3 messages and their respective state transitions. The output includes all the L2TP events, but only some of the more interesting packet and AVP details. The AVPs that were removed for brevity are indicated.

Example 12-17. *L2TPv3 Control Channel (Tunnel) Negotiation*

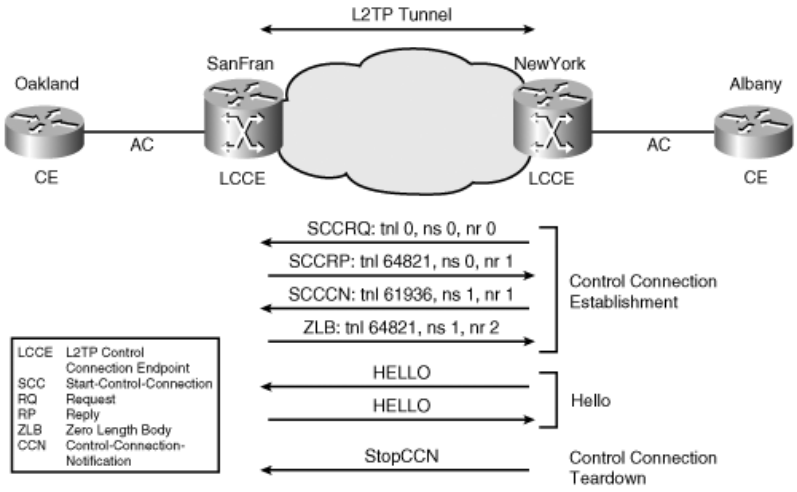
```
SanFran#
00:05:58: L2X: Parse AVP 0, len 8, flag 0x8000 (M)
00:05:58: L2X: Parse SCCRQ
! AVP 2 Protocol Version and AVP 6 Firmware Version omitted for brevity
00:05:58: L2X: Parse AVP 7, len 13, flag 0x8000 (M)
00:05:58: L2X: Hostname NewYork
! AVP 8 Vendor Name, AVP 10 Rx Window Size, AVP 11 Chlng omitted for brevity
! AVP 13 Chlng Resp, Cisco AVP 10 Cisco Draft omitted for brevity
00:05:58: L2X: Parse Cisco AVP 1, len 10, flag 0x8000 (M)
00:05:58: L2X: Assigned Control Connection ID 64821
00:05:58: L2X: Parse Cisco AVP 2, len 22, flag 0x8000 (M)
00:05:58: L2X: Pseudo Wire Capabilities List
00:05:58: L2X:   FR-DLCI [0001], ATM-AAL5 [0002], ATM-Cell [0003],
00:05:58: L2X:   Ether-Vlan [0004], Ether [0005], HDLC [0006],
00:05:58: L2X:   PPP [0007], ATM-VCC-Cell [0009],
00:05:58: L2X:   ATM-VPC-Cell [000A], IP [000B]
00:05:58: L2X: I SCCRQ, flg TLS, ver 3, len 144, tnl 0, ns 0, nr 0
00:05:58: L2TP: I SCCRQ from NewYork tnl 64821
00:05:58: Tnl61936 L2TP: Got a challenge in SCCRQ, SanFran
00:05:58: Tnl61936 L2TP: Control connection authentication skipped/passed.
00:05:58: Tnl61936 L2TP: New tunnel created for remote NewYork, address 10.0.0.203
00:05:58: Tnl61936 L2TP: O SCCRP to NewYork tnlid 64821
00:05:58: Tnl61936 L2TP: O SCCRP, flg TLS, ver 3, len 166, tnl 64821, ns 0, nr 1
00:05:58: Tnl61936 L2TP: Control channel retransmit delay set to 1 seconds
00:05:58: Tnl61936 L2TP: Tunnel state change from idle to wait-ctl-reply
00:05:58: Tnl61936 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
00:05:58: Tnl61936 L2TP: Parse SCCCN
00:05:58: Tnl61936 L2TP: No missing AVPs in SCCCN
00:05:58: Tnl61936 L2TP: I SCCCN, flg TLS, ver 3, len 42, tnl 61936, ns 1, nr 1
00:05:58: Tnl61936 L2TP: I SCCCN from NewYork tnl 64821
00:05:58: Tnl61936 L2TP: Got a response in SCCCN, from remote peer NewYork
00:05:58: Tnl61936 L2TP: Tunnel Authentication success
00:05:58: Tnl61936 L2TP: Control connection authentication skipped/passed.
00:05:58: Tnl61936 L2TP: Tunnel state change from wait-ctl-reply to established
00:05:58: Tnl61936 L2TP: O ZLB ctrl ack, flg TLS, ver 3, len 12, tnl 64821, ns 1, nr 2
00:05:58: Tnl61936 L2TP: SM State established
SanFran#
```

The prefix to the debug output varies from L2X to Tnl*XXXX* L2TP when more information is known or negotiated.

Figure 12-11 shows the control connection (tunnel) establishment as seen from SanFran PE, which is a graphical representation of the debug output from Example 12-17.

Figure 12-11. *L2TPv3 Control Connection (Tunnel) Establishment*

[View full size image]



You can correlate the output from Example 12-17 to the corresponding steps in Figure 12-11:

- **1 SCCRQ** In the first part of the three-way handshake, the SanFran PE receives a Start-Control-Connection Request (SCCRQ). At this point, the local tunnel is shown as 0 because it has not been created yet. Ns is also 0, because this is the first message sent from NewYork, and Nr is 0 because NewYork has not received messages from SanFran. The remote tunnel ID is 64821, as shown earlier in Example 12-13. This remote tunnel ID from SanFran's perspective is shown as Assigned Control Connection ID in the message received from NewYork in Example 12-17. A local tunnel is created with the tunnel ID of 61936, and it moves onto the next step. Some of the AVPs in the SCCRQ message are as follows:

  Control Message (AVP 0)

  Hostname NewYork

  Assigned Control Connection ID 64821

  Pseudowire Capabilities List

- **2 SCCRP** In the second part of the three-way handshake, SanFran PE sends a Start-Control-Connection Reply (SCCRP). The remote tunnel ID in the message sent is 64821. The Ns is 0 because this is the first message sent from SanFran, but Nr is now 1 because of the SCCRQ received from NewYork in Step 1 with Ns of 0. The tunnel state changes to wait-ctl-reply.

- **3 SCCCN** In the third part of the three-way handshake, the SanFran PE receives a Start-Control-Connection Connected (SCCCN). The tunnel ID in the packet received equals the local tunnel ID of 61936. Both Nr and Ns are 1. Nr is 1 because the Ns received in SCCRP is 0; Ns is 1 because the previous Ns sent in SCCRQ message was 0. At this point, the tunnel is established.

- **4 ZLB** The SanFran PE sends a Zero Length Body (ZLB) message as an acknowledgment. Ns is 1 (which is Ns in SCCRP sent plus 1), and Nr is 2 (which is Ns received in SCCCN received plus 1).

Note

Both in the tunnel and session establishment, many of the new AVPs that are defined for L2TPv3 in the base IETF L2TPv3 specification are sent with the Cisco Systems vendor ID of 9 (SMI Network Management Private Enterprise Codes from http://www.iana.org/assignments/enterprise-numbers). This is because the AVP types are yet to be assigned. When IANA assigns Cisco routers, the routers send the AVPs with IETF Vendor ID of 0 and accept both IETF and Cisco AVPs, giving priority to IETF AVPs.

The second control plane negotiation shown is the session (pseudowire) establishment. The debug output for the session establishment is shown in Example 12-18, highlighting the L2TPv3 messages and their respective state transitions.

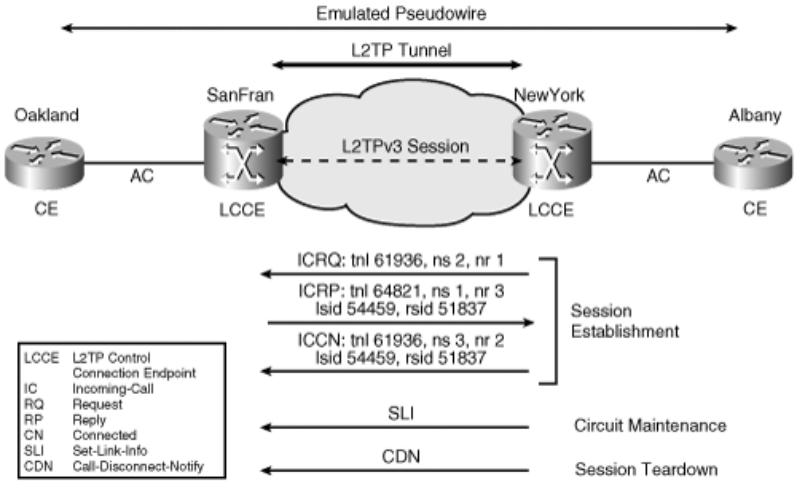
Example 12-18. *L2TPv3 Session Negotiation*

```
00:05:58: Tnl61936 L2TP: Parse ICRQ
! AVP 0 Control Message and AVP 15 Serial Number omitted for brevity
00:05:58: Tnl61936 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
00:05:58: Tnl61936 L2TP: Local Session ID 51837
00:05:58: Tnl61936 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
00:05:58: Tnl61936 L2TP: Remote Session ID 0
00:05:58: Tnl61936 L2TP: Parse Cisco AVP 5, len 10, flag 0x8000 (M)
00:05:58: Tnl61936 L2TP: Assigned Cookie
         9B 16 16 5E
00:05:58: Tnl61936 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
00:05:58: Tnl61936 L2TP: Pseudo Wire Type 7
00:05:58: Tnl61936 L2TP: Parse Cisco AVP 6, len 8, flag 0x0
00:05:58: Tnl61936 L2TP: End Identifier 60
! Cisco AVP 9 Session Tie Breaker, AVP 39 Seq Required omitted for brevity
00:05:58: Tnl61936 L2TP: No missing AVPs in ICRQ
00:05:58: Tnl61936 L2TP: I ICRQ, flg TLS, ver 3, len 96, tnl 61936, ns 2, nr 1
00:05:58: Tnl61936 L2TP: I ICRQ from NewYork tnl 64821
00:05:58: Tnl/Sn61936/54459 L2TP: Session sequencing enabled
00:05:58: Tnl/Sn61936/54459 L2TP: Session state change from idle to wait-connect
00:05:58: Tnl/Sn61936/54459 L2TP: Accepted ICRQ, new session created
00:05:58: Tnl/Sn61936/54459 L2TP: Session state change from wait-connect to wait-for
  service-selection-icrq
00:05:58: Tnl/Sn61936/54459 L2TP: O ZLB ctrl ack, flg TLS, ver 3, len 12, tnl 64821,
  lsid 54459, rsid 51837, ns 1, nr 3
00:05:58: Tnl/Sn61936/54459 L2TP: Started service selection, peer IP address
  10.0.0.203, VCID 60
00:05:58: Tnl/Sn61936/54459 L2TP: Session state change from wait-for-service-
  selection-icrq to wait-connect
00:05:58: Tnl/Sn61936/54459 L2TP: O ICRP to NewYork 64821/51837
00:05:58: Tnl/Sn61936/54459 L2TP: O ICRP, flg TLS, ver 3, len 64, tnl 64821, lsid 54459,
  rsid 51837, ns 1, nr 3
00:05:58: Tnl61936 L2TP: Parse ICCN
! AVP 0 Control Message AVP 24 Connect Speed omitted for brevity
00:05:58: Tnl61936 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
00:05:58: Tnl61936 L2TP: Remote Session ID 54459
00:05:58: Tnl61936 L2TP: No missing AVPs in ICCN
00:05:58: Tnl/Sn61936/54459 L2TP: I ICCN, flg TLS, ver 3, len 50, tnl 61936, lsid 54459,
  rsid 51837, ns 3, nr 2
00:05:58: Tnl/Sn61936/54459 L2TP: O ZLB ctrl ack, flg TLS, ver 3, len 12, tnl 64821,
  lsid 54459, rsid 51837, ns 2, nr 4
00:05:58: Tnl/Sn61936/54459 L2TP: I ICCN from NewYork tnl 64821, cl 51837
00:05:58: Tnl/Sn61936/54459 L2TP: Session state change from wait-connect to established
SanFran#
```

You can see the session establishment messaging in Figure 12-12.

Figure 12-12. *L2TPv3 Session Establishment*

[View full size image]



The session establishment also consists of three-way messaging:

1. **ICRQ** In the first part of the three-way handshake, the SanFran PE receives an Incoming-Call-Request (ICRQ) message from the NewYork PE. The tunnel ID is 61936, negotiated earlier. At this point, the session state machine changes to wait-connect, and a new session with session ID 54459 is created. You can see the local session ID for SanFran of 54459 in Example 12-13. The tunnel and session IDs are now prefixed to the debug output for message correlation. The SanFran PE responds in Step 2. Some of the most significant AVPs shown in the debug output for the ICRQ message are as follows:

   Local Session ID 51837 Example 12-18 shows this value in the received message from the NewYork PE. In Example 12-13, this same field is shown as the remote session ID from SanFran's perspective, because the AVP is with respect to the NewYork PE.

   Remote Session ID 0 The remote session ID is unknown to NewYork at this point.

   Assigned Cookie 9B 16 16 5E This is the cookie value that was assigned in the NewYork PE for this session. It is displayed in Example 12-18 as the parsed AVP value in the incoming message from NewYork.

   Pseudo Wire Type 7 This indicates PPP (Pseudowire Type 0x0007 from Table 12-1).

   End Identifier 60 This is the VC ID configured.

   Sequencing Required This indicates that sequencing for the pseudowire is on.

2. **ICRP** In the second part of the three-way handshake, the SanFran PE sends an Incoming-Call-Reply (ICRP) message to the NewYork PE, and the session state machine advances to the wait-connect state. This message includes SanFran's local session ID of 54459.

3. **ICCN** In the third part of the three-way handshake, the SanFran PE receives an Incoming-Call-Connected (ICCN) message from the NewYork PE, and the session state moves to established.

From the debug output and Figure 12-12, you can track the Ns and Nr values. Ns is always set to the previous Ns sent plus 1. For example, an ICRQ sent from NewYork contains Ns 2, and an ICCN sent from NewYork contains Ns 3. Nr is always set to the previous Ns received plus 1. For example, an ICRQ received in SanFran contains Ns 2, and an ICRP sent from SanFran contains Nr 3.

Data Plane Details

This section discusses some data plane details, such as encapsulation, imposition and disposition actions, and a data plane packet decode. You first see the PPPoL2TPv3 encapsulation details from the SanFran PE using the **show sss circuits** CLI command (see Example 12-19).

Example 12-19. *PPPoL2TPv3 Encapsulation Details from SanFran*

```
SanFran#show sss circuits

Current SSS Circuit Information: Total number of circuits 2

Common Circuit ID 0             Serial Num 2          Switch ID 18797112
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  2   FF03
   Y  AES  32  45000000 00000000 FF73A5F7 0A0000C9 0A0000CB
               0000CA7D 9B16165E 00000000
SanFran#
```

From Example 12-19, observe that the encapsulation of PPP frames into the tunnel is 32 bytes long, consisting of the following:

- **Delivery (IPv4) header** This is the 20-byte IP header indicating IP protocol 115 (0x73) for L2TPv3.
- **L2TPv3 Session Header** This is 8 bytes, including the following:

  Remote Session ID 4 bytes equal to 0x0000CA7D or 51837. You can see this in Example 12-13 in the **show l2tun** command output, in addition to Example 12-18 debug output as rsid 51837.

  Remote Cookie 4 bytes equal to 0x9B16165E. You can see this value in Example 12-15 in the output of the command **show l2tun session all vcid 60** displaying session details, and in Example 12-18 debug output as the Assigned Cookie value in the Cisco AVP 5 in the ICRQ message that SanFran received.

- **L2-Specific Sublayer** The Default L2-Specific Sublayer is used because **sequencing** has been configured. The **show sss circuit** command displays the 4 bytes as NULL and fills in the bytes for each packet with the appropriate value.

In Example 12-19, the PPP header address (0xFF) and control (0x03) fields are removed at imposition. Toward the attachment circuit, the encapsulation is 2 bytes long. It includes the following two fields that were removed at imposition and need to be prepended at disposition:

- **PPP Address** 1 byte equal to 0xFF.
- **PPP Control** 1 byte equal to 0x03.

To see the encapsulation in action, Example 12-20 captures two packets from the ping messages in Example 12-16. Use the two debug commands: **debug vpdn packet** and **debug vpdn packet detail**. The former one provides packet summary information, whereas the latter one displays a hexadecimal dump of the first bytes of the packet (see Example 12-20).

Example 12-20. *Capturing and Decoding PPPoL2TPv3 Packets*

```
SanFran#
00:17:15: L2TP:(Tnl0:Sn54459):FS/CEF Into tunnel (SSS): Sending pak
00:17:15: L2TP:(Tnl0:Sn54459):FS/CEF Into tunnel: Sending 134 byte pak
contiguous pak, size 134
         45 00 00 86 01 D7 00 00 FF 73 A3 9A 0A 00 00 C9
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         IPv4 Delivery Header (IP protocol L2TPv3)
```

```
         0A 00 00 CB 00 00 CA 7D 9B 16 16 5E 40 00 00 A0
      ...^^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^
IPv4 Delivery Header Rem. Sess Id Rem. Cookie L2-Specific Sublayer
                     ^^^^^^^^^^^^^^^^^^^^^^  S (Sequence flag) = 1
                     L2TP Session Header     Sequence Number = 160 (0xA0)

         00 21 45 00 00 64 00 05 00 00 FF 01 70 3F C0 A8
         ^^^^^ ^^^^^...
         |     Begins IP Packet
               PPP DLL Protocol Number - 0x0021 (IPv4)

         65 01 C0 A8 65 02 08 00 9B 51 00 01 00 00 00 00
         00 00 00 0F E2 E8 AB CD ...

00:17:15: L2TP:(Tnl0:Sn54459):CEF Into tunnel (SSS): Pak send successful
00:17:15: L2X:CEF From tunnel: Received 138 byte pak
contiguous pak, size 138
         0F 00 08 00 45 00 00 86 01 AC 00 00 FD 73 A5 C5
         ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         HDLC L2 IPv4 Delivery Header (IP protocol L2TPv3)


         0A 00 00 CB 0A 00 00 C9 00 00 D4 BB 5B AD 54 4D
      ...^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^
           IPv4 Delivery Header Loc.Sess Id Cookie (Local)
                                ^^^^^^^^^^^^^^^^^^^^^^^^
                                L2TP Session Header

            40 00 00 A1 00 21 45 00 00 64 00 05 00 00 FF 01
         ^^^^^^^^^^^ ^^^^^ ^^^^^...
         |           |     Begins IP Packet
                     |           PPP DLL Protocol Number - 0x0021 (IPv4)
                     L2-Specific Sublayer: S = 1; Sequence Number = 161 (0xA1)

         70 3F C0 A8 65 02 C0 A8 65 01 00 00 A3 51 00 01
         00 00 00 00 00 00 00 0F ...
00:17:15: L2TP:(Tnl0:Sn54459):CEF From tunnel: Pak send successful
```

Example 12-20 shows two packets captured in the SanFran PE. The highlighted portion of the hexadecimal dump indicates the overhead added to the PPP frames that are transported. The first packet labeled "Into tunnel" is an ICMP Echo that SanFran receives from Oakland and forwards into the L2TPv3 tunnel toward New York. The second packet labeled "From tunnel" is the ICMP Echo Reply received from Denver P and forwarded to Oakland CE. As before, the imposition packets (that is, the "Into tunnel" packets) display the IPv4 and L2TPv3 headers plus the PPP payload, whereas the disposition packets (that is, the "From Tunnel" packets) also include the data link layer header (HDLC in the case of the SanFran PE to the Denver P link).

The L2TPv3 portion of the first packet contains the following fields:

- Layer 2 Tunneling Protocol version 3

  Session ID: 51837

  Cookie: 9B16165E

- Default L2-Specific Sublayer

  .1.. .... = S-bit: True

  Sequence Number: 160

You can see that the payload contains part of the PPP frame, including the following:

- **PPP DLL Protocol Number** 0x0021 for IPv4.
- **IPv4 Packet** The PPP payload is the CE's IPv4 packet containing the ICMP Echo request.

However, the payload excludes the following:

- **Address** 0xFF
- **Control** 0x03

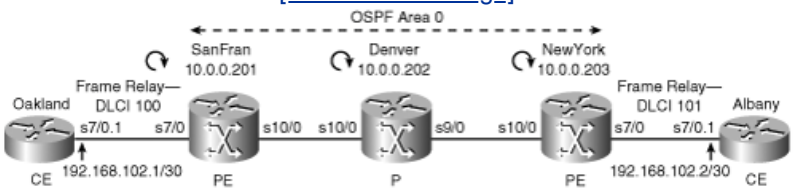
As shown in Example 12-19, the Address and Control fields with combined values of 0xFF03 are appended as a disposition encapsulation before the packet is forwarded onto the CE, and so are the 0x7E flag and recalculated FCS.

Case Study 12-3: Frame Relay DLCI over L2TPv3 with Dynamic Session

This case study covers the configuration and verification required to tunnel Frame Relay PVCs over L2TPv3. You learned a way to configure Frame Relay transport port-to-port using HDLCoL2TPv3 in Case Study 12-1. This case study deals with the DLCI-to-DLCI mode of FRoL2TPv3. Only two octet Frame Relay headers (that is, 10-bit DLCI) are supported in DLCI-to-DLCI mode. A requirement for 4-octet Frame Relay headers (that is, 23-bit DLCI) can be accommodated only in port mode using HDLCoL2TPv3. Cisco routers do not support Frame Relay extended addressing with 23-bit DLCIs as CE devices. The topology is shown in Figure 12-13.

Figure 12-13. ***FRoL2TPv3 DLCI Mode Dynamic Session Case Study Topology***

[View full size image]



You will be using different DLCIs at both ends to observe the DLCI rewrite.

Configuring FRoL2TPv3

The configuration for FRoL2TPv3 is slightly different from the other case studies. This is the first case in which the attachment circuit is a virtual circuit as opposed to an interface. The configuration of an attachment circuit in a PVC as opposed to an interface is accomplished by executing the **xconnect** command under a connect and not under the interface. In fact, after you set the encapsulation to **frame-relay** in a Serial or Packet over SONET (POS) interface, the interface no longer accepts the **xconnect** command. The attachment circuit occurs by creating the l2transport endpoint with the **connect** configuration command. This effectively generates a switched DLCI under the main interface with DLCI specified in the **connect** command. You can configure the switched DLCI by using the **frame-relay interface-dlci** command with the **switched** keyword.

This is also the first case study in which signaling messaging between PE and CE takes place. In particular, Frame Relay LMI runs on the links between PE and CE, providing a link keepalive mechanism and PVC status exchange. To achieve this, you configure the PE interfaces as Frame Relay LMI DCE after you enable the **frame-relay switching** command.

The configuration for the SanFran PE is included in Example 12-21.

Example 12-21. *Configuring the Frame Relay DLCI over the L2TPv3 PE*

```
!
hostname SanFran
!
frame-relay switching
l2tp-class l2tpv3-wan
 authentication
 password 0 cisco
 cookie size 4
!
pseudowire-class wan-l2tpv3-pw
 encapsulation l2tpv3
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
!
interface Serial7/0
 no ip address
 encapsulation frame-relay
 frame-relay intf-type dce
!
connect l2tpv3-fr-dlci Serial7/0 100 l2transport
 xconnect 10.0.0.203 70 pw-class wan-l2tpv3-pw
 !
!
```

Note

Configuring the **xconnect** statement under a **connect** as opposed to under a new subinterface saves memory and enhances the scalability of DLCI-to-DLCI mode by not requiring a Cisco IOS Software interface descriptor block (IDB) for each attachment circuit pseudowire in the PE device. The same is true for ATM PVC and permanent virtual path (PVP) modes by configuring the **xconnect** command under the PVC and PVP configuration mode, respectively. That assumes that the PVC or PVP are on the main interface, but it is not true for VLAN transport, in which a new subinterface is needed for each VLAN.

The CE configuration is included in Example 12-22. It does not differ if the Oakland CE is connected to a traditional Frame Relay switch.

Example 12-22. *Configuring the Frame Relay DLCI over the L2TPv3 CE*

```
!
hostname Oakland
!
interface Serial7/0
 no ip address
 encapsulation frame-relay
!
interface Serial7/0.1 point-to-point
 ip address 192.168.102.1 255.255.255.252
 frame-relay interface-dlci 100
!
```

Verifying FRoL2TPv3

For the Frame Relay DLCI over L2TPv3 (FR_DLCIoL2TPv3) verification, first check the connection in the SanFran PE (see Example 12-23).

Example 12-23. *Verifying the FR_DLCIoL2TPv3 Connection*

```
SanFran#show connection

ID   Name              Segment 1                Segment 2                State
==========================================================================
1    l2tpv3-fr-dlci   Se7/0 100                10.0.0.203 70            UP

SanFran#show connection name l2tpv3-fr-dlci

FR/Pseudo-Wire Connection: 1 - l2tpv3-fr-dlci
  Status - UP
  Segment 1 - Serial7/0 DLCI 100
    Segment status: UP
    Line status: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
  Segment 2 - 10.0.0.203 70
    Segment status: UP
    Requested AC state: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
SanFran#
```

You can display the two connection segments or endpoints by using the **show connection** command. The first segment is the attachment circuit, and the second segment is the pseudowire remote endpoint identified by peer IPv4 address and VC ID. You can see that all respective statuses and states are ACTIVE and UP.

Next, you can verify the switched DLCI created in the PE devices by using the **connect** command (see Example 12-24).

Example 12-24. *Verifying FR_DLCIoL2TPv3 DLCI*

```
SanFran#show frame-relay pvc summary

Frame-Relay VC Summary

              Active     Inactive      Deleted      Static
  Local          0           0            0            0
  Switched       1           0            0            0
  Unused         0           0            0            0
SanFran#
SanFran#show frame-relay pvc 100

PVC Statistics for interface Serial7/0 (Frame Relay DCE)

DLCI = 100, DLCI USAGE = SWITCHED, PVC STATUS = ACTIVE, INTERFACE = Serial7/0

  input pkts 335           output pkts 335           in bytes 119650
  out bytes 119320         dropped pkts 0            in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0           out BECN pkts 0
  in DE pkts 0             out DE pkts 0
  out bcast pkts 0         out bcast bytes 0
  switched pkts 335
  Detailed packet drop counters:
  no out intf 0            out intf down 0           no out PVC 0
  in PVC down 0            out PVC down 0            pkt too big 0
```

```
  pvc create time 05:29:16, last time pvc status changed 05:27:13
SanFran#
```

In Example 12-24 using the **summary** keyword, you can see that one switched DLCI is in the active state. Using the DLCI of 100, the command displays Frame Relay PVC details, which enables you to see that the DLCI with switched usage is created in the main interface Serial 7/0. The rest of the command output includes comprehensive PVC counters.

The next step is to verify the FR_DLCIoL2TPv3 session. You use the same command, **show l2tun session**, introducing the **brief** keyword (see Example 12-25).

Example 12-25. *Verifying the FR_DLCIoL2TPv3 Session*

```
SanFran#show l2tun session brief
 Session Information Total tunnels 1 sessions 3
 Tunnel control packets dropped due to failed digest 0

LocID       TunID       Peer-address    State     Username, Intf/
                                        sess/cir  Vcid, Circuit
54459       61936       10.0.0.203      est,UP    60, Se6/0
54467       61936       10.0.0.203      est,UP    70, Se7/0:100
221         0           10.0.0.203      est,UP    50, Se5/0
SanFran#
```

You can see that the session is established, the circuit is UP, and it is displayed as Se7/0:100 because the attachment circuit is now the logical connection with DLCI 100 in interface Serial 7/0. The details of the L2TPv3 session are shown in Example 12-26.

Example 12-26. *FR_DLCIoL2TPv3 Session Details*

```
SanFran#show l2tun session all vcid 70
 Session Information Total tunnels 1 sessions 3
 Tunnel control packets dropped due to failed digest 0


Session id 54467 is up, tunnel id 61936
Call serial number is 3084400001
Remote tunnel name is New York
  Internet address is 10.0.0.203
  Session is L2TP signalled
  Session state is established, time since change 22:37:10
    1365 Packets sent, 1365 received
    491480 Bytes sent, 490120 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 70
  Session Layer 2 circuit, type is Frame Relay, name is Serial7/0:100
  Circuit state is UP
    Remote session id is 51845, remote tunnel id 64821
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 4 bytes, value 58 47 4E 42
    remote cookie, size 4 bytes, value E6 FC CF 51
  FS cached header information:
    encap size = 28 bytes
```

```
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000
  Sequencing is off
SanFran#
```

You can see all the details of the session in Example 12-26. It is important to note that the encapsulation size is 28 bytes: 24 bytes minimum from IPv4 encapsulation plus the session ID, plus 4 bytes of cookie that was specified in the l2tpv3-wan l2tp-class. The circuit type is Frame Relay DLCI, which corresponds to 0x0001 from Table 12-1.

Finally, you can test connectivity from the Oakland CE, highlighting successful pings (see Example 12-27).

Example 12-27. *FR_DLCIoL2TPv3 Checking Connectivity from the CEs*

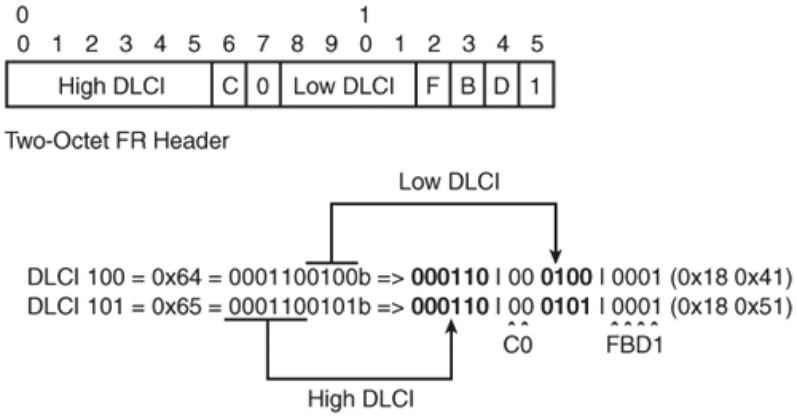
```
Oakland#ping 192.168.102.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.102.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/25/36 ms
Oakland#
```

Data Plane Details

To conclude the FRoL2TPv3 case study, this section explains FRoL2TPv3 data plane encapsulation details by capturing and decoding FRoL2TPv3 packets. However, before the actual packet decoding, you learn the details of the Frame Relay Q.922 encoding and its values as it pertain to this case study.

Figure 12-14 shows the 2-byte Q.922 header that was first introduced in Figure 12-5 in a reorganized format. It also calculates the Q.922 header value for the two DLCI values used in this case studynamely 100 and 101.

Figure 12-14. ***Frame Relay Q.922 Header Encoding***



From Figure 12-14, you can see that if the C/R, FECN, BECN, and DE bits are set to 0, the value for the Q.922 header with DLCI 100 is 0x1841. The value for the Frame Relay header with DLCI 101 is 0x1851. This is achieved by expressing the DLCI value as a 10-bit binary number and dragging the DLCI High and DLCI Low fields into their respective positions in the header.

You are now ready to capture FRoL2TPv3 packets from the ping shown in Example 12-27. To accomplish this capture, you use the commands **debug vpdn packet** and **debug vpdn packet detail** (see Example 12-28).

Example 12-28. *Capturing and Decoding FR_DLCIoL2TPv3 Packets*

```
SanFran#
*Jun 28 19:07:17.405: L2TP:(Tnl0:Sn54467):FS Into tunnel (SSS): Sending pak
*Jun 28 19:07:17.405: L2TP:(Tnl0:Sn54467):FS/CEF Into tunnel: Sending 132 byte pak
contiguous pak, size 132
         45 00 00 84 1A 20 00 00 FF 73 8B 53 0A 00 00 C9
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         IPv4 Delivery Header (IP protocol L2TPv3)

            0A 00 00 CB 00 00 CA 85 E6 FC CF 51 18 41 08 00
      ...^^^^^^^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^ ^^^^^ ^^^^^
IPv4 Delivery Header Rem.Sess Id Rem. Cookie |     etype = IPv4
                                          ^^^^^^^^^^^^^^^^^^^^^^ Q.922 Header
                                          L2TP Session Header     DLCI = 100

         45 00 00 64 00 27 00 00 FF 01 6E 1D C0 A8 66 01
         ^^^^^...
         Begins IP Packet


         C0 A8 66 02 08 00 47 A0 00 07 00 04 00 00 00 00
         09 07 2D 98 AB CD AB CD ...
*Jun 28 19:07:17.405: L2TP:(Tnl0:Sn54467):CEF Into tunnel (SSS): Pak send successful
*Jun 28 19:07:17.437: L2X:CEF From tunnel: Received 136 byte pak
contiguous pak, size 136
         0F 00 08 00 45 00 00 84 19 ED 00 00 FD 73 8D 86
         ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         HDLC L2     IPv4 Delivery Header (IP protocol L2TPv3)

         0A 00 00 CB 0A 00 00 C9 00 00 D4 C3 58 47 4E 42
      ...^^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^
            IPv4 Delivery Header Session Id Cookie (Local)
                                                         ^^^^^^^^^^^^^^^^^^^^^^^
                                                         L2TP Session Header
         18 51 08 00 45 00 00 64 00 27 00 00 FF 01 6E 1D
         ^^^^^ ^^^^^ ^^^^^...
         |     |     Begins IP Packet
               |     etype = IPv4
               Q.922 Header: DLCI = 101

         C0 A8 66 02 C0 A8 66 01 00 00 4F A0 00 07 00 04
         00 00 00 00 09 07 2D 98 ...
*Jun 28 19:07:17.437: L2TP:(Tnl0:Sn54467):CEF From tunnel: Pak send successful
SanFran#
```

In Example 12-28, you can see two FRoL2TPv3 packets captured in the SanFran PE. The portion highlighted in the hexadecimal dump corresponds to the overhead added to the Frame Relay frames that are being transported.

Similarly to previous examples, the first packet labeled "Into tunnel" is an ICMP Echo that SanFran receives from Oakland and forwards to the L2TPv3 tunnel toward New York. The second packet labeled "From tunnel" is the ICMP Echo Reply that the Denver P receives and forwards to the Oakland CE. The imposition packet that is sent into the tunnel displays the IPv4 and L2TPv3 headers. In contrast, the disposition packet that is coming from the tunnel and is later sent out of the attachment circuit also includes the data link layer header, IPv4, and L2TPv3.

The L2TPv3 portion of the first packet contains the following fields:

- Session ID: 51845
- Cookie: E6FCCF51

You can see that the payload corresponds to the complete Frame Relay frames that in turn carry IPv4 traffic, containing the following:

- **Q.922 Header** 2 bytes, including the following:

  DLCI 10 bits with a value of 100 (Higher DLCI: 0x06; Lower DLCI: 0x04)

  C/R 1 bit with a value of 0

  BECN 1 bit with a value of 0

  FECN 1 bit with a value of 0

  DE 1 bit with a value of 0

  EA Bits 2 bits with value of 0 for the first octet and 1 for the second octet.

- **Ethertype** 0x0800 indicating IPv4
- **IPv4 Packet** The IP packet from the CE being transported inside the Frame Relay encapsulation

Because you did not specify IETF Frame Relay encapsulation in the CE devices Oakland and Albany, the default of Cisco Frame Relay encapsulation is used (refer to Figure 12-5). The Ethertype is used as the upper-layer protocol identifier.

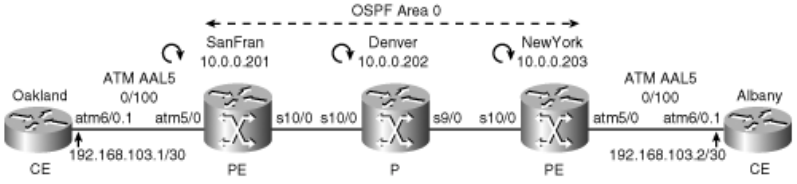
In the first packet sent out of SanFran toward New York, the Q.922 header equals 0x1841, indicating DLCI 100. The DLCI field is rewritten to 101 before the Frame Relay frame is sent out of the New York PE toward the Albany CE. In contrast, in the second packet received in the SanFran PE from the Denver P, the Q.922 header is 0x1851, designating a DLCI of 101 that the New York PE received from the Albany CE. The SanFran PE rewrites this DLCI field to a value of 100 before sending the frame to the Oakland CE.

Case Study 12-4: AAL5 SDU over L2TPv3 with Dynamic Session

This case study explains the configuration and verification of AAL5 SDU transport over L2TPv3 (AAL5_SDUoL2TPv3). The attachment circuits are the ATM PVC with VPI/VCI 0/100 in interface ATM5/0 on the SanFran end and the ATM PVC 0/100 in interface ATM5/0 on the New York PE. As you have learned, the ATM cell header including VPI and VCI is not transported in AAL5 mode. It is regenerated on segmentation at the disposition router before forwarding the packet to the CE. You use the same VPI/VCI values to support transport of raw ATM cells, such as ATM OAM cells, over the AAL5 pseudowire. The topology is shown in Figure 12-15. As you will learn in Chapter 13, "Advanced L2TPv3 Case Studies," the implication of using different VPI/VCI pairs with AAL5 SDU mode is that you cannot transport OAM cells over the pseudowire, and you need to use OAM emulation.

Figure 12-15. *AAL5_SDUoL2TPv3 Dynamic Session Case Study Topology*

[View full size image]



Configuring AAL5_SDUoL2TPv3 with Dynamic Session

The configuration steps are similar to the ones from previous case studies. You apply the **xconnect** command under the L2transport ATM PVC configuration mode. To specify the AAL5 SDU mode of tunneling and transport, configure the encapsulation as **aal5** (see Example 12-29).

Example 12-29. *Configuring AAL5_SDUoL2TPv3 in the SanFran PE*

```
!
hostname SanFran
!
pseudowire-class pw-l2tpv3-atm
 encapsulation l2tpv3
 ip local interface Loopback0
!
!
interface ATM5/0
 no ip address
 pvc 0/100 l2transport
  encapsulation aal5
  xconnect 10.0.0.203 27 pw-class pw-l2tpv3-atm
 !
!
```

Example 12-29 uses the default l2tp-class (l2tp_default_class), which means no authentication and no cookie. However, because you are using AAL5 mode, the ATM-Specific Sublayer is mandatory.

Example 12-30 shows the normal configuration of the CE device from the Oakland CE. OAM management is disabled.

Example 12-30. *Configuring the Oakland CE for the ATM PVC*

```
!
hostname Oakland
!
interface ATM6/0.1 point-to-point
 ip address 192.168.103.1 255.255.255.252
 pvc 0/100
  oam-pvc 0
  encapsulation aal5snap
 !
!
```

The configuration in the New York PE and the Albany CE is analogous to Examples 12-29 and 12-30, respectively.

Verifying AAL5_SDUoL2TPv3

To verify the status of the tunneling and transport of AAL5 SDU frames over L2TPv3, confirm the l2tun session using the summary and detailed versions of the **show** command (see Example 12-31).

Example 12-31. *Verifying the AAL5_SDUoL2TPv3 Session*

```
SanFran#show l2tun session
 Tunnel and Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0

LocID       RemID       TunID       Username, Intf/                          State
                                    Vcid, Circuit
43729       28232        23520      27, AT5/0:0/100                          est
SanFran#
SanFran#show l2tun session all
 Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0


Session id 43729 is up, tunnel id 23520
Call serial number is 2763400000
Remote tunnel name is New York
  Internet address is 10.0.0.203
  Session is L2TP signalled

  Session state is established, time since change 00:57:57
    0 Packets sent, 0 received
    0 Bytes sent, 0 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 27
  Session Layer 2 circuit, type is ATM AAL5, name is ATM5/0:0/100
  Circuit state is UP
    Remote session id is 28232, remote tunnel id 60864
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  No session cookie information available
  FS cached header information:
   encap size = 28 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000
  Sequencing is off
 SanFran#
```

You can see that the session is established, and similar to Frame Relay DLCI mode, the attachment circuit is shown as interface:virtual_circuit (in this case AT5/0:0/100). The detailed information shows the tunnel signaled and established using VC ID 27. The type is ATM AAL5 using VC Type (PW Type) 0x0002 from Table 12-1 for ATM AAL5 SDU VCC. Finally, the encapsulation size is 28 bytes, which corresponds to the following:

- Transport (IPv4) Header (20 bytes)
- L2TPv3 Header including Session ID (4 bytes) and ATM-Specific Sublayer Header (mandatory 4 bytes)

As usual, the definitive test is CE-CE connectivity (see Example 12-32).

Example 12-32. *Verifying CE-to-CE AAL5_SDUoL2TPv3 Connectivity*

```
SanFran#ping 192.168.103.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.103.2, timeout is 2 seconds:
```

```
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/20/24 ms
SanFran#
```

You can also display ATM PVC information. It is interesting to see how the ATM PVC information differs between the CE and PE routers (see Example 12-33).

Example 12-33. *ATM PVC Summary in the Oakland CE and the SanFran PE*

```
Oakland#show atm pvc interface ATM 6/0.1
               VCD /                                    Peak   Avg/Min Burst
Interface      Name          VPI   VCI   Type   Encaps   Kbps      Kbps Cells   Sts
6/0.1          1               0   100   PVC    SNAP   149760      N/A          UP
Oakland#

SanFran#show atm pvc interface ATM 5/0
               VCD /                                     Peak  Avg/Min Burst
Interface      Name          VPI   VCI   Type   Encaps    Kbps     Kbps Cells   Sts
5/0            1               0   100   PVC    AAL5    149760     N/A          UP
SanFran#
```

In the Oakland CE, the encapsulation is AAL5SNAP because you normally configure on an ATM PVC. However, in the SanFran side, the encapsulation is just AAL5, meaning AAL5 SDU L2Transport and tunneling. You can see the same distinction displaying PVC details in Example 12-34.

Example 12-34. *ATM PVC Details in the Oakland CE and the SanFran PE*

```
Oakland#show atm vc interface ATM 6/0.1 detail
ATM6/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5-LLC/SNAP, etype:0x0, Flags: 0xC20, VCmode: 0x0
OAM frequency: 0 second(s)
InARP frequency: 15 minutes(s)
Transmit priority 4
InPkts: 5, OutPkts: 5, InBytes: 540, OutBytes: 540
InPRoc: 5, OutPRoc: 5
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 125
OAM cells sent: 125
Status: UP
Oakland#


SanFran#show atm vc interface ATM 5/0 detail
ATM5/0: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5 L2transport, etype:0xF, Flags: 0x10000C2E, VCmode: 0x0
OAM Cell Emulation: not configured
Interworking Method: like to like
Remote Circuit Status = No Alarm, Alarm Type = None
InPkts: 130, OutPkts: 5, InBytes: 17179869224, OutBytes: 540
InPRoc: 0, OutPRoc: 0
InFast: 5, OutFast: 5, InAS: 0, OutAS: 0

InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
```

```
OAM cells received: 125
OAM cells sent: 125
Status: UP
SanFran#
```

Besides the encapsulation difference, you can see that the SanFran PE L2transport PVC indicates OAM Cell Emulation status and Interworking method. Chapter 13 describes OAM Cell Emulation. Chapter 14, "Layer 2 Interworking and Local Switching," covers interworking methods.

Control Plane Details

This section presents a complete session establishment control plane negotiation for L2TPv3 AA15 SDU transport. As before, the complete three-way handshake session establishment is shown highlighting the L2TPv3 control messages and AVPs that are specific to the pseudowire type. In all cases, all AVPs in the messages are parsed. Then the control message is accepted (see Example 12-35).

Example 12-35. *ATM AAL5 over L2TPv3 Session Establishment Control Plane Details*

```
SanFran#
SanFran#
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse ICRQ
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse AVP 15, len 10, flag 0x8000 (M)
*Jun 29 08:18:21.587: Tnl23520 L2TP: Serial Number -1531567296
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Jun 29 08:18:21.587: Tnl23520 L2TP: Local Session ID 28232
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
*Jun 29 08:18:21.587: Tnl23520 L2TP: Remote Session ID 0
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jun 29 08:18:21.587: Tnl23520 L2TP: Pseudo Wire Type 2
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse Cisco AVP 6, len 8, flag 0x0
*Jun 29 08:18:21.587: Tnl23520 L2TP: End Identifier 27
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse Cisco AVP 9, len 14, flag 0x8000 (M)
*Jun 29 08:18:21.587: Tnl23520 L2TP: Session Tie Breaker
         6A 57 1F 5A B3 1F 10 93
*Jun 29 08:18:21.587: Tnl23520 L2TP: Parse AVP 47, len 10, flag 0x0
*Jun 29 08:18:21.587: Tnl23520 L2TP: L2 Specific Sublayer 2
*Jun 29 08:18:21.587: Tnl23520 L2TP: No missing AVPs in ICRQ
*Jun 29 08:18:21.587: Tnl23520 L2TP: I ICRQ, flg TLS, ver 3, len 90, tnl 23520, ns 2, nr 1
*Jun 29 08:18:21.587: Tnl23520 L2TP: I ICRQ from New York tnl 60864
*Jun 29 08:18:21.587: Tnl/Sn23520/43729 L2TP: Session state change from idle to wait-
  connect
*Jun 29 08:18:21.587: Tnl/Sn23520/43729 L2TP: Accepted ICRQ, new session created
*Jun 29 08:18:21.587: Tnl/Sn23520/43729 L2TP: Session state change from wait-connect
  to wait-for-service-selection-icrq
*Jun 29 08:18:21.591: Tnl/Sn23520/43729 L2TP: Started service selection, peer IP
  address 10.0.0.203, VCID 27
*Jun 29 08:18:21.591: Tnl/Sn23520/43729 L2TP: Session state change from wait-for-
  service-selection-icrq to wait-connect
*Jun 29 08:18:21.591: Tnl/Sn23520/43729 L2TP: O ICRP to New York 60864/28232
*Jun 29 08:18:21.591: Tnl/Sn23520/43729 L2TP: O ICRP, flg TLS, ver 3, len 58, tnl 60864,
  lsid 43729, rsid 28232, ns 1, nr 3
*Jun 29 08:18:21.591: Tnl23520 L2TP: Control channel retransmit delay set to 1 seconds
*Jun 29 08:18:21.607: Tnl23520 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Jun 29 08:18:21.607: Tnl23520 L2TP: Parse ICCN
*Jun 29 08:18:21.607: Tnl23520 L2TP: Parse AVP 24, len 10, flag 0x8000 (M)
*Jun 29 08:18:21.607: Tnl23520 L2TP: Connect Speed 0
*Jun 29 08:18:21.607: Tnl23520 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Jun 29 08:18:21.607: Tnl23520 L2TP: Cisco AVP 3 is not for ICCN
*Jun 29 08:18:21.607: Tnl23520 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
*Jun 29 08:18:21.607: Tnl23520 L2TP: Remote Session ID 43729
*Jun 29 08:18:21.607: Tnl23520 L2TP: Parse AVP 47, len 10, flag 0x0
```

```
*Jun 29 08:18:21.607: Tnl23520 L2TP: L2 Specific Sublayer 2
*Jun 29 08:18:21.607: Tnl23520 L2TP: No missing AVPs in ICCN
*Jun 29 08:18:21.607: Tnl/Sn23520/43729 L2TP: I ICCN, flg TLS, ver 3, len 60, tnl 23520,
  lsid 43729, rsid 28232, ns 3, nr 2
*Jun 29 08:18:21.607: Tnl/Sn23520/43729 L2TP: I ICCN from New York tnl 60864, cl 28232
*Jun 29 08:18:21.607: Tnl/Sn23520/43729 L2TP: Session state change from wait-connect
  to established
SanFran#
```

You can note in Example 12-35 the following messages and AVPs:

- **1 ICRQ** The first way in the three-way handshake. The New York PE sends the SanFran PE an ICRQ message. The tunnel ID is 23520, as shown earlier in Example 12-31. At this point, the session state machine changes to wait-connect, and a new local session with session ID 43729 (also shown in Example 12-31) is created. Some of the most significant AVPs for AAL5 SDU shown in the debug output for the ICRQ message are as follows:

  Pseudowire Type 2 indicates ATM AAL5 SDU VCC (pseudowire type 0x0002 from Table 12-1).

  End Identifier 27 is the VC ID that is configured.

  Layer 2-Specific Sublayer 2 indicates the mandatory ATM-Specific Sublayer.

- **2 ICRP** The second way in the three-way handshake. The SanFran PE sends an ICRP message to the New York PE, and the session state machine advances to a wait-connect state.

- **3 ICCN** The third way in the three-way handshake. The New York PE sends the SanFran PE an ICCN message, and the session state moves to established. Again, the Layer 2-Specific Sublayer AVP contains a value of 2, indicating that the ATM Layer 2-Specific Sublayer is used.

Data Plane Details

This section includes the decode of AAL5oL2TPv3 packets captured in the SanFran PE using the commands **debug vpdn packet** and **debug vpdn packet detail**. Example 12-36 captures 36-byte pings to identify the end of the IPv4 packet.

Example 12-36. *Capturing and Decoding AAL5_SDUoL2TPv3 Packets*

```
SanFran#
*Jun 29 09:23:44.271: L2TP:(Tnl0:Sn43729):FS Into tunnel (SSS): Sending pak
*Jun 29 09:23:44.271: L2TP:(Tnl0:Sn43729):FS/CEF Into tunnel: Sending 72 byte pak
particle pak, size 72
         45 00 00 48 02 D9 00 00 FF 73 A2 D6 0A 00 00 C9
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         IPv4 Delivery Header (IP protocol L2TPv3)

         0A 00 00 CB 00 00 6E 48 00 00 00 00 AA AA 03 00
      ...^^^^^^^^^^^^ ^^^^^^^^^^^^ ^^^^^^^^^^^^ ^^^^^^^^^^^^...
IPv4 Delivery Header Sess Hdr    |           SNAP: LLC: AAAA03
                                          Sess Id     ATM-Specific Sublayer

         00 00 08 00 45 00 00 24 00 19 00 00 FF 01 6C 6B
      ...^^^^^^^^^^^^ ^^...
         |           Begins IP Packet
                SNAP: OUI: 000000; etype: 0x0800 (IPv4)

         C0 A8 67 01 C0 A8 67 02 08 00 78 3F 00 05 00 00
         00 00 00 00 00 67 7F 54
                       ^^^^^
```

```
                  Ends IP Packet

*Jun 29 09:23:44.271: L2TP:(Tnl0:Sn43729):CEF Into tunnel (SSS): Pak send successful
*Jun 29 09:23:44.283: L2X:CEF From tunnel: Received 76 byte pak
particle pak, size 76
         0F 00 08 00 45 00 00 48 02 DB 00 00 FE 73 A3 D4
         ^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
         HDLC L2     IPv4 Delivery Header (IP protocol L2TPv3)

         0A 00 00 CB 0A 00 00 C9 00 00 AA D1 00 00 00 00
      ...^^^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^
            IPv4 Delivery Header Sess Id     ATM-Specific Sublayer

         AA AA 03 00 00 00 08 00 45 00 00 24 00 19 00 00
         ^^^^^^^^^^^^^^^^^^^^^^^ ^^...
         |                       Begins IP Packet
                  SNAP: LLC: AAAA03; OUI: 000000; etype: 0x0800 (IPv4)

         FF 01 6C 6B C0 A8 67 02 C0 A8 67 01 00 00 80 3F
         00 05 00 00 00 00 00 00 ...
*Jun 29 09:23:44.283: L2TP:(Tnl0:Sn43729):CEF From tunnel: Pak send successful
```

Cells incoming to the SanFran PE from the Oakland CE are reassembled, and the AAL5 CPCS SDU (stripping the AAL5 CPCS-PDU trailer and padding) is encapsulated in L2TPv3, as shown in Figure 12-6, and sent over toward New York. You can see the ATM-Specific Sublayer Header being used with a NULL value because all significant ATM cell header bits are 0, and you have not enabled sequencing. You can also see the AAL5 LLC/SNAP encapsulation header, indicating an IPv4 packet. At disposition, the C-HDLC data-link layer encapsulation between the SanFran PE and the Denver P is included in the debug output.

The L2TPv3 portion of the first packet contains the following fields:

- Layer 2 Tunneling Protocol Version 3Session ID: 28232
- ATM-Specific Sublayer

  .0.. .... = S-bit: False

  .... 0... = T-bit: False

  .... .0.. = G-bit: False

  .... ..0. = C-bit: False

  ....0 = U-bit: False

  Sequence Number: 0

The payload shows part of the AAL5 frame including the LLC header:

- Logical-Link Control

  DSAP: SNAP (0xaa)

  IG Bit: Individual

  SSAP: SNAP (0xaa)

  CR Bit: Command

Control Field: U, func=UI (0x03)

Organization Code: Encapsulated Ethernet (0x000000)

Type: IP (0x0800)

In contrast to transporting AAL5 frames over the AAL5PW, you enable OAM cell management in the Oakland CE by using the PVC configuration mode command **oam-pvc manage** and capture an OAM cell that is being transported (see Example 12-37).

Example 12-37. *Capturing and Decoding OAM Cells over an AAL5_SDUoL2TPv3 Session*

```
SanFran#
*Jun 29 10:25:16.719: L2TP:(Tnl0:Sn43729):FS Into tunnel (SSS): Sending pak
*Jun 29 10:25:16.719: L2TP:(Tnl0:Sn43729):FS/CEF Into tunnel: Sending 80 byte pak
particle pak, size 80
         45 00 00 50 04 D3 00 00 FF 73 A0 D4 0A 00 00 C9
         0A 00 00 CB 00 00 6E 48 08 00 00 00 00 00 06 4A
                                ^^^^^^^^^^^ ^^^^^^^^^^^
                                            | ATM Cell Header: VPI/VCI = 0/100; PTI: 101b
                                ATM-Specific Sublayer: T-bit = 1
         18 01 00 00 00 01 FF FF FF FF FF FF FF FF FF FF
         FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6A 6A
         6A 6A 6A 6A 6A 6A 6A 6A ...
*Jun 29 10:25:16.719: L2TP:(Tnl0:Sn43729):CEF Into tunnel (SSS): Pak send successful
```

From Example 12-37, you can see that the OAM cell is encapsulated using Cell Relay over L2TPv3.

- The ATM-Specific Sublayer has the Transport bit set, an indication that it is carrying an ATM cell.
- An ATM Cell Header is included in the payload. The payload type identifier (PTI) value of 101 binary indicates an end-to-end OAM F5 flow cell.

  Note

  In AAL5 SDU mode, OAM cells that are received over the attachment circuit are sent immediately and might not maintain the relative cell order with respect to cells that comprise an AAL5 frame that is being reassembled.

The complete L2TPv3 encapsulation that indicates an admin cellincluding the T-bit in the ATM-Specific Sublayercontains the following fields:

- Layer 2 Tunneling Protocol Version 3Session ID: 28232
- ATM-Specific Sublayer

  .0.. .... = S-bit: False

  .... 1... = T-bit: True

  .... .0.. = G-bit: False

  .... ..0. = C-bit: False

....0 = U-bit: False

Sequence Number: 0

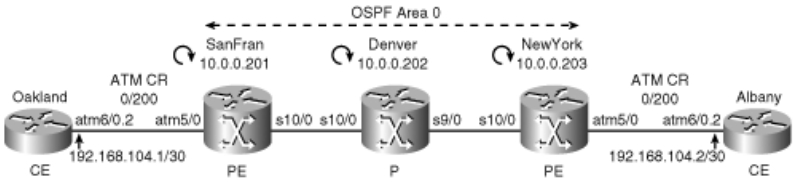
The first two nibbles in the OAM cell payload are 0x18. The OAM type 0x1 indicates a Fault Management, and the OAM Function type 0x8 specifies an OAM Cell Loopback. The next byte is the loopback indicator (LBI), and a value of 0x01 indicates that the cell must be looped back. It contains a 4-byte correlation tag (CTag) of 1 because you captured the first OAM cell after enabling OAM management. It follows with 16 bytes of binary ones for the location ID, which designates an end-to-end loopback. Unused bytes are filled with a 0x6A padding. Refer to the ITU-T Recommendation I.610, "B-ISDN Operation and Maintenance Principles and Functions," for ATM OAM details.

Case Study 12-5: ATM Cell Relay over L2TPv3 with Dynamic Session

Case Study 12-5 concentrates on ATM Cell Relay over L2TPv3 (ATM_CRoL2TPv3). Using the topology shown in Figure 12-16, you learn to configure and verify the operation of ATM VCC Cell transport over L2TPv3.

Figure 12-16. ***ATM VCC Cell Relay over L2TPv3 Dynamic Session Case Study Topology***

[View full size image]



As shown in Figure 12-16, the attachment circuits are now the PVCs with VPI/VCI pair of 0/200 in the ATM5/0 interfaces of the SanFran and New York PE routers.

Configuring ATM_CRoL2TPv3 with Dynamic Session

The configuration steps that are required to provision ATM_CRoL2TPv3 in VC mode are similar to the ones you just learned in Case Study 12-4. However, one difference is needed to specify ATM Cell Relay service as opposed to ATM AAL5 service. This difference involves specifying the encapsulation as **aal0**, meaning "no ATM Adaptation Layer," under the attachment circuit PVC (see Example 12-38).

Example 12-38. ***Configuring ATM_CRoL2TPv3 in the SanFran PE***

```
!
hostname SanFran
!
interface ATM5/0
 no ip address
 pvc 0/200 l2transport
  encapsulation aal0
  xconnect 10.0.0.203 28 pw-class pw-l2tpv3-atm
 !
```

You can see that Example 12-38 uses the same pseudowire class for comparison. The CE router's configuration is analogous to the previous case study and is the same as normal ATM PVC configuration using VPI/VCI 0/200.

Verifying ATM_CRoL2TPv3

To verify the correct functioning of the L2TPv3 pseudowire, use the **show l2tun session** command (see Example 12-39).

Example 12-39. *Verifying the ATM_CRoL2TPv3 Session*

```
SanFran#show l2tun session all vcid 28
 Session Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0

Session id 43738 is up, tunnel id 23520
Call serial number is 2763400001
Remote tunnel name is New York
  Internet address is 10.0.0.203
  Session is L2TP signalled
  Session state is established, time since change 00:00:44
    0 Packets sent, 0 received
    0 Bytes sent, 0 received
    Receive packets dropped:
      out-of-order:              0
      total:                     0
    Send packets dropped:
      exceeded session MTU:      0
      total:                     0
  Session vcid is 28
  Session Layer 2 circuit, type is ATM VCC CELL, name is ATM5/0:0/200
  Circuit state is UP
    Remote session id is 28241, remote tunnel id 60864
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  No session cookie information available
  FS cached header information:
    encap size = 24 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000
  Sequencing is off
SanFran#
```

The **show l2tun session** command for the session with the VC ID of 28 as configured in Example 12-38 shows that the session is signaled and established. The VC Type (pseudowire type) is ATM VCC Cell using pseudowire type 0x0009 for ATM n-to-one VCC cell from Table 12-1. The encapsulation size is now 24 bytes, which is the minimum possible encapsulation size. In ATM_CRoL2TPv3, you do not use the ATM-Specific Sublayer because the ATM cell headers are actually carried. The ATM L2TPv3 companion document specifies that, if needed, either the Default or the ATM Layer 2-Specific Sublayer can be used. Cisco IOS routers signal a request for the Default Layer 2-Specific Sublayer if sequencing is required.

You can also compare the ATM L2transport PVCs in the SanFran PE (see Example 12-40).

Example 12-40. *Verifying ATM over L2TPv3 PVC*

```
SanFran#show atm pvc interface ATM 5/0
               VCD /                                        Peak    Avg/Min Burst
Interface      Name          VPI   VCI   Type   Encaps     Kbps       Kbps Cells   Sts
5/0            1               0   100   PVC    AAL5     149760        N/A         UP
5/0            2               0   200   PVC    AAL0     149760        N/A         UP
SanFran#
```

Using the **show atm pvc** command, the encapsulation is now shown as AAL0, meaning ATM Cell Relay. From the CE router standpoint, the PVCs that are transported as AAL5 or Cell are the same.

Example 12-41 displays the details of the Cell Relay PVC 0/200.

Example 12-41. *ATM_CRoL2TPv3 PVC Details*

```
SanFran#show atm vc interface aTM 5/0 detail
ATM5/0: VCD: 2, VPI: 0, VCI: 200
UBR, PeakRate: 149760
AAL0-Cell Relay, etype:0x10, Flags: 0x10000C2D, VCmode: 0x0
OAM Cell Emulation: not configured
Interworking Method: like to like
Remote Circuit Status = No Alarm, Alarm Type = None
InBytes: 17179868768, OutBytes: 0
Cell-packing Disabled
OAM cells received: 104
OAM cells sent: 104
Status: UP
SanFran#
```

The encapsulation is shown as AAL0-Cell Relay. Also note that the next three lines showing OAM cell emulation, interworking, and remote circuit status are specific to ATM pseudowires and do not appear for a PVC in a CE device.

Cell Relay Details

The ATM Cell Relay tunneling and transport using L2TPv3 encompass three operational modes:

- VC mode
- VP mode
- Port mode

VP and Port mode do not support AAL5 transport because cells from different PVCs are interleaved and cannot be properly reassembled.

To configure the different ATM CR operational modes, use the same **xconnect** command with the same parameters and keywords. The difference is the context in which the command is applied, which in Cisco IOS CLI means the configuration mode. The configuration modes are as follows:

- **ATM CR VC Mode** Create the **xconnect** under **pvc l2transport** configuration mode.
- **ATM CR VP Mode** Create the **xconnect** under **atm pvp l2transport** configuration mode.
- **ATM CR Port Mode** Create the **xconnect** under ATM interface configuration mode.

Example 12-42 shows an example of each case.

Example 12-42. *Configuring Different ATM_CRoL2TPv3 Modes*

```
! ATM Cell Relay VC Mode Configuration
interface ATM5/0
 pvc 0/200 l2transport
  encapsulation aal0
  xconnect 10.0.0.203 28 pw-class pw-l2tpv3-atm
```

```
 !

! ATM Cell Relay VP Mode Configuration
interface ATM5/0
 atm pvp 5 l2transport
  xconnect 10.0.0.203 5 pw-class pw-l2tpv3-atm
!

! ATM Cell Relay Port Mode Configuration
interface ATM3/0
 xconnect 10.0.0.203 3 pw-class pw-l2tpv3-atm
 !
!
```

One key configuration difference is that in VC mode, you need to specify the encapsulation as **aal0** for Cell Relay, because AAL5 SDU is also supported on VC mode. This extra step is not needed in VP and Port modes.

By enabling the **debug vpdn l2x-packets**, you can see the control messages and parsed AVPs in the debug output. From here, you can compare the different pseudowire types used for the different CRoL2TPv3 modes by inspecting the Cisco AVP 7. Compare the values in Example 12-43 with the following pseudowire types from Table 12-1:

Example 12-43. *Pseudowire Type for Different ATM_CRoL2TPv3 Modes*

```
! ATM Cell Relay VC Mode PW Type
*Jun 29 10:32:00.071: Tnl23520 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jun 29 10:32:00.071: Tnl23520 L2TP: Pseudo Wire Type 9


! ATM Cell Relay VP Mode PW Type
*Jun 29 10:40:09.839: Tnl23520 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jun 29 10:40:09.839: Tnl23520 L2TP: Pseudo Wire Type 10


! ATM Cell Relay Port Mode PW Type
*Jun 29 10:43:29.767: Tnl23520 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jun 29 10:43:29.767: Tnl23520 L2TP: Pseudo Wire Type 3
```

- **0x0009** ATM n-to-one VCC cell (ATMoL2TPv3 Cell VC mode)
- **0x000A** ATM n-to-one VPC cell (ATMoL2TPv3 Cell VP mode)
- **0x0003** ATM Transparent cell (ATMoL2TPv3 Cell Port mode)

To conclude this section, Example 12-44 shows how these pseudowire types and attachment circuits for the different CRoL2TPv3 modes are displayed.

Example 12-44. *Pseudowire Type Display for Different ATM_CRoL2TPv3 Modes*

```
SanFran#show l2tun session
 Session Information Total tunnels 1 sessions 4
 Tunnel control packets dropped due to failed digest 0

LocID       RemID       TunID       Username, Intf/                          State
                                    Vcid, Circuit
```

```
43729      28232      23520      27, AT5/0:0/100                          est
43739      28242      23520      28, AT5/0:0/200                          est
43746      28249      23520      5, AT5/0:5                               est
43756      28259      23520      3, AT3/0:                                est
SanFran#
SanFran#! ATM AAL5 SDU Mode
SanFran#show l2tun session all vcid 27 | include type is
  Session Layer 2 circuit, type is ATM AAL5, name is ATM5/0:0/100
SanFran#

SanFran#! ATM Cell Relay VCC Mode
SanFran#show l2tun session all vcid 28 | include type is
  Session Layer 2 circuit, type is ATM VCC CELL, name is ATM5/0:0/200
SanFran#

SanFran#! ATM Cell Relay VPC Mode
SanFran#show l2tun session all vcid 5 | include type is
  Session Layer 2 circuit, type is ATM VPC CELL, name is ATM5/0:5
SanFran#

SanFran#! ATM Cell Relay Port Mode
SanFran#show l2tun session all vcid 3 | include type is
  Session Layer 2 circuit, type is ATM CELL, name is ATM3/0:
SanFran#
```

Summary

In this chapter, you learned the concepts and practical configuration, verification, and troubleshooting steps of the tunneling and transport of Layer 2 WAN protocols over an IPv4 infrastructure using L2TPv3. You worked through multiple case studies, including static and dynamic configuration of L2TPv3 sessions and transport of multiple WAN protocols such as HDLC, PPP, Frame Relay, and different flavors of ATM.

This chapter compared and contrasted L2TPv3 to transport diverse WAN protocols, in addition to L2TPv3 and AToM as pseudowire technology for WAN protocol transport. You learned L2TPv3 control plane theory and practice through multiple messaging exchange examples, data plane characteristics including the use of an Layer 2-Specific Sublayer, packet decodes, and MTU considerations.

Chapter 13. Advanced L2TPv3 Case Studies

This chapter covers the following topics:

- L2TPv3 path MTU discovery
- Advanced ATM transport over L2TPv3
- Quality of service

This chapter concentrates on advanced concepts and techniques in Layer 2 Tunnel Protocol Version 3 (L2TPv3) transport deployments. Building from the concepts and configurations covered in Chapters 10, "Understanding L2TPv3," through 12, "WAN Protocols over L2TPv3 Case Studies," this chapter covers diverse topics and case studies that involve a higher degree of complexity than previous chapters. Because the advanced deployment scenarios cover a wide range of concepts, the format of this chapter varies somewhat from other case study chapters.

This chapter starts by explaining path maximum transmission unit discovery (PMTUD), the problem it solves, the rationale behind its operation, and multiple examples. You learn details about combining PMTUD with setting the DF bit in the delivery header.

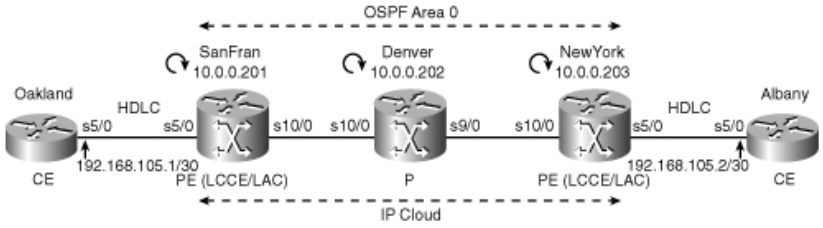
This chapter also covers two advanced cases of ATM over L2TPv3: ATM Operation, Administration, and Maintenance (OAM) emulation and ATM cell packing. This chapter concludes by describing quality of service (QoS) in L2TPv3 and explaining configuration examples.

Case Study 13-1: L2TPv3 Path MTU Discovery

In this section, you learn the PMTUD and fragmentation issues that are present with large packets in L2TPv3 networks. Just like any other encapsulation protocol, L2TPv3 adds a series of overheads or protocol control information (PCI) to a service data unit (SDU) that is being encapsulated to create the L2TPv3 protocol data unit (PDU). This section uses the network and L2TPv3 pseudowire shown in Figure 13-1 to cover PMTUD. The maximum transmission unit (MTU) is left as 1500 bytes default for all serial links in the network.

Figure 13-1. *L2TPv3 PMTUD Topology*

[View full size image]



The Problem: MTU and Fragmentation with L2TPv3

With any tunneling protocol, the packet that results from the encapsulation is N bytes longer than the original Layer 2 frame being tunneled. With L2TPv3 over IP as the tunneling protocol, the value of N has a range of 4 to 16 bytes, not including the outermost IP header. The exact value depends on the cookie size and the presence of Layer 2-Specific Sublayer. As such, the combined sizes of the encapsulated frame plus the encapsulation data might exceed the packet-switched network (PSN) path MTU, leading to the ingress PE performing fragmentation and the egress PE performing reassembly. The reassembly operation is an expensive one in terms of processing power. Avoid it in the PE device whenever possible. Cisco IOS has several configurations and features that prevent fragmentation and reassembly in the switching path if you adjust or "tune" the MTU.

> **Note**
>
> An L2TP node that exists at either end of an L2TP control connection is referred to as *L2TP control connection endpoint (LCCE)*. An LCCE can either be an L2TP access concentrator (LAC) when tunneled frames are processed at the data link layer (Layer 2) or an L2TP network server (LNS) when tunneled frames are processed at the network layer (Layer 3). A LAC cross-connects an L2TP session directly to a data link and is analogous to a Pseudowire Emulation Edge to Edge (PWE3) provider edge (PE). This chapter uses the terms LCCE, LAC (from L2TP nomenclature), and PE (from pseudowire name assignment) interchangeably to refer to an L2TPv3 tunnel endpoint, unless specifically noted.

You configure an L2TPv3 HDLC pseudowire with a remote and local cookie size of 4 bytes and with sequencing enabled. The configuration for the SanFran end is shown in Example 13-1. The NewYork PE configuration is analogous to this one.

Example 13-1. *L2TPv3 HDLC Pseudowire (HDLCPW) Configuration*

```
!
hostname SanFran
!
l2tp-class l2tpv3-wan
 cookie size 4
!
pseudowire-class wan-l2tpv3-pw
 encapsulation l2tpv3
 sequencing both
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
!
interface Serial5/0
 no ip address
 no cdp enable
 xconnect 10.0.0.203 50 pw-class wan-l2tpv3-pw
!
```

You can see in Example 13-2 that the L2TPv3 session is UP, and the encapsulation size is 32 bytes. The command **show sss circuit** also displays the encapsulation size.

Example 13-2. *L2TPv3 HDLCPW Verification*

```
SanFran#show l2tun session all vcid 50 | include Session is|state|encap
  Session is L2TP signalled
  Session state is established, time since change 00:05:41
  Circuit state is UP
    encap size = 32 bytes
SanFran#
```

The encapsulation size of 32 bytes comes from the following:

- 20 bytes of IPv4 Delivery header
- 4 bytes of L2TPv3 Session ID
- 4 bytes of L2TPv3 cookie
- 4 bytes of the default Layer 2-Specific Sublayer header used for sequencing

You also need to add the transport overhead for High-Level Data Link Control (HDLC), which is constant and equal to 4 extra bytes. Refer to Chapter 12 for details about MTU considerations.

In the transport and tunneling of HDLC frames over HDLC pseudowires (HDLCPW), encapsulating an HDLC frame received from a customer edge (CE) in L2TPv3 over IP adds 36 bytes in the core to the enclosed IP packet that the CE device sends. Before you can address what would happen if you were to send IP packets that were larger than the core MTU minus 36 bytes, you would need a baseline sample for comparison.

Start by sending 500 Internet Control Message Protocol (ICMP) ping packets that total 1464 bytes, which is exactly 1500 bytes (core MTU) - 36 bytes (total encapsulation overhead) from the Oakland CE to the Albany CE. While these packets are being sent, profile the IP Input IOS process by using the command **show processes cpu**, which displays detailed CPU utilization statistics on Cisco IOS processes. The IP Input process takes care of process switching received IP packets in Cisco IOS. Example 13-3 shows the CPU profile for the IP Input process when the packets are being transferred.

Example 13-3. *Baseline IP Input CPU Profile*

```
NewYork#show processes cpu | include util|PID|IP Input
CPU utilization for five seconds: 5%/0%; one minute: 5%; five minutes: 5%
 PID  Runtime(ms)   Invoked      uSecs   5Sec   1Min   5Min TTY Process
  18         5368       648       8283  0.07%  0.08%  0.12%   0 IP Input
NewYork#
```

You can see from Example 13-3 that the IP Input process CPU utilization is low. That is consistent with the fact that those L2TPv3 packets from the HDLC-PW are being Cisco Express Forwarding (CEF) switched. They are switched in the fast path and not in the process path.

Perform a similar experiment with packets that are larger than the core MTU minus the encapsulation overhead. Disable HDLC keepalives and Cisco Discovery Protocol (CDP) on the CE devices so that you have an accurate count of packets existing in the core routers (see Example 13-4).

Example 13-4. *CE Configuration*

```
!
hostname Oakland
!
interface Serial5/0
 ip address 192.168.105.1 255.255.255.252
 no keepalive
 serial restart-delay 0
 no cdp enable
!
```

Next, send 500 ICMP echo packets that total 1465 bytes (1 byte larger than 1500 bytes minus 36 bytes) and the don' fragment (DF) bit set in the IP header from the Oakland CE (see Example 13-5).

Example 13-5. *1465-Byte Packets from the Oakland CE*

```
Oakland#ping 192.168.105.2 repeat 500 size 1465 df-bit

Type escape sequence to abort.
Sending 500, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
Packet sent with the DF bit set
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!
Success rate is 100 percent (500/500), round-trip min/avg/max = 20/42/388 ms
Oakland#
```

Note that even though the DF bit is set and the packets do not fit, the pings are successful. Although the DF bit is set in packets that are sent from the Oakland CE device, they are further encapsulated in L2TPv3 over IPv4. The DF bit in this outer IPv4 delivery header is not set. Therefore, these oversized packets that are carrying ICMP over IPv4 over HDLC over L2TPv3 over IPv4 are being fragmented after tunnel encapsulation.

Check the CPU utilization for the IP Input process in the NewYork PE device (see Example 13-6).

Example 13-6. *IP Input CPU Profile During IP Fragmentation Reassembly*

```
NewYork#show processes cpu | include util|PID|IP Input
CPU utilization for five seconds: 20%/0%; one minute: 6%; five minutes: 1%
 PID Runtime(ms)   Invoked      uSecs   5Sec   1Min   5Min TTY Process
  18        1700       199       8542 16.04%  3.54%  0.41%   0 IP Input
NewYork#
```

The line labeled CPU utilization for five seconds shows that the CPU utilization was 20 percent, and 0 percent was spent at the interrupt level. This means that the process level used all the CPU, and the IP Input process added the delta. The reason for this huge difference is the reassembly of the fragmented IP packets carrying L2TPv3. Not surprisingly, the CPU utilization for the IP Input process jumps around 15 percent when compared to the CPU baseline profile shown in Example 13-3.

Note

Although this case study shows numbers for the CPU utilization and its variation because of reassembly, consider the numbers qualitatively and not quantitatively. The impact of reassembly in CPU performance varies significantly across platforms and traffic patterns. The intent of this case study is to show the effect of reassembly in a router CPU, although the actual variation usually deviates from the sample values shown.

In essence, IP fragmentation that is defined in RFC 791, "Internet Protocol," involves breaking up an IP datagram into several pieces that are sent in different IP packets and reassembling them later. The fact that fragmentation and reassembly is occurring is proven in Example 13-7 using the command **show ip traffic**.

Example 13-7. *IP Fragmentation and Reassembly in NewYork PE*

```
NewYork#show ip traffic | include IP stat|frag|reass
IP statistics:
  Frags: 500 reassembled, 0 timeouts, 0 couldn't reassemble
         501 fragmented, 0 couldn't fragment
NewYork#
```

A Cisco IOS router does not attempt to reassemble all IP fragments; it only fragments those that are destined to the router that need to be reassembled before decapsulation. Several issues could make you want to avoid IP fragmentation and reassembly. In a router, reassembly is an expensive operation. A router architecture is designed to switch packets as quickly as possible. Holding a packet for a relatively long period of time is not what a router is intended for; it is more of a host operation. When fragmenting a packet, a router needs to make copies of the original IP packet. When fragments are received, a Cisco IOS device chooses the largest buffer of 18 KB because the length of the total packet is unknown when the first fragment is received and before coalescing the fragments. This is an inefficient use of the buffers, but even more important is the fact that IP packets are process switched (process level switching path or slow path) for reassembly. This can degrade throughput and performance and increase CPU utilization.

Note

In Cisco IOS, multiple fragments from an IP packet are counted as a single IP packet. Therefore, the counters from Example 13-7 and Example 13-8 indicate 500 packets.

Example 13-8. *IP Reassembly Is Process Switched*

```
NewYork#show interfaces Serial 5/0 stats
Serial5/0
          Switching path    Pkts In   Chars In   Pkts Out  Chars Out
               Processor          0          0        500     734500
            Route cache        500     734500          0          0
                  Total        500     734500        500     734500
NewYork#show interfaces Serial 5/0 switching
Serial5/0
          Throttle count          0
         Drops         RP          0         SP          0
  SPD Flushes       Fast          0        SSE          0
  SPD Aggress       Fast          0
 SPD Priority     Inputs          0      Drops          0

     Protocol        Path    Pkts In   Chars In   Pkts Out  Chars Out
        Other     Process          0          0        500     734500
            Cache misses          0
                    Fast        500     734500          0          0
               Auton/SSE          0          0          0          0
NewYork#
```

To verify that the reassembly process takes place in the process path, you can use these two **show interface** commands from the NewYork PE: **show interfaces stats** and **show interfaces switching**. These commands are hidden in some IOS releases (see Example 13-8).

Example 13-8 shows that packets coming into Serial5/0 interface and sent into the tunnel are fast switched (CEF switched), but packets that are sent out of interface Serial5/0 coming from the L2TPv3 session and sent to Albany CE are process switched (switched in the process level switching path by a software process level component). You can see that the 500 IP packets sent from the Oakland CE and fragmented by the SanFran PE are process switched at the NewYork PE because of reassembly and then sent to the Albany CE device.

In summary, stay away from reassembly by avoiding fragmentation by means of MTU tuning. As you will learn in the upcoming examples, Sweep Ping is a useful tool to identify fragmentation issues and their boundary conditions.

The Solution: Path MTU Discovery

The solution to the MTU and fragmentation problem is L2TPv3 PMTUD, which is defined in RFC 1191. To prevent reassembly by the L2TPv3 edge routers, PMTUD allows the PE to dynamically adjust the Session MTU and is only supported for dynamic sessions.

Understanding PMTUD

Enabling PMTUD in the PE device further enables a set of new behaviors, as follows:

- The ingress LCCE copies the DF bit from the IP header in the CE IPv4 packet into the IPv4 delivery header. The DF bit is reflected from the inner IP header to the tunnel IP header.
- The ingress LCCE listens to ICMP Unreachable messages with code 4 to find out the path MTU and records the discovered path MTU for the session.

- The ingress LCCE inspects the IPv4 packet inside the Layer 2 frame it receives from the CE. If the IPv4 packet has the DF bit cleared and the resulting L2TPv3 packet exceeds the discovered MTU, it determines the number of fragments so that each fragment plus the encapsulation overhead is smaller than the path MTU. It fragments the CE IPv4 packet, copies the original Layer 2 header and appends it into each of the generated fragments, and sends multiple L2TPv3 packets. This procedure effectively pushes the computational expensive IPv4 reassembly into the receiving CE device and relieves the PE from being a centralized reassembly point. Note that this action occurs only after the path MTU is discovered.

- The ingress LCCE generates ICMP unreachable messages to the CE device when the IPv4 CE packet contains the DF bit set and the resulting L2TPv3 packet exceeds the discovered MTU. The MTU value informed by the PE to the CE in this ICMP unreachable is called the *adjusted MTU*. The adjusted MTU is the discovered path MTU (PMTU) in the core minus the L2TPv3 overhead (IPv4 header, Session ID, cookie, and Layer 2-Specific Sublayer). Consequently, this adjusted MTU plus the L2TPv3 overhead adds up to the core discovered PMTU and enables PMTUD applications in the customer (C) network to work correctly. Note that this action occurs only after the path MTU is discovered.

- If the path MTU has not been discovered, the ingress LCCE performs only the actions in the first two bulleted points.

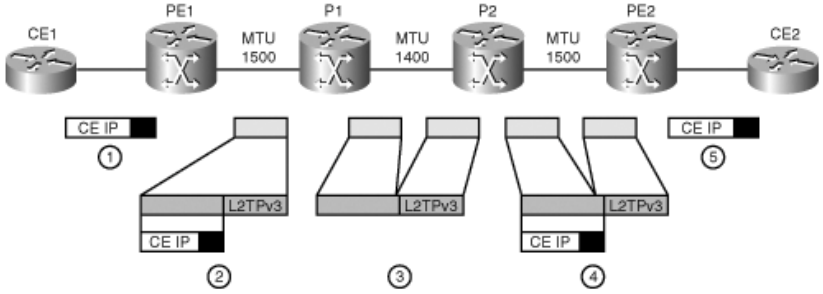
  Note

  With PMTUD enabled, the PE device decodes ICMP Destination Unreachable (Type 3) messages with code 4 ("The datagram is too big. Packet fragmentation is required, but the DF bit in the IP header is set") and updates the session MTU accordingly. This ICMP message is also referred to as the Datagram Too Big message and is defined in RFC 792, "Internet Control Message Protocol."

To illustrate the behavior and the new rules, compare a sample network behavior with and without PMTUD. Figure 13-2 shows a sample network where the PMTUD feature is disabled.

Figure 13-2. *Processing with L2TPv3 PMTUD Disabled*

[View full size image]



The operational procedures are as follows:

1. CE1 sends an IP packet that is encapsulated in HDLC.

2. PE1 encapsulates the Layer 2 frame in L2TPv3 and sends the single L2TPv3 packet onto P1. The outer IPv4 header always has the DF bit cleared.

3. P1 determines that the MTU of the outgoing interface is smaller than the L2TPv3 over IPv4 packet size. Because the DF bit in the delivery header is cleared, P1 fragments the packet and sends two fragments of an IPv4 packet to P2.

4. P2 switches the two fragments that PE2 receives. PE2 reassembles the IPv4 packet that contains the L2TPv3 packet and decapsulates the reassembled L2TPv3 packet.

5. PE2 sends the Layer 2 PDU that contains the CE IPv4 packet toward CE2.

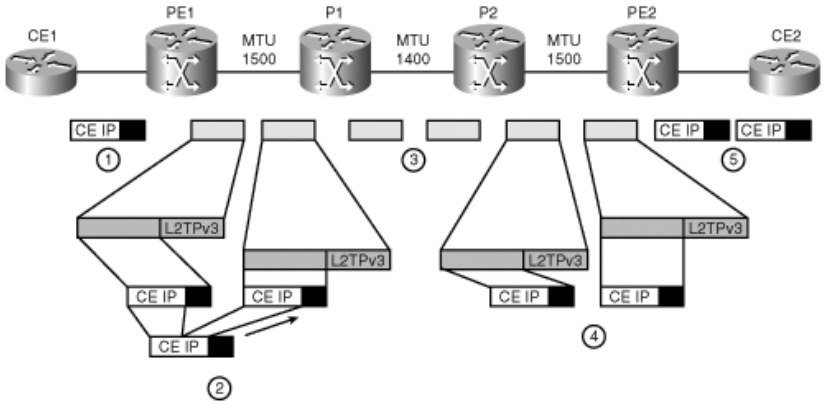
Note

The process that is described in Step 4 is the reassembly of post-fragmentation because it fragments the IPv4 delivery packet. The prefix "post" is used in reference to encapsulation. Therefore, *post-fragmentation* means fragmentation after L2TPv3 encapsulation. PE2 carries out processor-intensive reassembly in Step 4.

In contrast, Figure 13-3 presents the case in which PMTUD is enabled. This assumes that PE1 has already discovered the path MTU by processing an ICMP unreachable "datagram too big" message from the core P1 router, and the session MTU has been updated with a value equal to 1400 bytes.

Figure 13-3. *Processing with L2TPv3 PMTUD Enabled*

[View full size image]



The following steps take place when PMTUD is enabled:

1. CE1 sends an IP packet that is encapsulated in HDLC.

2. PE1 determines that the resulting L2TPv3 over IPv4 packet is greater than the path that MTU discovered. PE1 proceeds to fragment the IPv4 CE packet inside the HDLC frame and appends a copy of the HDLC header to the seconds fragment. The result is that two Layer 2 frames are passed onto L2TPv3 for encapsulation and two L2TPv3 over IPv4 packets are sent onto P1.

3. P1 router does not need to perform fragmentation.

4. PE2 receives the two L2TPv3 data packets, and as far as PE2 knows, they are from two different

Layer 2 frames. PE2 then decapsulates the two L2Tpv3 packets to end up with two Layer 2 frames containing two fragments of a single IPv4 packet from CE1.

5. PE2 sends two Layer 2 PDUs toward CE2, each containing a fragment of the CE IPv4 packet. CE2 reassembles the two fragments into an IPv4 packet.

> **Note**
>
> The process that is described in Step 2 is called prefragmentation because it fragments the data and not the delivery header packet. The prefix "pre" is used in reference to encapsulation. Therefore, *pre-fragmentation* means fragmentation before L2TPv3 encapsulation. CE2 carries out processor-intensive reassembly in Step 5.

You can see that PMTUD forces the CPU-intensive reassembly to happen in the receiving CE device. In essence, fragmentation of IP packets from the CE occurs before data enters the pseudowire (prefragmentation). The goal is that tunneled L2TPv3 packets are not fragmented along the way through the IP PSN, so the receiving PE does not perform reassembly. You have learned that the default behavior is to fragment L2TPv3 packets that are larger than the MTU.

> **Note**
>
> Another important aspect of MTU handling is that the Layer 2 frames being tunneled should fall within the MTU of the remote attachment circuit. In a bidirectional communication, this means that attachment circuit MTUs need to match. As opposed to Any Transport over MPLS (AToM), where pseudowires do not come up if an MTU mismatch occurs between the attachment circuits, the attachment circuit MTU is not advertised or enforced in L2TPv3.

Implementing PMTUD

Now that you have learned the operational procedures of PMTUD, it is time to see it in action. Example 13-9 shows the configuration changes that are required to enable PMTUD. This configuration is applied in the SanFran and NewYork PE devices.

Example 13-9. *Enabling PMTUD*

```
!
hostname SanFran
!
pseudowire-class wan-l2tpv3-pw-pmtu
 encapsulation l2tpv3
 sequencing both
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
 ip pmtu
!
interface Serial5/0
 no ip address
 no ip directed-broadcast
 no cdp enable
 no clns route-cache
```

```
 xconnect 10.0.0.203 50 pw-class wan-l2tpv3-pw-pmtu
!
```

Example 13-9 shows the **ip pmtu** command added into a new pseudowire class for the L2TPv3 pseudowire. The **ip pmtu** command can also hard-code the maximum path MTU for the session by adding the **max** keyword and the maximum path MTU value to the **ip pmtu** command. This is most useful to account for the extra overheads when the core network has further encapsulations. Example 13-10 highlights a new line of output that specifies that PMTUD is enabled for the session.

Example 13-10. *Verifying PMTUD*

```
SanFran#show l2tun session all vcid 50
 Session Information Total tunnels 1 sessions 3
 Tunnel control packets dropped due to failed digest 0


Session id 61603 is up, tunnel id 51402
Call serial number is 2310500000
Remote tunnel name is NewYork
  Internet address is 10.0.0.203
  Session is L2TP signalled
  Session state is established, time since change 00:00:23
    0 Packets sent, 0 received
    0 Bytes sent, 0 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 50
  Session Layer 2 circuit, type is HDLC, name is Serial5/0
  Circuit state is UP
    Remote session id is 5399, remote tunnel id 51995
  Session PMTU enabled, path MTU is not known
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 4 bytes, value 0B B4 A2 90
    remote cookie, size 4 bytes, value BA 12 10 7F
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

  Sequencing is on
    Ns 0, Nr 0, 0 out of order packets received
SanFran#
```

Session PMTU is now enabled, but the path MTU is still unknown because the PE has not received an ICMP unreachable "packet too big" message yet. Nevertheless, until the path MTU is known, the default behavior is the same as before. Fragmentation is not possible until the MTU is known; therefore, if you were to perform the initial experiment with this current setup, the result would be analogous to before.

To trigger the path MTU to be discovered and the session PMTU to be updated, you need to send an IP packet from the Oakland CE that is at least 1465 bytes long and has the DF bit set that will be copied over to the delivery header. Meanwhile, enable **debug ip icmp**. This example uses the same network from Figure 13-1 (see Example 13-11).

Example 13-11. *Triggering PMTU Discovery*

```
Oakland#debug ip icmp
ICMP packet debugging is on
Oakland#ping 192.168.105.2 size 1465 df-bit

Type escape sequence to abort.
Sending 5, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
Packet sent with the DF bit set
.MMMM
Success rate is 0 percent (0/5)
Oakland#
02:13:02: ICMP: dst (192.168.105.1) frag. needed and DF set unreachable rcv from
  192.168.105.2
02:13:02: ICMP: dst (192.168.105.1) frag. needed and DF set unreachable rcv from
  192.168.105.2
02:13:02: ICMP: dst (192.168.105.1) frag. needed and DF set unreachable rcv from
  192.168.105.2
02:13:02: ICMP: dst (192.168.105.1) frag. needed and DF set unreachable rcv from
  192.168.105.2
Oakland#
```

You can see in the Oakland CE that the first ping times out ("."). This is because the SanFran PE drops the first ping packet in the P network, which triggers the ICMP unreachable message that the SanFran PE absorbs, inspects, and uses to discover the path MTU. For the remaining four ICMP echo packets, you see an M character standing for MTU, which means "Could not fragment." The four M characters correspond to the four ICMP frag. needed and DF set unreachable messages sent by the SanFran PE, received by the Oakland CE, and shown in the debug output. Although the source for these ICMP unreachables is 192.168.105.2, the SanFran PE generates these ICMP unreachable messages by using a source IP address that is equal to the destination IP address in the ICMP echo packets.

The first ping that is dropped triggers an ICMP packet too big unreachable in the core that is sent toward and terminated in the SanFran PE, because the DF bit is copied onto the IPv4 core delivery header. In the basic network in Figure 13-1, the ICMP error is sent from SanFran to SanFran because all the MTUs are the same and equal to 1500 bytes. After the first ping is dropped, the PMTU is discovered. Example 13-12 shows the respective output in the SanFran PE with **debug ip icmp** and **debug vpdn l2x-events** enabled.

Example 13-12. *Discovering PMTUD in the SanFran PE*

```
SanFran#debug ip icmp
ICMP packet debugging is on
SanFran#debug vpdn l2x-events
L2X protocol events debugging is on
SanFran#
*Jul  6 03:09:47.799: ICMP: dst (10.0.0.203) frag. needed and DF set unreachable sent
to 10.0.0.201
*Jul  6 03:09:47.835: ICMP: dst (10.0.0.201)  frag. needed and DF set unreachable rcv
from 10.0.0.201
*Jul  6 03:09:47.835: Tnl46820 L2TP: Socket  MTU changed to 1500
SanFran#
SanFran#show l2tun session all vcid 50 | include PMTU
  Session PMTU enabled, path MTU is 1500 bytes
SanFran#
```

You can also see the ICMP unreachables that the SanFran PE generates with a sweep ping with verbose output. This is, in fact, how devices in the C network learn about the adjusted path MTU when they

perform their own PMTUD by setting the DF bit (see Example 13-13).

Example 13-13. *ICMP Unreachables Sent from the SanFran PE with PMTUD Enabled*

```
Oakland#ping
Protocol [ip]:
Target IP address: 192.168.105.2
Repeat count [5]: 1
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface:
Type of service [0]:
Set DF bit in IP header? [no]:  y
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: v
Loose, Strict, Record, Timestamp, Verbose[V]:
Sweep range of sizes [n]: y
Sweep min size [36]: 1460
Sweep max size [18024]: 1470
Sweep interval [1]:
Type escape sequence to abort.
Sending 11, [1460..1470]-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
Packet sent with the DF bit set
Reply to request 0 (20 ms) (size 1460)
Reply to request 1 (20 ms) (size 1461)
Reply to request 2 (36 ms) (size 1462)
Reply to request 3 (20 ms) (size 1463)
Reply to request 4 (28 ms) (size 1464)
Unreachable from 192.168.105.2, maximum MTU 1464 (size 1465)
Unreachable from 192.168.105.2, maximum MTU 1464 (size 1466)
Unreachable from 192.168.105.2, maximum MTU 1464 (size 1467)
Unreachable from 192.168.105.2, maximum MTU 1464 (size 1468)
Unreachable from 192.168.105.2, maximum MTU 1464 (size 1469)
Unreachable from 192.168.105.2, maximum MTU 1464 (size 1470)
Success rate is 45 percent (5/11), round-trip min/avg/max = 20/24/36 ms
Oakland#
```

To prove the benefits of PMTUD, perform the original experiment sending 500 1465-byte packets from the Oakland CE to the Albany CE and checking the NewYork PE and Albany CE counters. First clear all counters (see Example 13-14).

Example 13-14. *1465-Byte Packets from the Oakland CE with PMTUD*

```
Oakland#ping 192.168.105.2 size 1465 repeat 500

Type escape sequence to abort.
Sending 500, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!
Success rate is 100 percent (500/500), round-trip min/avg/max = 16/32/360 ms
Oakland#
```

Example 13-15 shows the switching statistics in the NewYork PE.

Example 13-15. *Switching Statistics in the NewYork PE*

```
NewYork#show interfaces Serial 5/0 stats
Serial5/0
          Switching path     Pkts In   Chars In   Pkts Out   Chars Out
               Processor           0          0          0           0
             Route cache         500     734500       1000      746500
                   Total         500     734500       1000      746500
NewYork#
NewYork#show interfaces Serial 5/0 switching
Serial5/0
           Throttle count          0
         Drops         RP          0           SP                  0
   SPD Flushes       Fast          0          SSE                  0
   SPD Aggress       Fast          0
  SPD Priority     Inputs          0        Drops                  0

      Protocol       Path    Pkts In    Chars In          Pkts Out    Chars Out
         Other    Process          0           0                 0            0
            Cache misses           0
                     Fast         500      734500              1000       746500
                Auton/SSE           0           0                 0            0
NewYork#
```

Example 13-15 shows that from the 500 packets that the Oakland CE sent and the SanFran PE received, the NewYork PE received 1000 packets from the Denver PE and sent them to the Albany CE. This is twice as many. The CE IPv4 packet inside each frame that was received from SanFran was divided into two fragments and sent to two separate HDLC over L2TPv3 packets with their respective HDLC transport overhead. See Example 13-16 for the SanFran PE statistics.

Example 13-16. *CE IPv4 Fragmentation and Packet Statistics in the SanFran PE*

```
SanFran#show ip traffic | include IP stat|frag|reass
IP statistics:
  Frags: 0 reassembled, 0 timeouts, 0 couldn't reassemble
         500 fragmented, 1 couldn't fragment
SanFran#
SanFran#show l2tun session packets vcid 50
 Session Information Total tunnels 2 sessions 3
 Tunnel control packets dropped due to failed digest 0

LocID       RemID      TunID        Pkts-In     Pkts-Out   Bytes-In   Bytes-Out
4437        64786      48966        1000        1001       746500     748001
SanFran#
```

Example 13-16 shows that 500 packets were fragmented. One packet could not be fragmented but triggered PMTUD. From the 500 fragmented IPv4 CE packets, 1000 L2TPv3 packets were sent into the tunnel. You can also validate the results using **debug ip packet** so that you can see the fragments in the CE device, as shown in Example 13-17.

Example 13-17. *IP Fragments in the Oakland CE*

```
Oakland#debug ip packet
IP packet debugging is on
Oakland#ping 192.168.105.2 size 1465 repeat 1
Type escape sequence to abort.
```

```
Sending 1, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
!
Success rate is 100 percent (1/1), round-trip min/avg/max = 32/32/32 ms
Oakland#
03:41:58: IP: s=192.168.105.1 (local),  d=192.168.105.2 (Serial5/0), len 1465, sending
03:41:58: IP: s=192.168.105.1 (local), d=192.168.105.2 (Serial5/0), len 1465, sending
full packet
03:41:58: IP: s=192.168.105.2 (Serial5/0), d=192.168.105.1, len 44, rcvd 2
03:41:58: IP: recv fragment from 192.168.105.2 offset 0 bytes
03:41:58: IP: s=192.168.105.2 (Serial5/0), d=192.168.105.1, len 1441, rcvd 2
03:41:58: IP: recv fragment from 192.168.105.2 offset 24 bytes
03:41:58: ICMP: echo reply rcvd, src 192.168.105.2, dst 192.168.105.1
Oakland#
```

Notice in Example 13-17 that although the Oakland and Albany CE devices send full unfragmented IP packets, they receive fragmented IP packets. You can see an IP packet composed of two fragments of lengths 44 bytes and 1441 bytes. Coalescing the two and removing the extra IP header make the 1465-byte packet (44 bytes + 1441 bytes 20 bytes = 1465 bytes).

The most important thing, however, is that the NewYork PE device does not perform reassembly, and the 1000 packets are switched in the fast switching path, which is CEF-switched in this case. As far as the NewYork PE and the Albany CE can see, it is as if the Oakland CE fragmented the packets. The CPU-intensive reassembly is now pushed onto the Albany CE (see Example 13-18).

Example 13-18. *Fragmentation Statistics in the Albany CE*

```
Albany#show ip traffic | include IP stat|frag|reass
IP statistics:
  Frags: 500 reassembled, 0 timeouts, 0 couldn't reassemble
         0 fragmented, 0 couldn't fragment
Albany#
```

You can see from Example 13-18 that the Albany CE reassembles the 500 packets. This is the goal of prefragmentation: to effectively push the costly reassembly operation to the CE device.

Combining PMTUD with DF Bit

As powerful as the PMTUD feature is, by itself it contains the weakest link. The key to the correct functioning of PMTUD is the discovery of the path MTU. As you have seen already, to discover the path MTU, you need to have a large packet with the DF bit set sent from the CE device, which means the whole process is controlled by the CE. Moreover, after a specified timer that defaults to 10 minutes, the path that MTU discovers is restored to a default value (see Example 13-19).

Example 13-19. *PMTUD Timeout*

```
SanFran#debug vpdn l2x-events
L2X protocol events debugging is on
SanFran#
*Jul 6 03:19:47.852: L2X: Restoring default pmtu for peer 10.0.0.203
```

Also, PE devices that have PMTUD enabled but do not have the path that MTU discovered copy the DF bit from the inner CE IPv4 header into the outer IPv4 delivery header but otherwise act as if PMTUD is

disabled. If PMTUD is configured but the path MTU is not discovered and CE packets do not have the DF bit set, the reassembly occurs in the PE device (see Example 13-20).

Example 13-20. *PMTU Not Discovered*

```
SanFran#show l2tun session all vcid 50 | include PMTU
  Session PMTU enabled, path MTU is not known
SanFran#

Oakland#ping 192.168.105.2 size 1465 repeat 500

Type escape sequence to abort.
Sending 500, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!
Success rate is 100 percent (500/500), round-trip min/avg/max = 28/41/108 ms
Oakland#

NewYork#show l2tun session all vcid 50 | include Packets
    500 Packets sent, 500 received
NewYork#show interface Serial5/0 stats
Serial5/0
          Switching path    Pkts In   Chars In   Pkts Out   Chars Out
               Processor          0          0        500      734500
            Route cache        500     734500          0           0
                  Total        500     734500        500      734500
NewYork#
```

You can see in Example 13-20 that PMTU is not discovered and reassembly occurs in the NewYork PE. If you trigger the discovery of path MTU and perform the same exercise, IP fragmentation reassembly is pushed onto the CE device (see Example 13-21).

Example 13-21. *PMTU Discovered*

```
Oakland#
! Triggering PMTUD
Oakland#ping 192.168.105.1 size 1465 df-bit repeat 1

Type escape sequence to abort.
Sending 1, 1465-byte ICMP Echos to 192.168.105.1, timeout is 2 seconds:
Packet sent with the DF bit set
.
Success rate is 0 percent (0/1)
Oakland#ping 192.168.105.2 size 1465 repeat 500

Type escape sequence to abort.
Sending 500, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!
Success rate is 100 percent (500/500), round-trip min/avg/max = 8/31/252 ms
Oakland#


NewYork#show l2tun session all vcid 50 | include Packets
    1000 Packets sent, 1500 received
NewYork#show interface Serial5/0 stats
Serial5/0
          Switching path    Pkts In    Chars In   Pkts Out   Chars Out
```

```
                Processor          0           0        500      734500
             Route cache        1000     1469000       1000      746500
                   Total        1000     1469000       1500     1481000
NewYork#
```

Observe in Example 13-21 that the 1465-byte packet with the DF bit set triggers PMTUD, and the 500 packets that the Oakland CE sends to the Albany CE are received as 1000 packets in the NewYork PE and then switched in the fast path. The highlighted Route cache line in the **show interface stats** command shows that these packets are fast switched. On the other hand, PMTUD is not triggered in the other direction (return path from the Albany CE to the Oakland CE); therefore, only 500 packets are sent from the NewYork PE to the SanFran PE on the way back, and the SanFran PE does the reassembly.

> **Note**
>
> Because of this reason and to add predictability to the PMTUD process and decouple the CE devices from driving the process, use PMTUD in conjunction with the DF bit set. Otherwise, packets are not prefragmented. They are post-fragmented unless the CE device sends a large packet with the DF bit set to trigger PMTUD as usual. Combining PMTUD with setting the DF bit allows the PE to obtain the PMTU more quickly and predictably.

The PE device can take a more active role in the PMTUD process and strengthen the whole concept. You can bring this about by setting the DF bit in all packets in the outer delivery IPv4 header. As a result, reassembly is prevented in the PE devices. The required configuration, shown in Example 13-22, is accomplished by using the **ip dfbit set** command in the pseudowire class. You create a new pseudowire class exactly like the previous one, with the addition of the **ip dfbit set** command, which you use in the Serial 5/0 **xconnect**.

Example 13-22. *PMTUD Combined with DF Bit Setting Configuration*

```
!
hostname SanFran
!
pseudowire-class wan-l2tpv3-pw-pmtu-df
 encapsulation l2tpv3
 sequencing both
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
 ip pmtu
 ip dfbit set
!
interface Serial5/0
 no ip address
 no cdp enable
 xconnect 10.0.0.203 50 pw-class wan-l2tpv3-pw-pmtu-df
!
```

> **Caution**
>
> Before you enable PMTUD, make sure that end-to-end PMTUD works. If it does not work, you could break applications just by setting the DF bit. PMTUD might not operate correctly if ICMP unreachables are blocked or end devices are noncompliant.

You can see the DF bit configuration in the **show l2tun session** command output, as shown in Example 13-23.

Example 13-23. *PMTUD Combined with DF Bit Setting Verification*

```
SanFran#show l2tun session all vcid 50
 Session Information Total tunnels 1 sessions 3
 Tunnel control packets dropped due to failed digest 0


Session id 58502 is up, tunnel id 56513
Call serial number is 905100000
Remote tunnel name is NewYork
  Internet address is 10.0.0.203
  Session is L2TP signalled
  Session state is established, time since change 00:09:13
    0 Packets sent, 0 received
    0 Bytes sent, 0 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 50
  Session Layer 2 circuit, type is HDLC, name is Serial5/0
  Circuit state is UP
    Remote session id is 38115, remote tunnel id 47670
  Session PMTU enabled, path MTU is not known
  DF bit on, ToS reflect disabled, ToS value 0, TTL value 255
  Session cookie information:
    local cookie, size 4 bytes, value 17 72 1D 8B
    remote cookie, size 4 bytes, value 3D 49 99 2F
  FS cached header information:
    encap size = 32 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000

  Sequencing is on
    Ns 0, Nr 0, 0 out of order packets received
SanFran#
```

With this configuration, the PE sets the DF bit in the IPv4 delivery header and participates in PMTUD regardless of the DF bit setting in the packets it receives. If the CE devices do not set the DF bit in IPv4 packets, the session PMTU is still discovered and acted upon.

Next, examine a complete example of PMTUD when the PE has **ip dfbit set** explicitly configured. Example 13-24 shows 500 1465-byte IP packets without the DF bit set sent from the Oakland CE to the Albany CE.

Example 13-24. *PMTUD Combined with DF Bit Setting Operation*

```
Oakland#ping 192.168.105.2 size 1465 repeat 500

Type escape sequence to abort.
Sending 500, 1465-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
..!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!Output omitted for brevity
!!!!!!!!!!
Success rate is 99 percent (498/500), round-trip min/avg/max = 4/19/120 ms
Oakland#
```

You can see that only 498 of the 500 pings are successful. The following steps outline the process:

1. The Oakland CE sends the first IP/ICMP request packet over HDLC. The SanFran PE receives it, encapsulates it with the L2TPv3 and IPv4 delivery headers, and sends it to the IP layer. The SanFran PE sets the DF bit in the delivery header because of the **ip dfbit set** command.

2. The IP packet is dropped in the SanFran PE in the IP layer because it is too big and has the DF bit set. This triggers an ICMP type 3 code 4 message that is used to discover the path MTU. The ICMP type 3 code 4 packet is generated source from and destined to the SanFran PE; the source IP address is from the outgoing interface of the originating device, and the destination address comes from the source IP address of the dropped L2TPv3 packet. In the general case, the ICMP type 3 code 4 packet would be sourced from a router in the IP cloud that is destined to the PE device. Note that as far as L2TPv3 is concerned, this packet was sent and will show up in L2TPv3 counters. This packet times out in the Oakland CE.

3. The Oakland CE sends the second IP packet over HDLC. The SanFran PE prefragments (before encapsulation) this packet and sends two L2TPv3 packets to the NewYork PE and onto the Albany CE.

4. The Albany CE reassembles the IP packet and replies with a 1465-byte ICMP Echo reply, which is encapsulated in L2TPv3 in the NewYork PE but later dropped at the IP layer in the NewYork PE. This triggers an ICMP type 3 code 4 message that is used to discover the path MTU on the NewYork PE for the return path. The ICMP type3 code 4 packet is generated source from and destined to the NewYork PE. The source IP address is from the outgoing interface of the originating device, and the destination address comes from the source IP address of the dropped L2TPv3 packet. This packet times out in the Oakland CE.

5. The Oakland CE sends a third IP packet over HDLC. The SanFran PE receives it, prefragments it, and sends it as two L2TPv3 packets to the NewYork PE and as two IPv4 over HDLC fragments to the Albany CE.

6. The Albany CE performs the reassembly and replies with an ICMP Echo reply message. The NewYork PE, which now knows the PMTU, receives this reply, prefragments the IPv4 CE packet, and sends two L2TPv3 packets to the SanFran PE. The SanFran PE sends the two IPv4 over HDLC frames to the Oakland CE.

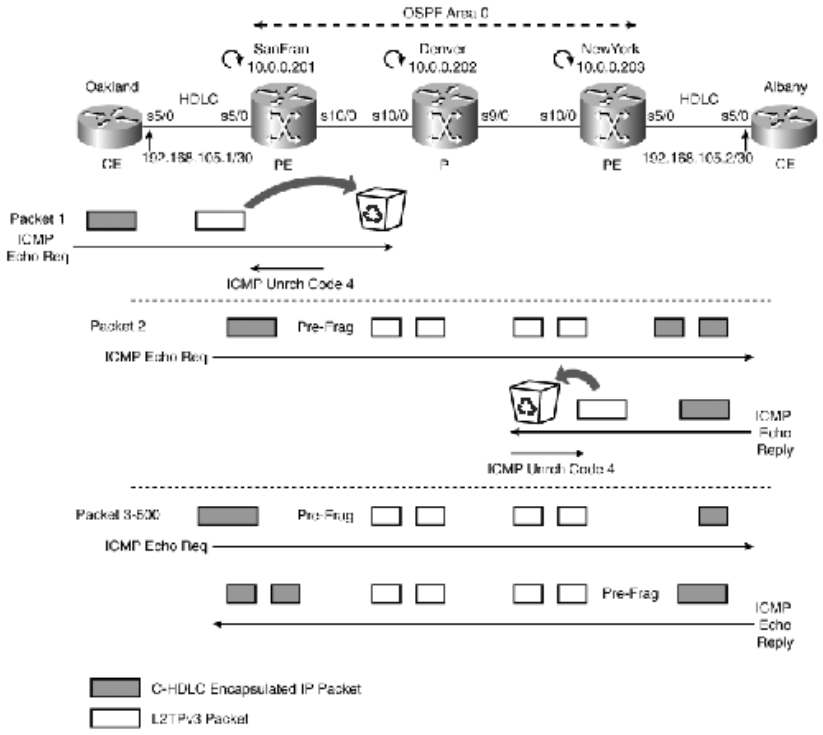
7. The Oakland CE receives the two fragments, reassembles the IP packet, and receives the ping reply. This third ping is successful.

8. The remaining 497 packets follow the same process as this third packet.

This complete process is depicted in Figure 13-4.

Figure 13-4. *Processing with L2TPv3 PMTUD and DF Bit Setting Enabled*

[View full size image]



You can also track various packet counters along the way, starting from the SanFran PE (see Example 13-25).

Example 13-25. *PMTUD and DF Bit Counters in the SanFran PE*

```
SanFran#show ip traffic | include IP stat|frag|reass
IP statistics:
  Frags: 0 reassembled, 0 timeouts, 0 couldn't reassemble
         499 fragmented, 1 couldn't fragment
SanFran#show l2tun session packet vcid 50
 Session Information Total tunnels 1 sessions 3
 Tunnel control packets dropped due to failed digest 0

LocID       RemID       TunID       Pkts-In     Pkts-Out    Bytes-In    Bytes-Out
12967       49396       62563       996         999         743514      746508
SanFran#
```

In the output of the **show ip traffic** command, you can see that the SanFran PE could not fragment one packet: packet number 1.

This packet also shows up in the L2TPv3 session counters. The SanFran PE fragmented the remaining 499 packets, creating 2 * 499 = 998 L2TPv3 packets. These 999 (1 + 998) packets are shown as packets sent out and into the tunnel in the L2TPv3 session packet counters. Example 13-26 shows the respective counters in the NewYork PE, including the 998 L2TPv3 packets that are counted as packets from the SanFran PE.

Example 13-26. *PMTUD and DF Bit Counters in the NewYork PE*

```
NewYork#show ip traffic | include IP stat|frag|reass
IP statistics:
  Frags: 0 reassembled, 0 timeouts, 0 couldn't reassemble
         498 fragmented, 1 couldn't fragment
NewYork#show l2tun session packet vcid 50
 Session Information Total tunnels 1 sessions 3
 Tunnel control packets dropped due to failed digest 0

LocID      RemID     TunID       Pkts-In     Pkts-Out    Bytes-In    Bytes-Out
49396      12967     35876       998         997         745007      745015
NewYork#
```

The output of the command **show ip traffic** from the NewYork PE shows that one packetthe reply to packet number 2could not be fragmented. The remaining 498 packets (from packet 3 through packet 500) were prefragmented, creating 996 L2TPv3 packets that you can see in Example 13-25 in the SanFran PE as packets in from the tunnel from the NewYork PE. These 997 (1 + 996) packets appear as packets out, meaning into the L2TPv3 tunnel in the output of the command **show l2tun session packet** in NewYork.

Example 13-27. *PMTUD and DF Bit Counters in the Albany CE*

```
Albany#show ip traffic | i IP stat|frag|reass|ICMP|echo
IP statistics:
  Frags: 499 reassembled, 0 timeouts, 0 couldn't reassemble
         0 fragmented, 0 couldn't fragment
ICMP statistics:
        499 echo, 0 echo reply, 0 mask requests, 0 mask replies, 0 quench
  Sent: 0 redirects, 0 unreachable, 0 echo, 499 echo reply
Albany#
```

The new configuration effectively pushes the reassembly into the CE devices. Albany reassembled 499 packets (all except packet number 1) from ICMP Echo messages and replied to them.

Example 13-28 shows the IP traffic counters for the Oakland CE.

Example 13-28. *PMTUD and DF Bit Counters in the Oakland CE*

```
PMTUD and DF Bit Counters in the Oakland CE
Oakland#show ip traffic | i IP stat|frag|reass|ICMP|echo
IP statistics:
  Frags: 498 reassembled, 0 timeouts, 0 couldn't reassemble
         0 fragmented, 0 couldn't fragment
ICMP statistics:
        0 echo, 498 echo reply, 0 mask requests, 0 mask replies, 0 quench
  Sent: 0 redirects, 0 unreachable, 500 echo, 0 echo reply
Oakland#
```

You can see that the Oakland CE reassembled 498 packets from the 498 respective Echo replies (all except packets 1 and 2, which were dropped in the core to discover the PMTU).

Advanced ATM Transport over L2TPv3

The previous section covered PMTUD that applies to the transport of all Layer 2 protocols over L2TPv3. This section covers the transport and tunneling of ATM over L2TPv3. First you learn about ATM OAM local emulation mode for AAL5 CPCS-SDU Transport. Then you learn about ATM cell packing for cell relay (CR) over L2TPv3.

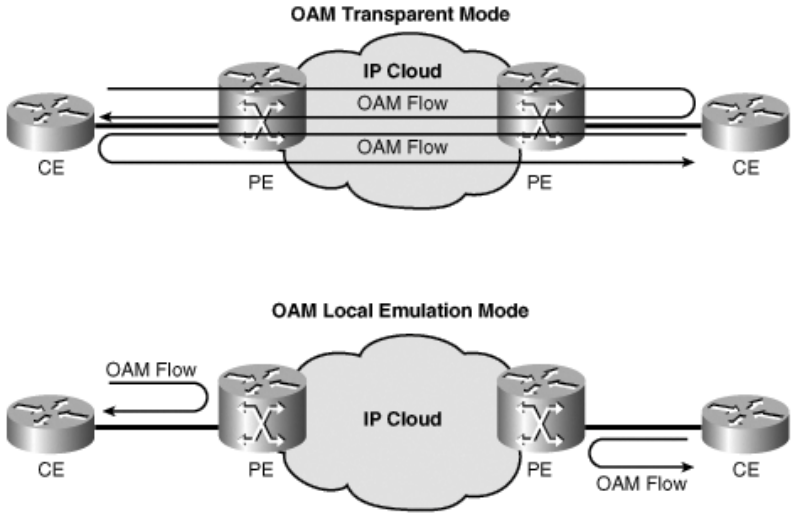
Case Study 13-2: ATM OAM Emulation

Two operational modes exist for managing OAM cells in an AAL5 L2TPv3 pseudowire:

- **OAM transparent mode** In this mode, PE devices transport OAM cells transparently across the pseudowire. They do this via the use of the Transport bit (T-bit) in the mandatory ATM-Specific Sublayer to indicate that the L2TPv3 packet contains an ATM admin cell.
- **OAM local emulation mode** In this mode, PE devices do not transport OAM cells over the pseudowire. Instead, they locally terminate and process VC OAM cells (F5 OAM cells).

You can see a graphic representation of these two operational modes for OAM flows in Figure 13-5.

Figure 13-5. ***Operational Modes for OAM Flows in L2TPv3***



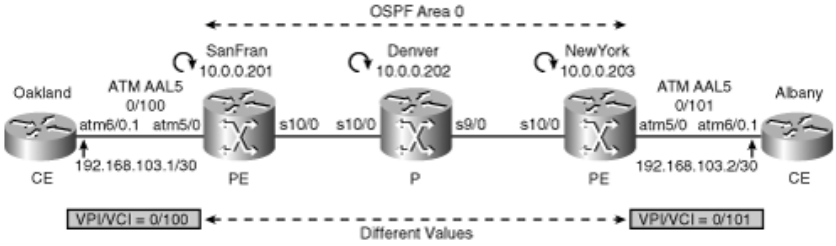
You might wonder when you would use ATM OAM local emulation. You can use OAM emulation to locally terminate or loop the OAM cells in two realistic scenarios:

- When a PE device does not support the transport of OAM cells across the AAL5 L2TPv3 session.
- When you are using different virtual path identifier (VPI) or virtual circuit identifier (VCI) values at both ends of an AAL5 L2TPv3 pseudowire. Rewriting the VPI/VCI values for admin cells that are transported over AAL5 SDU L2TPv3 sessions is not supported.

In this case study, you will learn OAM emulation in L2TPv3 using the topology shown in Figure 13-6. Note that the VPI/VCI pair is different in both endpoints of the pseudowire.

Figure 13-6. *L2TPv3 OAM Emulation Topology*

[View full size image]



In AAL5 SDU L2TPv3 sessions, the VPI/VCI pair that makes up the attachment circuits for the PVCs can be different at both endpoints, as in Figure 13-6. However, such a scenario has a limitation: The VPI/VCI cannot be rewritten for cells that are transported over the AAL5 pseudowire. This limitation exists only for ATM cells that are transported over the AAL5 SDU pseudowire; it does not pose a problem for AAL5 SDUs that are transported over it. For successful transport of raw ATM cells (such as F5 OAM cells) over an AAL5 SDU pseudowire, the VPI/VCI pair needs to match at both ends. The only way of supporting OAM management of CE PVCs with different VPI/VCIs is by enabling OAM local emulation. With OAM local emulation, OAM cells are looped back or terminated and acted upon in the PE's attachment circuit. They are not transported over the pseudowire.

> **Note**
>
> When an L2TPv3 PE that is configured for OAM emulation receives an OAM cell indicating an alarm condition such as OAM AIS, a Set-Link-Info (SLI) message is triggered to notify the remote PE of the defect instead of tearing down the L2TPv3 session. This in turn triggers the generation of OAM alarm signals in the remote end of the L2TPv3 session and toward the remote CE. This achieves end-to-end alarm indication, maintaining the session as UP but alarmed.

Example 13-29 shows the L2TPv3 sessions used in both the SanFran and NewYork PE devices.

Example 13-29. *OAM Emulation Sessions*

```
SanFran#show l2tun session circuit vcid 27 | begin Loc
LocID      TunID      Peer-address    Type Stat Username, Intf/
                                              Vcid, Circuit
10314      22650      10.0.0.203      ATM UP    27, AT5/0:0/100
SanFran#


NewYork#show l2tun session circuit vcid 27 | begin Loc
LocID    TunID       Peer-address    Type Stat Username, Intf/
                                              Vcid, Circuit
38764    45445       10.0.0.201      ATM  UP   27, AT5/0:0/101
NewYork#
```

Now enable OAM PVC management in the Oakland and Albany PVCs with the command **oam-pvc manage**. Example 13-30 shows the configuration for the Oakland endpoint.

Example 13-30. *OAM PVC Management Configuration in the CEs*

```
!
hostname Oakland
!
interface ATM6/0.1 point-to-point
 ip address 192.168.103.1 255.255.255.252
 pvc 0/100
  oam-pvc manage
  encapsulation aal5snap
 !
```

The CE PVCs go into a DOWN state when OAM cells that are looped back are not received (see Example 13-31).

Example 13-31. *CE PVCs Go DOWN Without OAM-AC Emulation*

```
Oakland#show atm pvc interface ATM 6/0.1
                VCD /                                   Peak Avg/Min Burst
Interface       Name          VPI    VCI Type   Encaps   Kbps    Kbps Cells Sts
6/0.1           1               0    100 PVC    SNAP   149760     N/A       DOWN
```

You can use the command **show atm vc** to display the OAM state and counters (see Example 13-32).

Example 13-32. *CE PVCs DOWN and OAM Counters*

```
Oakland#show atm vc interface ATM 6/0.1 detail
ATM6/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5-LLC/SNAP, etype:0x0, Flags: 0xC20, VCmode: 0x0
OAM frequency: 10 second(s)
InARP frequency: 15 minutes(s)
Transmit priority 4
InPkts: 0, OutPkts: 0, InBytes: 0, OutBytes: 0
InPRoc: 0, OutPRoc: 0
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 0
OAM cells sent: 16
Status: DOWN
```

The command **show atm pvc** presents enhanced statistics and counters, including the fact that the ATM PVC state is managed by OAM events (see Example 13-33).

Example 13-33. *CE PVCs DOWN and Enhanced OAM Counters*

```
Oakland#show atm pvc 0/100
ATM6/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5-LLC/SNAP, etype:0x0, Flags: 0xC20, VCmode: 0x0
OAM frequency: 10 second(s), OAM retry frequency: 1 second(s)
OAM up retry count: 3, OAM down retry count: 5
OAM Loopback status: OAM Sent
OAM VC state: Not Verified
ILMI VC state: Not Managed
VC is managed by OAM.
InARP frequency: 15 minutes(s)
Transmit priority 4
InPkts: 0, OutPkts: 0, InBytes: 0, OutBytes: 0
InPRoc: 0, OutPRoc: 0
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 0
F5 InEndloop: 0, F5 InSegloop: 0, F5 InAIS: 0, F5 InRDI: 0
OAM cells sent: 16
F5 OutEndloop: 16, F5 OutSegloop: 0, F5 OutAIS: 0, F5 OutRDI: 0
OAM cell drops: 0
Status: DOWN, State: NOT_VERIFIED
Oakland#
```

By comparing Example 13-32 and Example 13-33, you can see the difference in output between the old **show atm vc** command and the new **show atm pvc** command. The latter contains much more detailed information than the former.

For OAM emulation to work effectively, you must enable it in both ends simultaneously. The L2TPv3 extensions for ATM pseudowires define the new OAM Emulation Required Attribute-Value Pair (AVP) to be used in AAL5 CPCS-SDU mode to signal OAM Emulation. OAM Emulation AVP is a boolean AVP that has no attribute value. Its mere presence or absence indicates a TRUE or FALSE value, respectively.

If OAM cell emulation is configured or detected on one side, the other LCCE also must support it, which is the purpose of the OAM Emulation Required AVP signaling method. If the other LCCE cannot support the OAM cell emulation, you must tear down the associated L2TP session via a Call-Disconnect-Notify (CDN) message.

Example 13-34 shows the command to enable OAM emulation in the SanFran PE.

Example 13-34. *Enabling OAM Emulation in the SanFran PE*

```
interface ATM5/0
 pvc 0/100 l2transport
  oam-ac emulation-enable 2
  encapsulation aal5
  xconnect 10.0.0.203 27 pw-class pw-l2tpv3-atm
 !
```

The rate in seconds at which OAM alarm indication signal (AIS) cells are sent follows the command **oam-ac emulation-enable**.

Note

After you enable OAM emulation with the **oam-ac emulation-enable** command, you can make the usual OAM Management commands such as **oam-pvc manage** available in the Layer 2 transport PVC. A Layer 2 transport PVC (attachment circuit) that has been configured for OAM emulation can periodically send OAM loopback cells toward the CE router and manage the Layer 2 transport PVC status based on the reply to those OAM loopback cells.

At this point, you have enabled OAM emulation only at one endin the SanFran PE. As you have learned already in this section, this tears down the session with a CDN message because the OAM emulation value does not match at both ends. Examples 13-33 through 13-35 show the L2X debugs in the SanFran and NewYork PEs. The debugs that are enabled are **debug vpdn l2x-errors**, **debug vpdn l2x-events**, and **debug vpdn l2x-packets**. L2X means that the command is applicable to both Layer 2 Forwarding (L2F) and Layer 2 Tunnel Protocol (L2TP) protocols.

Example 13-35 shows an Incoming-Call-Request (ICRQ) message that the SanFran PE receives. It has a pseudowire Type 2 for AAL5 SDU and a VC ID (end identifier) of 27.

Example 13-35. *Mismatch OAM-AC Emulation ConfigurationICRQ*

```
SanFran#debug vpdn l2x-errors
L2X protocol errors debugging is on
SanFran#debug vpdn l2x-events
L2X protocol events debugging is on
SanFran#debug vpdn l2x-packets
L2X control packets debugging is on
SanFran#
SanFran#show debugging
VPN:
  L2X protocol events debugging is on
  L2X control packets debugging is on
  L2X protocol errors debugging is on
SanFran#
*Jul  6 18:27:26.792: Tnl56204 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Jul  6 18:27:26.792: Tnl56204 L2TP: Parse ICRQ
!Output omitted for brevity
*Jul  6 18:27:26.792: Tnl56204 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jul  6 18:27:26.792: Tnl56204 L2TP: Pseudo Wire Type 2
*Jul  6 18:27:26.792: Tnl56204 L2TP: Parse Cisco AVP 6, len 8, flag 0x0
*Jul  6 18:27:26.792: Tnl56204 L2TP: End Identifier 27
!Output omitted for brevity
*Jul  6 18:27:26.792: Tnl56204 L2TP: Parse AVP 47, len 10, flag 0x0
*Jul  6 18:27:26.792: Tnl56204 L2TP: L2 Specific Sublayer 2
```

The ICRQ message also includes the Layer 2-Specific Sublayer AVP specifying the ATMSpecific Sublayer with a value of 2. No OAM Emulation Required AVP is available because it has not been configured in the NewYork PE. Example 13-36 shows the Incoming-Call-Reply (ICRP) message in reply as received by the NewYork PE.

Example 13-36. *Mismatch OAM-AC Emulation ConfigurationICRP*

```
NewYork#debug vpdn l2x-errors
L2X protocol errors debugging is on
NewYork#debug vpdn l2x-events
L2X protocol events debugging is on
NewYork#debug vpdn l2x-packets
L2X control packets debugging is on
NewYork#
NewYork#
*Jul 12 06:19:12.309: Tnl65533 L2TP: Parse ICRP
*Jul 12 06:19:12.309: Tnl65533 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Jul 12 06:19:12.313: Tnl65533 L2TP: Local Session ID 12575
*Jul 12 06:19:12.313: Tnl65533 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
*Jul 12 06:19:12.313: Tnl65533 L2TP: Remote Session ID 38123
*Jul 12 06:19:12.313: Tnl65533 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jul 12 06:19:12.313: Tnl65533 L2TP: Pseudo Wire Type 2
*Jul 12 06:19:12.313: Tnl65533 L2TP: Parse Cisco AVP 108, len 6, flag 0x0
*Jul 12 06:19:12.313: Tnl65533 L2TP: OAM Emulation Required
*Jul 12 06:19:12.313: Tnl65533 L2TP: Parse AVP 47, len 10, flag 0x0
*Jul 12 06:19:12.313: Tnl65533 L2TP: L2 Specific Sublayer 2
*Jul 12 06:19:12.313: Tnl65533 L2TP: No missing AVPs in ICRP
*Jul 12 06:19:12.313: Tnl/Sn65533/38123 L2TP: OAM Emulation Required AVP in ICRP
contradicts with local config. Tearing down the session.
```

From Example 13-36, you can see that the ICRP as received from NewYork contains a pseudowire Type 2 for AAL5 and the Layer 2-Specific Sublayer with a value of 2 for ATM. In addition, the ICRP contains the OAM Emulation Required AVP because it has been configured in the SanFran PE. The OAM Emulation Required AVP uses Cisco AVP 108 waiting for IANA assignment.

Example 13-36 shows the mismatch of OAM emulation configuration detected in the NewYork PE. The NetYork PE tears down the session by sending a CDN message (see Example 13-37).

Example 13-37. *Mismatch OAM-AC Emulation ConfigurationCDN*

```
SanFran#
*Jul  6 18:27:26.812: Tnl56204 L2TP: Parse CDN
*Jul  6 18:27:26.812: Tnl56204 L2TP: Parse AVP 1, len 10, flag 0x8000 (M)
*Jul  6 18:27:26.816: L2X: Result code(4): 4: Call failed, not enough resources
(temporary)
*Jul  6 18:27:26.816:      Error code(0): No error
*Jul  6 18:27:26.816: Tnl56204 L2TP: Parse Cisco AVP 3, len 10, flag 0x8000 (M)
*Jul  6 18:27:26.816: Tnl56204 L2TP: Local Session ID 38123
*Jul  6 18:27:26.816: Tnl56204 L2TP: Parse Cisco AVP 4, len 10, flag 0x8000 (M)
*Jul  6 18:27:26.816: Tnl56204 L2TP: Remote Session ID 12575
*Jul  6 18:27:26.816: Tnl56204 L2TP: No missing AVPs in CDN
*Jul  6 18:27:26.816: Tnl/Sn56204/12575 L2TP: I CDN from NewYork tnl 65533, cl 38123
*Jul  6 18:27:26.816: Tnl/Sn56204/12575 L2TP: Destroying session
*Jul  6 18:27:26.816: Tnl/Sn56204/12575 L2TP: Session state change from wait-connect
to idle
```

From Example 13-37, you can see that the SanFran PE receives the CDN that destroys the session. The CDN message contains the Result Code AVP (IETF AVP 1) as defined in RFC 2661, "Layer Two Tunneling Protocol 'L2TP.'" The result code of 4 indicates that the call failed because appropriate facilities were not available (temporary condition). Error code 0 specifies no general error.

Because the L2TPv3 pseudowire is down, ATM OAM AIS cells are sent out of the attachment circuits toward the CEs. The CEs in turn reply with remote defect indication (RDI) OAM cells that the PE receives (see Example 13-38).

Example 13-38. ***OAM AIS and RDI Cells***

```
SanFran#show atm pvc 0/100
ATM5/0: VCD: 3, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5 L2transport, etype:0xF, Flags: 0x30000C2E, VCmode: 0x0
OAM Cell Emulation: enabled, F5 End2end AIS Xmit frequency: 2 second(s)
Interworking Method: like to like
Remote Circuit Status = Undefined, Alarm Type = Undefined
OAM frequency: 0 second(s), OAM retry frequency: 1 second(s)
OAM up retry count: 3, OAM down retry count: 5
OAM Loopback status: OAM Disabled
OAM VC state: Not Managed - AIS Xmitted
ILMI VC state: Not Managed
InPkts: 21, OutPkts: 0, InBytes: 1680, OutBytes: 0
InPRoc: 0, OutPRoc: 0
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 73
F5 InEndloop: 21, F5 InSegloop: 0, F5 InAIS: 0, F5 InRDI: 52
OAM cells sent: 52
F5 OutEndloop: 0, F5 OutSegloop: 0, F5 OutAIS: 52, F5 OutRDI: 0
OAM cell drops: 0
Status: UP
SanFran#


Oakland#show atm pvc 0/100
ATM6/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5-LLC/SNAP, etype:0x0, Flags: 0xC20, VCmode: 0x0
OAM frequency: 10 second(s), OAM retry frequency: 1 second(s)
OAM up retry count: 3, OAM down retry count: 5
OAM Loopback status: OAM Sent
OAM VC state: AIS/RDI
ILMI VC state: Not Managed
VC is managed by OAM.
InARP frequency: 15 minutes(s)
Transmit priority 4
InPkts: 0, OutPkts: 0, InBytes: 0, OutBytes: 0
InPRoc: 0, OutPRoc: 0
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 52
F5 InEndloop: 0, F5 InSegloop: 0, F5 InAIS: 52, F5 InRDI: 0
OAM cells sent: 73
F5 OutEndloop: 21, F5 OutSegloop: 0, F5 OutAIS: 0, F5 OutRDI: 52
OAM cell drops: 0
Status: DOWN, State: NOT_VERIFIED
Oakland#
```

Example 13-38 shows that OAM emulation is enabled for the AAL5 Layer 2 transport pseudowire in SanFran. The SanFran attachment circuit has sent 52 AIS cells (F5 OutAIS) toward the Oakland CE because the L2TPv3 session is down, and it has received 52 RDI cells (F5 InRDI) from the Oakland CE. OAM AIS cells are equivalent to a blue alarm, and OAM RDI cells are equivalent to a yellow alarm. Consistently, the Oakland CE's PVC shows the receipt of 52 OAM AIS cells (F5 InAIS) from the

SanFran PE, which resulted in 52 RDI cells (F5 OutRDI) generated toward the SanFran PE. The Oakland OAM PVC status is now DOWN because of AIS/RDI.

To conclude the OAM emulation configuration, enable OAM emulation in the NewYork PE and verify that the L2TPv3 session comes up. The Oakland PVC receives end-to-end loopback cells and comes up (see Example 13-39).

Example 13-39. *Oakland CE PVC UP*

```
Oakland#show atm pvc interface ATM 6/0.1
                VCD /                                   Peak Avg/Min Burst
Interface       Name        VPI   VCI Type   Encaps    Kbps    Kbps Cells Sts
6/0.1           1             0   100 PVC    SNAP    149760     N/A       UP
Oakland#
Oakland#show atm pvc 0/100
ATM6/0.1: VCD: 1, VPI: 0, VCI: 100
UBR, PeakRate: 149760
AAL5-LLC/SNAP, etype:0x0, Flags: 0xC20, VCmode: 0x0
OAM frequency: 10 second(s), OAM retry frequency: 1 second(s)
OAM up retry count: 3, OAM down retry count: 5
OAM Loopback status: OAM Received
OAM VC state: Verified
ILMI VC state: Not Managed
VC is managed by OAM.
InARP frequency: 15 minutes(s)
Transmit priority 4
InPkts: 0, OutPkts: 0, InBytes: 0, OutBytes: 0
InPRoc: 0, OutPRoc: 0
InFast: 0, OutFast: 0, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 0
OAM cells received: 162
F5 InEndloop: 9, F5 InSegloop: 0, F5 InAIS: 153, F5 InRDI: 0
OAM cells sent: 183
F5 OutEndloop: 30, F5 OutSegloop: 0, F5 OutAIS: 0, F5 OutRDI: 153
OAM cell drops: 0
Status: UP
Oakland#
```

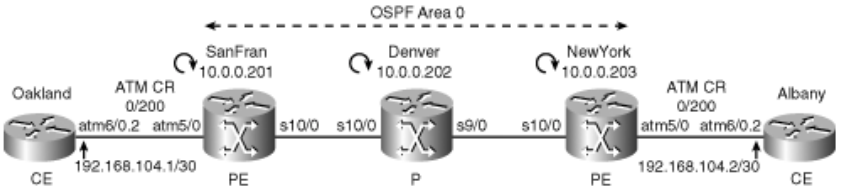
Case Study 13-3: ATM Cell Packing

The second ATM advanced topic pertains to the cell transport over L2TPv3 (CRoL2TPv3). The cell packing feature, also called ATM cell concatenation, consists of packing or concatenating multiple ATM cells into a single L2TPv3 packet from a minimum of one cell (that is, no cell packing performed) to the maximum allowed by the MTU. Refer to Chapter 12 if you need a refresher about cell relay over L2TPv3.

In contrast to OAM emulation, you can configure the cell packing feature independently in each endpoint using different values. You will learn to configure and verify cell packing using the topology shown in Figure 13-7.

Figure 13-7. *L2TPv3 ATM Cell Packing Topology*

[View full size image]



To configure cell packing, you must configure the maximum cell packing timeout (MCPT) timers in the main ATM interface. You have three different timers to pick from when configuring cell packing in the ATM port or all PVCs or PVPs within the physical interface. To configure these three timers, shut down the main interface (see Example 13-40).

Example 13-40. *Configuring MCPT Timers*

```
SanFran#conf t
Enter configuration commands, one per line. End with CNTL/Z.
SanFran(config)#interface ATM 5/0
SanFran(config-if)#shutdown
SanFran(config-if)#atm mcpt-timers 100 1000 4095
SanFran(config-if)#no shutdown
```

With the MCPT timers preconfigured in Example 13-40, you can enable cell packing in both the SanFran and NewYork PEs. The one mandatory argument is the maximum number of cells that can be packed (see Example 13-41).

Example 13-41. *Configuring Cell Packing*

```
!
hostname SanFran
!
interface ATM5/0
 atm mcpt-timers 100 1000 4095
 pvc 0/200 l2transport
  encapsulation aal0
  cell-packing 14 mcpt-timer 3
  xconnect 10.0.0.203 28 pw-class pw-l2tpv3-atm
 !
```

In Example 13-41, you have enabled cell packing for the pseudowire with attachment circuit, with VPI/VCI 0/200 in the SanFran PE specifying a maximum number of cells to be packed equal to 14 cells, and using the third preconfigured timer. Although you can configure a different value in the remote PE, you can also configure a maximum number of cells packed (MNCP) of 14 cells in the NewYork PE.

The value of MNCP is signaled in the ATM Maximum Concatenated Cells AVP. MNCP indicates how many concatenated cells (maximum value) the LCCE node can process as a disposition capability. The values that are advertised in both directions do not need to match. Furthermore, the absence of this AVP indicates no cell packing. This AVP and cell packing in general apply only to ATM Cell Relay pseudowire types (see Example 13-42).

Example 13-42. *ATM Maximum Concatenated Cells AVP*

```
SanFran#
*Jul  1 10:45:09.119: Tnl22650 L2TP: Parse AVP 0, len 8, flag 0x8000 (M)
*Jul  1 10:45:09.119: Tnl22650 L2TP: Parse ICRQ
!Output omitted for brevity
*Jul  1 10:45:09.123: Tnl22650 L2TP: Parse Cisco AVP 7, len 8, flag 0x8000 (M)
*Jul  1 10:45:09.123: Tnl22650 L2TP: Pseudo Wire Type 9
!Output omitted for brevity
*Jul  1 10:45:09.123: Tnl22650 L2TP: Parse Cisco AVP 11, len 8, flag 0x0
*Jul  1 10:45:09.123: Tnl22650 L2TP: ATM Maximum Number of cells that can be packed 14
*Jul  1 10:45:09.123: Tnl22650 L2TP: No missing AVPs in ICRQ
```

Example 13-42 shows the ATM Maximum Concatenated Cells AVP included in the ICRQ and using Cisco AVP 11. You can verify the configuration of ATM cell packing from the SanFran PE. See Example 13-43, which shows the local and remote MNCP values of 14 cells.

Example 13-43. *ATM Cell-Packing Verification*

```
SanFran#show atm cell-packing
                                average               average
       circuit             local nbr of cells    peer nbr of cells    MCPT
       type                MNCP  rcvd in one pkt MNCP sent in one pkt (us)
ATM5/0           vc 0/200   14        0           14       0          4095
SanFran#
```

As you know, each ATM cell carries 48 bytes of payload. When you concatenate 14 cells, you can carry an SDU of 48 bytes/cell * 14 cells = 672 bytes. You configured the Oakland and Albany CE's PVCs with AAL5-LLC/SNAP encapsulation, which means that an IP packet would have 16 bytes of encapsulation overhead as follows: 8 bytes of CPCS-PDU trailer plus 8 bytes of LLC/SNAP header. Therefore, the largest IP packet that you can fit into a single L2TPv3 Cell Relay packet that is concatenating 14 cells is 672 bytes 16 bytes = 656 bytes.

To verify these calculations, send 100 656-bytes packets from the Oakland CE to the Albany CE. Then display the average number of cells sent and received per packet (see Example 13-44).

Example 13-44. *Optimal Cell-Packing Utilization*

```
Oakland#ping
Protocol [ip]:
Target IP address: 192.168.104.2
Repeat count [5]: 100
Datagram size [100]:  656
Timeout in seconds [2]:
 Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 100, 656-byte ICMP Echos to 192.168.104.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 96/98/108 ms
Oakland#
```

```
SanFran#show atm cell-packing
                                average                 average
       circuit             local nbr of cells      peer nbr of cells      MCPT
       type                MNCP  rcvd in one pkt   MNCP sent in one pkt  (us)
```

```
ATM5/0              vc 0/200 14       14             14        14           4095
SanFran#
```

You can see from Example 13-44 that the average number of cells sent and received in one packet equals 14 cells, which is the optimal usage for the pseudowire overhead.

Consider what would happen if you used a slightly larger IP packet. See Example 13-45, which uses 657-byte packets.

Example 13-45. *Suboptimal Cell-Packing Utilization*

```
Oakland#ping
Protocol [ip]:
Target IP address: 192.168.104.2
Repeat count [5]: 100
Datagram size [100]: 657
Timeout in seconds [2]:
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 100, 657-byte ICMP Echos to 192.168.104.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 108/108/112 ms
Oakland#
```

```
SanFran#show atm cell-packing
                                 average                 average
       circuit            local  nbr of cells       peer nbr of cells     MCPT
       type               MNCP   rcvd in one pkt    MNCP sent in one pkt  (us)
ATM5/0             vc 0/200   14        7             14        7           4095
SanFran#
```

You can see in Example 13-45 that the average number of cells sent and received per L2TPv3 packet drastically dropped to 7 bytes. Each IP packet now needs 15 ATM cells; thus, it uses a first L2TPv3 packet with 14 cells and a second L2TPv3 packet with just 1 cell. The average equates to 7 cells per l2TPv3 packet.

Quality of Service

In this section, you learn concepts and configuration of QoS in L2TPv3 networks. Although there are some L2TPv3 specific edge QoS services, L2TPv3 runs over an IP PSN; therefore, all IP QoS models apply. Specifically, you explore the IP differentiated services (DiffServ) model, whose architecture is defined in RFC 2475, "An Architecture for Differentiated Services."

DiffServ separates edge behaviorssuch as classification, marking, policing, metering, and complex per-user and per-application tasksfrom core functions or per-hop behaviors (PHB)including queuing, shaping, dropping, and simple tasks. DiffServ partitions IP traffic into a small number of classes (eight classes per RFC are recommended) and allocates resources on a per-class basis. At the edge, the classification information is summarized in the DiffServ code point (DSCP), which gives a new interpretation to the type of service (ToS) IPv4 header octet and IPv6 traffic class octet as defined in RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers."

The configuration of IP QoS in Cisco IOS follows the Modular QoS CLI (MQC) model. You can break down the MQC configuration into three distinctive steps:

Step 1. **Classification**Traffic is classified with a class-map.

Step 2. **Policy creation**Policies are applied to the traffic classes that were defined previously in a policy-map.

Step 3. **Policy application**The policies that were defined previously are applied in a direction to a specific interface, subinterface, or ATM and Frame Relay VCs using a service policy.

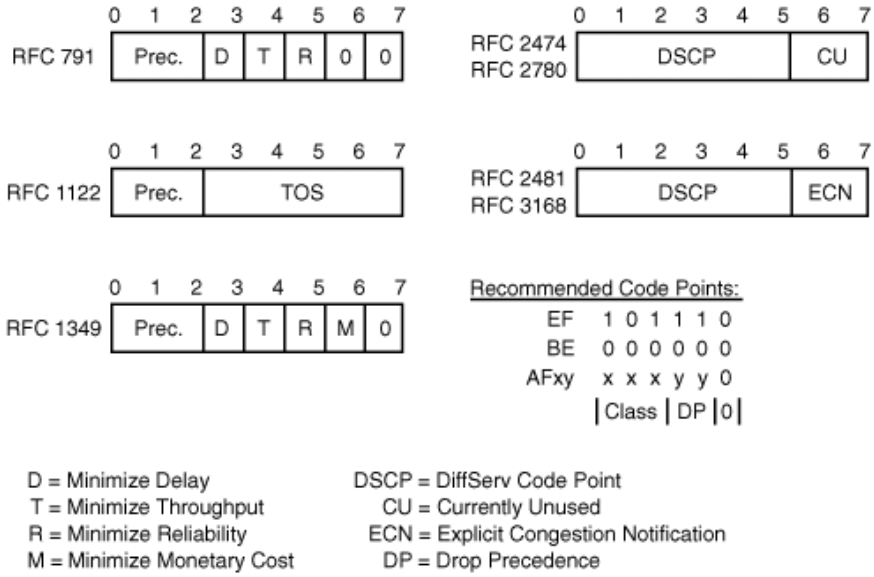
The following case studies explain examples of different QoS modules, including traffic marking, traffic policing, queuing and shaping, and Layer 2-specific matching and setting.

Case Study 13-4: Traffic Marking

One of the constitutive pieces of the DiffServ model is traffic marking. Before exploring specific traffic marking methods, this section presents a historical perspective on the IPv4 header TOS octet that was defined originally in RFC 791. The ToS byte is the second byte in the IPv4 header and can be interpreted in multiple ways. Figure 13-8 shows the evolution of the ToS octet in different RFCs. You can see that the three most significant bits are called the precedence bits, and they correspond to the class selector in the DSCP.

Figure 13-8. ***Type of Service IPv4 Header Octet Evolution***

[View full size image]



Note

Traffic marking alone does nothing for QoS. PHBs need to exist in the core network to apply different policies to the traffic that is marked in different classes.

In the next sections, you learn different ways to perform traffic marking. The first two methods are L2TPv3 specific, whereas the third one is a generic marking.

ToS Setting

The first method for marking traffic consists of setting the ToS value under a pseudowire class for all L2TPv3 IP packets. You carry out this configuration with the command **ip tos value** {*value*} as shown in Example 13-46.

Example 13-46. *ToS Setting Configuration*

```
!
hostname SanFran
!
pseudowire-class wan-l2tpv3-pw-pmtu-df
 encapsulation l2tpv3
 sequencing both
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
 ip pmtu
 ip dfbit set
 ip tos value 96
!
```

Example 13-46 shows the ToS set to a value of 96 (0x60 in hexadecimal). This is the value for the complete ToS byte. The number 96 that is represented in binary equals 01100000, from which you can infer that the IP precedence is 011b or 3 (flash). To verify this configuration, configure inbound (that is, packets received) IP accounting by IP precedence in the NewYork PE in the interface that connects to the Denver P (see Example 13-47).

Example 13-47. *IP Accounting Configuration in the NewYork PE*

```
!
hostname NewYork
!
interface Serial10/0
 ip unnumbered Loopback0
 ip accounting precedence input
!
```

Now send 1000 ping packets from the Oakland to the Albany CEs, which use the default precedence of 0. Check the IP accounting information that has been collected (see Example 13-48).

Example 13-48. *ToS Setting Verification in the NewYork PE*

```
NewYork#show interfaces precedence
Serial10/0
  Input
    Precedence 3: 1000 packets, 140000 bytes
    Precedence 6: 2 packets, 88 bytes
NewYork#
```

You can see that, as expected, 1000 packets were received with an IP precedence of 3. You also see a couple of IP precedence 6 packets that correspond to routing updates.

> **Note**
>
> Although IP accounting can be useful and is a quick way to gather per-precedence accounting information, it suffers from performance limitations. For example, IP accounting is not supported in distributed CEF (dCEF). It forces packets to be process switched in distributed platforms, such as the Cisco 7500 and 12000 series routers. Whenever possible, use NetFlow to collect statistics in a more scalable way. IP accounting is a great proof of concept tool, but you should not use it in distributed platforms.

ToS Reflection

Another traffic-marking feature that is specific to L2TPv3 is called *ToS reflection*. In ToS reflection marking, the ToS octet is copied over or reflected from the inner CE IP packet header into the outer IP tunnel packet header. This behavior cannot be mimicked with MQC.

To configure TOS reflection, you can use the pseudowire-class command **ip tos reflect**. The command **ip tos** {**reflect** | **value**} appears only in pseudowire-class configuration when the encapsulation is set to L2TPv3 (see Example 13-49).

Example 13-49. *ToS Reflection Configuration*

```
!
hostname SanFran
!
pseudowire-class wan-l2tpv3-pw-pmtu-df
 encapsulation l2tpv3
 sequencing both
 protocol l2tpv3 l2tpv3-wan
 ip local interface Loopback0
 ip pmtu
 ip dfbit set
 ip tos reflect
!
```

To verify the ToS reflection functioning, generate 1000 packets with IP precedence 5 (critical) from the Oakland CE to the Albany CE. Use an extended **ping** command, in which you must specify the ToS octet value. You can express an IP precedence of 5 in binary as 101. Therefore, the binary representation of the ToS is 10100000, which is 0xA0 or 160 (see Example 13-50).

Example 13-50. *Generating Precedence 5 Traffic Between CEs*

```
Oakland#ping ip
Target IP address: 192.168.105.2
Repeat count [5]: 1000
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface:
Type of service [0]: 0xa0
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 1000, 100-byte ICMP Echos to 192.168.105.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output omitted for brevity
!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (1000/1000), round-trip min/avg/max = 16/21/52 ms
Oakland#
```

Using the same IP accounting method, verify the precedence in the packets sent over the IP PSN and received by the NewYork PE (see Example 13-51).

Example 13-51. *ToS Reflection Verification in the NewYork PE*

```
NewYork#show interfaces precedence
Serial10/0
  Input
    Precedence 3:  1000 packets, 140000 bytes
    Precedence 5:  1000 packets, 140000 bytes
    Precedence 6:  40 packets, 2352 bytes
NewYork#
```

You can see 1000 new packets received with precedence 5. The precedence 5 was copied over from the TS byte in the CE IP packet.

You can also configure both **ip tos** actions of **value** and **reflect** into a single pseudowire class. In that case, the ToS value in the outer IP header defaults to the fixed set value but is overwritten with the reflected value when the Layer 2 tunneled frame contains an IP packet.

MQC IP Precedence or DSCP Setting

The third traffic-marking mechanism uses MQC. The MQC **set ip precedence** and **set ip dscp** policy commands have been extended to include the **tunnel** keyword to indicate that the policy applies to the outer L2TPv3 tunnel IPv4 delivery header.

When you are using MQC to perform marking with L2TPv3, only the inbound direction (that is, coming from the CE device) is meaningful for classification; therefore, the classification criterion needs to be Layer 2 fields at the attachment circuit. A PE device normally marks classified traffic with a tunnel as IP DSCP. The primary goal for tunnel marking is to control QoS for a particular tunneled customer within the provider core network. Customer-specific PHB should be pushed out to the CE devices. For simplicity, Example 13-52 shows the configuration required to perform tunnel marking with a precedence of 2 (immediate) using MQC to classify all traffic incoming into the attachment circuit. Apply the service policy in a Layer 2 transport ATM PVC.

Example 13-52. *Tunnel Marking with MQC Configuration*

```
!
hostname SanFran
!
class-map match-all all_traffic
  match any
!
policy-map prec-2
  class all_traffic
   set ip precedence tunnel 2
!
interface ATM5/0
pvc 0/100 l2transport
  oam-ac emulation-enable 2
  encapsulation aal5
  xconnect 10.0.0.203 27 pw-class pw-l2tpv3-atm
  service-policy in prec-2
 !
!
```

Now generate 500 packets by using an extended **ping** command from the Oakland CE to the Albany CE, and check the MQC policy-map counters (see Example 13-53).

Example 13-53. *Tunnel Marking with MQC Verification in the SanFran PE*

```
SanFran#show policy-map interface
 ATM5/0: VC 0/100 -

  Service-policy input: prec-2 (1330)

    Class-map: all_traffic (match-all) (1331/3)
      500 packets, 56000 bytes
      5 minute offered rate 4000 bps, drop rate 0 bps
      Match: any (1332)
      QoS Set
        ip precedence tunnel 2
          Packets marked 500

   Class-map: class-default (match-any) (1334/0)
     0 packets, 0 bytes
     5 minute offered rate 0 bps, drop rate 0 bps
     Match: any (1335)
       0 packets, 0 bytes
       5 minute rate 0 bps
SanFran#
```

You can see that 500 packets were classified and marked with tunnel precedence 2. You can also check the PE-P interface in the NewYork PE for IP accounting statistics after you clear the counters (see Example 13-54).

Example 13-54. *Tunnel Marking with MQC Verification in the NewYork PE*

```
NewYork#show interfaces precedence
Serial4/0
  Input
    Precedence 2: 500 packets, 70000 bytes
NewYork#
```

You can configure the three marking methods simultaneously. In that case, the relative priority from highest to lowest is as follows:

1. MQC **set ip** {**precedence** | **dscp**} **tunnel**
2. ToS reflection
3. ToS setting

For this reason, it is recommended that you use the MQC configuration in favor of the legacy ToS setting or marking. The policy for setting the IP precedence or DSCP in an L2TPv3 tunnel can be selected as a policing action by using the keywords **set-prec-tunnel-transmit** or **setd-scp-tunnel-transmit**.

Case Study 13-5: Traffic Policing

Using the MQC model, you can configure traffic policing using single- or dual-rate policers that have multiple conform, exceed, and violate actions. Example 13-55 shows the possible actions for conforming traffic, including the highlighted tunnel marking. The same actions are available for exceeding and violating traffic.

Example 13-55. *Traffic-Policing Actions*

```
SanFran(config-pmap-c-police)#conform-action ?
  drop                                drop packet
  set-clp-transmit                    set atm clp and send it
  set-cos-transmit                    set cos and send it
  set-discard-class-transmit          set discard-class and send it
  set-dscp-transmit                   set dscp and send it
  set-dscp-tunnel-transmit            rewrite tunnel packet dscp and send it
  set-frde-transmit                   set FR DE and send it
  set-mpls-exp-imposition-transmit    set exp at tag imposition and send it
  set-prec-transmit                   rewrite packet precedence and send it
  set-prec-tunnel-transmit            rewrite tunnel packet precedence and send it
  set-qos-transmit                    set qos-group and send it
  transmit                            transmit packet

SanFran(config-pmap-c-police)#conform-action
```

Example 13-56 shows the configuration of a policer with marking actions.

Example 13-56. *Traffic-Policing Configuration*

```
!
hostname SanFran
!
class-map match-all all_traffic
  match any
!
policy-map my_policer
  class all_traffic
   police cir 64000 pir 128000
     conform-action set-prec-tunnel-transmit 5
     exceed-action set-prec-tunnel-transmit 1
     violate-action drop
interface ATM5/0
pvc 0/100 l2transport
  oam-ac emulation-enable 2
  encapsulation aal5
  xconnect 10.0.0.203 27 pw-class pw-l2tpv3-atm
  service-policy in my_policer
 !
!
```

MQC is versatile enough through the use of nested or hierarchical policies to allow the configuration of ATM Forum Traffic Management 4.0 (TM 4.0) policers using the policy police rate.

With this capability, a PE device can behave like a traditional ATM switch. In particular, you can configure the following policing policies:

- **CBR Policing** Using a single class and **police** statement.
- **VBR.1 Policing** Using hierarchical policies:

  A parent policy includes a **police** statement to policy at peak cell rate (PCR) for all cells.

  A child policy includes a **police** statement to policy at sustained cell rate (SCR) for all cells.

- **VBR.2 Policing** Using hierarchical policies:

  A parent policy includes a **police** statement to policy at PCR for all cells.

  A child policy includes a **police** statement to policy at SCR for all cell loss priority of zero (CLP0) cells classified under a new class-map.

- **VBR.3 Policing** Using hierarchical policies:

  A parent policy includes a **police** statement to policy at PCR for all cells.

  A child policy includes a **police** statement to policy at SCR for all CLP0 cells that are classified under a new class-map, and mark as exceeded any cells with CLP1.

- **UBR.1 Policing** Using a single class and **police** statement.

Example 13-57 shows the ATMF TM 4.0 policer configuration, including tunnel IP precedence marking using the policy **set ip precedence tunnel**. The highlighted policy-maps are to be applied as input service policies under the ATM PVC.

Example 13-57. *ATM TM 4.0 Policers*

```
!
class-map match-all CLP0
  match not atm clp
!
!
policy-map CBR.1
  class class-default
   police rate 10000 cps delay-tolerance 500
     conform-action transmit
     exceed-action drop
   set ip precedence tunnel 5
policy-map VBR.1_child
  class class-default
   police rate 5000 cps atm-mbs 1000
     conform-action transmit
     exceed-action drop
policy-map VBR.1
  class class-default
```

```
   police rate 10000 cps delay-tolerance 200
     conform-action transmit
     exceed-action drop
   set ip precedence tunnel 4
   service-policy VBR.1_child
 policy-map VBR.2_child
   class CLP0
    police rate 5000 cps atm-mbs 1000
      conform-action transmit
      exceed-action drop
policy-map VBR.2
  class class-default
   police rate 10000 cps delay-tolerance 200
     conform-action transmit
     exceed-action drop
   set ip precedence tunnel 3
   service-policy VBR.2_child
policy-map VBR.3_child
  class CLP0
   police rate 5000 cps atm-mbs 750
     conform-action transmit
     exceed-action set-clp-transmit
policy-map VBR.3
  class class-default
   police rate 10000 cps delay-tolerance 400
     conform-action transmit
     exceed-action drop
   set ip precedence tunnel 2
   service-policy VBR.3_child
policy-map UBR.plus
  class class-default
   police rate 10000 cps delay-tolerance 2000
     conform-action transmit
     exceed-action drop
   set ip precedence tunnel 1
!
```

The ATMF TM4.0 specification uses the PCR as the rate in the first bucket and cell delay variation tolerance (CDVT) as the height of the first bucket. It also defines the use of the SCR as the rate in the second bucket and a function of maximum burst size (MBS) as the height of the second bucket.

You can see that, as specified in ATMF TM4.0, Example 13-57 uses the PCR and CDVT combination in the first bucket specified by the parent policy and uses the SCR and MBS combination in the second bucket specified by the child policy for VBR service types.

> **Note**
>
> Some platforms do not support the **atm-mbs** keyword. In those cases, you should define the SCR only in the child policies.

On some platforms, Layer 2 marking is supported only in the outbound direction. In these platforms, you cannot implement VBR.3 because it cannot use **set atm-clp** in an input policy. However, you can usually configure around this by marking with a new tunnel IP precedence

instead of ATM CLP at ingress. You can map the tunnel IP precedence back to ATM CLP at egress with an intermediate **qos-group**:

- **Ingress PE** In the ingress PE, perform VBR.2 policing as an inbound policy but set a different tunnel IP precedence or DSCP for the exceed-action in the VBR.3_child policymap instead of performing a drop action. This effectively performs an IP marking. You can refer to this IP precedence as *CLP precedence*.

- **Egress PE** In the egress PE, at ingress from the P router, match the CLP precedence tunnel IP precedence with an inbound policy and set the **qos-group**. This classifies traffic that will later be marked with CLP. At egress in the attachment circuit that is outbound toward the CE, match the **qos-group** and **set atm-clp**.

Two local markings exist: **qos-group** and **discard-class**. They preserve the marking tunnel IP precedence or DSCP information (or MPLS Experimental bits in AToM) before tunnel or label disposition. Used as an input feature, the **qos-group** ID identifies or selects a class, and the **discard-class** identifies a drop precedence. These two local markings are important when input Layer 2 marking is not supported. They allow you to match on PSN information, such as IP DSCP or MPLS EXP, while acting on them at egress on an attachment circuit when that PSN class information is lost because of disposition.

The intermediate step is the **qos-group** ID, which conveys the received class to the output interface. A **qos-group** and **discard-class** are required when you use the input PHB marking to classify packets on the output interface. Example 13-58 shows a sample configuration setting the ATM CLP at attachment circuit egress based on the tunnel IP precedence at ingress from the P router.

Example 13-58. *Conveying Class Classification from Egress to Egress*

```
!
hostname NewYork
!
class-map match-all pre1
  match ip precedence 1
class-map match-all qosg
  match qos-group 1
!
!
policy-map clp
  class qosg
   set atm-clp
policy-map qosg
  class pre1
   set qos-group 1
!
interface Serial4/0
 ip unnumbered Loopback0
 service-policy input qosg
!
interface ATM5/0
 no ip address
 pvc 0/101 l2transport
  oam-ac emulation-enable 2
  encapsulation aal5
  xconnect 10.0.0.201 27 pw-class pw-l2tpv3-atm
  service-policy out clp
```

```
 !
!
```

You can see that the Serial4/0 interface coming into the NewYork PE from the Denver P has the inbound service policy qosg. The qosg policy-map classifies traffic with IP precedence of 1 in the class-map pre1 and sets the **qos-group** to 1 for the classified packets.

At egress and toward the attachment circuit, the PVC 0/101 in interface ATM5/0 has the outbound service policy clp. The policy-map clp sets the ATM CLP bit for traffic that is classified with the qosg class-map that matches qos-group 1. In effect, you are applying an outbound policy in the outgoing interface from an inbound classification in the incoming interface. The **qos-group** marking and matching acts like the middle man. Example 13-59 shows the verification for this pair of service policies.

Example 13-59. *Verifying the QoS Group Configuration*

```
NewYork#show policy-map interface
 Serial4/0

  Service-policy input: qosg (1581)

    Class-map: pre1 (match-all) (1582/9)
      5 packets, 700 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: ip precedence 1  (1583)
      QoS Set
        qos-group 1
          Packets marked 5

    Class-map: class-default (match-any) (1585/0)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any  (1586)
        0 packets, 0 bytes
        5 minute rate 0 bps
ATM5/0: VC 0/101 -

Service-policy output: clp (1561)

  Class-map: qosg (match-all) (1562/8)
    5 packets, 560 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: qos-group 1 (1570)
    QoS Set
      atm-clp
        Packets marked 5

  Class-map: class-default (match-any) (1565/0)
    0 packets, 0 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: any (1566)
      0 packets, 0 bytes
      5 minute rate 0 bps
NewYork#
```

The input service policy qosg in Serial4/0 matches 5 packets and sets the **qos-group** to 1. In turn, the output service policy clp in ATM5/0 VC 0/101 matches these 5 packets by **qos-group** and sets the **atm-clp**.

Case Study 13-6: Queuing and Shaping

MQC supports multiple queuing and shaping of outbound features in L2TPv3 environments. Some of these features are as follows:

- **Low-latency queuing (LLQ)** The LLQ is a strict priority first-in, first out (FIFO) queue. Strict priority queuing allows delay-sensitive data to receive a preferential queuing treatment by being dequeued and serviced before other queues.
- **Class-based weighted fair queuing (CBWFQ)** CBWFQ provides a fair queuing based on defined classes with no strict priority. The weight for a packet that belongs to a specific class is given from the bandwidth that you assigned to the class when you configured it.
- **Weighted Random Early Detection (WRED)** WRED drops packets selectively based on IP precedence. The higher the IP precedence, the less likely the packets are to be dropped.

Example 13-60 shows an example of egress queuing policies to provide committed information rate (CIR) guarantees.

Example 13-60. *Queuing Configuration for Frame Relay Pseudowires*

```
!
hostname SanFran
!
class-map match-all cust1
  match fr-dlci 100
class-map match-all cust2
  match fr-dlci 101
!
policy-map cir_guarantee
  class cust1
    bandwidth 128
  class cust2
    bandwidth 256
!
interface Serial3/1
 no ip address
 service-policy output cir_guarantee
 encapsulation frame-relay
 frame-relay intf-type dce
!
```

Example 13-60 matches Frame Relay DLCIs in two different class-maps and applies the bandwidth policy to each class. Example 13-60 does not show the two pseudowires in Serial 3/1 configured with the global **connect** command with attachment circuits of DLCI 100 and 101, respectively.

You can accomplish per-class traffic shaping for ATM PVC and permanent virtual path (PVP) attachment circuits with the **pvc** and **atm pvp** mode ATM service type configuration using the

following commands:

- **CBR** {**PCR**}
- **UBR** {**PCR**}
- **VBR-RT** {**PCR**} {**SCR**} [**MBS**]
- **VBR-NRT** {**PCR**} {**SCR**} [**MBS**]

Case Study 13-7: Layer 2-Specific Matching and Setting

L2TPv3 transports and tunnels multiple and diverse Layer 2 technologies. It is reasonable that MQC supports different matching and setting criteria for different Layer 2 protocols. Table 13-1 summarizes some of these Layer 2 technology-specific criteria.

Table 13-1. ***Layer 2-Specific Matching and Marking Criteria***

Layer 2	**Matching**	**Setting**
Ethernet	**match cos** **match vlan** (including VLAN ranges)	**set cos**
Frame Relay	**match fr-de** **match fr-dlci**	**set fr-de** **set fr-fecn-becn**
ATM	**match atm clp**	**set atm-clp**

With these combinations, you can configure the following:

- **Input service policies** Matching on the Frame Relay DE bit, the ATM CLP bit, or Ethernet 802.1P CoS bits coming from the attachment circuit, and setting the IP tunnel precedence or DSCP into the L2TPv3 tunnel accordingly.
- **Output service policies** Matching on a **qos-group** or **discard-class** conveying the input IP tunnel precedence or DSCP that is incoming from the IP PSN, and setting the Frame Relay DE or forward explicit congestion notification/backward explicit congestion notification (FECN/BECN) bits, ATM CLP bit, or 802.1P CoS bits as desired toward the attachment circuit.

For FECN/BECN marking, both FECN and BECN bits are set when they are above the marking threshold.

Example 13-61 shows an inbound service policy applied to an ATM PVC attachment circuit that sets the tunnel IP precedence to 2 (immediate) for ATM packets that do not have the CLP bit set.

Example 13-61. *Matching on ATM CLP*

```
!
hostname SanFran
!
class-map match-all not-clp
  match not atm clp
policy-map prec-2
  class not-clp
   set ip precedence tunnel 2
!
interface ATM5/0
pvc 0/100 l2transport
  oam-ac emulation-enable 2
  encapsulation aal5
  xconnect 10.0.0.203 27 pw-class pw-l2tpv3-atm
  service-policy in prec-2
 !
!
```

Example 13-62 shows how to set the ATM CLP bit for all traffic.

Example 13-62. *Setting on ATM CLP-Configuration*

```
!
hostname NewYork
!
class-map match-all everything
  match any
policy-map atm-clp
  class everything
   set atm-clp
interface ATM5/0
pvc 0/101 l2transport
  oam-ac emulation-enable 2
  encapsulation aal5
  xconnect 10.0.0.201 27 pw-class pw-l2tpv3-atm
  service-policy out atm-clp
 !
!
```

Example 13-63 Shows the verification results in the NewYork PE when sending 200 pings from the Oakland CE to the Albany CE.

Example 13-63. *Setting on ATM CLPVerification*

```
NewYork#show policy-map interface
 ATM5/0: VC 0/101 -
```

```
  Service-policy output: atm-clp (1132)

    Class-map: everything (match-all) (1133/1)
      200 packets, 22400 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any  (1134)
      QoS Set
        atm-clp
          Packets marked 200

    Class-map: class-default (match-any) (1136/0)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any  (1137)
        0 packets, 0 bytes
        5 minute rate 0 bps
NewYork#
NewYork#show atm pvc 0/101
ATM5/0: VCD: 3, VPI: 0, VCI: 101
UBR, PeakRate: 149760
AAL5 L2transport, etype:0xF, Flags: 0x32000C2E, VCmode: 0x0
OAM Cell Emulation: enabled, F5 End2end AIS Xmit frequency: 2 second(s)
Interworking Method: like to like
Remote Circuit Status = No Alarm, Alarm Type = None
OAM frequency: 0 second(s), OAM retry frequency: 1 second(s)
OAM up retry count: 3, OAM down retry count: 5
OAM Loopback status: OAM Disabled
OAM VC state: Not Managed
ILMI VC state: Not Managed
InPkts: 200, OutPkts: 200, InBytes: 21600, OutBytes: 21600
InPRoc: 0, OutPRoc: 0
InFast: 200, OutFast: 200, InAS: 0, OutAS: 0
InPktDrops: 0, OutPktDrops: 0
CrcErrors: 0, SarTimeOuts: 0, OverSizedSDUs: 0
Out CLP=1 Pkts: 200
OAM cells received: 0
F5 InEndloop: 0, F5 InSegloop: 0, F5 InAIS: 0, F5 InRDI: 0
OAM cells sent: 0
F5 OutEndloop: 0, F5 OutSegloop: 0, F5 OutAIS: 0, F5 OutRDI: 0
OAM cell drops: 0
Status: UP
NewYork#
```

You can see from both the policy-map and ATM PVC counters that 200 packets were marked with CLP1.

Summary

This chapter explored various L2TPv3 advanced concepts and topics through the use of case studies. It covered the detailed theoretical and practical operation of PMTUD in L2TPv3. It also highlighted the more theoretical aspects of MTU handling and PMTUD with "the life of a packet" scenarios and figures both with and without PMTUD. When you are adding encapsulation headers to an SDU, MTU and fragmentations issues arise in the IP PSN. PMTUD combined with DF bit setting provides a practical approach to solving the core MTU and IP reassembly processing problems.

You learned the control plane and data plane considerations of ATM OAM cell emulation and ATM cell packing. New L2TPv3 AVPs were presented detailing their use, functionality, and interaction.

In the final few pages of this chapter, you learned QoS essentials, configuration, and QoS particulars regarding L2TPv3. You saw how to configure and apply traffic marking, policing, queuing, shaping, and advanced QoS matching and setting criteria for each Layer 2 technology that is tunneled and transported using L2TPv3.

Part V: Additional Layer 2 VPN Architectures

Chapter 14. Layer 2 Interworking and Local Switching

This chapter covers the following topics:

- Layer 2 Interworking technology overview
- Layer 2 Interworking case studies
- Layer 2 local switching
- Layer 2 local switching with interworking
- Understanding advanced interworking and local switching

This chapter comprises important topics that are part of the Cisco Unified VPN Suite. The first part covers Layer 2 any-to-any pseudowires (also known as *interworking* [IW]), by which a pseudowire links two attachment circuits with different data-link layer protocols. You can provision Layer 2 any-to-any IW circuits using either Any Transport over MPLS (AToM) in Multiprotocol Label Switching (MPLS) packet-switched networks (PSNs) or Layer 2 Tunnel Protocol Version 3 (L2TPv3) in IP cores. This chapter covers both scenarios.

The second part of this chapter details Layer 2 local switching, which allows you to switch Layer 2 packets between two interfaces within the same router. Layer 2 local switching is not a pseudowire technology per-say because pseudowire control plane signaling is not involved. However, local switching is considered part of the Layer 2 virtual private network (VPN) solution because operators who implement Layer 2 VPNs come across situations in which local switching is required.

The final part of this chapter consists of a section about Layer 2 local switching with IW and a section on advanced topics that are related to Layer 2 IW and local switching.

The case studies in this chapter explain how to configure and manage Layer 2 IW, Layer 2 local switching, and mixed Layer 2 IW plus local switching scenarios. They provide extensive examples using multiple attachment circuit technology combinations.

Layer 2 Interworking Technology Overview

Previous chapters dealt with a diverse variety of AToM and L2TPv3 examples on what is sometimes referred to as *like-to-like attachment circuits*. As the name implies, like-to-like means that the two attachment circuits use the same Layer 2 technology (that is, either two Ethernet VLAN attachment circuits or two Frame Relay attachment circuits). This section offers you a mechanism and the underlying theory to configure any-to-any IW pseudowires. IW functions perform the translation and adaptation necessary to interconnect disparate attachment circuits by means of a native service processor (NSP) function. The NSP requires knowledge of the semantics of the payload to be adapted. It resides between the pseudowire termination and the attachment circuit.

Following are the two types of Layer 2 VPN IW:

- **Bridged (Ethernet) Internetworking** Ethernet frames that are extracted from the attachment circuit are sent over the pseudowire. In the case of 802.1q, the VLAN tag is removed. The pseudowire functions in Ethernet (VC type 0x0005) like-to-like mode, and the IW function at the NSP performs the required adaptation based on the attachment circuit technology. Non-Ethernet frames are dropped.

- **Routed (IP) Internetworking** IP packets that are extracted from the attachment circuit are sent over the pseudowire. The pseudowire functions in IP Layer 2 Transport (VC type 0x000B) like-to-like mode, and the IW function (NSP) performs the required adaptation based on the attachment circuit technology. Non-IPv4 packets are dropped.

In general, you use Layer 2 IW to connect different Layer 2 technologies at both attachment circuits by means of a pseudowire. The actual type of IW typically depends on the end-to-end application type, such as bridged or routed. If you want to interconnect different attachment circuit technologies and carry protocols other than IP, the only current option is bridged IW.

Note

Although the real use of any-to-any is to connect dissimilar attachment circuit types, no rule states that the circuit types need to be different. You can configure a bridge IW pseudowire between two ATM virtual circuit (VC) attachment circuits. However, just because you can do it does not mean you should, because not using the native VC type has disadvantages, as you learn in the next sections. Specifically, only a subset of the possible packets received in the attachment circuit can be transported.

In the IW case, the pseudowire consists of two endpoints with the same VC type. With Ethernet IW, the two pseudowire endpoints advertise a VC type of 5 for Ethernet. In contrast, with IP IW, the two pseudowire endpoints advertise a VC type of 11 for IP Layer 2 Transport. The attachment circuits can be different, however, so an IW function deals with processing the native service. The consequence is of paramount importance: Unlike the like-to-like case in which the attachment circuit is transported transparently and unmodified, in the IW case, the attachment circuits are terminated locally. The behavior of locally terminating attachment circuits imposes some limitations, as you learn in the next two sections.

Another consequence is that the maximum transmission unit (MTU) considerations for IW scenarios differ from the like-to-like ones, not only because the PDU that is transported is different, but also because various interface types might have diverse default MTU values.

Bridged Interworking

In the Ethernet IW case, Ethernet frames are bridged across the pseudowire. The customer edge (CE) devices can either be running native Ethernet bridging or using Integrated Routing and Bridging (IRB) or Routed Bridge Encapsulation (RBE), for example, on ATM subinterfaces. This scenario shows you one of the usages for bridged IW: when a customer wants to bridge between two sites with LAN segments but the service provider access technology in one of the sites is either ATM or Frame Relay.

Bridged IW has some Layer 2-specific encapsulation behaviors, specifically when carrying bridged protocols over ATM or Frame Relay. With Ethernet over ATM, two translations by the NSP are supported when using Logical Link Control (LLC) of 0xAA-AA-03, indicating a Subnetwork Access Protocol (SNAP) header and an Organizationally Unique Identifier (OUI) of 0x00-80-C2, which means bridged protocols. The same two translations by the NSP are supported for Ethernet over Frame Relay using Control of 0x03; Pad of 0x00; Network Layer Protocol Identifier (NLPID) of 0x80, indicating a SNAP header; and an OUI of 0x00-80-C2, indicating bridged protocols:

- **PID 0x0007** 802.3/Ethernet without preserved frame check sequence (FCS)
- **PID 0x000E** Bridge protocol data unit (BPDU), as defined by 802.1 or 802.1(g).

Figure 14-1 shows the bridged Ethernet/802.3 frame encapsulation over Frame Relay and ATM.

Figure 14-1. ***Bridged Ethernet/802.3 Frame Encapsulation over Frame Relay and ATM***

[View full size image]

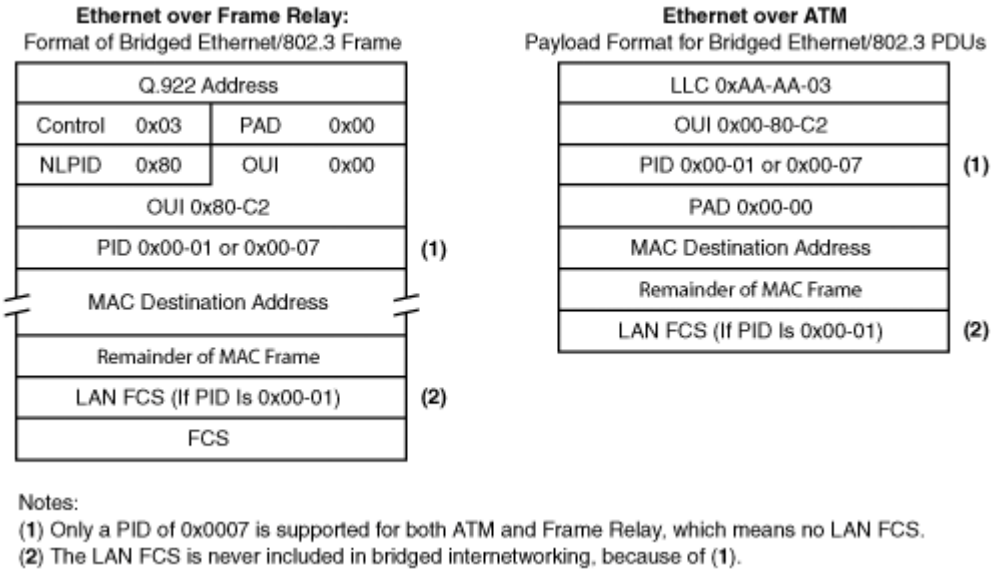
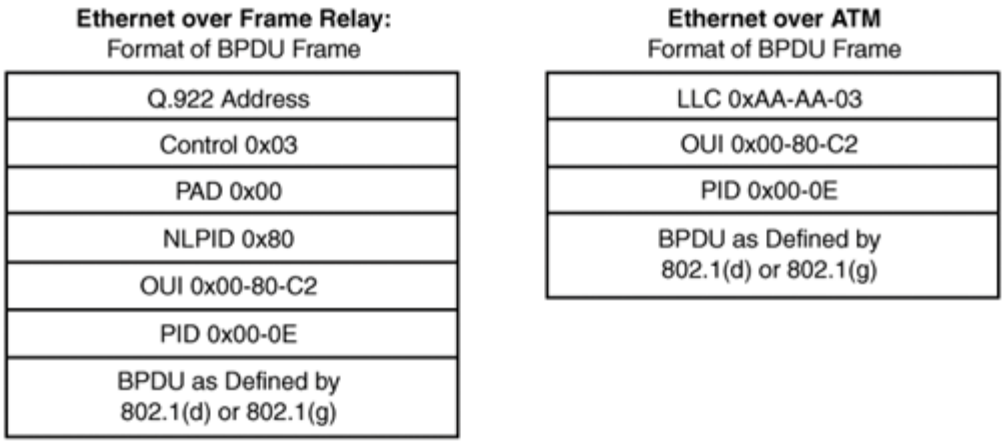


Figure 14-2 shows the BPDU encapsulation over Frame Relay and ATM.

Figure 14-2. *BPDU Encapsulation over Frame Relay and ATM*



The encapsulation of bridged Ethernet over Frame Relay and ATM is defined in RFC 2427 (which obsoletes RFC 1490 and RFC 1294), "Multiprotocol Interconnect over Frame Relay," and RFC 2684 (which obsoletes RFC 1483), "Multiprotocol Encapsulation over ATM Adaptation Layer 5," respectively.

Routed Interworking

In the routed IW case, only IPv4 packets are sent over the pseudowire. Any non-IPv4 packets are dropped. Depending on the Layer 2 technology being used, IPv4 packets are identified by a specific upper layer protocol identification field: either Ethertype (0x0800), PPP dynamic link library (DLL) (0x0021), or NLPID (0xCC) for Ethernet, PPP, and Frame Relay, respectively. Because Layer 2 technologies encapsulate IP datagrams differently, the NSP IW function is required to perform translation.

Only IP packets are transported; therefore, address resolution packets are dropped. Address resolution is handled differently in diverse Layer 2 technologies:

- **Ethernet** Using Address Resolution protocol (ARP)
- **Frame Relay and ATM** Using Inverse ARP
- **PPP** Using Internet Protocol Control Protocol (IPCP)

These address resolution packets are dropped, because they are not IPv4 packets. The protocol type in all cases is as follows:

- **ARP** Ethertype 0x0806 defined in RFC 826
- **Inverse ARP** Ethertype 0x0806 defined in RFC 2390
- **IPCP** PPP DLL Protocol number 0x8021 defined in RFC 1332

In consequence, you need to give specific considerations to address resolution. Following are some tips for routed IW:

- **General consideration** Because the address resolution mechanisms and packets are different, no direct translation is possible other than an NSP function involving address resolution termination and signaling.
- **Ethernet** With native Ethernet, the provider edge (PE) device acts as a proxy-ARP to all ARP requests received from the CE.
- **Point-to-point ATM and Frame Relay** Inverse ARP does not run by default in point-to-point Frame Relay or ATM subinterfaces, because the IP address and subnet mask define the connected prefix. Therefore, no configuration is required in the CE devices.
- **Multipoint ATM and Frame Relay** Inverse ARP is enabled and runs by default in multipoint ATM and Frame Relay subinterfaces. Because routed IW simply drops inverse ARP packets and does not support inverse ARP, the IPv4 address at the remote end of an ATM or Frame Relay permanent virtual circuit (PVC) cannot be discovered dynamically. Therefore, in this case, manual configuration of static IPv4 to PVC mapping is needed in the CE devices.

- **PPP** For PPP attachment circuits, manual configuration of the remote CE's IPv4 address for IPCP negotiation is needed in the PE device configuration.

  Note

  The address resolution considerations are applicable to routed IW because address resolution packets are not transported end-to-end over the pseudowire.

Interworking MTU Considerations

As you learned in Chapter 6, "Understanding Any Transport over MPLS," in all Layer 2 VPN IW using AToM cases except the transport of ATM cells, MTUs need to match in both attachment circuits for the pseudowire to come up. The MTU value is advertised in the MTU interface parameter.

This prerequisite takes a new meaning with IW because different interface types have different default MTU values. The default MTU values in Cisco IOS are shown in Table 14-1.

Table 14-1. ***Default MTU Values for Different Medias***

Interface Type	Default MTU
Serial	1500 bytes
Ethernet	
HSSI	4470 bytes
ATM	
POS	

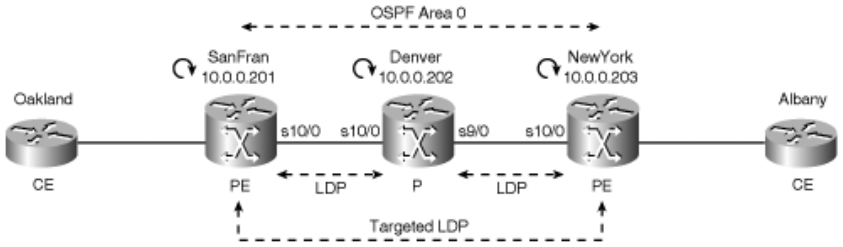
When you are configuring pseudowires between interfaces that have default MTU values (such as Packet over SONET [POS] to Ethernet), the MTU values need to match. Frame Relay has a special circumstance that is covered in the case studies.

Layer 2 Interworking Case Studies

This section covers case studies of Layer 2 VPN IW. Multiple variations of Ethernet and IP IW using both AToM and L2TPv3 are covered using the network shown in Figure 14-3.

Figure 14-3. *L2VPN IW Case Study Topology*

[View full size image]



Note

The LDP and targeted LDP configuration and sessions apply only to the AToM cases.

All case studies use a common initial configuration, which includes the following:

- Create a loopback interface and assign a /32 IP address to it.
- Enable IP Cisco Express Forwarding (CEF) globally.
- For AToM cases, enable MPLS globally and select Label Distribution Protocol (LDP) as the label distribution protocol. Specify the loopback interface's IP address as the LDP router ID.
- By using point-to-point serial interfaces that are running Cisco High-Level Data Link Control (HDLC) between PE and P routers, unnumber those interfaces' IP addresses to the previously defined loopback. This effectively reduces the number of IP addresses to one for all core routers.
- For AToM cases, enable LDP in all PE-to-P interfaces.
- Enable an Interior Gateway Protocol (IGP) among the core routers. These case studies use Open Shortest Path First (OSPF) with a single area 0.

The initial configuration including MPLS is shown for the SanFran PE router. The Denver and New York configurations are analogous to this one (see Example 14-1).

Example 14-1. *Required Preconfiguration*

```
!
hostname SanFran
```

```
!
ip cef
mpls ip
mpls label protocol ldp
mpls ldp router-id Loopback0 force
!
interface Loopback0
 ip address 10.0.0.201 255.255.255.255
!
interface Serial10/0
 ip unnumbered Loopback0
 mpls ip
!
router ospf 1
 log-adjacency-changes
 network 10.0.0.201 0.0.0.0 area 0
```

The configuration for Layer 2 IW transport requires the use of a pseudowire class and involves the use of the command **interworking** {**ip** | **ethernet**}. In the upcoming sections, you will learn the configuration and verification for the following IW case studies:

- Ethernet (Bridged) IW:

  Case Study 14-1: Ethernet-to-VLAN Using AToM

  Case Study 14-2: Ethernet-to-VLAN Using L2TPv3

  Case Study 14-3: ATM AAL5-to-VLAN Using AToM

- IP (Routed) IW:

  Case Study 14-4: Frame Relay-to-VLAN Using AToM

  Case Study 14-5: Frame Relay-to-PPP Using L2TPv3

  Case Study 14-6: IP L2-Transport MTU Considerations

  Case Study 14-7: Frame Relay-to-ATM Interworking Best Practices

Ethernet (Bridged) Interworking Case Studies

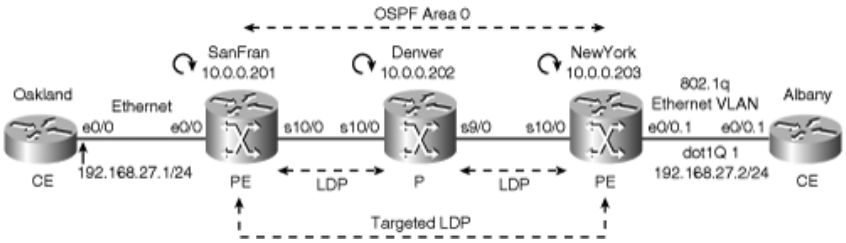
In this section, you learn to configure bridged IW using both AToM and L2TPv3.

Case Study 14-1: Ethernet-to-VLAN Using AToM

The first case study covers Ethernet-to-VLAN bridged IW using AToM. The topology used is shown in Figure 14-4.

Figure 14-4. *Ethernet-to-VLAN Bridged IW Using AToM*

[View full size image]



In the case of IW, the attachment circuit configuration required in a PE device does not mirror the remote PE device configuration.

Example 14-2 shows the configuration for the Ethernet side in the SanFran PE.

Example 14-2. *Ethernet Side Configuration for Ethernet IW Using AToM*

```
!
hostname SanFran
!
pseudowire-class atom-iw-eth-vlan
 encapsulation mpls
 interworking ethernet
!
interface Ethernet0/0
 no ip address
 no cdp enable
 xconnect 10.0.0.203 1 pw-class atom-iw-eth-vlan
!
```

From Example 14-2, you can see that the specific configuration requires the **interworking ethernet** command in the pseudowire-class configuration submode. The **xconnect** directive is applied to the Ethernet interface. Also note that the **no cdp enable** command was entered to prevent the PE from sending CDP packets to the CE. The PE device does not discover the CE device as a Cisco Discovery Protocol (CDP) neighbor, because the PE device does not inspect CDP packets coming from the CE; instead, these packets are transported over the pseudowire. However, if you did not disable CDP in the attachment circuit, the PE sends CDP packets out the attachment circuit, and the local CE discovers the PE in addition to the remote CE. You disable CDP to prevent the CE from "seeing" the PE device.

Example 14-3 shows the configuration for the 802.1q VLAN side in the NewYork PE.

Example 14-3. *VLAN Side Configuration for Ethernet Interworking Using AToM*

```
!
hostname NewYork
!
pseudowire-class atom-iw-eth-vlan
 encapsulation mpls
 interworking ethernet
!
interface Ethernet0/0
 no ip address
 no cdp enable
!
```

```
interface Ethernet0/0.1
 encapsulation dot1Q 1
 no cdp enable
 xconnect 10.0.0.201 1 pw-class atom-iw-eth-vlan
!
```

The VLAN PE side is similar to the Ethernet side, except that the **xconnect** command is applied to the dot1Q subinterface. Also, remember to disable CDP in the Ethernet main interface and subinterface so that you do not send CDP packets to the CE device.

The CE configuration is included in Example 14-4 for both Oakland and Albany CEs for comparison.

Example 14-4. *Ethernet-to-VLAN CE Configuration*

```
! Oakland CE Ethernet attachment circuit configuration
!
hostname Oakland
!
interface Ethernet0/0
 ip address 192.168.27.1 255.255.255.0
!


! Albany CE VLAN attachment circuit configuration
!
hostname Albany
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.1
 encapsulation dot1Q 1
 ip address 192.168.27.2 255.255.255.0
!
```

Example 14-5 shows that the AToM L2transport circuit is UP. It also shows other detailed information captured from the NewYork side.

Example 14-5. *AToM Bridged Interworking Verification*

```
NewYork#show mpls l2transport vc

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------  --------------  ---------  ----------
Et0/0.1        Eth VLAN 1              10.0.0.201      1          UP
NewYork#show mpls l2transport vc detail
Local interface: Et0/0.1 up, line protocol up, Eth VLAN 1 up
  MPLS VC type is Ethernet, interworking type is Ethernet
  Destination address: 10.0.0.201, VC ID: 1, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 17, next hop point2point
    Output interface: Se10/0, imposed label stack {17 18}
  Create time: 00:20:51, last status change time: 00:20:06
  Signaling protocol: LDP, peer 10.0.0.201:0 up
    MPLS VC labels: local 18, remote 18
```

```
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  Sequence number: receive 0, send 0
  VC statistics:
    packet totals: receive 151, send 30
    byte totals:   receive 16854, send 8642
    packet drops:  receive 0, seq error 0, send 0

NewYork#
```

The first verification is performed using the command **show mpls l2transport vc** to check that the L2transport VC is UP. You can note some important points using the **detail** keyword of that command in the NewYork side (VLAN side). The following list explains the first three lines of the output:

- **Local interface and state** Et0/0.1 up, line protocol up. Note that line protocol cannot be detected in the PE Ethernet interfaces because that would imply generating loop packets out of the attachment circuit toward the CE device and intercepting them. Instead, all Ethernet packets received are transported without inspection.
- **Attachment circuit type and state** Eth VLAN 1 up. Note that this refers only to the local attachment circuit type.
- **VC type** Ethernet (VC type 0x0005). Although the attachment circuit (AC) is Ethernet VLAN, the VC type is always Ethernet for bridged IW. Bridged IW uses this VC type for all AC technologies.
- **Interworking type** Ethernet (bridged) as configured under the **pseudowire-class atomiw-eth-vlan** template. This triggers the use of the Ethernet VC Type.
- **10.0.0.201** This is the remote PE's router ID configured in the **xconnect** command.
- **VC ID** 1 as configured in the **xconnect** command. The VC ID needs to match in both PEs.
- **VC status: UP** The VC status UP means that the VC can carry data between the two endpoints. (Imposition and disposition are programmed.) Two conditions need to hold true:

  Disposition interfaces programmed The VC has been configured and the CE interface is up.

  Imposition interfaces programmed The disposition interface is programmed, and there is a remote VC label and an IGP label (label-switched path to the peer). Note that for the imposition interface to be programmed, the disposition interface must have been programmed previously.

Even though the first line of the output shows that the attachment circuit is the Ethernet VLAN with a VLAN ID of 1, the VC type that is signaled is Ethernet because of the IW type Ethernet. You can also see this in Example 14-6.

Example 14-6. *Checking the AToM Bindings*

```
NewYork#show mpls l2transport binding
  Destination Address: 10.0.0.201, VC ID: 1
    Local Label:  18
        Cbit: 1,    VC Type: Ethernet,   GroupID: 0
```

```
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 18
        Cbit: 1,    VC Type: Ethernet,   GroupID: 0
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2

NewYork#
```

In all the pseudowire endpoints that use Ethernet IW, the VC type is 0x0005 for Ethernet regardless of whether the attachment circuits are 802.1q VLAN, ATM AAL5, or Frame Relay.

Each endpoint does behave differently, however, because the attachment circuits differ:

- **Ethernet** The Ethernet attachment circuit behaves exactly as described in Chapter 7, "LAN Protocols over MPLS Case Studies." There is no difference in imposition or disposition of Ethernet frames or in CLI command output. In fact, the Ethernet side is completely unaware that a remote IW function exists, so its state is no different.

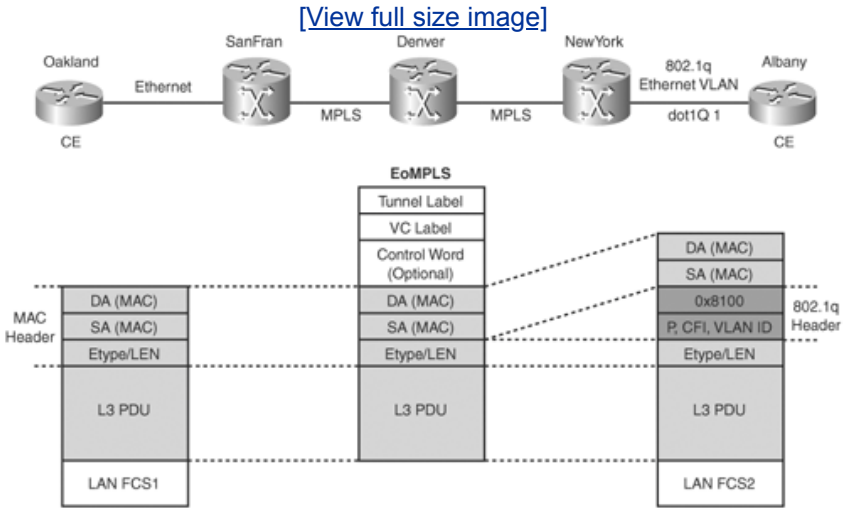
- **802.1q VLAN** The VLAN attachment circuit performs the IW function by the NSP both at imposition and disposition. For example, at disposition, Ethernet frames that are received from the pseudowire are inserted with the 4-byte 802.1q header after the source MAC address. These extra 4 bytes include the 2-byte Ethertype value of 0x8100, indicating 802.1q/802.1p VLAN, followed by the 2 bytes of tag control information (3 bits of class of service [CoS], 1 bit of Canonical Format Identifier [CFI], and 12 bits of VLAN ID equal to 1 in this example). Following these 4 bytes, the next 2 bytes are the ones that originally came after the source MAC addressthat is, Ethertype for Ethernet II and length in the case of 802.3. This new packet is sent over the attachment circuit to the CE device.

It is also worth noting that ARP is end-to-end, because in Ethernet IW, ARP packets are just Ethernet frames with Ethertype 0x0806.

You can see in Figure 14-5 the way the encapsulation changes as the packet traverses from the Oakland CE through the AToM network to the Albany CE and vice versa.

Figure 14-5. ***Ethernet-to-VLAN AToM Bridged Interworking Encapsulation Details***

[View full size image]



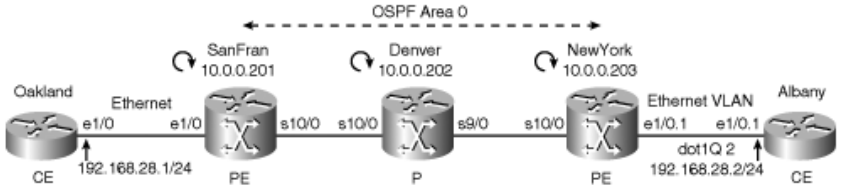
You can see that the NewYork PE inserts the 802.1q header at imposition and removes it at disposition. The 802.1q header is not carried over the pseudowire. You can also see that the LAN FCS is not transported over the pseudowire. The PE routers need to regenerate it at disposition and remove it at imposition.

Case Study 14-2: Ethernet-to-VLAN Using L2TPv3

This case study presents a similar example to Case Study 14-1 but uses L2TPv3 as the pseudowire technology. The topology is included in Figure 14-6. Note that MPLS is not configured.

Figure 14-6. *Ethernet-to-VLAN Bridged IW Using L2TPv3*

[View full size image]



You can view the configuration required for this case study as a like-to-like L2TPv3 configuration with the additional pseudowire class command **interworking ethernet**. You can also view it as the same configuration as Case Study 14-1, using the pseudowire class **encapsulation l2tpv3** command plus any additional L2TPv3 settings.

Example 14-7 shows the configuration required for the SanFran PE. To highlight the specific IW commands, this example uses default L2TPv3 dynamic configuration under the *l2tpv3-iweth-vlan* **pseudowire-class**. Therefore, it employs the L2TP class **l2tp_default_class**.

Example 14-7. *Ethernet Side Configuration for Ethernet IW Using L2TPv3*

```
!
hostname SanFran
!
pseudowire-class l2tpv3-iw-eth-vlan
 encapsulation l2tpv3
 interworking ethernet
 ip local interface Loopback0
!
interface Ethernet1/0
 no ip address
 no cdp enable
 xconnect 10.0.0.203 2 pw-class l2tpv3-iw-eth-vlan
!
```

Example 14-8 shows the configuration required for the NewYork PE, where the attachment circuit is configured in an 802.1q subinterface.

Example 14-8. *VLAN Side Configuration for Ethernet IW Using L2TPv3*

```
!
hostname NewYork
!
pseudowire-class l2tpv3-iw-eth-vlan
 encapsulation l2tpv3
 interworking ethernet
 ip local interface Loopback0
!
interface Ethernet1/0
 no ip address
 no cdp enable
!
interface Ethernet1/0.1
 encapsulation dot1Q 2
 no cdp enable
 xconnect 10.0.0.201 2 pw-class l2tpv3-iw-eth-vlan
!
```

The CE configuration is analogous to the previous case study. You can perform verifications using different permutations of the command **show l2tun session** (see Example 14-9).

Example 14-9. *L2TPv3 Bridged IW Verification*

```
NewYork#show l2tun session
 Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0


LocID       RemID       TunID       Username, Intf/              State
                                    Vcid, Circuit
39772       21748       12926       2, Et1/0.1:2                 est
NewYork#
NewYork#show l2tun session all
 Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0


Session id 39772 is up, tunnel id 12926
Call serial number is 1808100000
Remote tunnel name is SanFran
  Internet address is 10.0.0.201
  Session is L2TP signalled
  Session state is established, time since change 08:33:51
    587 Packets sent, 120 received
    200195 Bytes sent, 13658 received
    Receive packets dropped:
      out-of-order:              0
      total:                     0
    Send packets dropped:
      exceeded session MTU:      0
      total:                     0


  Session vcid is 2
  Session Layer 2 circuit, type is Ethernet Vlan, name is Ethernet1/0.1:2
  Circuit state is UP
  L2TP VC type is Ethernet, interworking type is Ethernet
    Remote session id is 21748, remote tunnel id 48316
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  No session cookie information available
```

```
  FS cached header information:
    encap size = 24 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000
  Sequencing is off
NewYork#
```

Using the command **show l2tun session** without keywords or arguments displays summary information about the L2TPv3 sessions. You can see that the state for the only session is established. You can use the keyword **all** to obtain detailed information. This information tells you that the session was established successfully using L2TP signaling, that the Layer 2 circuit type is Ethernet VLAN, and that the VC type is Ethernet. Like Ethernet IW using AToM, the VC type is always 0x0005 for Ethernet despite the attachment circuit technology, which is Ethernet VLAN in this case. This command clearly shows that the IW type is Ethernet.

Another command that uses the **interworking** keyword presents IW-specific information. Refer to Example 14-10, and compare the differences in output from the two PE routers.

Example 14-10. *Displaying IW Details in L2TPv3*

```
SanFran#show l2tun session interworking
 Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0


LocID       TunID       Peer-address    Type IWrk Username, Intf/
                                                  Vcid, Circuit
21750       48316       10.0.0.203      ETH  -    2, Et1/0
SanFran#

NewYork#show l2tun session interworking
 Session Information Total tunnels 1 sessions 1
 Tunnel control packets dropped due to failed digest 0


LocID       TunID       Peer-address     Type IWrk Username, Intf/
                                                   Vcid, Circuit
39772       12926       10.0.0.201       VLAN ETH  2, Et1/0.1:2
NewYork#
```

From Example 14-10, you can see that in the SanFran end, the type is represented as ETH for Ethernet, because that is the attachment circuit type. The IW type (Iwrk) is dashed out, meaning that an NSP IW function is nonexistent, or to be more precise, a NULL IW function exists between Ethernet and Ethernet.

As you know, the Ethernet side is oblivious about the remote NSP IW function. For all the SanFran side knows, the remote end might also be Ethernet.

In contrast, the NewYork side shows the type as VLAN because that is the attachment circuit type. The NSP IW type is shown as ETH for Ethernet, making explicit the fact that there is an NSP function.

ARP packets are also handled end-to-end just the same way as in the AToM bridged IW Case Study 14-1. As a final note that is applicable to both AToM and L2TPv3 Ethernet VLAN attachment circuits, you can use the command **show interface** to display xconnect statistics (see Example 14-11).

Example 14-11. *Displaying Xconnect Statistics in the Ethernet VLAN*

```
NewYork#show interfaces ethernet 1/0.1
Ethernet1/0.1 is up, line protocol is up
  Hardware is Lance, address is 0000.0c00.cb01 (bia 0000.0c00.cb01)
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 255/255, load 1/255
  Encapsulation 802.1Q Virtual LAN, Vlan ID 2.
  ARP type: ARPA, ARP Timeout 04:00:00
  Xconnect switched:
    Pkts In 556, Chars In 75990, Pkts Out 795, Chars Out 91196
NewYork#
```

Case Study 14-3: ATM AAL5-to-VLAN Using AToM

This bridged IW case study shows Ethernet IW between ATM AAL5 and Ethernet VLAN attachment circuits using AToM. The objective is to present a minor difference and to emphasize concepts. See Example 14-12 for the SanFran ATM AAL5-specific configuration.

Example 14-12. *ATM Side Configuration for Ethernet IW Using AToM*

```
!
hostname SanFran
!
pseudowire-class atom-iw-atm-vlan
 encapsulation mpls
 interworking ethernet
!
interface ATM4/0.500 point-to-point
 mtu 1500
 pvc 0/500 l2transport
  encapsulation aal5snap
  xconnect 10.0.0.203 500 pw-class atom-iw-atm-vlan
 !
!
```

The highlighted lines outline the configuration that is specific to the AAL5 attachment circuit with bridged IW. ATM IW endpoints are sensible only using ATM AAL5 attachment circuits, not cell relay.

Note the MTU setting in the ATM4/0.500 subinterface to match the FastEthernet subinterface MTU and enable the Layer 2 circuit to come up. Some output refers to the Layer 2 circuit as *L2Ckt*. The other PE configuration is included in Example 14-13.

Example 14-13. *VLAN Side Configuration for Ethernet IW Using AToM*

```
!
hostname NewYork
!
pseudowire-class atom-iw-atm-vlan
 encapsulation mpls
 interworking ethernet
!
interface FastEthernet0/0.500
 encapsulation dot1Q 500
 xconnect 10.0.0.201 500 pw-class atom-iw-atm-vlan
!
```

You can use the usual verification commands of **show mpls l2transport vc** and **show mpls l2transport binding** to confirm control plane status (see Example 14-14).

Example 14-14. *Bridged IW Verification*

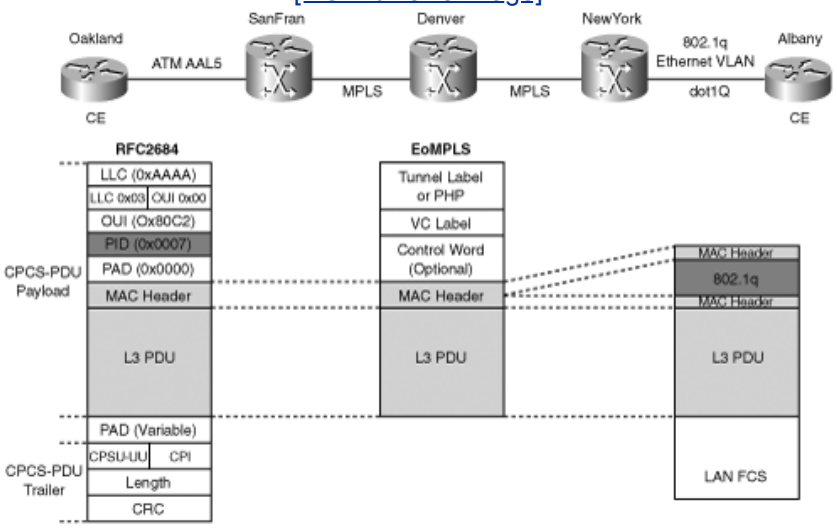
```
SanFran#show mpls l2transport vc 500

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------- --------------- ---------- ----------
AT4/0.500      ATM AAL5 0/500          10.0.0.203      500        UP
SanFran#show mpls l2transport vc 500 detail
Local interface: AT4/0.500 up, line protocol up, ATM AAL5 0/500 up
  MPLS VC type is Ethernet, interworking type is Ethernet
  Destination address: 10.0.0.203, VC ID: 500, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: imp-null, next hop 10.0.1.203
    Output interface: Fa0/0, imposed label stack {16 25}
  Create time: 00:14:22, last status change time: 00:14:22
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 17, remote 25
    Group ID: local 5, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  Sequence number: receive 0, send 0
  VC statistics:
    packet totals: receive 0, send 0
    byte totals:   receive 0, send 0
    packet drops:  receive 0, seq error 0, send 0

SanFran#show mpls l2transport binding 500
  Destination Address: 10.0.0.203,  VC ID: 500
    Local Label:  17


        Cbit: 1,    VC Type: Ethernet,     GroupID: 5
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 25
        Cbit: 1,    VC Type: Ethernet,     GroupID: 0
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2
SanFran#
```

From the output of the command **show mpls l2transport vc 500**, you can see that the local circuit is an ATM AAL5 VC and the status is UP. When appending the **detail** keyword, you can also see that although the attachment circuit is ATM AAL5 0/500, the MPLS VC type (PW Type) is 0x0005 for Ethernet. The IW type is also Ethernet. Finally, using the command **show mpls l2transport binding 500**, you can further prove that for both the ATM AAL5 VC and Ethernet VLAN attachment circuits, the VC type is Ethernet and the MTU that is advertised is 1500 bytes.

In Figure 14-7, the encapsulation changes as the packet traverses from the Oakland CE as bridged Ethernet/802.3 PDUs over AAL5, through the AToM network as Ethernet over MPLS, to the Albany CE as a tagged Ethernet frame and vice versa.

Figure 14-7. *ATM AAL5 to VLAN AToM Bridged IW Encapsulation Details*

[View full size image]



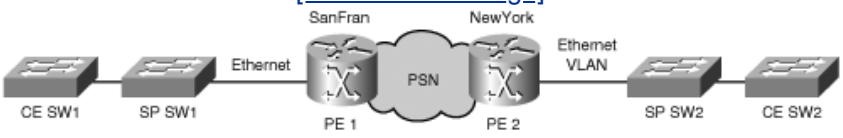
You can see that the NewYork PE inserts the 802.1q header at imposition and removes it at disposition. The LAN FCS is not transported over the pseudowire. The NewYork PE needs to regenerate the LAN FCS at disposition and remove it at imposition. For a bridged frame over AAL5, the organization code is 0x0080C2 for 802.1. Only the PID of 0x0007 indicating 802.3/Ethernet without preserved FCS is supported. Therefore, no LAN FCS exists in the AAL5 attachment circuit.

Ethernet-VLAN IW Switch Environment Considerations

You need to take specific considerations into account in a switched environment, as shown in Figure 14-8, that are not present in a router environment.

Figure 14-8. *Switch Environment Topology*

[View full size image]



You can see from Figure 14-8 that you have an IW scenario between Ethernet and VLAN in a LAN switch environment. From Figure 14-8, SP SW 1 and SP SW 2 are service provider switches, and CE SW 1 and CE SW 2 are customer switches. The PSN can be either MPLS with AToM pseudowire or IP with L2TPv3 pseudowire.

In this case, you are using bridged IW between Ethernet and VLAN attachment circuits and allowing per-VLAN spanning tree plus (PVST+). Because of the NSP IW function, the VLAN tag is modified, removed, or added. Spanning Tree Protocol (STP) BPDUs sent from the switch contain the source VLAN that the BPDU is sent on in the 802.1q header, but PVST+ also contains the Port VLAN ID (PVID) TLV (type, length, value) field inside the BPDU that identifies the VLAN number of the source

port. The result is that the remote CE switch received a BPDU with an outer 802.1q VLAN tag that is different from the VLAN number in the PVID TLV field in the PVST+ BPDU, or even missing. Because of this inconsistency, the BPDU puts the port into a PVID-inconsistent state, blocking the traffic in that VLAN to prevent forwarding loops. This error condition is a result of violating one of the PVST+ rules, ensuring a consistent native VLAN on all bridges. When the inconsistency is detected, a switch logs error messages such as %SPANTREE-2-RX_1QPVIDERR, % SPANTREE-2-RX_BLKPORTPVID, or others depending on the specific configuration and traffic direction. These errors indicate inconsistency between the PVID and the VLAN tag. This consideration also applies to like-to-like Ethernet VLAN mode pseudowires with VLAN rewrite and Frame Relay or ATM to VLAN bridged IW, where a VLAN tag needs to be inserted or removed.

Routed Interworking

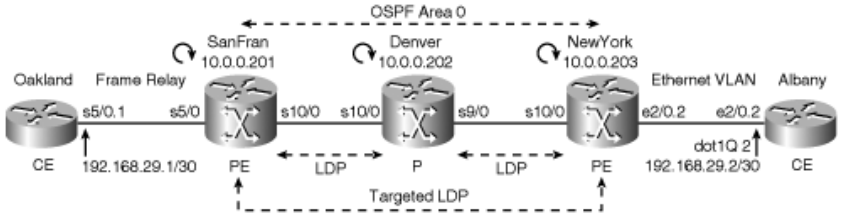
This section explores configuring routed IW using both AToM and L2TPv3. Routed IW uses a new VC type of 11 (0x000B), which was not used before. The AToM imposition function directly encapsulates a raw IP packet inside AToM. That is why the name of VC type 11 is IP Layer2 Transport.

Case Study 14-4: Frame Relay-to-VLAN Using AToM

This case study involves configuring and verifying IP IW between Frame Relay and VLAN attachment circuit endpoints. This topology is included in Figure 14-9.

Figure 14-9. *Frame Relay-to-VLAN Routed IW Using AToM*

[View full size image]



SanFran and Oakland use Frame Relay Internet Engineering Task Force (IETF) encapsulation. SanFran is the LMI data communication equipment (DCE) (Frame Relay switch behavior), and Oakland is the Local Management Interface (LMI) data terminal equipment (DTE). In the case of IP IW, the configuration command that directs the use of IP VC type and consequent transport of IP only is **interworking ip**. Example 14-15 shows the configuration for the SanFran PE.

Example 14-15. *Frame Relay Side Configuration for IP IW Using AToM*

```
!
hostname SanFran
!
frame-relay switching
!
pseudowire-class atom-iw-fr-vlan
 encapsulation mpls
 interworking ip
!
interface Serial5/0
 no ip address
```

```
 encapsulation frame-relay IETF
 frame-relay intf-type dce
!
connect fr-vlan Serial5/0 100 l2transport
 xconnect 10.0.0.203 100 pw-class atom-iw-fr-vlan
 !
!
```

Example 14-15 shows Frame Relay switching globally enabled, so that you can configure the Serial5/0 in the SanFran PE as LMI DCI. In general terms, the configuration is similar to the like-to-like examples, with the addition of the **interworking ip** pseudowire-class configuration mode command.

> **Note**
>
> On IP IW with Frame Relay attachment circuits, both NLPID (0xCC) and SNAP with Ethertype (0x0800) encapsulations identifying IP as upper layer are recognized.

Because it is necessary to specify the encapsulation (either MPLS or L2TPv3) and the IW type (either Ethernet or IP) in all Ethernet and IP IW cases, the use of a pseudowire-class is mandatory.

The rest of the configuration uses the global connect command with the **l2transport** keyword to enter the fr-pw-switching configuration mode and then the actual **xconnect**. See Example 14-16 for the NewYork side of the configuration.

Example 14-16. *VLAN Side Configuration for IP IW Using AToM*

```
!
hostname NewYork
!
pseudowire-class atom-iw-fr-vlan
 encapsulation mpls
 interworking ip
!
interface Ethernet2/0
 no ip address
 no cdp enable
!
interface Ethernet2/0.2
 encapsulation dot1Q 2
 no cdp enable
 xconnect 10.0.0.201 100 pw-class atom-iw-fr-vlan
!
```

The configuration is similar to the Ethernet VLAN like-to-like case, with the addition of the **interworking ip** directive. The CE configuration uses a point-to-point Frame Relay subinterface in the Oakland router and does not need inverse ARP or static mapping (see Example 14-17).

Example 14-17. *CE Configuration for IP IW Using AToM*

```
Oakland#
! Oakland CE configuration
Oakland#show running-config interface serial 5/0.100
```

```
Building configuration...

Current configuration : 153 bytes


!
interface Serial5/0.100 point-to-point
 ip address 192.168.29.1 255.255.255.252
 frame-relay interface-dlci 100 IETF
end

Oakland#

Albany#
! Albany CE configuration
Albany#show running-config interface ethernet 2/0.2
Building configuration...

Current configuration : 121 bytes
!
interface Ethernet2/0.2
 encapsulation dot1Q 2
 ip address 192.168.29.2 255.255.255.252
end

Albany#
```

You can issue the usual verification shown in Example 14-18 from the SanFran side. You can use the command **debug frame-relay pseudowire** to display events and errors that occur, binding a Frame Relay data-link connection identifier (DLCI) to a pseudowire.

Example 14-18. *AToM Routed IW Verification*

```
SanFran#show mpls l2transport vc 100

Local intf     Local circuit           Dest address    VC ID      Status
-------------  ----------------------- --------------- ---------- ----------
Se5/0          FR DLCI 100             10.0.0.203      100        UP
SanFran#show mpls l2transport vc 100 detail
Local interface: Se5/0 up, line protocol up, FR DLCI 100 up
  MPLS VC type is IP, interworking type is IP
  Destination address: 10.0.0.203, VC ID: 100, VC status: up
    Preferred path: not configured
    Default path: active
    Tunnel label: 16, next hop point2point
    Output interface: Se10/0, imposed label stack {16 19}
  Create time: 22:26:36, last status change time: 22:24:17
  Signaling protocol: LDP, peer 10.0.0.203:0 up
    MPLS VC labels: local 20, remote 19
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
  Sequencing: receive disabled, send disabled
  Sequence number: receive 0, send 0
  VC statistics:
    packet totals: receive 14, send 19


    byte totals:   receive 1512, send 2052
    packet drops:  receive 0, seq error 0, send 0
```

```
SanFran#show mpls l2transport binding 100
  Destination Address: 10.0.0.203,  VC ID: 100
    Local Label:  20
        Cbit: 1,    VC Type: IP,    GroupID: 0
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2
    Remote Label: 19
        Cbit: 1,    VC Type: IP,    GroupID: 0
        MTU: 1500,   Interface Desc: n/a
        VCCV Capabilities: Type 1, Type 2
SanFran#
```

In the SanFran PE, the command **show mpls l2transport vc** shows that the VC is UP and the attachment circuit is FR DLCI 100. When you use the **detail** keyword, the VC type of IP (using 0x000B) and the IW type of IP become explicit. The command **show mpls l2transport binding** displays the VC type as IP for both the local and remote endpoints.

Note

In routed IW, no attachment circuit natively uses the VC type of IP, unlike bridged IW, where Ethernet interfaces use the VC type of Ethernet. As a consequence, the l2transport VCs in both PEs always perform the IP IW function. This is to say that for routed IW, two NSPs are needed. You can see this with the output of the command **show mpls l2transport vc detail**, by verifying that the attachment circuit type does not match the VC type in either PE.

To see the VC type being advertised, you can use the debug command **debug mpls l2transport signaling message**, as shown in Example 14-19.

Example 14-19. *Displaying the IP Layer 2 Transport VC Type*

```
SanFran#debug mpls l2transport signaling message
AToM LDP message debugging is on
SanFran#
22:44:46: AToM LDP  [10.0.0.203]: Sending label mapping msg
vc type 11, cbit 1,  vc id 100, group id 0, vc label 20, status 0, mtu 1500
22:45:19: AToM LDP [10.0.0.203]: Received label mapping msg, id 3141, graceful restart
instance 0
vc type 11, cbit 1, vc id 100, group id 0, vc label 16, status 0, mtu 1500
```

The highlighted sections of Example 14-19 show the use of the IP Layer 2 transport VC type with a value of 11, both for the LDP label mapping received and sent.

Because you are using a switched Frame Relay DLCI attachment circuit created by means of the **connect** command, you can utilize the **show connection** command to view connection status and information. The **show connection** command is also a troubleshooting tool (see Example 14-20).

Example 14-20. *Using the* **connection** *Command*

```
SanFran#show connection

ID   Name            Segment 1               Segment 2               State
=============================================================================
1    fr-vlan         Se5/0 100               10.0.0.203 100          UP

SanFran#show connection name fr-vlan

FR/Pseudo-Wire Connection: 1 - fr-vlan
  Status    - UP
  Segment 1 - Serial5/0 DLCI 100
    Segment status: UP
    Line status: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
  Segment 2 - 10.0.0.203 100
    Segment status: UP
    Requested AC state: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
  Interworking - ip
SanFran#
```

Without keywords, you can see that **show connection** command displays a summary of connections with their respective state. You can also see that a connection has two segments: segment 1 and segment 2. Identifying a specific connection by connection name or ID gives the connection details. In particular, it lists the status of each segment, including the segment status, line or attachment circuit status, PVC status, and NNI status. The PVC status refers to the local DLCI status, and the NNI status refers to the status of the DLCI as learned through NNI from the CE device.

The detailed version of the **show connection** command also shows that the IW type is IP between segment 1, which is the local Frame Relay DLCI, and segment 2, which is the pseudowire plus the remote endpoint attachment circuit of the pseudowire connection identified by the remote peer IP address and VC ID.

In the earlier section titled "Interworking MTU Considerations," you learned that because IW pseudowires use diverse interfaces, they are more prone toward MTU mismatches between attachment circuits. In a FR-VLAN IW case, the Frame Relay attachment circuit might be located in a POS interface with a default MTU of 4470, and the VLAN might reside in an Ethernet subinterface with a default MTU of 1500. To solve this problem, you might be tempted to consider changing the MTU in the POS interface to match the 1500 bytes. However, that would affect every DLCI on that interface, including the IW DLCIs that have a remote attachment circuit in an ATM interface with a default MTU of 4470. Example 14-21 presents the optimal solution for this problem, achieved by modifying the MTU per DLCI under connect mode.

Example 14-21. *Changing the MTU per FR DLCI*

```
SanFran(config)#connect POS_to_Ethernet POS 8/0 200 l2transport
SanFran(config-fr-pw-switching)#mtu 1500
SanFran(config-fr-pw-switching)#xconnect 10.0.0.203 200 encapsulation mpls
SanFran(config-fr-pw-switching)#end
SanFran#
```

Note

Specifying the MTU under the **connect** configuration mode is Frame Relay specific, because the **xconnect** command is not applied to a subinterface for Frame Relay DLCI attachment circuits. In other attachment circuit types such as Ethernet or ATM, the MTU can be changed under the subinterface where the **xconnect** command is applied and affect a single pseudowire.

To finalize this case study, capture an IP IW AToM packet in the C-HDLC link between the SanFran PE router and the Denver P router using the debug command **debug mpls l2transport packet data**. See Example 14-22, including an inline decode.

Example 14-22. *Capturing and Decoding an IP IW Packet*

```
22:43:38: ATOM imposition: out Se10/0, size 116, EXP 0x0, seq 0, control word 0x0
22:43:38: 0F 00 88 47 00 01 00 FF 00 01 31 02 00 00 00 00
          ^^ ^^ ^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^ ^^^^^^^^^^^
          |  |  |     top_shim    VC_Label    Ctrl-word
          |  |  |     Label=16    Label=19
          |  |  |     S=0         S=1
          |  |  |     TTL=255     TTL=2
          |  |  +-etype = IPv4
          |  +-Control
          +-Address = Unicast Frame

22:43:38: 45 00 00 64 00 13 00 00 FF 01 00 32 C0 A8 1D 01
          ^^^ ...
          Begins IP Packet

22:43:38: C0 A8 1D 02 08 00 0B B7 00 03 00 04 00 00 00 00
22:43:38: 04 E0 6D AC AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD
22:43:38: ATOM disposition: in Se10/0, size 100, seq 0, control word 0x0
22:43:38: 45 00 00 64 00 13 00 00 FF 01 00 32 C0 A8 1D 02
          ^^^ ...
          Begins IP Packet


22:43:38: C0 A8 1D 01 00 00 13 B7 00 03 00 04 00 00 00 00
22:43:38: 04 E0 6D AC AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD
22:43:38: AB CD AB CD
```

Note

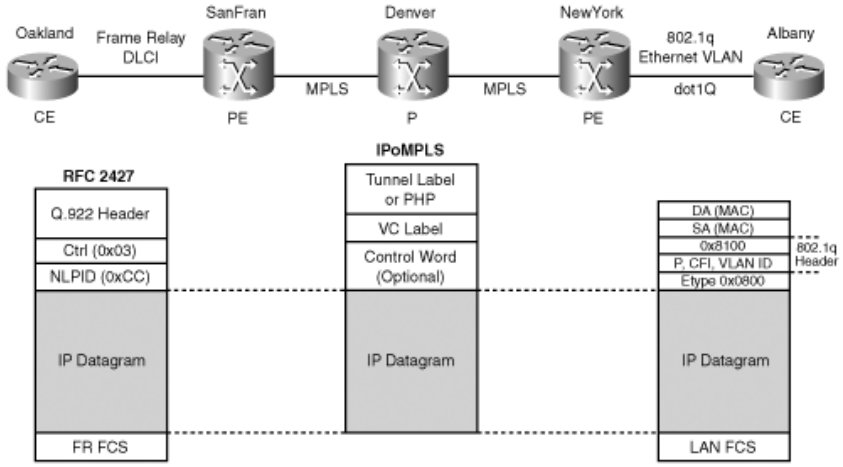
Note that in Example 14-22, the offline hand decoding of the packets is shown in bold font.

From Example 14-22, you can clearly see that after the control word, raw IP is transported. Remember from previous chapters that at disposition, only AToM Payload or SDU is displayed, whereas at imposition, the complete AToM packet is dumped.

You can see in Figure 14-10 how the encapsulation changes as the packet traverses from the Oakland CE as a Frame Relay encapsulated IP datagram (RFC 2427), through the AToM network to the Albany CE as a tagged-Ethernet encapsulated IP datagram and vice versa.

Figure 14-10. ***Frame Relay DLCI to VLAN AToM Routed IW Encapsulation Details***

[View full size image]



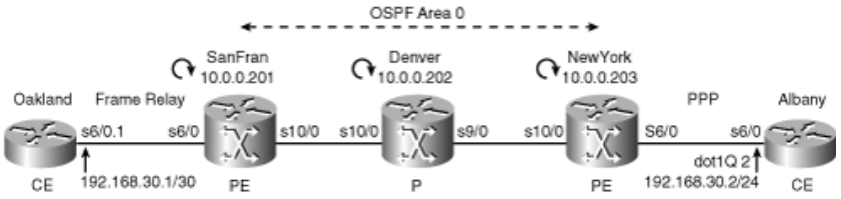
You can see that only the raw IP datagram is transported over the AToM pseudowire. The PE routers remove the complete Layer 2 encapsulation at imposition and re-create it at disposition.

Case Study 14-5: Frame Relay-to-PPP Using L2TPv3

In this case study, you learn the configuration and verification of IP IW between Frame Relay and PPP endpoints using L2TPv3 on the topology included in Figure 14-11.

Figure 14-11. ***Frame Relay-to-PPP Routed IW Using L2TPv3***

[View full size image]



This case study includes specific details on address resolution in IP IW both in Frame Relay DLCI and PPP port attachment circuits. These details apply to both AToM and L2TPv3 pseudowires. Example 14-23 shows the configuration for the SanFran PE side.

Example 14-23. *Frame Relay Side Configuration for IP IW Using L2TPv3*

```
!
hostname SanFran
!
frame-relay switching
!
pseudowire-class fr-ppp-l2tpv3
 encapsulation l2tpv3
 interworking ip
 ip local interface Loopback0
!
interface Serial6/0
 no ip address
 encapsulation frame-relay
 frame-relay intf-type dce
!
connect fr-ppp Serial6/0 60 l2transport
 xconnect 10.0.0.203 60 pw-class fr-ppp-l2tpv3
 !
!
```

The routed IW behavior is configured explicitly with the **interworking ip** command. As usual with Frame Relay DLCI attachment circuits, this configuration uses the **connect** command and the cross-connect inside the fr-pw-switching configuration mode.

Example 14-24 shows the configuration in the NewYork side.

Example 14-24. *PPP Side Configuration for IP IW Using L2TPv3*

```
!
hostname NewYork
!
pseudowire-class fr-ppp-l2tpv3
 encapsulation l2tpv3
 interworking ip
 ip local interface Loopback0
!
interface Serial6/0
 no ip address
 encapsulation ppp
 ppp ipcp address proxy 192.168.30.1
 xconnect 10.0.0.201 60 pw-class fr-ppp-l2tpv3
!
```

The configuration in Example 14-24 is similar to a normal L2TPv3 session configuration, with the addition of the **interworking ip** directive. However, an additional command exists under the PPP interface. The command **ppp ipcp address proxy** specifies the remote CE's IP address, which is the IP address of the Oakland CE. This is necessary because ARP mediation does not take place.

In the like-to-like scenario with PPP pseudowires, PPP negotiations including IPCP are between the two CE devices. In IP IW, Layer 2 from the CE is terminated in the PE. Therefore, in the case of a PPP attachment circuit, IPCP as specified in RFC 1332 needs to be negotiated between the PE and CE because PPP is terminated at the PE. IPCP negotiation includes the exchange of IP addresses, but the NewYork PE has no knowledge of Oakland's IP address. The Oakland CE is the IP peer to the Albany CE. You need to manually assign Oakland's IP address in the NewYork PE so that it can be included in

IPCP packets in the IPCP IP-Address Configuration option (type number 3). In this case, the PE device acts as an IPCP address proxy for the remote CE.

IPCP has a PPP DLL protocol number of 0x8021 and terminates on the PE. Only IP packets that have a PPP DLL protocol number of 0x0021 are transported over the pseudowire.

When the PE performs address resolution with the local CE, you can achieve the same result without configuring the PE device by using the command **peer default ip address** in the local CE, indicating the remote CE's IP address.

The next two examples show the CE router configuration. Example 14-25 starts with the Oakland configuration.

Example 14-25. *Frame Relay CE Configuration*

```
!
hostname Oakland
!
interface Serial6/0
 no ip address
 encapsulation frame-relay
!
interface Serial6/0.60 multipoint
 ip address 192.168.30.1 255.255.255.252
 no ip directed-broadcast
 frame-relay map ip 192.168.30.2 60 broadcast
!
```

You can see from Example 14-25 that you are now using a multipoint subinterface. This is to show the difference in configuration between using point-to-point versus multipoint subinterfaces. You do not need the **frame-relay map** configuration in a point-to-point subinterface, because the IP address and mask already define the connected prefix. The main interface has multipoint behavior, and you need to configure a **frame-relay map** (or inverse ARP when possible) for DLCIs in the main interface.

Example 14-25 shows the usage of the command **frame-relay map** instead of **frame-relay interface-dlci**. The **frame-relay map** command creates a DLCI but also maps it to a next-hop network protocol address. The example shows DLCI 60 mapped to the IP address 192.168.30.2 (the remote CE's IP on the PPP side) and specifies that broadcast packets such as routing protocol updates are to be sent over the DLCI.

Example 14-26 shows the configuration at the PPP-speaking Albany CE.

Example 14-26. *PPP CE Configuration*

```
!
hostname Albany
!
interface Serial6/0
 ip address 192.168.30.2 255.255.255.252
 encapsulation ppp
!
```

This configuration is the same as previous PPP-CE configurations.

Next, verify functionality and connectivity. Start the checks in the SanFran PE, as shown in Example 14-27.

Example 14-27. *L2TPv3 IP IW Pseudowire Verification*

```
SanFran#show l2tun session interworking vcid 60
 Session Information Total tunnels 1 sessions 2
 Tunnel control packets dropped due to failed digest 0


LocID       TunID       Peer-address    Type IWrk Username, Intf/
                                                  Vcid, Circuit
11756       51283       10.0.0.203      FR IP     60, Se6/0:60


SanFran#
SanFran#show l2tun session all ip-addr 10.0.0.203 vcid 60

L2TP Session

Session id 11756 is up, tunnel id 51283
Call serial number is 3043600001
Remote tunnel name is NewYork
  Internet address is 10.0.0.203
  Session is L2TP signalled
  Session state is established, time since change 21:54:06
    5 Packets sent, 5 received
    500 Bytes sent, 500 received
    Receive packets dropped:
      out-of-order:             0
      total:                    0
    Send packets dropped:
      exceeded session MTU:     0
      total:                    0
  Session vcid is 60
  Session Layer 2 circuit, type is Frame Relay, name is Serial6/0:60
  Circuit state is UP
  L2TP VC type is IP, interworking type is IP
    Remote session id is 9875, remote tunnel id 15753
  DF bit off, ToS reflect disabled, ToS value 0, TTL value 255
  No session cookie information available
  FS cached header information:
    encap size = 24 bytes
    00000000 00000000 00000000 00000000
    00000000 00000000
  Sequencing is off
SanFran#
```

The command **show l2tun session** with the **interworking** keyword displays the fact that although the attachment circuit type is Frame Relay (and PPP on the NewYork end), the IW type is IP. The same command with the **all** keyword displays the details of the dynamic L2TPv3 pseudowire, including the following:

- The dynamic session is L2TP signaled.
- The session state is established.
- The circuit state is UP.

- The VC type is IP (0x000B), as is the IW type.

You can use the **show connection** command in the Frame Relay end (see Example 14-28).

Example 14-28. *Using the* show connection *Command*

```
SanFran#show connection name fr-ppp

FR/Pseudo-Wire Connection: 4 - fr-ppp
  Status     - UP
  Segment 1 - Serial6/0 DLCI 60
    Segment status: UP
    Line status: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
  Segment 2 - 10.0.0.203 60
    Segment status: UP
    Requested AC state: UP
    PVC status: ACTIVE
    NNI PVC status: ACTIVE
  Interworking - ip
SanFran#
```

After you check the PE devices, do not forget that the goal is to provide CE-CE IP connectivity using disparate Layer 2 access technologies; therefore, check the CE devices. Example 14-29 shows some captures from the Oakland CE.

Example 14-29. *Frame Relay CE Verification*

```
Oakland#ping 192.168.30.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.30.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/28/36 ms
Oakland#show frame-relay map
Serial5/0.100 (up): point-to-point dlci, dlci 100(0x64,0x1840), broadcast, IETF
          status defined, active
Serial6/0.60 (up): ip 192.168.30.2 dlci 60(0x3C,0xCC0), static,
               broadcast,
               CISCO, status defined, active

Oakland#
```

Example 14-29 shows that IP connectivity exists between CE devices using ping. It also shows the output of the **show frame-relay map** command. You can see that DLCI 60 in interface Serial 6/0.60, which uses Frame Relay Cisco encapsulation, has a static map to 192.168.30.2 as configured, is active, and allows broadcasts. You can compare it to the map created in Case Study 14-4 using Frame Relay. In that case, a map is defined automatically for point-to-point subinterfaces, because the router can infer the mapping given the IP address and the subnet mask.

Note

The command **show frame-relay map** includes two hexadecimal numbers between parentheses beside the DLCI. The first number is the DLCI in hexadecimal representation.

The second number is the 2-byte Q.922 header with the BECN, FECN, and DE bits zeroed out.

Next, analyze the PPP interface at the Oakland CE (see Example 14-30).

Example 14-30. *PPP-CE Verification*

```
Albany#show interfaces s6/0
Serial6/0 is up, line protocol is up
  Hardware is M4T
  Internet address is 192.168.30.2/30
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation PPP, loopback not set
  Keepalive set (10 sec)
  LCP Open
  Open: IPCP
  Last input 00:00:02, output 00:00:02, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
!Output omitted for brevity
Albany#
Albany#show ip route connected | include Serial6/0
C       192.168.30.0/24 is directly connected, Serial6/0
C       192.168.30.1/32 is directly connected, Serial6/0
Albany#
```

You can use the command **show interfaces** to see that Link Control Protocol (LCP) and IPCP are in an open state, which means they have been negotiated successfully, and that the /32 connected route to Oakland's IP address was installed through IPCP.

In the routed IW case, only IPv4 is transported, and Layer 2 terminates in the PE device. From Chapters 8, "WAN Protocols over MPLS Case Studies," and 12, "WAN Protocols over L2TPv3 Case Studies," you learned that in the like-to-like cases, PPP does not run in the PE device. It goes into a closed state when the **xconnect** command is entered. That is to say, in the like-to-like case, PE devices do not participate in PPP negotiation.

You can verify whether PPP runs in the PE by enabling **debug ppp negotiation** in the NewYork PE (see Example 14-31).

Example 14-31. *PPP Running at the PPP-PE Verification*

```
NewYork#
*Jun 10 01:42:41.394: %LINK-3-UPDOWN: Interface Serial6/0, changed state to up
*Jun 10 01:42:41.394: Se6/0 PPP: Treating connection as a dedicated line
*Jun 10 01:42:41.394: Se6/0 PPP: Phase is ESTABLISHING, Active Open
*Jun 10 01:42:41.394: Se6/0 LCP: O CONFREQ [Closed] id 226 len 10
*Jun 10 01:42:41.394: Se6/0 LCP:    MagicNumber 0xC0A64078 (0x0506C0A
*Jun 10 01:42:41.418: Se6/0 LCP: I CONFREQ [REQsent] id 6 len 10
*Jun 10 01:42:41.418: Se6/0 LCP:    MagicNumber 0xC0A60E24 (0x0506C0A60E24)
*Jun 10 01:42:41.418: Se6/0 LCP: O CONFACK [REQsent] id 6 len 10
*Jun 10 01:42:41.418: Se6/0 LCP:    MagicNumber 0xC0A60E24 (0x0506C0A60E24)
*Jun 10 01:42:41.418: Se6/0 LCP: I CONFACK [ACKsent] id 226 len 10
```

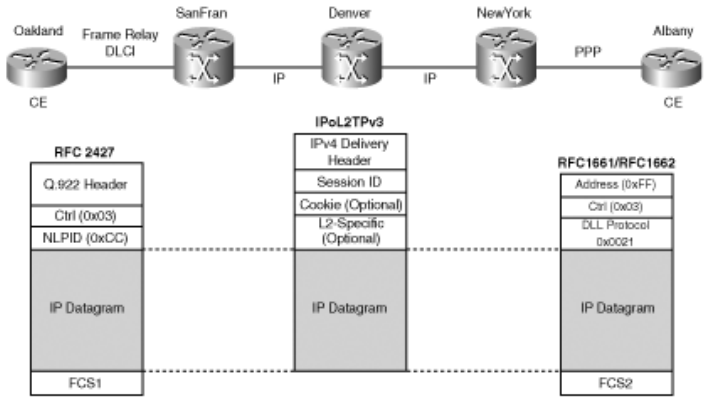
```
*Jun 10 01:42:41.418: Se6/0 LCP:    MagicNumber 0xC0A64078 (0x0506C0A64078)
*Jun 10 01:42:41.418: Se6/0 LCP: State is Open
*Jun 10 01:42:41.418: Se6/0 PPP: XCONNECT has gated NCP starts
*Jun 10 01:42:41.418: Se6/0 PPP: Phase is UP
*Jun 10 01:42:41.418: Se6/0 PPP: XCONNECT is preventing NCP starts
*Jun 10 01:42:41.438: Se6/0 PPP XCONNECT request to START IPCP using 0.0.0.0
*Jun 10 01:42:41.438: Se6/0 IPCP: O CONFREQ [Closed] id 8 len 10
*Jun 10 01:42:41.438: Se6/0 IPCP:    Address 192.168.30.1 (0x0306C0A81E01)
*Jun 10 01:42:41.470: Se6/0 IPCP: I CONFACK [REQsent] id 8 len 10
*Jun 10 01:42:41.470: Se6/0 IPCP:    Address 192.168.30.1 (0x0306C0A81E01)
*Jun 10 01:42:42.454: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial6/0,
changed state to up
*Jun 10 01:42:43.454: Se6/0 IPCP: TIMEout: State ACKrcvd
*Jun 10 01:42:43.454: Se6/0 IPCP: O CONFREQ [ACKrcvd] id 9 len 10
*Jun 10 01:42:43.454: Se6/0 IPCP:    Address 192.168.30.1 (0x0306C0A81E01)
*Jun 10 01:42:43.454: Se6/0 IPCP: I CONFREQ [REQsent] id 7 len 10
*Jun 10 01:42:43.454: Se6/0 IPCP:    Address 192.168.30.2 (0x0306C0A81E02)
*Jun 10 01:42:43.454: Se6/0 IPCP: O CONFACK [REQsent] id 7 len 10
*Jun 10 01:42:43.454: Se6/0 IPCP:    Address 192.168.30.2 (0x0306C0A81E02)
*Jun 10 01:42:43.470: Se6/0 IPCP: I CONFACK [ACKsent] id 9 len 10
*Jun 10 01:42:43.470: Se6/0 IPCP:    Address 192.168.30.1 (0x0306C0A81E01)
*Jun 10 01:42:43.470: Se6/0 IPCP: State is Open
NewYork#
NewYork#show interfaces serial 6/0
Serial6/0 is up, line protocol is up
  Hardware is M4T
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation PPP, loopback not set
  Keepalive set (10 sec)
  LCP Open
  Open: IPCP
  Last input 00:00:00, output 00:00:00, output hang never
!Output omitted for brevity
NewYork#
```

You can see that LCP and IPCP are terminated at the PE router, IPv4 is transported, and everything else is dropped. Example 14-31 shows that all the debug information before and including the timestamp of Jun 10 01:42:41.418 is LCP negotiation that succeeds with the LCP state being open. The debug information after and including the timestamp of Jun 10 01:42:41.438 pertains to the only NCP that is IPCP negotiation and concludes with the open state for IPCP. You can also use the command **show interface** in the PE device and see that in contrast to the like-to-like case, LCP and IPCP are now open at the PE attachment circuit.

Regarding the data plane encapsulation details, you can see in Figure 14-12 how the encapsulation changes as the packet traverses from the Oakland CE as a Frame Relayencapsulated IP datagram (RFC 2427), through the IP service provider network over L2TPv3, to the Albany CE as a PPP-encapsulated IP datagram (RFC 1661) and vice versa.

Figure 14-12. ***Frame Relay DLCI-to-PPP L2TPv3 Routed IW Encapsulation Details***

[View full size image]



You can see that only the raw IP datagram is transported over the L2TPv3 pseudowire, and the PE routers remove the complete Layer 2 encapsulation at imposition and re-create it at disposition. Non-IP datagrams coming from the CE device are dropped at the PE.

Case Study 14-6: IP L2-Transport MTU Considerations

This section presents the important and recurring topic of MTU as it pertains to IP IW, both in the AToM and L2TPv3 cases. IP IW is the most efficient from an overhead perspective. It holds the best-case scenario in regards to MTU adjustments because only raw IP is transported, and Layer 2 overhead does not exist. For IP IW, the Layer 2 from the CE is terminated at the PE and not transported; therefore, no additional overhead is present.

Table 14-2 summarizes the different overheads added in the PE router to an IP packet received from the CE device. Table 14-2 lists the overheads for both AToM and L2TPv3, including the overhead definition and the actual overhead value from Case Studies 14-4 and 14-5, respectively.

Table 14-2. *MTU Considerations for IP Layer 2 Transport*

PSN	PSN Tunnel Overhead	Demultiplexer Overhead	Layer 2 Specific	Total
AToM (Case Study 14-4)	MPLS Tunnel header	MPLS VC header	Control word	**12 bytes**
	4 bytes	4 bytes	4 bytes	
L2TPv3 (Case Study 14-5)	IP header without options	Session ID + cookie	L2-Specific Sublayer	**24 bytes**
	20 bytes	4 bytes + 0 bytes	0 bytes	

From Table 14-2, you can see that the IP IW AToM Case Study 14-4 has 12 bytes of overhead, whereas the IP IW L2TPv3 Case Study 14-5 has 24 bytes of overhead (because of the cookie absence and because sequencing is disabled).

Knowing these overheads, you can calculate the largest IP packet that can be sent from the CE and make it unfragmented, where all interfaces have a default MTU of 1500 bytes:

- **AToM** 1500 bytes 12 bytes = 1488 bytes
- **L2TPv3** 1500 bytes 24 bytes = 1476 bytes

To prove the preceding calculations, perform the following experiment:

- For AToM, send extended ping from the Oakland CE while setting the don't fragment (DF) bit in the IP header and using "Sweep range of sizes" and verbose output.
- For L2TPv3, configure the L2TPv3 session to set the DF bit in the IP header, adding **ip dfbit set** in the **fr-ppp-l2tpv3** pseudowire class, as explained in Chapter 13, "Advanced L2TPv3 Case Studies." In this case, the extended ping from the Oakland CE does not need to set the DF bit, because the DF bit that will be used in the cloud is the one in the L2TPv3 IPv4 delivery header.

See Example 14-32 for the results of these tests.

Example 14-32. *MTU Considerations with IP IW*

```
Oakland#
! Now test the AToM IP interworking case study 14-4
Oakland#ping
Protocol [ip]:
Target IP address: 192.168.29.2
Repeat count [5]: 1
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Source address or interface:
Type of service [0]:
Set DF bit in IP header? [no]: y
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: v
Loose, Strict, Record, Timestamp, Verbose[V]:
Sweep range of sizes [n]: y
Sweep min size [36]: 1480
Sweep max size [18024]: 1490
Sweep interval [1]:
Type escape sequence to abort.
Sending 11, [1480..1490]-byte ICMP Echos to 192.168.29.2, timeout is 1 seconds:
Reply to request 0 (96 ms) (size 1480)
Reply to request 1 (24 ms) (size 1481)
Reply to request 2 (32 ms) (size 1482)
Reply to request 3 (20 ms) (size 1483)
Reply to request 4 (28 ms) (size 1484)
Reply to request 5 (36 ms) (size 1485)
Reply to request 6 (28 ms) (size 1486)
Reply to request 7 (48 ms) (size 1487)
Reply to request 8 (32 ms) (size 1488)
Request 9 timed out (size 1489)
Request 10 timed out (size 1490)
Success rate is 81 percent (9/11), round-trip min/avg/max = 20/38/96 ms
```

```
Oakland#
Oakland#
Oakland#
Oakland#
! Now test the L2TPv3 IP interworking case study 14-5
Oakland#ping
Protocol [ip]:
Target IP address: 192.168.30.2
Repeat count [5]: 1
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface:
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: v
Loose, Strict, Record, Timestamp, Verbose[V]:
Sweep range of sizes [n]: y
Sweep min size [36]: 1470
Sweep max size [18024]: 1480
Sweep interval [1]:
Type escape sequence to abort.
Sending 11, [1470..1480]-byte ICMP Echos to 192.168.30.2, timeout is 2 seconds:
Reply to request 0 (20 ms) (size 1470)
Reply to request 1 (20 ms) (size 1471)
Reply to request 2 (32 ms) (size 1472)
Reply to request 3 (20 ms) (size 1473)
Reply to request 4 (24 ms) (size 1474)
Reply to request 5 (28 ms) (size 1475)
Reply to request 6 (36 ms) (size 1476)
Request 7 timed out (size 1477)
Request 8 timed out (size 1478)
Request 9 timed out (size 1479)
Request 10 timed out (size 1480)
Success rate is 63 percent (7/11), round-trip min/avg/max = 20/25/36 ms
Oakland#
```

You can conclude that the calculation is correct. From Example 14-32, you learn that in the AToM case, the largest IP packet that makes it through is 1488 bytes because of the 12-byte overhead. In contrast, in the L2TPv3 case, the largest IP packet that makes it through is 1476 bytes because of the 24-byte overhead.

By knowing the IP MTU limitation and calculation with IP IW, you can tune the core MTU based on the largest expected IP datagram from the CE device.

Case Study 14-7: Frame Relay-to-ATM Interworking Best Practices

In this last IW case study, you will learn some best practices about routed IW between Frame Relay and ATM. First, understand that IP IW pseudowire between Frame Relay and ATM VCs is not FRF.8, "Frame Relay/ATM PVC Service Interworking Implementation Agreement," (SIW) in which an IW function (IWF) translates between RFC 2427 encapsulated Frame Relay and RFC 2684 encapsulated AAL5 with a null SSCP. Only IP packets are transported over the IP IW pseudowire.

In contrast to FRF.8 SIW, which supports address resolution translation, inverse ARP is not supported in IP IW pseudowires; therefore, the ATM and Frame Relay CEs need to be configured on point-to-point subinterfaces or use static maps if they are on multipoint subinterfaces.

In the ATM attachment circuit, only AAL5 SDU VC mode is supported, and the ATM PVC encapsulation can either be AAL5SNAP or AAL5MUX (in which no translation is required because LLC/SNAP is nonexistent, and only one protocol is carried). Either AAL5SNAP or AAL5MUX needs to be configured in both the CE and PE devices under the PVC configuration mode. The encapsulations of AAL0 or AAL5 are not valid for IP IW, because AAL5 terminates at the PE device, and the PE device needs to know the AAL5 encapsulation type. However, in contrast to the CE configuration, when using AAL5MUX encapsulation in the PE Layer 2 PVC, you do not need to specify **ip** as the protocol carried.

A typical configuration for a CE that has multipoint subinterfaces is included in Example 14-33.

Example 14-33. *CE Configuration with Multipoint ATM Subinterfaces*

```
!
hostname Oakland
!
interface ATM3/0.27 multipoint
 ip address 192.168.31.1 255.255.255.0
 pvc 0/270
  protocol ip 192.168.31.2 broadcast
  encapsulation aal5snap
 !
!
interface ATM3/0.28 multipoint
 ip address 192.168.32.1 255.255.255.0
 pvc 0/271
  protocol ip 192.168.32.2 broadcast
  encapsulation aal5mux ip
 !
!
```

Note that both AAL5SNAP and AAL5MUX IP PVC encapsulations are supported. The protocol to VC mapping is achieved with the PVC mode command **protocol**. Example 14-34 shows how to verify ATM protocol to VC mappings.

Example 14-34. *Verifying ATM Maps*

```
Oakland#show atm map
Map list ATM3/0.27pvc10E : PERMANENT
ip 192.168.31.2 maps to VC 4, VPI 0, VCI 270, ATM3/0.27
        , broadcast

Map list ATM3/0.28pvc10F : PERMANENT
ip 192.168.32.2 maps to VC 9, VPI 0, VCI 271, ATM3/0.28
        , broadcast, aal5mux

Oakland #
```

You can see that IP addresses map to ATM VCs in a permanent way, meaning manually or statically configured. The broadcast keyword indicates pseudobroadcasting.

Layer 2 Local Switching

The local switching feature is another building block of the Cisco Unified VPN suite. Layer 2 local switching allows you to switch Layer 2 frames between two different attachment circuits on the same PE. The following permutations are supported, some of which do not require IW because the attachment circuit technologies are the same, whereas others do because they use different attachment circuit technologies:

- Interfaces of the same type:

  ATM-to-ATM

  Frame Relay-to-Frame Relay

  Ethernet-to-Ethernet/VLAN-to-VLAN

- Interfaces of different types:

  ATM-to-Ethernet/VLAN

  ATM-to-Frame Relay

  Frame Relay-to-Ethernet/VLAN

Local switching is not a pseudowire technology by the rigorous definition because a signaling protocol (such as LDP or L2TPv3) is not involved. However, local switching is a useful tool in the Layer 2 VPN solutions.

The main building block of the local switching configuration is the **connect** command, which allows you to create locally switched cross connections.

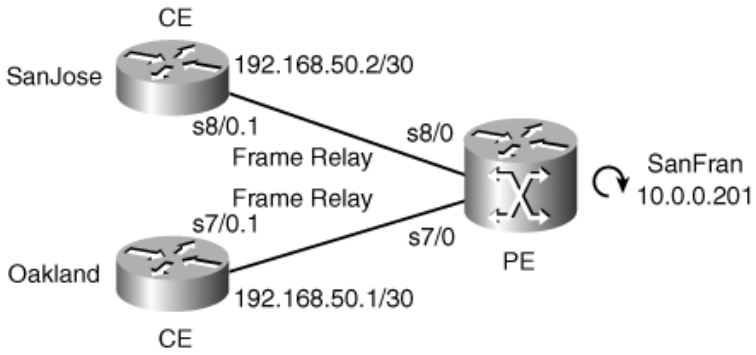
This section covers local switching between interfaces of the same type, and a later section details local switching between interfaces of different types. (See the section titled "Layer 2 local switching with interworking.") In this section, you learn the configuration and verification for the following local switching case studies:

- Case Study 14-8: Frame Relay-to-Frame Relay Local Switching
- Case Study 14-9: ATM-to-ATM Local Switching
- Case Study 14-10: Ethernet-to-Ethernet Local Switching

Case Study 14-8: Frame Relay-to-Frame Relay Local Switching

Frame Relay-to-Frame Relay local switching is a feature that was introduced before the development of the complete Layer 2 VPN local switching suite and was welcomed as part of that. This Frame Relay-to-Frame Relay local switching case study uses the topology shown in Figure 14-13, introducing the SanJose node.

Figure 14-13. ***Frame Relay-to-Frame Relay Local Switching Topology***



Example 14-35 shows the configuration for the SanFran PE.

Example 14-35. *Frame Relay-to-Frame Relay Local Switching*

```
!
hostname SanFran
!
frame-relay switching
!
interface Serial7/0
 no ip address
 encapsulation frame-relay
 frame-relay interface-dlci 70 switched
 frame-relay intf-type dce
!
interface Serial8/0
 no ip address
 encapsulation frame-relay
 frame-relay intf-type dce
!
connect fr_local_sw Serial7/0 70 Serial8/0 80
 !
!
```

From Example 14-35, you can see that the global configuration for **frame-relay switching** is a requirement. You can also see a switched Frame Relay interface DLCI configured in interface Serial 7/0 but not for interface Serial 8/0. Configuring the switched DLCI is an optional step. If the switched DLCI is not created in the interface, it is created with the **connect** command.

The heart of the configuration is in the **connect** command, by which you create the connection named **fr_local_sw**, which ties up DLCI 70 in Serial 7/0 to DLCI 80 in Serial 8/0. All the commands and tools that were covered in this and previous chapters regarding Frame Relay LMI, DLCI, maps, and connections are applicable to this case. For completeness, Example 14-36 includes the output of the command **show connection**.

Example 14-36. show connection *Command in Local Switching*

```
SanFran#show connection name fr_local_sw
3    fr_local_sw     Se7/0 70             Se8/0 80            UP
SanFran#
```

With Frame Relay attachment circuits, you can use the debug command **debug frame-relay pseudowire** to troubleshoot problems.

Case Study 14-9: ATM-to-ATM Local Switching

The case of ATM-to-ATM local switching is quite similar to Frame Relay-to-Frame Relay local switching. However, specific considerations are necessary, the most important ones related to the PVC encapsulation. Only two types of PVC encapsulation are supported in ATM-to-ATM local switching:

- **AAL5** Using **encapsulation aal5**
- **SCR** Single Cell Relay VC mode using **encapsulation aal0**

When you are using Single Cell Relay in some platforms, the virtual path identifier and virtual channel identifier (VPI/VCI) pair must match in both endpoints of the local switched connection.

When you are using AAL5, VPI/VCI values do not need to match in both endpoints. However, if you are transporting OAM cells over the local switched connection, the VPI/VCI must match because OAM cells are transported as cells, and you have the same limitation stated for single cell relay (SCR).

Besides the local switching of ATM PVCs, some platforms support the local switching of ATM permanent virtual paths (PVP) and packed cell relay (PCR) for local switching of ATM PVCs and ATM PVPs. In addition, local switching of ATM PVCs and ATM PVPs in the same port is supported. The configuration is analogous to this case study, using the same ATM interface for both connection endpoints.

Example 14-37 shows a sample configuration for PVC ATM-to-ATM local switching.

Example 14-37. *ATM-to-ATM Local Switching Configuration*

```
!
hostname SanFran
!
interface ATM1/0
 pvc 0/100 l2transport
  encapsulation aal5
!
interface ATM2/0
 pvc 0/200 l2transport
  encapsulation aal5
!
connect aal5_local_sw atm 1/0 0/100 atm 2/0 0/200
```

Notice that Example 14-37 shows cross-connecting PVCs with different VPI/VCI values, and the configuration uses encapsulation AAL5. Observe that the ATM PVCs are created using the **l2transport** keyword to identify the PVC as switched and not as terminated. The **connection** configuration is analogous to the Frame Relay-to-Frame Relay example, using VPI/VCI instead of DLCI.

Another similarity with Frame Relay-to-Frame Relay local switching is that you can enter the **connect** command without previously configuring the ATM PVCs, in which case the PVCs are created

automatically in the respective interfaces that are specified (see Example 14-38).

Example 14-38. *ATM-to-ATM Local Switching with Automatic PVCs*

```
SanFran(config)#connect atm_local ATM 4/0 0/40 ATM 3/0 0/40
SanFran(config-connection)#

SanFran#show connection

ID   Name            Segment 1               Segment 2               State
================================================================================
1    atm_local       AT4/0 CELL 0/40         AT3/0 CELL 0/40         UP

SanFran#
SanFran#show connection id 1
Connection: 1 - atm_local
 Current State: UP
 Segment 1: ATM4/0 CELL 0/40 u
 Segment 2: ATM3/0 CELL 0/40 up

SanFran#
```

You can see from Example 14-38 that only the **connect** command is entered, and it automatically creates the local switched connection. Note from the **show connection** output that the default encapsulation for automatically created l2transport PVCs is AAL0that is, VC Cell Relay mode. Example 14-39 shows how to check for automatically created l2transport PVCs and their default encapsulation.

Example 14-39. *Displaying Automatic PVCs*

```
SanFran#show atm vc | include 40| VC
               VCD /                                          Peak Avg/Min Burst
Interface      Name          VPI    VCI Type    Encaps      Kbps    Kbps Cells Sts
3/0            11              0     40 PVC-A   AAL0      155000     N/A       UP
4/0            19              0     40 PVC-A   AAL0      149760     N/A       UP
SanFran#
SanFram#show atm pvc 0/40 | begin ATM4/0
ATM4/0: VCD: 19, VPI: 0, VCI: 40
UBR, PeakRate: 149760
AAL0-Cell Relay, etype:0x10, Flags: 0x10000C2D, VCmode: 0x0
OAM Cell Emulation: not configured
Interworking Method: like to like
Remote Circuit Status = No Alarm, Alarm Type = None
InBytes: 208963575912, OutBytes: 1088149400
Cell-packing Disabled
OAM cells received: 1
F5 InEndloop: 0, F5 InSegloop: 0, F5 InAIS: 1, F5 InRDI: 0
OAM cells sent: 1
F5 OutEndloop: 0, F5 OutSegloop: 0, F5 OutAIS: 1, F5 OutRDI: 0
OAM cell drops: 0
Auto-created by Connection Manager
Status: UP
W2N-7.11-c7206VXR-A#m
```

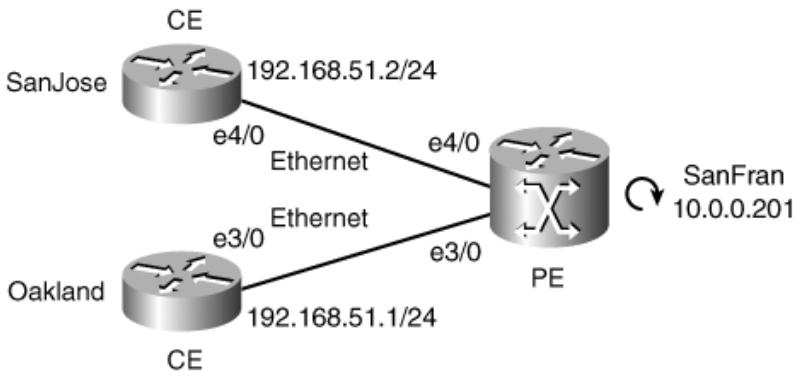
Displaying the ATM PVC summary, you can see that the PVC type is PVC-A, which stands for PVC automatically created. You can also see that the encapsulation is AAL0 single-cell relay by default, which works fine for like-to-like ATM-to-ATM connections but does not work for IW ones. A detailed list of valid and default encapsulations for IW and local switching ATM VC attachment circuits is included in "Case Study 14-12: ATM Attachment Circuits and Local Switching."

With ATM PVC attachment circuits, you can use the debug command **debug atm l2transport** as a troubleshooting tool.

Case Study 14-10: Ethernet-to-Ethernet Local Switching

This case study shows Ethernet-to-Ethernet port mode local switching. The same configuration and verification presented here is analogous to Ethernet dot1Q VLAN-to-VLAN local switching using subinterfaces instead of the main interface. This topology is included in Figure 14-14.

Figure 14-14. *Ethernet-to-Ethernet Local Switching*



The actual configuration required is equivalent to the previous ones using the **connect** command (see Example 14-40).

Example 14-40. *Ethernet-to-Ethernet Local Switching*

```
!
hostname SanFran
!
connect eth-eth Ethernet3/0 Ethernet4/0
 !
!
```

The configuration that is required at the PE is to activate the Ethernet interfaces with a **no shutdown** and issue the **connect** command. You can verify that the local switched connection is working (see Example 14-41).

Example 14-41. *Ethernet-to-Ethernet Local Switching Verification*

```
SanFran#show connection name eth-eth
Connection: 4 - eth-eth
 Current State: UP
 Segment 1: Ethernet3/0 up
 Segment 2: Ethernet4/0 up

SanFran#

SanJose#ping 192.168.51.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.51.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/28/36 ms
SanJose#
```

For the sake of argument, you can use the local switching feature without configuring IP addresses in the PE device, because no signaling is involved. An interface or a router that does not have an IP address does not process IP packets; therefore, it cannot process signaling messages carried over IP, as is the case with LDP and L2TPv3. This idea emphasizes the point that no signaling protocol is implicated in local switching.

You can witness this fact when enabling the debug command **debug acircuit event** to debug events that occur on the attachment circuits (see Example 14-42).

Example 14-42. *Debugging Attachment Circuit for a Local Switched Connection*

```
SanFran(config)#do debug acircuit event
Attachment Circuit events debugging is on
SanFran(config)#interface Ethernet 3/0
SanFran(config-if)#no shutdown
SanFran(config-if)#
00:28:44: ACLIB [0.0.0.0, 0]: SW AC interface UP for Ethernet interface Et3/0
00:28:44: ACLIB: Added circuit to retry queue, type 6, id 5, idb Et3/0
00:28:44: ACLIB [0.0.0.0, 0]: pthru_intf_handle_circuit_up() calling acmgr_circuit_up
00:28:44: ACLIB [0.0.0.0, 0]: Setting new AC state to Ac-Connecting
00:28:44: ACLIB: Update switching plane with circuit UP status
00:28:44: ACLIB [0.0.0.0, 0]: SW AC interface UP for Ethernet interface Et3/0
00:28:44: ACLIB [0.0.0.0, 0]: pthru_intf_handle_circuit_up() ignoring up event.
  Already connected or connecting.
00:28:44: ACLIB [0.0.0.0, 0]: pthru_intf_handle_circuit_up() ignoring up event.
  Already connected or connecting.
00:28:44: Et3/0 ACMGR: Receive <Circuit Up> msg
00:28:44: Et3/0 ACMGR: circuit up event, SIP state chg fsp up to connected, action
  is p2p up forwarded
00:28:44: ACLIB: pthru_intf_response hdl is 7F000012, response is 2
00:28:44: ACLIB [0.0.0.0, 0]: Setting new AC state to Ac-Connected
00:28:44: Et4/0 ACMGR: Receive <Remote Up Notification> msg
00:28:44: Et4/0 ACMGR: remote up event, FSP state chg fsp up to connected, action
  is respond forwarded
00:28:44: ACLIB: pthru_intf_response hdl is 41000016, response is 2
00:28:44: ACLIB [0.0.0.0, 0]: Setting new AC state to Ac-Connected
00:28:46: %LINK-3-UPDOWN: Interface Ethernet3/0, changed state to up
00:28:47: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet3/0, changed
  state to up
SanFran(config-if)#
```

You can see by inspecting [0.0.0.0, 0] that the IP address of the remote peer is displayed as 0.0.0.0, and the VC ID is shown as 0. This is because it is a local switching connection.

Note

A consequence of no pseudowire signaling protocol being involved in local switching cases is that MTU mismatches between the attachment circuits do not prevent the circuit from coming up. The downside is that a circuit not coming up might trigger you to revisit the MTU settings.

Layer 2 Local Switching with Interworking

This section brings together the two previous sections of IW and local switching to provide another building block of the Cisco Unified VPN suite: any-to-any local switching. The configuration aspect also uses the two core commands of each of the previous sections: the **connect** command and the **interworking** keyword.

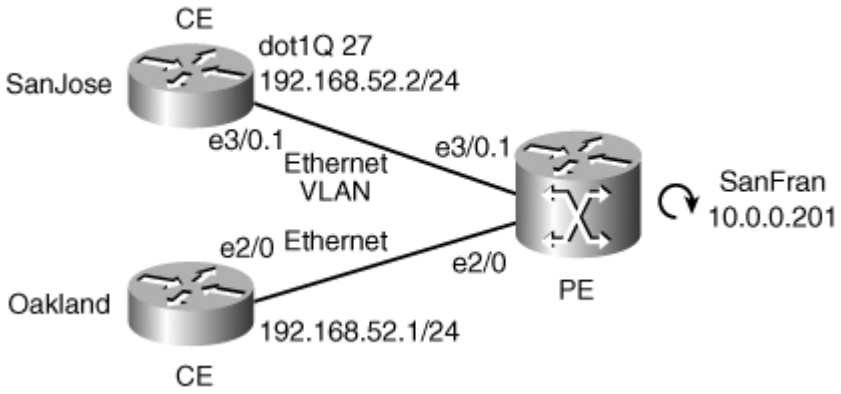
In this section, you learn the configuration and verification of the following case studies:

- Case Study 14-11: Ethernet-to-VLAN Local Switching
- Case Study 14-12: ATM Attachment Circuits and Local Switching

Case Study 14-11: Ethernet-to-VLAN Local Switching

This section discusses the mechanisms to implement local switching between Ethernet and Ethernet VLAN attachment circuits, using the topology shown in Figure 14-15.

Figure 14-15. *Ethernet-to-VLAN Local Switching*



The configuration that is required in the SanFran PE is shown in Example 14-43.

Example 14-43. *Configuring Ethernet-to-VLAN Local Switching*

```
!
hostname SanFran
!
```

```
interface Ethernet2/0
 no ip address
!
interface Ethernet3/0
 no ip address
!
interface Ethernet3/0.1
 encapsulation dot1Q 27
!
connect eth-vlan Ethernet2/0 Ethernet3/0.1 interworking ethernet
 !
!
```

You can see that this example uses the global command **connect** with diverse interfaces, and it prompts you for the IW type. The IW types are also IP or Ethernet.

You can perform connection verification from the SanFran PE and connectivity checking from the SanJose CE (see Example 14-44).

Example 14-44. *Verifying Ethernet-to-VLAN Local Switching*

```
SanFran#show connection name eth-vlan
Connection: 4 - eth-vlan
 Current State: UP
 Segment 1: Ethernet2/0 up
 Segment 2: Ethernet3/0.1 up
 Interworking Type: ethernet
SanFran#

SanJose#ping 192.168.52.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.52.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/29/36 ms
SanJose#
```

The **show connection** command shows the IW type as Ethernet.

Case Study 14-12: ATM Attachment Circuits and Local Switching

This section deals with Layer 2 VPN local switching with IW and goes over some details and ideas that are helpful when using ATM PVC attachment circuits.

When you are configuring ATM-to-Ethernet Local Switching Interworking (LSIW) or ATM-to-Ethernet-VLAN LSIW, both IP and Ethernet types of IW are supported. With IP IW, you can configure the ATM PVC for either AAL5SNAP (in which translation takes place) or AAL5MUX (without translation because the AAL5 packet begins with raw IP). On the other hand, when you are using Ethernet IW, only AAL5SNAP PVC encapsulation is supported.

Creating the ATM PVC is not required. When you enter the **connect** command without the existing PVC, the PVC-A is created automatically and assigned AAL5SNAP encapsulation. If, however, you create the ATM PVC, you need to specify the **l2transport** keyword to indicate that it is a switched PVC and not a terminating PVC.

When you are configuring ATM-to-Frame Relay IW, only IP IW is supported. The ATM PVC encapsulation can either be AAL5SNAP or AAL5NLPID.

Table 14-3 summarizes the PVC encapsulation options while using ATM local switching with and without IW.

Table 14-3. ***ATM PVC Encapsulation Usage for IW and Local Switching***

Segment 1 (Attachment Circuit)	**Segment 2 (Attachment Circuit)**	**Interworking (IW) Type**	**ATM PVC Encapsulations**
ATM PVC	ATM PVC	N/A	**aal0** (default)
			aal5
ATM PVC	Ethernet/VLAN	Ethernet	**aal5snap**
		IP	**aal5snap** (default)
			aal5mux
ATM PVC	Frame Relay DLCI	IP	**aal5snap**(default)
			aal5nlpid

Understanding Advanced Interworking and Local Switching

This final section covers two topics:

- **connect command** You learn about the different behaviors of this command.
- **Encapsulation** You learn detailed information about the encapsulations of IW and local switching.

connect Command

At this point, you know how to use the **connect** command in multiple contexts. You used it to create AToM and L2TPv3 pseudowire endpoints in Frame Relay DLCI attachment circuits to perform local switching and Frame Relay local switching. This section compares the different modes of this command using examples.

You have used the **connect** command in three different contexts and created different configuration modes. Example 14-45 shows the **connect** command that performs Frame Relay pseudowire switching.

Example 14-45. connect *Command and Frame Relay Pseudowire Switching*

```
SanFran(config)#connect fr-vlan Serial5/0 100 l2transport
SanFran(config-fr-pw-switching)#
```

When you use the **connect** command with the **l2transport** keyword, you are taken into config-fr-pw-switching configuration mode. Example 14-46 shows the **connect** command that performs local Frame Relay switching.

Example 14-46. connect *Command and Frame Relay Switching*

```
SanFran(config)#connect fr_local_sw serial 7/0 70 serial 8/0 80
SanFran(config-fr-switching)#
```

When you use the **connect** command to cross connect two local Frame Relay DLCIs, you are taken into config-fr-switching configuration mode. Example 14-47 shows the **connect** command that performs local cross connection between two attachment circuits.

Example 14-47. connect *Command for Local Switching Connections*

```
SanFran(config)#connect eth-eth ethernet 3/0 ethernet 4/0
SanFran(config-connection)#

SanFran(config)#connect eth-fr ethernet 3/0 serial 7/0 100 interworking ip
SanFran(config-connection)#

SanFran(config)#connect eth-eth ethernet 3/0.1 ethernet 4/0.1
SanFran(config-connection)#exit

SanFran(config)#connect atm_local ATM 4/0 0/40 ATM 3/0 0/40
SanFran(config-connection)#
```

Example 14-47 shows various cases used throughout this chapter. In the case of any-to-any (not like-to-like) attachment circuits, the IW option is presented. These different configuration submodes also present different commands that are applicable to the specific function that is being performed. (For example, the local switching submodes have no **xconnect** command.)

Encapsulation

This section introduces you to encapsulation details for some of the case studies presented in this chapter. To view the encapsulation details, use Subscriber Service Switch (SSS) exec commands. The connections are represented as attachment circuit session types to SSS. All the examples use the command **show sss circuits**, which provides the status, encapsulation length, and encapsulation hexadecimal dump for the circuits. The encapsulation that is presented as output of this command, also called rewrite, indicates data that is added to the packet.

This section's goal is that you obtain a better understanding of the underlying processes and protocols in the case studies. The following examples are presented:

- Encapsulation 1: Ethernet-to-VLAN Local Switching Ethernet IW
- Encapsulation 2: Frame Relay-to-VLAN IP IW Using AToM
- Encapsulation 3: VLAN-to-Ethernet Bridged IW Using L2TPv3
- Encapsulation 4: Frame Relay-to-PPP IP-IW Using L2TPv3

Encapsulation 1: Ethernet-to-VLAN Local Switching Ethernet IW

This scenario presents the local switching Case Study 14-10in the SanFran PE router between Ethernet 2/0 and VLAN 27 in Ethernet 3/0.1 (see Example 14-48).

Example 14-48. *SSS Circuit Encapsulation for Ethernet-to-VLAN Local Switching Ethernet-IW*

```
SanFran#show sss circuits

Current SSS Circuit Information: Total number of circuits 5

!Output omitted for brevity
Common Circuit ID 0             Serial Num 5          Switch ID 18796512
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  0
   Y  AES  4   8100001B
SanFran#
```

Focusing on the encapsulation, you know that in Ethernet IW, Ethernet frames are sent over the Layer 2 circuit. Therefore, the Ethernet endpoint has no specific encapsulation, as denoted in the encapsulation length of 0.

For the VLAN side, however, the encapsulation includes the 4-byte VLAN tag. In this example, the encapsulation is represented as 0x8100001B, from which you see the following:

- The 802.1q VLAN Ethertype of 0x8100
- The 802.1p CoS bits of 0 and CFI of 0
- The VLAN ID of 0x1B or 27 as configured

Encapsulation 2: Frame Relay-to-VLAN IP IW Using AToM

This scenario presents the AToM IP IW Case Study 14-4 from both the SanFran and New York PE routers. The scenario starts from the SanFran PE router, in which the local cross-connection is Frame Relay-to-AToM (see Example 14-49).

Example 14-49. *SSS Circuit Encapsulation for Frame Relay IP-IW Using AToM*

```
SanFran#show sss circuits

Current SSS Circuit Information: Total number of circuits 5

!Output omitted for brevity
Common Circuit ID 0             Serial Num 3          Switch ID 18796912
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  4   184103CC
   Y  AES  0
```

```
SanFran#
```

The **show sss circuits** command in the SanFran PE shows the encapsulations that are local to this PE router; it does not show the remote VLAN encapsulation. In IP IW, only raw IP packets are sent over the pseudowire; therefore, the rewrite includes the complete Layer 2 encapsulation.

You can see that the encapsulation for the Frame Relay endpoint is 4 bytes in length, is equal to 0x184103CC, and is made out of the following:

- The first 2 octets represent the Q.922 header:

  The leftmost 6 bits of the first octet are equal to 000110 and form the high-order DLCI; the leftmost 4 bits of the second octet are equal to 0100 and make up the low-order DLCI. Therefore, the 10-bit DLCI is expressed in binary 0001100100 or 100 in decimal, which is the DLCI that is configured in Case Study 14-4.

  The least significant bit of the second byte is set to 1, to indicate the lack of an extended address (EA).

- The third octet in the encapsulation is the control of 0x03.

- The last octet is the NLPID value of the IP Protocol that is equal to 0xCC.

Because IP is received over the pseudowire, after you append this 4-byte encapsulation and set the values of FECN, BECN, and DE, you send the packet over the Frame Relay attachment circuit. Toward the AToM side, the encapsulation is always shown as NULL.

The second part of this example presents the VLAN side of the AToM routed IW pseudowire from the New York PE, in which the local cross connection is VLAN to AToM (see Example 14-50).

Example 14-50. *SSS Circuit Encapsulation for VLAN IP-IW Using AToM*

```
New York#show sss circuits

Current SSS Circuit Information: Total number of circuits 4

[snip]
Common Circuit ID 0             Serial Num 3          Switch ID 18796912
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  18  FFFFFFFF FFFF000C CF552408 81000002 0800
   Y  AES  0
!Output omitted for brevity
```

```
New York#
```

To reiterate, the **show sss circuits** command in the New York PE illustrates the encapsulations that are local to this PE router and does not show the remote Frame Relay encapsulation. In IP IW, only raw IP packets are transmitted over the pseudowire; therefore, the VLAN side should show a complete 18-byte VLAN rewrite.

When you compare the VLAN encapsulation for both bridged and routed IW, you see the following:

- **Bridged** The encapsulation is only 4 bytes and includes the 802.1q header only. This is because an Ethernet frame is received over the pseudowire, and adding the 802.1q header creates a complete VLAN frame.
- **Routed** The encapsulation is 18 bytes and includes the complete Layer 2 encapsulation, including Ethernet II and 802.1q headers. This is because an IP datagram is received over the pseudowire, so appending the 18-byte encapsulation creates a complete VLAN frame.

You can see that the encapsulation length is 18 bytes and is composed of the following:

- The first 6 bytes are the destination MAC address where 0xFFFFFFFFFFFF is a broadcast Ethernet address, meaning that the PE has not yet learned the CE's MAC address. This MAC address is changed to the actual value after it is learned.
- The next 6 bytes are the source MAC address.
- The next 4 bytes are the VLAN tag, including the following:

  VLAN etype of 0x8100

  CoS and CFI of 0

  VLAN ID of 2 as configured

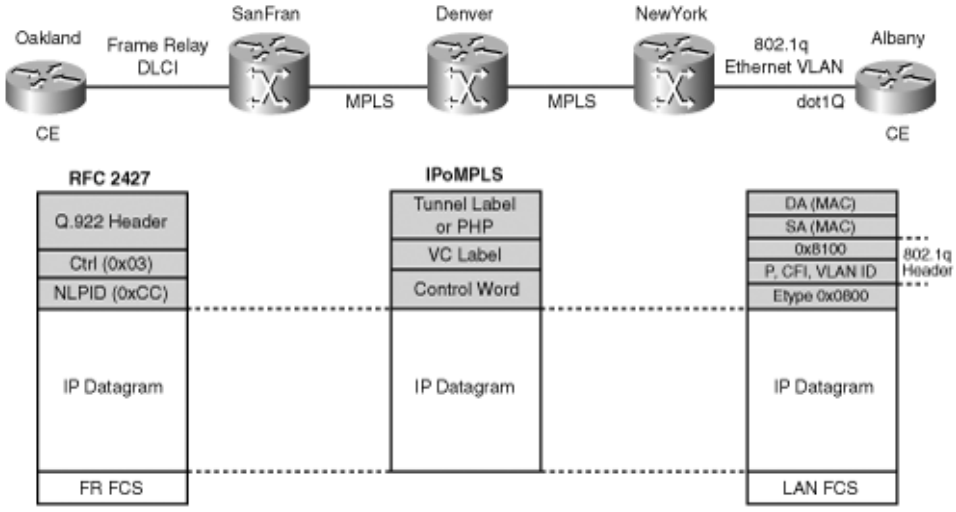
- The final two bytes are the IP Ethertype of 0x0800.

Prepending this encapsulation to an IP packet that is received over the pseudowire and setting CoS bits appropriately makes a valid frame to be sent out of the Ethernet 2/0.1 interface in the New York PE.

Figure 14-16 shows the encapsulation added in both SanFran and New York PEs for this scenario. The fields in gray represent the rewrite that has been described verbally.

Figure 14-16. ***Frame Relay DLCI-to-VLAN AToM Routed IW Encapsulation Details***

[View full size image]



Encapsulation 3: VLAN-to-Ethernet Bridged IW Using L2TPv3

This scenario presents the L2TPv3 Ethernet IW Case Study 14-2 from the VLAN attachment circuit in the New York PE router. The local cross-connection in the New York router is VLAN-to-L2TPv3 (see Example 14-51).

Example 14-51. *SSS Circuit Encapsulation for VLAN Ethernet-IW Using L2TPv3*

```
New York#show sss circuits

Current SSS Circuit Information: Total number of circuits 4

Common Circuit ID 0             Serial Num 2          Switch ID 18797112
------------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  4   81000002
   Y  AES  28  45000000 00000000 FF73A5F7 0A0000CB 0A0000C9
               000058FB 00000000

New York#
```

The local encapsulations to the New York PE router are VLAN toward the attachment circuit and L2TPv3 toward the PSN. The attachment circuit side is equivalent to previous encapsulation scenario 1. Because this is bridged IW and Ethernet frames are received over the pseudowire, no other encapsulation is needed.

However, the PSN side that uses L2TPv3 is new and shows an encapsulation length of 28 bytes, consisting of the following:

- 20 bytes of IPv4 header, including the following:

  Version 4 Header length 5 32-bit words

  Protocol 115 (0x73) for L2TPv3

  Source address 10.0.0.203

  Destination address 10.0.0.201

- 4 bytes of L2TPv3 Session Header (0x000058FB): 32-bit Session ID of 22779
- 4 bytes of L2-Specific Sublayer (sequencing was configured):

  Sequence bit clear

  Sequence number cleared

Note

You can also display the L2TPv3 encapsulation by using the **show adjacency detail** command in the pseudowire IP adjacency.

An Ethernet frame is encapsulated. After you complete the empty fields, such as DSCP in the IP Header and sequencing information in the L2-Specific Sublayer, you can send the L2TPv3 packet toward the PSN.

Encapsulation 4: Frame Relay-to-PPP IP-IW Using L2TPv3

This scenario presents the L2TPv3 Ethernet IW Case Study 14-5 from the SanFran and New York PE routers. The discussion starts from the SanFran PE router, in which the local cross connection is Frame Relay to L2TPv3 (see Example 14-52).

Example 14-52. *SSS Circuit Encapsulation for Frame Relay IP-IW Using L2TPv3*

```
SanFran#show sss circuits

Current SSS Circuit Information: Total number of circuits 5

!Output omitted for brevity
```

```
Common Circuit ID 0             Serial Num 4          Switch ID 18796712
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  4   0CC103CC
   Y  AES  24  45000000 00000000 FF73A5F7 0A0000C9 0A0000CB
               0000269D

SanFran#
```

The Frame Relay attachment circuit and L2TPv3 encapsulation in SanFran PE are similar to the previous examples, where you can see the following for the Frame Relay attachment circuit:

- The first 2 octets represent the Q.922 header: DLCI = 111100b = 60.
- The third octet in the encapsulation is the control of 0x03.
- The last octet is the NLPID value of IP Protocol, equal to 0xCC.

The Frame Relay encapsulation is a full Frame Relay header that, when appended to the received raw IP datagram, creates a complete Frame Relay frame carrying IP.

The L2TPv3 side shows an encapsulation length of 24 bytes, because sequencing is not enabled and no L2-Specific Sublayer exists:

- 20 bytes of IPv4 header, including the following:

  Version 4 Header Length 5 32-bit words

  Protocol 115 (0x73) for L2TPv3

  Source address 10.0.0.201

  Destination address 10.0.0.203

- 4 bytes of L2TPv3 Session Header (0x0000269D):

  32-bit Session ID of 9885

The second part of this scenario presents the PPP side of the L2TPv3 routed IW pseudowire from the New York PE, in which the local cross-connection is PPP-to-L2TPv3 (see Example 14-53).

Example 14-53. *SSS Circuit Encapsulation for PPP IP-IW Using L2TPv3*

```
New York#show sss circuits

Current SSS Circuit Information: Total number of circuits 4

!Output omitted for brevity
Common Circuit ID 0             Serial Num 4          Switch ID 18796712
------------------------------------------------------------------------
   Status  Encapsulation
   UP flg  len dump
   Y  AES  4   FF030021
   Y  AES  24  45000000 00000000 FF73A5F7 0A0000CB 0A0000C9
                  000058FA
!Output omitted for brevity

New York#
```

You can see in the New York PE that the encapsulation length for the PPP side is 4 bytes, comprising a full PPP header, and contains the following:

- Address of 0xFF
- Control of 0x03
- PPP DLL protocol number of 0x0021 for IPv4

Prepending this encapsulation to an IP packet that is received over the pseudowire creates a PPP frame to be sent out of the Serial 6/0 interface in the New York PE. The L2TPv3 side is equivalent to the analysis in the SanFran PE with mirror source and destination IP addresses and a different Session ID. The L2TPv3 encapsulation includes the following:

- 20 bytes of IPv4 header, including the following:

  Version 4 Header Length 5, 32-bit words

  Protocol 115 (0x73) for L2TPv3

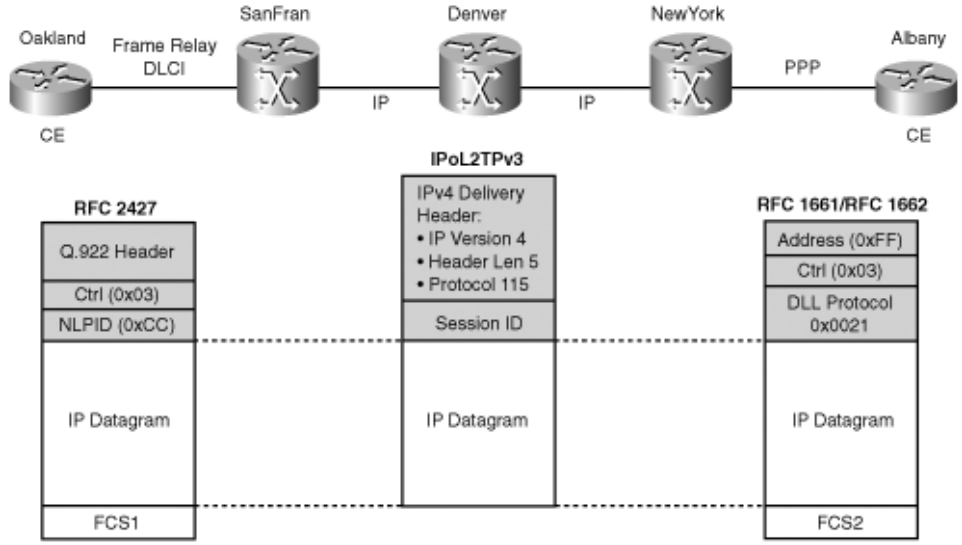
  Source address 10.0.0.203, destination address 10.0.0.201

- 4 bytes of L2TPv3 Session Header (0x000058FA): 32-bit Session ID of 22778

Figure 14-17 shows the encapsulations added in both the SanFran and New York PEs for this scenario in addition to the L2TPv3 encapsulation added to the carried PDU. The fields in gray represent the rewrite that has been described verbally.

Figure 14-17. *Frame Relay DLCI-to-PPP L2TPv3 Routed IW Encapsulation Details*

[View full size image]



Summary

In this chapter, you learned the theory, operation, and configuration of Layer 2 bridged and routed IW pseudowires through the use of a technology overview and case studies. You then learned about like-to-like and any-to-any (IW) local switching. This chapter concluded with advanced topics in Layer 2 IW and local switching, including encapsulation details related to some of the chapter's case studies.

Chapter 15. Virtual Private LAN Service

This chapter covers the following topics:

- Understanding VPLS fundamentals
- VPLS deployment models
- VPLS configuration case studies

The Layer 2 VPN architectures that have been discussed in this book so far share one common characteristic: They provide only point-to-point connectivity.

Virtual Private LAN Service (VPLS), on the other hand, is a Layer 2 VPN architecture that was built for multipoint connectivity and has broadcast capability.

Like other Layer 2 VPN architectures, customer edge (CE) routers are connected through provider edge (PE) routers and pseudowires, but they no longer have the point-to-point peering relationship. Instead, VPLS enables CE routers to communicate with one another as if they were attached to a common LAN.

Interestingly, pseudowires that are used in VPLS are the same type of pseudowire as that used in the point-to-point Layer 2 VPN architectures. The point-to-point versus multipoint behavior is determined by the data packet forwarding behaviors of a given Layer 2 VPN architecture. This also implies that the pseudowire encapsulation is orthogonal to the functionality that VPLS provides. In theory, both Multiprotocol Label Switching (MPLS) and L2TP pseudowires satisfy the forwarding requirements of VPLS. In reality, the rapid growth of MPLS network deployment drives the momentum behind the MPLS-based VPLS in terms of standardization activities and product implementations. Therefore, this chapter focuses on VPLS concepts and examples that involve MPLS pseudowires.

This chapter begins with an overview of VPLS that describes the service definitions, signaling protocols, and more importantly the concept of virtual switch and its data forwarding characteristics. Then it describes VPLS deployment issues of network topology, complexity, and scalability. VPLS configuration case studies conclude the chapter.

Understanding VPLS Fundamentals

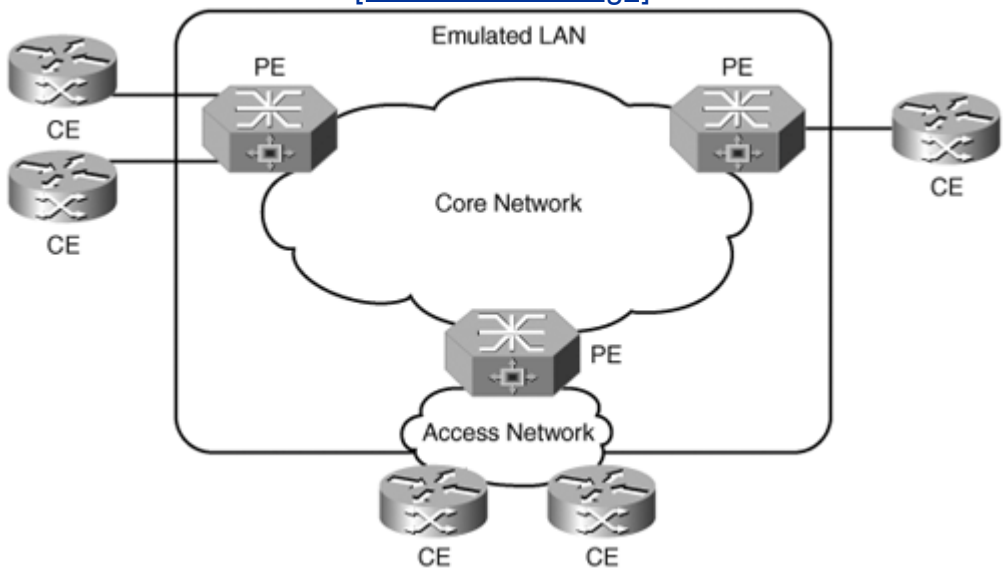
VPLS is generating considerable interest with enterprises and service providers because it offers the Transparent LAN Service (TLS). Previously, TLS was available only in Ethernet-switched networks and had limited geographic reach. VPLS not only overcomes the distance limitation of Ethernet-switched networks, but it also enables new value-added features and services by leveraging advanced MPLS features, such as traffic engineering.

The inherent broadcast nature of Ethernet makes it easy for networked devices to discover one another. VPLS extends that broadcast capability to the reach that is possible only with a WAN infrastructure. In VPLS, end users perceive that the network devices are connected directly to a common LAN segment, which is in fact an emulated LAN created by VPLS, also known as a *VPLS domain*.

Figure 15-1 shows the VPLS network reference model, where PE devices act as virtual switches such that CE routers of a particular VPLS domain appear to be on a single bridged Ethernet network. CE routers can connect to PE routers either through direct links or through an access network.

Figure 15-1. *VPLS Network Reference Model*

[View full size image]



As a multipoint architecture, VPLS allows a single physical or logical CE-PE link to be used for transmitting Ethernet packets to multiple remote CE routers. Therefore, fewer connections are required to get full connectivity among customer sites. To provide the same level of connectivity with a point-to-point architecture, more

connections are required. The "VPLS Configuration Case Studies" section later in this chapter examines the magnitude of such savings.

What makes VPLS particularly attractive to service providers is its plug-and-play nature. After you provision PE routers with multipoint connectivity for VPLS customers, you need to reconfigure only the directly attached PE router when adding, removing, or relocating a CE router within a Layer 2 VPN. In contrast, you must reconfigure every peering PE router if the Layer 2 VPN is based on a point-to-point architecture. With VPLS, packets are no longer forwarded based on the one-to-one mapping between an attachment circuit and a pseudowire on a PE router. Rather, a PE router uses a Layer 2 forwarding table to determine the outgoing paths based on the destination MAC addresses. A Layer 2 forwarding table is populated dynamically with MAC addresses and next-hop interfaces through the learning process. The next few sections explain the types of service that VPLS provides, protocol signaling, and packet forwarding behaviors.

Service Definitions

VPLS offers two types of service:

- TLS
- Ethernet Virtual Connection Service (EVCS)

The services are differentiated by the way that MAC addresses are learned and the way that bridging protocol data units (BPDU) are processed. TLS performs unqualified learning, in which all customer VLANs of a Layer 2 VPN are treated as if they were in the same broadcast domain.

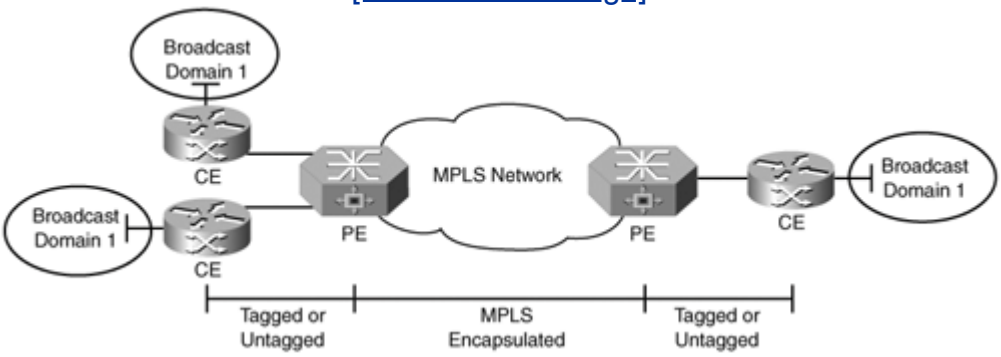
Source MAC addresses are learned and forwarding entries are populated in the same Layer 2 forwarding table regardless of whether they are tagged or untagged. This means that MAC addresses have to be unique among all customer VLANs. Overlapping MAC addresses can cause confusion in the Layer 2 forwarding table and result in loss of customer packets.

Besides tagged and untagged Ethernet packets, a PE router that provides TLS also forwards BPDUs that it receives from the CE-facing interface to other interfaces or pseudowires without processing. Such transparency in BPDU forwarding makes the CE routers perceive that they are connected directly through an Ethernet hub instead of through a series of virtual switches, which you learn more about in the next section. Virtual switches, like real physical switches, terminate and process BPDUs by default.

Figure 15-2 illustrates an example of TLS.

Figure 15-2. *TLS Example*

[View full size image]

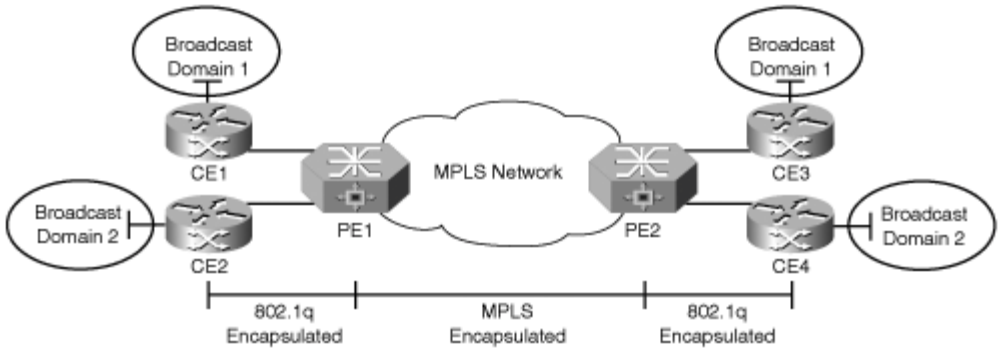


For customers who want to keep a separate broadcast domain for each VLAN, EVCS is a more appropriate choice. In EVCS, the outer VLAN tag on the Ethernet packet differentiates one customer VLAN instance from another. Each VLAN has its own MAC address space, which allows qualified learning. In qualified learning, MAC addresses of different VLANs might overlap with one another, and each VLAN has a separate Layer 2 forwarding table.

EVCS keeps the broadcast domain on a per-VLAN basis and does not extend the spanning tree across the MPLS network. BPDU packets from CE routers are dropped or processed at PE routers. In such cases, CE routers do not see each other directly in the spanning tree. Figure 15-3 shows an example of EVCS. Suppose that a VPLS customer has four sites that form two separate broadcast domains. CE1 and CE2 connect to the same PE router but belong to different broadcast domains. IEEE 802.1q VLAN encapsulation is used between the CE routers and PE router to separate the traffic of different broadcast domains.

Figure 15-3. *EVCS Example*

[View full size image]



Note

If it is necessary to exchange Layer 3 traffic among different broadcast domains, PE routers can provide Layer 3 connectivity using the Integrated Routing and Bridging (IRB) capability. Layer 2 traffic cannot be forwarded from one broadcast domain to another.

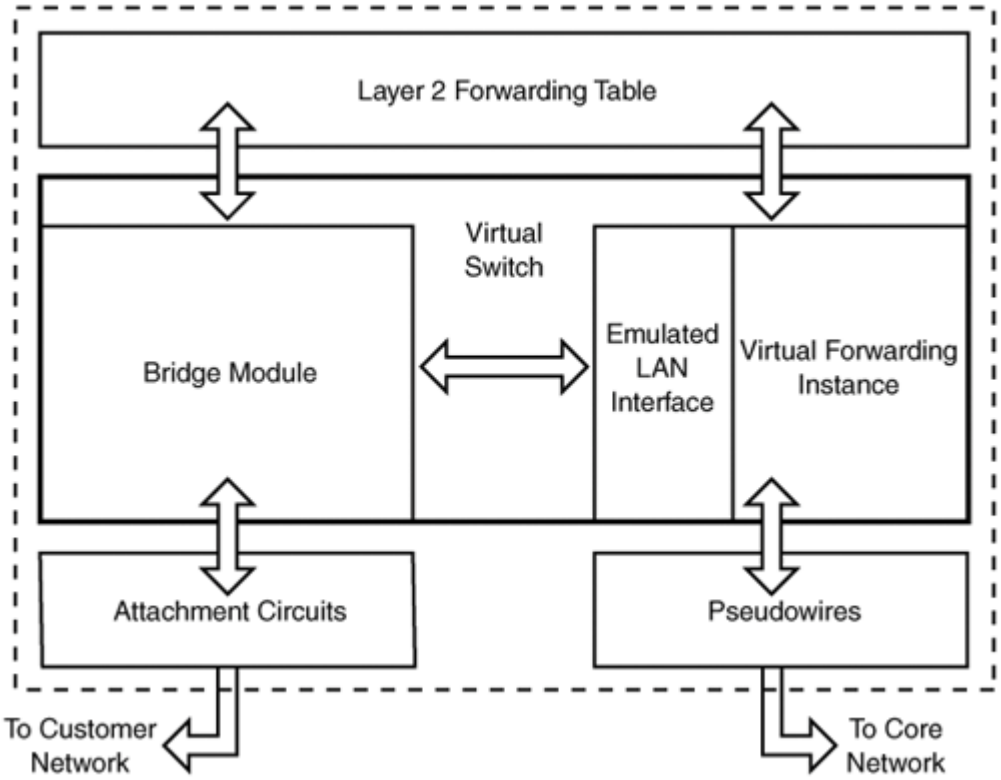
Virtual Switch

Each service that is defined in the previous section is offered by a virtual switch inside a PE router. When provisioned to support multiple VPLS customers, the PE router effectively is partitioned into multiple virtual switches. A given PE router has at most one virtual switch for every VPLS domain.

A virtual switch consists of a bridge module, an emulated LAN interface, and a virtual forwarding instance (VFI), as shown in Figure 15-4. CE routers are connected to the bridge module through attachment circuits. Each bridge also has one emulated LAN interface that connects to the VFI.

Figure 15-4. *Virtual Switch*

[View full size image]



The bridge module in a virtual switch has the equivalent role of that in a physical Ethernet switch. It makes no distinction between the emulated LAN interface and any physical LAN interface in terms of bridging functions, such as MAC address learning and aging, and packet flooding. Besides the bridge module maintaining a forwarding table that maps MAC addresses to attachment circuits, it can run spanning-tree protocols on them.

A VFI has similar functionality to a bridge but performs bridging operations on pseudowires instead of attachment circuits. It maintains a forwarding table that maps MAC addresses to pseudowires. The forwarding table is populated through the MAC address learning process based on packets it receives on pseudowires. It never learns the MAC addresses of the packets it receives on attachment circuits.

> **Note**
>
> In some literature, virtual switching instance (VSI) is used in place of VFI. They are inter-changeable terms.

Conceptually, the forwarding table of a bridge module and that of a VFI are different entities. In practice, VPLS implementations can choose either to create separate tables for each or combine them into a single table. Because the actual form of the data structures does not affect VPLS operations, this chapter assumes a single Layer 2 forwarding table for every VPLS domain for the sake of simplicity.

VPLS Forwarding and Flooding

In a point-to-point Layer 2 VPN architecture, an attachment circuit and a pseudowire have a one-to-one mapping. Packets that are received from a CE router are forwarded to only one pseudowire based on the attachment circuit that packets arrive on.

In VPLS, attachment circuits and pseudowires are connected through a virtual switch. Because more than one attachment circuit and pseudowire can attach to the same virtual switch, the correlation between attachment circuits and pseudowires becomes many-to-many.

The forwarding decision is made in two stages in VPLS, as follows:

1. A packet is mapped to a virtual switch and its corresponding Layer 2 forwarding table based on the attachment circuit or pseudowire that the packet arrives on.

2. The virtual switch looks up the forwarding table using the destination MAC address and determines the proper forwarding action. Unless a policy is in

place to block this particular packet, the forwarding action can be either broadcast or unicast.

Initially, the Layer 2 forwarding table does not include dynamically learned entries.

When packets arrive on attachment circuits or pseudowires, MAC address learning takes place as part of the forwarding process. If the source MAC address is not present in the forwarding table, it is added to the table with the arriving attachment circuit or pseudowire as the outgoing interface. Also, an aging timer is started for the new forwarding entry. If the source MAC address is already in the forwarding table, no new entry is created, and the aging timer is refreshed so that an active MAC address is not flushed out prematurely.

Unlike Layer 3 forwarding, in which packets are dropped if no forwarding entry matches the Layer 3 destination address, VPLS employs a flooding process when the virtual switch receives a packet that has an unknown destination MAC address. The flooding process also applies to multicast and broadcast packets. Resembling that of a real bridge, VPLS flooding also has its distinct nuances that pertain to pseudowires. Depending on whether the packet receives on an attachment circuit or a pseudowire and whether Layer 2 split horizon is enabled, flooding can take different courses of action.

When a packet with an unknown destination MAC address arrives on an attachment circuit, it is flooded to all other attachment circuits and all pseudowires that are bound to the virtual switch. When Layer 2 split horizon is enabled on a pseudowire, packets that arrive on this pseudowire are flooded to all attachment circuits, but not a pseudowire. When Layer 2 split horizon is disabled, packets are flooded to all other pseudowires and all attachment circuits that are bound to the virtual switch.

Layer 2 split horizon is a loop prevention mechanism specifically devised for forwarding VPLS traffic over pseudowires. When virtual switches of a VPLS domain are interconnected by a fully meshed network of pseudowires, you must enable Layer 2 split horizon on all pseudowires to prevent forwarding loops. Service providers typically do not run spanning-tree protocols over pseudowires. The "VPLS Deployment Models" section later in this chapter examines the correlations between split horizon and different deployment models.

VPLS Signaling

Chapter 2, "Pseudowire Emulation Framework and Standards," explained two MPLS pseudowire emulation frameworks, known as draft-martini and draft-kompella, in the context of point-to-point Layer 2 VPN architectures. Each architecture defines a signaling protocol to establish and manage pseudowires. Just as the networking community debates which signal protocol is superior in the point-to-point Layer 2 VPN architectures, a similar debate arose when VPLS debuted as a multipoint Layer 2 VPN architecture. The two competing proposals that were made to the networking community are based on the same ideas as in draft-martini and draft-kompella, where one is based on Label Distribution Protocol (LDP) and the other is based on Border Gateway Protocol (BGP). Despite being applied to a new architecture like VPLS, the fundamental property of each protocol still remains.

The LDP-based VPLS solution, like its point-to-point counterpart, receives much wider acceptance in terms of vendor implementation and network deployment. The VPLS solution that Cisco IOS offered is an LDP-based solution.

To comprehend the details of the BGP-based VPLS solution, refer to the relevant documents of the Layer 2 VPN working group at the IETF web site (http://www.ietf.org). This chapter focuses on the LDP-based VPLS solution and its deployment scenarios. Note that both solutions have the same data forwarding specifications despite the difference in signaling.

The procedure of setting up pseudowires for VPLS is quite similar to that for point-to-point Ethernet over MPLS (EoMPLS). First, a targeted LDP session is created between each pair of PE routers that participate in a given VPLS domain. In a full-mesh deployment model, *N* * (*N* 1) / 2 LDP sessions need to be established, where *N* is the number of PE routers participating in VPLS. These LDP sessions can be shared among different VPLS domains. In other words, you can use a single LDP session between a pair of PE routers to establish pseudowires for all VPLS domains that are provisioned on the PE routers.

After LDP sessions are established among participating PE routers, the next step is to create pseudowires to interconnect the virtual switches. Again, in a full-mesh deployment model, each VPLS domain requires *N* * (*N* 1) / 2 pseudowires throughout the network, where *N* is the number of virtual switches.

Note

As part of the deployment planning, you should calculate the number of signaling sessions and pseudowires to be established throughout the network to measure the scale of VPLS deployment. One benefit of VPLS architecture is that the numbers stay the same if more customer sites need to be attached to existing virtual switches.

The protocol messages and encodings that are used for VPLS signaling are identical to those that are described in Chapter 6, "Understanding Any Transport over MPLS." The Pseudowire Type field in the Pseudowire ID FEC element is Ethernet for VPLS, which is the same as for the point-to-point EoMPLS.

The Pseudowire ID field in the Pseudowire ID FEC element serves the binding of two remotely located entities, such as attachment circuits or VFIs. For point-to-point Layer 2 VPNs, the Pseudowire ID only needs to be unique between a pair of PE routers. For VPLS Layer 2 VPNs, each VPLS domain is identified by a globally unique VPN ID, which means that you have to provision VFIs of the same VPLS domain with the same VPN ID on all participating PE routers. When you are setting up pseudowires for a VPLS domain, instead of VC IDs or Pseudowire IDs of individual point-to-point pseudowires, you have the VPN ID in the Pseudowire ID field. Because point-to-point and VPLS Layer 2 VPNs share the same Pseudowire ID

space, you need to ensure that VPN IDs that are used in VPLS do not overlap with Pseudowire IDs that are used in point-to-point Layer 2 VPNs.

VPLS Deployment Models

Compared to point-to-point Layer 2 VPNs, deploying multipoint Layer 2 VPNs that are built on the VPLS architecture is far more complicated. One of the reasons is that a point-to-point Layer 2 VPN resembles a traditional Frame Relay-or ATM-based network in areas such as topologies and forwarding characteristics. For this reason, it is natural to overlap a point-to-point Layer 2 VPN on top of the WAN-based core network.

LAN services, as the name suggests, are designed for networks that are confined to a local or metropolitan area. One fundamental assumption that many LAN services make is that plenty of cheap bandwidth is available in the local or metro-area network (MAN). Many LAN services also rely on broadcasting and flooding to function properly. Despite the phenomenal growth in building high-speed network infrastructures, WAN bandwidth has always been one of the most expensive pieces in the overall network cost. Without carefully engineered VPLS deployment, new VPLS services will not be the only ones to suffer. Broadcast storms seen in a LAN environment can propagate to the multiservice backbone and affect other non-VPLS services. This section examines a few deployment issues, with considerations to loop-free forwarding, broadcast traffic, and scalability.

Basic Topologic Models

In a given VPLS domain, virtual switches are interconnected by pseudowires. The topology formed by these pseudowires plays a critical role in loop-free forwarding and contributes to the overall scalability and performance. A VPLS domain can have the following forms:

- Full mesh
- Hub and spoke
- Partial mesh

Full Mesh

The "Understanding VPLS Fundamentals" section briefly addressed the forwarding and signaling characteristics of a full-mesh VPLS topology, which is the most common deployment model today. In a full-mesh model, every virtual switch has exactly one pseudowire to every other virtual switch in the same VPLS domain. The loop-free forwarding is guaranteed by enabling Layer 2 split horizon on every pseudowire in this topology. Split horizon prevents packets that are received on one pseudowire from being sent to any other pseudowire. It applies to normal forwarding if the destination MAC address matches an entry in the forwarding table,

and it also applies to flooding if the packet has an unknown unicast, multicast, or broadcast destination MAC address.

A full-mesh model provides loop-free full connectivity among all virtual switches. It also eliminates the need to run spanning-tree protocols over the backbone, which saves valuable WAN bandwidth. However, such benefits come at the cost of other network resources. Remember from the previous section that the number of signaling sessions and pseudowires grows exponentially when the number of PE routers and virtual switches increases. Supporting numerous signaling sessions and pseudowires requires more bandwidth for exchanging protocol messages and more processing power on PE routers for signaling and packet forwarding. The forwarding performance also degrades when flooding has been performed on a large number pseudowires.

Hub and Spoke

In a hub-and-spoke model, exactly one PE router that is acting as a hub connects all other PE routers that act as spokes in a given VPLS domain. The virtual switch on a spoke PE router has exactly one pseudowire connecting to the virtual switch on the hub PE router. No pseudowire interconnects the virtual switches on spoke PE routers. A hub-and-spoke topology by definition is loop-free, so it does not need to enable spanning-tree protocols or split horizon on pseudowires. To provide Layer 2 connectivity among the virtual switches on spoke PE routers, the hub PE router must turn off split horizon on the pseudowires. When split horizon is disabled, you can forward or flood packets among different pseudowires at the hub PE router.

The simplicity of a hub-and-spoke model makes it an attractive choice for small VPLS deployment. Realize, though, that the hub PE router is a single point of failure. Because all traffic has to go through the hub PE router, the router requires ample processing power to relay and flood packets across pseudowires.

Partial Mesh

The most flexible topologic model is partial mesh. To guarantee loop-free forwarding in an arbitrary partial-mesh model, you need to run spanning-tree protocols on pseudowires throughout the backbone. Spanning-tree protocols are typically chatty and take a considerable amount of expensive WAN bandwidth. In addition, deploying spanning-tree protocols in a large-scale network is always a great challenge. Network design considerations such as root bridge selection, redundancy, and load balancing are highly complex, which means they are more vulnerable to configuration and operation mistakes. Service providers typically do not deploy a partial-mesh model because they want to avoid running spanning-tree protocols in the core network.

Hierarchical VPLS

Aiming at having the benefits of both basic topologic models while mitigating their problems, a hybrid between the full-mesh and hub-and-spoke models is now available, known as *hierarchical VPLS*. A hierarchical VPLS consists of a top tier and a bottom tier. Depending on the type of network that is deployed at the bottom tier, hierarchical VPLS comes in two forms:

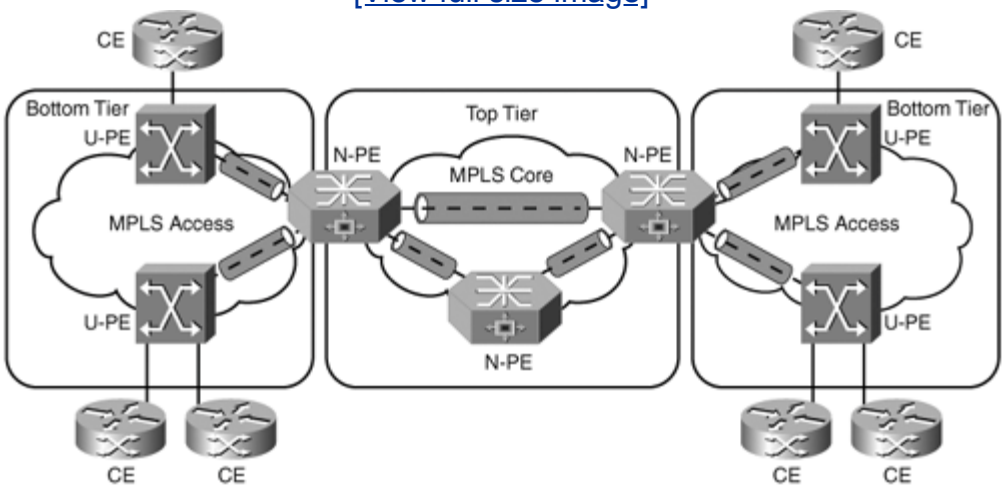
- Hierarchical VPLS with MPLS access network
- Hierarchical VPLS with QinQ access network

Hierarchical VPLS with MPLS Access Network

As shown in Figure 15-5, for a given VPLS domain, virtual switches in the top tier are fully meshed through pseudowires. Each virtual switch in the bottom tier has exactly one pseudowire that connects to a top-tier virtual switch, which is effectively a hub-and-spoke model. This form of hierarchical VPLS is known as *hierarchical VPLS with MPLS access*. PE routers in the top tier and bottom tier are also known as *network-facing PE (N-PE)* routers and *user-facing PE (U-PE)* routers, respectively. To ensure loop-free forwarding, an N-PE router must enable Layer 2 split horizon on all pseudowires that connect to other N-PE routers and disable split horizon on all pseudowires that connect to U-PE routers. On an N-PE router, packets are forwarded to other pseudowires only if they arrive on a pseudowire that connects a U-PE router. Packets that arrive on a pseudowire that connects an N-PE router can be forwarded to pseudowires that connect to U-PE routers only.

Figure 15-5. *Hierarchical VPLS with MPLS Access*

[View full size image]

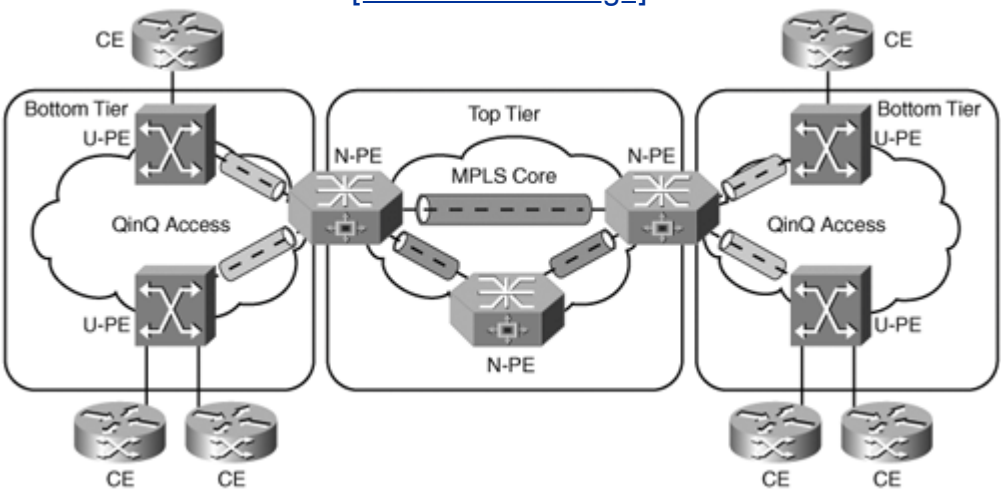


Hierarchical VPLS with QinQ Access Network

Hierarchical VPLS has an alternate form that uses Ethernet QinQ tunnels between U-PE and N-PE routers, as depicted in Figure 15-6. It is also known as *hierarchical VPLS with QinQ access*. Instead of a pseudowire, you can use an Ethernet QinQ tunnel between a U-PE router and an N-PE router. Despite the absence of pseudowires in the bottom tier, the overall bridging architecture is still based on two logically separated layers, where an N-PE router forwards packets to pseudowires that connect to other N-PE routers only if they arrive on QinQ tunnels that connect to U-PE routers. The hierarchical VPLS models significantly reduce the total number of signaling sessions and pseudowires; therefore, they improve network scalability and performance. The scalability benefit also surfaces when you add or relocate a PE router. If the object is an N-PE router, you need to reconfigure only other N-PE routers. If the object is a U-PE router, you need to reconfigure only the attached N-PE router.

Figure 15-6. *Hierarchical VPLS with QinQ Access*

[View full size image]



Like point-to-point Layer 2 VPN architectures, you can deploy VPLS in an inter-autonomous system (AS) or multidomain environment using a hierarchical model. In their simplest form, the peering VPLS PE routers of different administrative domains operate in such a fashion that each PE router treats itself as an N-PE router, and treats the peering PE as a U-PE router in the hierarchical model.

VPLS Redundancy

In the hierarchical VPLS model, an N-PE router can still be a single point of failure for attached U-PE routers. To solve this problem, each U-PE can connect to multiple N-PE routers through redundant pseudowires or QinQ tunnels. This method for providing redundancy is also known as *multihoming*. In this case, Layer 2 split

horizon alone is no longer sufficient for providing loop-free forwarding. You need to enable spanning-tree protocols between U-PE and N-PE routers.

In Metro Ethernet deployment, you can view each metro area as an island of which U-PE and N-PE routers are closely located and are connected through a LAN. You can view VPLS as a collection of islands interconnected by pseudowires. When you confine spanning-tree protocols within individual islands, each island becomes a separate spanning-tree domain of which the boundary stays within the LAN, and the core network does not suffer bridging protocolrelated problems such as bandwidth inefficiency, operation complexity, and other drawbacks.

When a U-PE router multihomes with N-PE routers, you must enable spanning-tree protocols on the U-PE router for all the pseudowires or QinQ tunnels that exist between the U-PE and N-PE routers. However, an N-PE router can choose whether to participate in spanning-tree protocols. If it does, it behaves like an Ethernet bridge that exchanges and processes BPDUs with U-PE and other N-PE routers of the same island. If it does not, it acts as an Ethernet hub that simply relays BPDUs without processing. In the next section, a case study shows how to achieve VPLS redundancy using multihoming.

VPLS Configuration Case Studies

The feature requirements for VPLS originated from service providers that were deploying Metro Ethernet services and wanted to extend the coverage beyond the boundary of a metro using their WAN infrastructure. As part of the Cisco Metro Ethernet service portfolio, the integrated VPLS solution in Cisco IOS fulfills such requirements.

This section describes how to configure VPLS on a Cisco router. The case studies that are commonly seen in Metro Ethernet deployment, which are by no means exhaustive, can help you further understand the Cisco VPLS solution. Configuration examples in this section are based on the Cisco 7600 series router. Refer to Cisco.com to obtain the information on the latest platform and hardware support.

Case Study 15-1: Basic Configuration

Before you configure VPLS, you need to ensure that IP routing and MPLS forwarding are configured properly and that the minimal Layer 2 VPN connectivity requirements are met:

- Every PE router has a loopback interface that is configured with an IP address and a /32 network mask. This address is used as the Router ID in LDP signaling for the PE router.
- PE routers have IP connectivity to each other, and the IP routing tables contain those host routes that were previously configured on the loopback interfaces. This ensures that you can establish the LDP sessions for pseudowire signaling. You can verify the reachability for the host routes by using the **show ip route** command.
- PE routers have MPLS label switched paths (LSPs) for those host routes. This ensures that MPLS encapsulated pseudowire packets are not sent to a black hole caused by a broken LSP. You can verify this by using the **show mpls forwarding-table** command.

The next few sections discuss the tasks involved for baseline VPLS configuration, demonstrate with a complete example, and verify the configuration results.

Configuring Attachment Circuit

Attachment circuits that are used in VPLS can be Layer 2 switch-port interfaces, Gigabit Ethernet interfaces on intelligent line cards, or other interfaces with bridged encapsulation. Because of the low cost and high port density, Layer 2 switchport interfaces are the most commonly deployed attachment circuits.

Before going into the configuration steps for Layer 2 switchport interface, it is necessary to explain the difference and the correlation between a service-delimiting VLAN tag and an internal VLAN tag. Chapter 6 introduced the concept of a service-delimiting VLAN tag. To recap, service providers use service-delimiting VLAN tags to identify different types of customer traffic. Because a service-delimiting VLAN tag usually has only local significance,

it is removed at the ingress PE router. The egress PE router might have a different service-delimiting VLAN tag, which is added to the packets that are sending to a CE router.

An internal VLAN tag identifies a bridge domain on a PE router. In the context of VPLS, it is the virtual switch. Conceptually, service-delimiting VLAN tags and internal VLAN tags are two independent entities. A bridge domain, represented by an internal VLAN tag, might have multiple attachment circuits, where each is provisioned with a different service-delimiting VLAN tag. Such independence allows service providers to offer multiple value-added services to a single VPLS customer using the same physical connection. Currently, this provisioning model is available only on interfaces of high-end intelligent access line cards.

Traffic from VPLS customers does not always have service-delimiting VLAN tags, such as untagged customer traffic. In addition, having an 802.1q VLAN tag in the packet does not automatically make it a service-delimiting VLAN tag. Later in this section, you will study the characteristics of a service-delimiting VLAN tag and an internal VLAN tag when used in different switchport modes of the Layer 2 switchport interface.

A Layer 2 switchport interface can operate in one of three mutually exclusive switchport modes. The following list recaps how each mode is used in normal bridging applications:

- **access** The interface sends and accepts untagged Ethernet packets only. Tagged Ethernet VLAN packets are dropped.
- **trunk** The interface sends and receives tagged Ethernet VLAN packets and native VLAN packets.
- **dot1q-tunnel** Any packet, tagged or untagged, is forwarded through a QinQ tunnel. A QinQ tunnel is identified by the access VLAN tag that is configured on the Layer 2 switchport interface. The access VLAN tag is added to the packet at the ingress tunnel interface and removed at the egress tunnel interface, which means that the VLAN tags must be identical at both interfaces for a given QinQ tunnel.

In VPLS, the switchport modes work in a similar fashion from an end user's perspective, but some of the internal operations vary slightly.

The following configuration steps highlight how service-delimiting VLAN tags and internal VLAN tags are used in each switchport mode.

Configuring the Access Mode

The access mode in VPLS is identical to that in normal bridging. Only untagged Ethernet packets are sent and received on the Layer 2 switchport interface, and the Ethernet header is sent over the pseudowire unmodified because no service-delimiting VLAN tag exists. You can configure the access mode as follows:

Step 1. Configure the interface as a switchport:

```
VPLS-PE1(config)#interface FastEthernet 4/3
VPLS-PE1(config-if)#switchport
VPLS-PE1(config-if)#
```

Step 2. Configure the switchport as an access mode:

```
VPLS-PE1(config-if)#switchport mode access
VPLS-PE1(config-if)#
```

Step 3. Assign the Layer 2 switchport interface to a bridge domain, which is represented by an internal VLAN tag:

```
VPLS-PE1(config-if)#switchport access vlan 2
VPLS-PE1(config-if)#
```

The interface configuration after these steps is shown in Example 15-1.

Example 15-1. *Access Mode Interface Configuration*

```
interface FastEthernet4/3
 no ip address
 switchport
 switchport access vlan 2
 switchport mode access
```

Configuring the Trunk Mode

When you configure a Layer 2 switchport interface as trunk mode in VPLS, the VLAN tag maps packets received from a CE router to a bridge domain. In other words, the VLAN tag in the customer traffic is considered the service-delimiting VLAN tag. A Layer 2 switchport interface does not support a configurable service-delimiting VLAN tag; therefore, the service-delimiting VLAN tag has to match the internal VLAN tag of the bridge domain for a given VPLS customer. Because the trunk mode supports multiple VLAN tags, traffic of different VPLS customers can be sent and received over a Layer 2 switchport interface.

In trunk mode, a PE router removes the service-delimiting VLAN tag on the Ethernet header before applying the pseudowire encapsulation. For the opposite direction, a PE applies the internal VLAN tag to the Ethernet header after the pseudowire encapsulation is removed from the pseudowire packet. Use the following steps to configure the trunk mode on a Layer 2 switchport interface:

Step 1. Configure the interface as a switchport:

```
VPLS-PE1(config)#interface FastEthernet 4/3
VPLS-PE1(config-if)#switchport
VPLS-PE1(config-if)#
```

Step 2. Configure the interface to use 802.1q VLAN encapsulation:

```
VPLS-PE1(config-if)#switchport trunk encapsulation dot1q
VPLS-PE1(config-if)#
```

Step 3. (Optional) Assign a list of VLANs allowed on this trunk, such as VLAN 2 to VLAN 10:

```
VPLS-PE1(config-if)#switchport trunk allowed vlan 2-10
VPLS-PE1(config-if)#
```

Step 4. Configure the switchport as trunk mode:

```
VPLS-PE1(config-if)#switchport mode trunk
VPLS-PE1(config-if)#
```

Example 15-2 shows the interface configuration after you complete the steps for the trunk mode.

Example 15-2. *Trunk Mode Interface Configuration*

```
interface FastEthernet4/3
 no ip address
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk allowed vlan 2-10
 switchport mode trunk
```

Configuring dot1q-tunnel Mode

QinQ tunneling is an Ethernet native tunneling mechanism that stacks VLAN tags together in a similar fashion to the MPLS labels. The outer VLAN tag that is added at the tunnel ingress interface is the access VLAN tag that is configured on the Layer 2 switchport interface. The purpose of the outer VLAN tag is similar to that of the tunnel label in an MPLS-encapsulated pseudowire packet. The outer VLAN tag is to forward the packet from the ingress tunnel endpoint to the egress tunnel endpoint and hide the inner VLAN tag from the transit network.

In VPLS, the transit network is an MPLS network, and a tunnel label is used to move packets from the LSP ingress endpoint to the egress endpoint. Because the function of an outer VLAN tag is effectively replaced by an MPLS tunnel label, the outer VLAN tag is no longer added to the Ethernet header when the Layer 2 switchport interface is configured as

dot1q-tunnel mode. That is the main difference in the way dot1q-tunnel mode operates in VPLS versus normal bridging.

The following is an example of configuring a Layer 2 switchport interface as dot1q-tunnel mode:

Step 1. Configure the interface as a switchport:

```
VPLS-PE1(config)#interface FastEthernet 4/3
VPLS-PE1(config-if)#switchport
VPLS-PE1(config-if)#
```

Step 2. Configure the switchport as dot1q-tunnel mode:

```
VPLS-PE1(config-if)#switchport mode dot1q-tunnel
VPLS-PE1(config-if)#
```

Step 3. Assign the Layer 2 switchport interface to a bridge domain, which is represented by an internal VLAN tag:

```
VPLS-PE1(config-if)#switchport access vlan 2
VPLS-PE1(config-if)#
```

The interface configuration for dot1q-tunnel mode is shown in Example 15-3.

Example 15-3. *dot1q-tunnel Mode Interface Configuration*

```
interface FastEthernet4/3
 no ip address
 switchport
 switchport access vlan 2
 switchport mode dot1q-tunnel
```

Layer 2 switchport interfaces are the predominant interface type in VPLS deployment, and new attachment circuit types are being added as the VPLS solution is being enhanced. Refer to Cisco.com to obtain the latest attachment circuit support.

Configuring VFI

When you configure AToM or L2TPv3 pseudowire emulation, the pseudowire portion of the configuration is done from the attachment circuit configuration mode. For example, interface mode is used for PPP and HDLC pseudowires, PVC mode for ATM AAL5, and connect mode for Frame Relay data-link connection identifier (DLCI). This implicitly builds the one-to-one mapping between an attachment circuit and a pseudowire.

VPLS needs to build a many-to-many mapping for each VPLS domain. For the pseudowire portion of the configuration, VPLS uses the VFI configuration mode to specify a set of pseudowires and associated properties for a given VPLS domain. Before enabling any other command, configure the VFI needs with a VPN ID. As explained earlier in the "VPLS Signaling" section, a VPN ID identifies a VPLS domain throughout the network. It is encoded in the Pseudowire ID field of the protocol messages. The VFI configuration mode also specifies the addresses of the peering PE routers, the type of pseudowire signaling, and the encapsulation method for each peer.

The following steps show an example of configuring a VFI:

Step 1. Create a multipoint VFI by enabling the VFI configuration mode:

```
VPLS-PE1(config)#l2 vfi blue manual
VPLS-PE1(config-vfi)#
```

The keyword **manual** indicates that you will enter the peering relationship with remote PE routers manually.

Step 2. Configure a VPN ID for the VFI:

```
VPLS-PE1(config-vfi)#vpn id 100
VPLS-PE1(config-vfi)#
```

The VPN ID is configured in the form of an unsigned integer. The range of values is from 1 to 4294967295, or 0xFFFFFFFF in hex.

Step 3. Specify the peering PE router ID and pseudowire encapsulation:

```
VPLS-PE1(config-vfi)#neighbor 10.0.0.2 encapsulation mpls
VPLS-PE1(config-vfi)#
```

Step 4. Repeat Step 3 for every peering PE router.

Note

Currently, the manual mode is the only provisioning option available for multipoint VFI. When an autodiscovery mechanism is introduced for VPLS in the future, a VFI can be provisioned automatically.

Example 15-4 shows an example of a VFI configuration.

Example 15-4. *VFI Configuration*

```
l2 vfi blue manual
 vpn id 100
 neighbor 10.0.0.2 encapsulation mpls
 neighbor 10.0.0.3 encapsulation mpls
 neighbor 10.0.0.4 encapsulation mpls
```

Associating Attachment Circuits to the VFI

The final step in building the many-to-many mapping involves how to associate attachment circuits to a VFI in configuration.

In the switchport mode configuration steps, a bridge domain or an internal VLAN is assigned to attachment circuits for a given VPLS domain. For example, Layer 2 switchport interfaces in access and dot1q-tunnel mode use the command **switchport access vlan** to specify the bridge domain explicitly. Those in trunk mode use the service-delimiting VLAN tags instead.

You can view a VLAN interface as a virtual interface representation of a bridge domain. By associating the VFI under the VLAN interface configuration mode, the many-to-many association is finally accomplished. To configure the VFI under a VLAN interface, use the following steps:

Step 1. Create or access a VLAN interface, also known as a switched virtual interface:

```
VPLS-PE1(config)#interface vlan 2
VPLS-PE1(config-if)#
```

Note that the VLAN ID needs to be identical to the service-delimiting VLAN tag when using Layer 2 switchport trunk mode. Otherwise, it can be the tag value of an unused VLAN.

Step 2. Attach the VFI to the VLAN interface:

```
VPLS-PE1(config-if)#xconnect vfi blue
VPLS-PE1(config-if)#
```

You must have a valid VFI configured before this command can be accepted.

The next section shows the complete configuration example and ways to verify whether it is working.

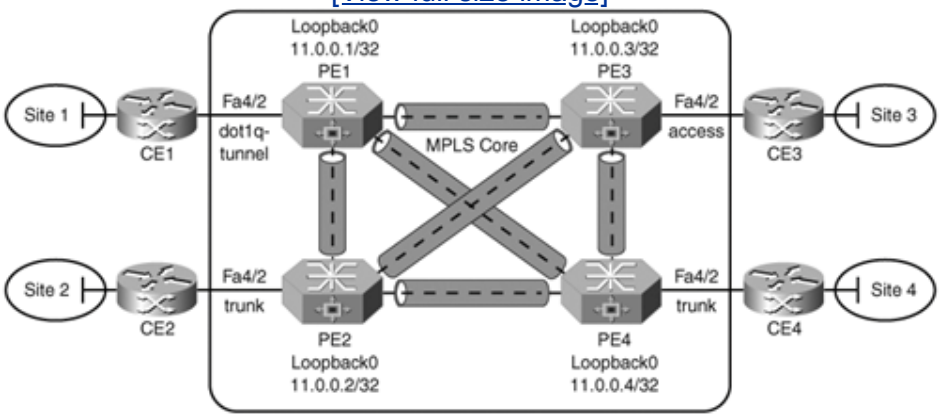
Configuration Example

With the basic VPLS configuration building blocks, network operators can construct fairly sophisticated multipoint Layer 2 VPNs. Figure 15-7 shows an example of a full-mesh VPLS

Layer 2 VPN with four CE routers of the same VPLS customer.

Figure 15-7. *VPLS Configuration Example*

[View full size image]



To illustrate the flexibility of how you can connect CE devices, the configuration example uses different switchport modes and service-delimiting VLAN tags on each PE router as follows:

- CE1 sends and receives untagged Ethernet packetsthat is, null service-delimiting VLAN tags. PE1 configures the switchport mode as dot1q-tunnel to forward packets that have an unmodified Ethernet header. The internal VLAN that is associated with the switchport is 2.
- CE2 sends and receives tagged Ethernet VLAN packets of which the service-delimiting VLAN tag is 4. PE2 configures the switchport mode as a trunk to remove or add the service-delimiting VLAN tag accordingly. The internal VLAN that is associated with the switchport is 4.
- CE3 sends and receives untagged Ethernet packetsthat is, null service-delimiting VLAN tags. PE2 configures the switchport mode as access to forward all untagged packets. The internal VLAN that is associated with the switchport is 8.
- CE4 sends and receives tagged Ethernet VLAN packets of which the service-delimiting VLAN tag is 10. PE4 configures the switchport mode as a trunk to remove or add the service-delimiting VLAN tag accordingly. The internal VLAN that is associated with the switchport is 10.

Example 15-5 shows the configuration on PE1.

Example 15-5. *PE1 Configuration*

```
hostname PE1
!
```

```
mpls label protocol ldp
mpls ldp logging neighbor-changes
mpls ldp router-id Loopback0
!
l2 vfi l2vpn manual
 vpn id 1
 neighbor 10.0.0.2 encapsulation mpls
 neighbor 10.0.0.3 encapsulation mpls
 neighbor 10.0.0.4 encapsulation mpls
!
interface Loopback0
 ip address 10.0.0.1 255.255.255.255
!
interface POS3/1
 ip address 10.0.1.1 255.255.255.252
 mpls ip
!
interface FastEthernet4/2
 no ip address
 switchport
 switchport access vlan 2
 switchport mode dot1q-tunnel
!
interface Vlan2
 no ip address
 xconnect vfi l2vpn
```

Example 15-6 shows the configuration on PE2.

Example 15-6. *PE2 Configuration*

```
hostname PE2
!
mpls label protocol ldp
mpls ldp logging neighbor-changes
mpls ldp router-id Loopback0
!
l2 vfi l2vpn manual
 vpn id 1
 neighbor 10.0.0.1 encapsulation mpls
 neighbor 10.0.0.3 encapsulation mpls
 neighbor 10.0.0.4 encapsulation mpls
!
interface Loopback0
 ip address 10.0.0.2 255.255.255.255
!
interface POS3/1
 ip address 10.0.2.1 255.255.255.252
 mpls ip
!
interface FastEthernet4/2
 no ip address
 switchport
```

```
 switchport trunk encapsulation dot1q
 switchport trunk allowed vlan 4
 switchport mode trunk
!
interface Vlan4
 no ip address
 xconnect vfi l2vpn
```

Example 15-7 shows the configuration on PE3.

Example 15-7. *PE3 Configuration*

```
hostname PE3
!
mpls label protocol ldp
mpls ldp logging neighbor-changes
mpls ldp router-id Loopback0
!
l2 vfi l2vpn manual
 vpn id 1
 neighbor 10.0.0.1 encapsulation mpls
 neighbor 10.0.0.2 encapsulation mpls
 neighbor 10.0.0.4 encapsulation mpls
!
interface Loopback0
 ip address 10.0.0.3 255.255.255.255
!
interface POS3/1
 ip address 10.0.3.1 255.255.255.252
 mpls ip
!
interface FastEthernet4/2
 no ip address
 switchport
 switchport access vlan 8
 switchport mode access
!
interface Vlan8
 no ip address
 xconnect vfi l2vpn
```

Example 15-8 shows the configuration on PE4.

Example 15-8. *PE4 Configuration*

```
hostname PE4
!
mpls label protocol ldp
mpls ldp logging neighbor-changes
```

```
mpls ldp router-id Loopback0
!
l2 vfi l2vpn manual
 vpn id 1
 neighbor 10.0.0.1 encapsulation mpls
 neighbor 10.0.0.2 encapsulation mpls
 neighbor 10.0.0.3 encapsulation mpls
!
interface Loopback0
 ip address 10.0.0.4 255.255.255.255
!
interface POS3/1
 ip address 10.0.4.1 255.255.255.252
 mpls ip
!
interface FastEthernet4/2
 no ip address
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk allowed vlan 10
 switchport mode trunk
!
interface Vlan10
 no ip address
 xconnect vfi l2vpn
```

You can examine the VFI using the command **show vfi** (see Example 15-9).

Example 15-9. *Verifying the VFI Status*

```
PE1#show vfi l2vpn
VFI name: l2vpn, state: up
  Local attachment circuits:
    Vlan2
  Neighbors connected via pseudowires:
    10.0.0.2 10.0.0.3 10.0.0.4
```

Table 15-1 lists the MAC addresses that are associated with each CE router. When a CE router also functions as an Ethernet switch, it bridges customer Ethernet traffic toward the attached PE router. In that scenario, the PE router learns multiple source MAC addresses from the CE router.

Table 15-1. *MAC Addresses from CE Routers*

Router	MAC Address

Router	MAC Address
CE1	000b.5fb5.0080
CE2	000b.5fad.e580
CE3	000b.5fb1.5780
CE4	000b.5fb1.5480

After full connectivity is established among all CE routers, every PE router should learn all MAC addresses from the CE routers. To verify the learning process on each PE router, use the command **show mac-address-table vlan**, as shown in Example 15-10.

Example 15-10. *Verifying the Learning Process on Each PE Router*

```
PE1#show mac-address-table vlan 2
Legend: * - primary entry

  vlan   mac address     type  learn            ports
------+---------------+-------+-----+-----------------------
*    2  000b.5fb5.0080 dynamic  Yes   Fa4/2
*    2  000b.5fad.e580 dynamic  Yes
*    2  000b.5fb1.5780 dynamic  Yes
*    2  000b.5fb1.5480 dynamic  Yes


PE2#show mac-address-table vlan 4
Legend: * - primary entry

  vlan   mac address     type   learn            ports
------+---------------+-------+-----+-----------------------
*    4  000b.5fb5.0080  dynamic  Yes
*    4  000b.5fad.e580  dynamic  Yes   Fa4/2
*    4  000b.5fb1.5780  dynamic  Yes
*    4  000b.5fb1.5480  dynamic  Yes


PE3#show mac-address-table vlan 8
Legend: * - primary entry

  vlan   mac address     type   learn            ports
------+---------------+-------+-----+-----------------------
*    8  000b.5fb5.0080  dynamic  Yes
*    8  000b.5fad.e580  dynamic  Yes
*    8  000b.5fb1.5780  dynamic  Yes    Fa4/2
*    8  000b.5fb1.5480  dynamic  Yes
```

```
PE4#show mac-address-table vlan 10
Legend: * - primary entry
  vlan   mac address     type   learn            ports
------+---------------+-------+-----+------------------------
*   10  000b.5fb5.0080  dynamic  Yes
*   10  000b.5fad.e580  dynamic  Yes
*   10  000b.5fb1.5780  dynamic  Yes
*   10  000b.5fb1.5480  dynamic  Yes    Fa4/2
```

To display the status of the pseudowires that interconnect the virtual switches, use the command **show mpls l2transport vc** on PE routers, as shown in Example 15-11.

Example 15-11. *Displaying the Status of the Pseudowires*

```
PE1#show mpls l2transport vc

Local intf     Local circuit          Dest address    VC ID      Status
-------------- ---------------------- --------------- ---------- ----------
VFI l2vpn      VFI                    10.0.0.2        1          UP
VFI l2vpn      VFI                    10.0.0.3        1          UP
VFI l2vpn      VFI                    10.0.0.4        1          UP
```

Case Study 15-2: Per-VLAN MAC Address Limiting

Service providers are concerned that a rogue VPLS customer will take too much system and network resources and affect normal services for other customers. One of the limited system resources on which different VPLS customers compete is the MAC address table.

Generally speaking, the size of the MAC address table on a given system is finite, and the portion allocated for each bridge domain directly impacts the forwarding performance. The larger the portion allocated is, the less likely a packet is subject to flooding. Flooding is always an expensive operation in terms of processing power and the network bandwidth it takes; it penalizes overall packet forwarding performance.

To limit the maximum number of MAC address entries on a per-VLAN basis, use the **mac-address-table limit** command, as shown in Example 15-12. Cisco VPLS allows setting a limit for each bridge domain, which is represented by an internal VLAN.

Example 15-12. mac-address-table limit *Command*

```
PE1(config)#mac-address-table limit vlan 2 maximum 1000
PE1(config)#
```

To display the MAC address limiting status for a VLAN, use the **show mac-address-table limit vlan** command, as shown in Example 15-13.

Example 15-13. show mac-address-table limit vlan *Command*

```
PE1#show mac-address-table limit vlan 2
  vlan   module    action    maximum  Total entries   flooding
-------+--------+-----------+-------+--------------+------------
  2         2      warning     1000       0           enabled
  2         4      warning     1000       0           enabled
```

Case Study 15-3: Quality of Service

On Cisco 7600 series routers, Layer 2 switchport interfaces use Policy Feature Card (PFC)based QoS configuration, and the core-facing interfaces use Modular QoS CLI (MQC). The general topics on PFC-based and MQC-based configuration alone warrant a book. This book does not cover the details on these topics. Refer to Cisco.com for the PFC-based and MQC-based QoS commands and examples. This QoS case study shows an example that is related to VPLS.

Per-VLAN traffic shaping in VPLS specifies the shaping rate of individual MPLS uplinks for a given bridge domain, not the aggregated rate of all MPLS uplinks. For example, if a VLAN is configured with a shaping rate of 10 Mbps, and there are two MPLS uplinks toward the MPLS core network, the shaper allows up to 20 Mbps of VPLS traffic forwarded into the core network.

In Example 15-14, PE1 matches all traffic coming from CE1 and shapes the VPLS traffic on each core-facing interface to 10 Mbps.

Example 15-14. *VPLS Per-VLAN Traffic Shaping*

```
hostname PE1
!
class-map match-all all-traffic
  match any
!
policy-map vpls-policy
  class all-traffic
    shape average 1000000 4000 4000
!
interface Vlan2
 no ip address
 xconnect vfi l2vpn
 service-policy output vpls-policy
```

To verify QoS configuration status, use the **show policy-map interface** command, as shown in Example 15-15.

Example 15-15. *Verifying QoS Status*

```
PE1#show policy-map interface
 Vlan2

  Service-policy output: vpls-policy

    Class-map: all-traffic (match-all)
      6 packets, 2316 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      queue size 0, queue limit 0
      packets output 6, packet drops 0
      tail/random drops 0, no buffer drops 0, other drops 0
      shape (average) cir 1000000 bc 4000 be 4000
      target shape rate 1000000
    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Case Study 15-4: Layer 2 Protocol Tunneling

Layer 2 protocol tunneling allows Layer 2 PDUs, such as Cisco Discovery Protocol (CDP), Spanning-Tree Protocol (STP), and VLAN Trunking Protocol (VTP), to be tunneled through an Ethernet-switched network. Without Layer 2 protocol tunneling, Layer 2 switchport interfaces drop STP and VTP packets and process CDP packets.

To allow CE1 and CE3 in Figure 15-7 to view each other as CDP neighbors, the interfaces on PE1 and PE3 that connect to CE1 and CE3 respectively need to enable Layer 2 protocol tunneling (see Example 15-16).

Example 15-16. *Enabling Layer 2 Protocol Tunneling*

```
PE1(config)#interface FastEthernet 4/2
PE1(config-if)#l2protocol-tunnel cdp

PE3(config)#interface FastEthernet 4/2
PE3(config-if)#l2protocol-tunnel cdp
```

To verify the effectiveness, use the command **show cdp neighbors** on the CE devices, as demonstrated in Example 15-17.

Example 15-17. *Verifying CDP Neighbors with the* show cdp neighbors *Command*

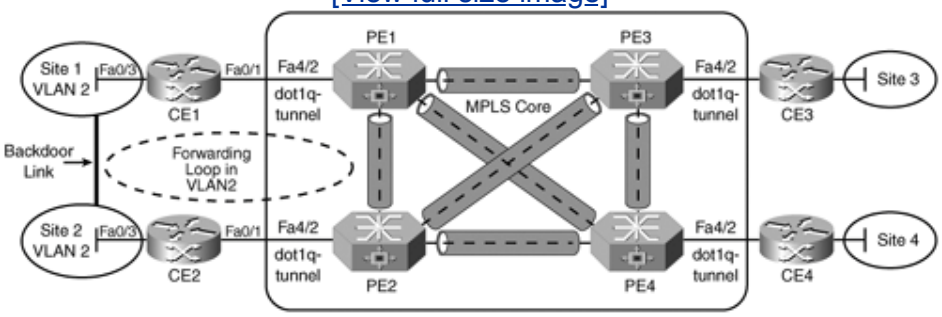
```
CE1#show cdp  neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater, P - Phone

Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
CE3              Fas 0/1             157         R S I     WS-C3550-2Fas 0/1

CE3#show cdp  neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater

Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
CE1              Fas 0/1             170        R S I      WS-C3550-2Fas 0/1
```

For STPs, a separate spanning tree is created at each customer site if Layer 2 protocol tunneling is not enabled on PE routers. Using the network shown in Figure 15-7 as an example, bridging devices at Site 1including CE1build a spanning tree solely for Site 1 without considering convergence parameters of other customer sites. In this particular example, the disjointed spanning tree domains do not lead to potential forwarding loops because of the use of Layer 2 split horizon in the service provider network. However, if the customer sites have backdoor links, it is imperative that you have a single spanning-tree domain for the VPLS customer to avoid forwarding loops in the customer network.

Figure 15-8 shows a backdoor link that connects CE1 and CE2. A possible forwarding loop exists between CE1 and CE2 when packets can be sent over the links that are connected to the service provider and the backdoor link. To identify the possible forwarding loop, examine the spanning-tree status on both CE routers.

Figure 15-8. *VPLS with Backdoor Link*

[View full size image]



On CE1, the interface FastEthernet0/1 connected to PE1 acts as a designated port for VLAN 2 and is in the forwarding state. The interface FastEthernet0/3 connected to Site 1 is a root port for VLAN 2 and is also in the forwarding state (see Example 15-18).

Example 15-18. *VLAN 2 Spanning-Tree Status on CE1 Before the Forwarding Loop Is Fixed*

```
CE1#show spanning-tree vlan 2

VLAN0002
  Spanning tree enabled protocol ieee
  Root ID     Priority    32770
              Address     000b.5fadfie580
              Cost        19
              Port        3 (FastEthernet0/3)
              Hello Time   2 sec Max Age 20 sec Forward Delay 15 sec

  Bridge ID   Priority    32770 (priority 32768 sys-id-ext 2)
              Address     000b.5fb5.0080
              Hello Time   2 sec Max Age 20 sec Forward Delay 15 sec
              Aging Time 300

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Fa0/1            Desg FWD 19        128.1    P2p
Fa0/3            Root FWD 19        128.3    P2p
```

CE2 is the root bridge for VLAN 2 because it has a lower bridge address than CE1 when both have the same bridge priority. Both the interface FastEthernet0/1 that connects to PE2 and the interface FastEthernet0/3 that connects to Site 2 have a role of designated port for VLAN 2, and they are both in the forwarding state (see Example 15-19).

Example 15-19. *VLAN 2 Spanning-Tree Status on CE2 Before the Forwarding Loop Is Fixed*

```
CE2#show spanning-tree vlan 2

VLAN0002
  Spanning tree enabled protocol ieee
  Root ID     Priority    32770
              Address     000b.5fadfie580
              This bridge is the root
              Hello Time   2 sec Max Age 20 sec Forward Delay 15 sec

  Bridge ID Priority    32770 (priority 32768 sys-id-ext 2)
            Address     000b.5fadfie580
            Hello Time   2 sec Max Age 20 sec Forward Delay 15 sec
            Aging Time 300

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Fa0/1            Desg FWD 19        128.1    P2p
Fa0/3            Desg FWD 19        128.3    P2p
```

Because the spanning-tree status on all four ports is in the forwarding state for VLAN 2, the forwarding loop is inevitable. The problem apparently is caused by the two ports connected through the pseudowire that cannot exchange BPDUs.

To fix this problem, PE routers need to configure Layer 2 protocol tunneling for STP traffic (see Example 15-20).

Example 15-20. *Configuring Layer 2 Protocol Tunneling for STP Traffic*

```
PE1(config)#int FastEthernet4/2
PE1(config-if)#l2protocol-tunnel stp

PE2(config)#int FastEthernet4/2
PE2(config-if)#l2protocol-tunnel stp
```

The interface configuration on PE1 and PE2 is shown in Example 15-21. Notice that the VLAN that is configured on the dot1q-tunnel interface is an internal VLAN. It can be different from the VLAN that is used in customer traffic.

Example 15-21. *PE1 and PE2 Interface Configuration*

```
interface FastEthernet4/2
 no ip address
 no keepalive
 switchport
 switchport access vlan 4
 switchport mode dot1q-tunnel
 l2protocol-tunnel stp
```

To display Layer 2 protocol tunneling status on PE routers, use the **show l2protocol-tunnel** command, as in Example 15-22.

Example 15-22. *Displaying the Layer 2 Protocol Tunneling Status*

```
PE1#show l2protocol-tunnel summary
COS for Encapsulated Packets: 5
Drop Threshold for Encapsulated Packets: 0

Port    Protocol    Shutdown         Drop             Status
                    Threshold        Threshold
                    (cdp/stp/vtp)    (cdp/stp/vtp)
------- ----------- ---------------- ---------------- ----------
Fa4/2   --- stp --- ----/----/----   ----/----/----   up
```

After you enable Layer 2 protocol tunneling on PE1 and PE2, the forwarding loop no longer exists. On CE1, the interface FastEthernet0/3 is now in the blocking state for VLAN 2, and packets in VLAN 2 are forwarded through the interface FastEthernet0/1 only (see Example 15-23).

Example 15-23. ***VLAN 2 Spanning-Tree Status on CE1 After the Forwarding Loop Is Fixed***

```
CE1#show spanning-tree vlan 2

VLAN0002
  Spanning tree enabled protocol ieee
  Root ID     Priority    32770
              Address     000b.5fad580
              Cost        19
              Port        1 (FastEthernet0/1)
              Hello Time   2 sec Max Age 20 sec Forward Delay 15 sec

  Bridge ID   Priority    32770 (priority 32768 sys-id-ext 2)
              Address     000b.5fb5.0080
              Hello Time   2 sec  Max Age 20 sec Forward Delay 15 sec
              Aging Time 300

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Fa0/1            Root FWD 19        128.1    P2p
Fa0/3            Altn BLK 19        128.3    P2p
```

Case Study 15-5: Multihoming

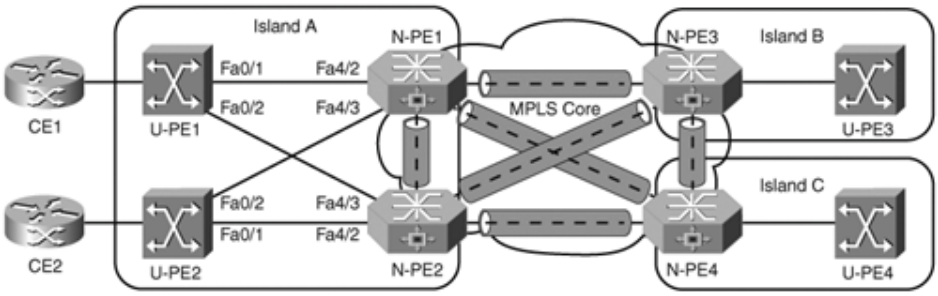
With the basic VPLS deployment models, each CE router has a single connection with a PE router. If the PE router fails or is taken out of service for maintenance, the CE router loses connectivity to the rest of the VPLS service. Likewise, in a hierarchical model, a U-PE router has only a single connection with an N-PE router, which makes the N-PE router a single point of failure.

By multihoming with more than one PE or N-PE router, a CE or U-PE router can achieve fault tolerance through the redundant connections. Whenever redundant Ethernet connections exist, bridging loops form as a result. Layer 2 split horizon is not designed to deal with redundant connections; therefore, STPs need to be enabled to create loop-free forwarding paths.

As described in the "VPLS Redundancy" section, each metro area or island consists of a group of U-PE and N-PE routers that are connected through a LAN. Figure 15-9 shows a network with three separate islands. The goal is to run STPs within each island for redundancy while preventing the spanning trees from spreading across the WAN. In a Metro Ethernet environment, devices from different network vendors are often deployed and required to work together, which means the network needs to run standard network protocols. For STPs, IEEE 802.1S Multiple Spanning Tree Protocol (MSTP) fits the purpose.

Figure 15-9. ***VPLS Redundancy Using Multihoming***

[View full size image]



Both U-PE1 and U-PE2 are peered with N-PE1 and N-PE2. To reduce the amount of complexity and processing power required, N-PE1 and N-PE2 do not run STPs themselves but simply relay BPDUs from one link to another. To prevent BPDUs from leaking to the WAN, you need to separate customer traffic from the BPDUs that originated in each island. In this case study, you accomplish this by marking these two types of traffic with different service provider VLAN tags. After the two types of traffic are separated into different VLANs, you can configure N-PE routers in such a way that only VLAN traffic that is marked as customer traffic can be forwarded to other islands. VLAN traffic that is marked as BPDU is only forwarded to other N-PE routers of the same island.

In Island A, two separate forwarding loops exist:

- From U-PE1 to N-PE1, N-PE2, and back to U-PE1
- From U-PE2 to N-PE1, N-PE2, and back to U-PE2.

Because U-PE1 and U-PE2 do not have direct connections, they can construct separate spanning trees.

On U-PE1, MST traffic is carried in the native VLAN for which the tag value is 200. VLAN 2 carries customer traffic for a particular VPLS customer. The configuration on U-PE1 is shown in Example 15-24.

Example 15-24. *U-PE1 Configuration*

```
hostname U-PE1
spanning-tree mode mst
!
spanning-tree mst configuration
 name MST-1
 revision 1
 instance 1 vlan 2
!
vlan dot1q tag native
!
interface FastEthernet0/1
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 200
 switchport trunk allowed vlan 2,200
 switchport mode trunk
```

```
 no ip address
!
interface FastEthernet0/2
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 200
 switchport trunk allowed vlan 2,200
 switchport mode trunk
 no ip address
```

On U-PE2, MST traffic is carried in the native VLAN for which the tag value is 400. VLAN 2 carries the customer traffic. The configuration on U-PE2 is shown in Example 15-25.

Example 15-25. *U-PE2 Configuration*

```
hostname U-PE2
spanning-tree mode mst
!
spanning-tree mst configuration
 name MST-2
 revision 1
 instance 1 vlan 2
!
vlan dot1q tag native
!
interface FastEthernet0/1
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 400
 switchport trunk allowed vlan 2,400
 switchport mode trunk
 no ip address
!
interface FastEthernet0/2
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 400
 switchport trunk allowed vlan 2,400
 switchport mode trunk
 no ip address
```

STPs are disabled on N-PE1 and N-PE2. To relay BPDUs for each MST instance transparently, in addition to configuring Layer 2 protocol tunneling N-PE1 and N-PE2, you must configure a dedicated VFI for each MST instance, where the neighbors are N-PE routers in the same island. The configuration on N-PE1 is shown in Example 15-26.

Example 15-26. *N-PE1 Configuration*

```
hostname N-PE1
!
mpls label protocol ldp
mpls ldp router-id Loopback0
```

```
!
l2 vfi l2vpn manual
 vpn id 1
 neighbor 10.0.0.2 encapsulation mpls
 neighbor 10.0.0.3 encapsulation mpls
 neighbor 10.0.0.4 encapsulation mpls
!
l2 vfi mst-1 manual
 vpn id 1001
 neighbor 10.0.0.2 encapsulation mpls
!
l2 vfi mst-2 manual
 vpn id 2001
 neighbor 10.0.0.2 encapsulation mpls
!
no spanning-tree vlan 2,200,400
!
vlan dot1q tag native
!
interface Loopback0
 ip address 10.0.0.1 255.255.255.255
!
interface POS3/1
 ip address 10.0.1.1 255.255.255.252
 mpls ip
!
interface FastEthernet4/2
 no ip address
 no keepalive
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 200
 switchport trunk allowed vlan 2,200
 switchport mode trunk
 l2protocol-tunnel stp
!
interface FastEthernet4/3
 no ip address
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 400
 switchport trunk allowed vlan 2,400
 switchport mode trunk
 l2protocol-tunnel stp
 no cdp enable
!
interface Vlan2
 no ip address
 xconnect vfi l2vpn
!
interface Vlan200
 no ip address
 xconnect vfi mst-1
!
interface Vlan400
 no ip address
 xconnect vfi mst-2
```

The configuration on N-PE2 is shown in Example 15-27.

Example 15-27. *N-PE2 Configuration*

```
hostname N-PE2
!
mpls label protocol ldp
mpls ldp router-id Loopback0
!
l2 vfi l2vpn manual
 vpn id 1
 neighbor 10.0.0.1 encapsulation mpls
 neighbor 10.0.0.3 encapsulation mpls
 neighbor 10.0.0.4 encapsulation mpls
!
l2 vfi mst-1 manual
 vpn id 1001
 neighbor 10.0.0.1 encapsulation mpls
!
l2 vfi mst-2 manual
 vpn id 2001
 neighbor 10.0.0.1 encapsulation mpls
!
no spanning-tree vlan 2,200,400
!
vlan dot1q tag native
!
interface Loopback0
 ip address 10.0.0.2 255.255.255.255
!
interface POS3/1
 ip address 10.0.2.1 255.255.255.252
 mpls ip
!
interface FastEthernet4/2
 no ip address
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 400
 switchport trunk allowed vlan 2,400
 switchport mode trunk
 l2protocol-tunnel stp
!
interface FastEthernet4/3
 no ip address
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 200
 switchport trunk allowed vlan 2,200
 switchport mode trunk
 l2protocol-tunnel stp
!
interface Vlan2
```

```
 no ip address
 xconnect vfi l2vpn
!
interface Vlan200
 no ip address
 xconnect vfi mst-1
!
interface Vlan400
 no ip address
 xconnect vfi mst-2
```

To verify that MSTP removes the forwarding loops, use the **show spanning-tree mst** command on U-PE1 and U-PE2 (see Example 15-28). Notice that each router is the root bridge for its own MST instance, the interface FastEthernet0/1 acts as a designated port and is in a forwarding state, and the interface FastEthernet0/2 acts as a backup port and is in a blocking state.

Example 15-28. *MST Instance Status on U-PE Routers*

```
U-PE1#show spanning-tree mst 1

###### MST01        vlans mapped:  2
Bridge      address 000b.5fb5.0080 priority 32769 (32768 sysid 1)
Root        this switch for MST01

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Fa0/1            Desg FWD 200000    128.1    P2p
Fa0/2            Back BLK 200000    128.2    P2p


U-PE2#show spanning-tree mst 1

###### MST01        vlans mapped:   2
Bridge      address 000b.5fad580 priority 32769 (32768 sysid 1)
Root        this switch for MST01

Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Fa0/1            Desg FWD 200000    128.1    P2p
Fa0/2            Back BLK 200000    128.2    P2p
```

Summary

Extending LAN services across multiple metropolitan areas is one of the critical requirements for large Metro Ethernet deployment. Service providers want to leverage their existing Layer 3 WAN infrastructure to bridge the distance gap of LAN services.

VPLS is a multipoint Layer 2 VPN architecture designed to fulfill such a requirement with considerations to scalability and performance. Comparing point-to-point Layer 2 VPN architectures, VPLS is relatively more complex in both implementation and deployment.

This chapter started with an overview of VPLS architecture that defined the service requirements, explained the concept and components of a virtual switch, and identified the characteristics of the VPLS control plane and data plane.

When it comes to topologic models, VPLS is quite flexible depending on the size and complexity of individual deployment. The basic VPLS models include full-mesh and hub-and-spoke, and the more advanced hierarchical VPLS is a hybrid of both basic models. Hierarchical VPLS can have either MPLS access networks or QinQ networks. Redundancy is always a top priority for service providers. You can accomplish VPLS redundancy by using multihoming and STPs. In a Metro Ethernet deployment, you should limit the scope of spanning trees within individual islands for bandwidth efficiency and network stability in the backbone.

This chapter showed basic VPLS configuration step-by-step and gave a VPLS example that used different switchport modes on the interfaces connecting to customer networks. The case studies highlighted the scenarios commonly seen in VPLS deployment, such as per-VLAN MAC address limiting, QoS, Layer 2 protocol tunnel, backdoor links, and multihoming.

Appendix 1. L2TPv3 AVP Attribute Types

This appendix presents a comparison between the Attribute-Value Pair (AVP) attribute types that Cisco routers use prior to the Internet Assigned Numbers Authority (IANA) assignment of the values and the IANA assigned values. The message types in which those AVPs are present are also included (see Table A-1).

Table A-1. *Comparing L2TPv3 AVP Attribute Types*

AVP Name	Cisco AVP	IETF[1] AVP	Messages
Extended Vendor ID AVP	N/A	58	All messages
Message Digest	12	59	All messages
Router ID	N/A	60	SCCRQ[2], SCCRP[3]
Assigned Control Connection ID	1	61	SCCRQ, SCCRP, StopCCN[4]
Pseudowire Capabilities List	2	62	SCCRQ, SCCRP
Local Session ID	3	63	ICRQ[5], ICRP[6], ICCN[7], CDN[8], WEN[9], SLI[10]
Remote Session ID	4	64	ICRQ, ICRP, ICCN, CDN, WEN, SLI
Assigned Cookie	5	65	ICRQ, ICRP
Remote End ID	6	66	ICRQ
Application Code [Unused]	N/A	67	Not defined
Pseudowire Type	7	68	ICRQ

AVP Name	Cisco AVP	IETF[1] AVP	Messages
Layer 2-Specific Sublayer	Uses IETF	69	ICRQ, ICRP, ICCN
Data Sequencing	N/A	70	ICRQ, ICRP, ICCN
Circuit Status	8	71	ICRQ, ICRP, ICCN, SLI
Preferred Language	N/A	72	SCCRQ, SCCRP
Control Message Authentication Nonce	13	73	SCCRQ, SCCRP
Tx Connect Speed	N/A	74	ICRQ, ICRP, ICCN
Rx Connect Speed	N/A	75	ICRQ, ICRP, ICCN
Session Tie Breaker	9	N/A	ICRQ
ATM Maximum Concatenated Cells	11	N/A	ICRQ, ICRP, SLI
OAM Emulation Required	108	N/A	ICRQ, ICRP, SLI
ATM Alarm Status	109	N/A	SLI

[1] IETF = Internet Engineering Task Force

[2] SCCRQ = Start-Control-Connection-Request

[3] SCCRP = Start-Control-Connection-Reply

[4] StopCCN = Stop-Control-Connection-Notification

[5] ICRQ = Incoming-Call-Request

[6] ICRP = Incoming-Call-Reply

[7] ICCN = Incoming-Call Connected

[8] CDN = Circuit-Disconnect-Notify

[9] WEN = WAN-Error-Notify

[10] SLI = Set-Link-Info

You can find a complete list of L2TP registries and numbers managed by the IANA at http://www.iana.org/assignments/l2tp-parameters.

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

MTU considerations

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

MTU considerations

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index