ipexpert

IPexpert's

IPv4/IPv6 Quality of Service (QoS)

Operation and Troubleshooting

Authored By: Anthony Sequeira CCIE# 15626 (R&S), CCDP, CCSP.
Terry Vinson CCIE #35347 (R&S), CCNP
Technical Editor: Carl Yost Jr CCIE# 30486 (R&S), Jason Gooley CCNP
Editor: Tiffany Pagan

Before We Begin

This product is part of the IPexpert suite of materials that provide CCIE candidates and network engineers with a comprehensive training program. For information about the full solution, contact an IPexpert Training Advisor today.

Telephone: +1.810.326.1444
Email: sales@ipexpert.com

Congratulations! You now possess one of the ULTIMATE CCIE™ Lab preparation and network operation resources available today! Senior engineers, technical instructors, and authors boasting decades of internetworking experience produced this resource.

In order to enjoy technical support from IPexpert and your CCIE community, be sure to visit the following Internet resources:

http://blog.ipexpert.com
http://onlinestudylist.com

IPexpert is proud to lead the industry with multiple support options at your disposal free of charge. Our online communities have attracted a membership of over 20,000 of your peers from around the world! At blog.ipexpert.com, you can keep up to date with everything IPexpert does and read the latest in technical articles from word-renowned IPexpert instructors. At OnlineStudyList.com, you may subscribe to multiple "SPAM-free," moderated CCIE-focused email lists.

Feedback

Do you have a suggestion or other feedback regarding this book or other IPexpert products? At IPexpert, we look to you – our valued clients – for the real world, frontline evaluation that we believe is necessary so that we may always improve. Please send an email with your thoughts to feedback@ipexpert.com or call 1.866.225.8064 (international callers dial +1.810.326.1444).

In addition, for those using this book as CCIE™ preparation, when you pass the CCIE™ Lab exam, we want to hear about it! Email your CCIE™ number to success@ipexpert.com and let us know how IPexpert helped you succeed. We would like to send you a gift of thanks and congratulations.

Additional CCIE™ Preparation Material

IPexpert, Inc. is committed to developing the most effective Cisco CCIE™ R&S, Security, Voice and Wireless Lab certification preparation tools available. Our team of certified networking professionals develops the most up-to-date and comprehensive materials for networking certification, including self-paced workbooks, online Cisco hardware rental, classroom training, online (distance learning) instructor-led training, audio products, and video training materials. Unlike other certification-training providers, we employ the most experienced and accomplished teams of experts to create, maintain, and constantly update our products. At IPexpert, we are focus on making your CCIE™ Lab preparation more effective.

Issues with this Book

This book is carefully edited to ensure the accuracy of all content. Should you find any error whatsoever, please email a page reference and detailed comment to compsolv@me.com. Your email will be responded to promptly.

IPEXPERT END-USER LICENSE AGREEMENT

END USER LICENSE FOR ONE (1) PERSON ONLY

IF YOU DO NOT AGREE WITH THESE TERMS AND CONDITIONS,
DO NOT OPEN OR USE THE TRAINING MATERIALS.

This is a legally binding agreement between you and IPEXPERT, the “Licensor,” from whom you have licensed the IPEXPERT training materials (the “Training Materials”). By using the Training Materials, you agree to be bound by the terms of this License, except to the extent these terms have been modified by a written agreement (the “Governing Agreement”) signed by you (or the party that has licensed the Training Materials for your use) and an executive officer of Licensor. If you do not agree to the License terms, the Licensor is unwilling to license the Training Materials to you. In this event, you may not use the Training Materials, and you should promptly contact the Licensor for return instructions.

The Training Materials shall be used by only ONE (1) INDIVIDUAL who shall be the sole individual authorized to use the Training Materials throughout the term of this License.

Copyright and Proprietary Rights

The Training Materials are the property of IPEXPERT, Inc. ("IPEXPERT") and are protected by United States and International copyright laws. All copyright, trademark, and other proprietary rights in the Training Materials and in the Training Materials, text, graphics, design elements, audio, and all other materials originated by IPEXPERT at its site, in its workbooks, scenarios and courses (the "IPEXPERT Information") are reserved to IPEXPERT.

The Training Materials cannot be used by or transferred to any other person. You may not rent, lease, loan, barter, sell or time-share the Training Materials or accompanying documentation. You may not reverse engineer, decompile, or disassemble the Training Materials. You may not modify, or create derivative works based upon the Training Materials in whole or in part. You may not reproduce, store, upload, post, transmit, download or distribute in any form or by any means, electronic, mechanical, recording or otherwise any part of the Training Materials and IPEXPERT Information other than printing out or downloading portions of the text and images for your own personal, non-commercial use without the prior written permission of IPEXPERT.

You shall observe copyright and other restrictions imposed by IPEXPERT. You may not use the Training Materials or IPEXPERT Information in any manner that infringes the rights of any person or entity.

Exclusions of Warranties

THE TRAINING MATERIALS AND DOCUMENTATION ARE PROVIDED "AS IS." LICENSOR HEREBY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE LIMITATION OF INCIDENTAL DAMAGES OR LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU. This agreement gives you specific legal rights, and you may have other rights that vary from state to state.

Choice of Law and Jurisdiction

This Agreement shall be governed by and construed in accordance with the laws of the State of Michigan, without reference to any conflict of law principles. You agree that any litigation or other proceeding between you and Licensor in connection with the Training Materials shall be brought in the Michigan state or courts located in Port Huron, Michigan, and you consent to the jurisdiction of such courts to decide the matter. The parties agree that the United Nations Convention on Contracts for the International Sale of Goods shall not apply to this License. If any provision of this Agreement is held invalid, the remainder of this License shall continue in full force and effect.

Limitation of Claims and Liability

ANY ACTION ON ANY CLAIM AGAINST IPEXPERT MUST BE BROUGHT BY THE USER WITHIN ONE (1) YEAR FOLLOWING THE DATE THE CLAIM FIRST ACCRUED, OR SHALL BE DEEMED WAIVED. IN NO EVENT WILL THE LICENSOR'S LIABILITY UNDER, ARISING OUT OF, OR RELATING TO THIS AGREEMENT EXCEED THE AMOUNT PAID TO LICENSOR FOR THE TRAINING MATERIALS. LICENSOR SHALL NOT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, REGARDLESS OF WHETHER LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. WITHOUT LIMITING THE FOREGOING, LICENSOR WILL NOT BE LIABLE FOR LOST PROFITS, LOSS OF DATA, OR COSTS OF COVER.

Entire Agreement

This is the entire agreement between the parties and may not be modified except in writing signed by both parties.

U.S. Government - Restricted Rights

The Training Materials and accompanying documentation are “commercial computer Training Materials” and “commercial computer Training Materials documentation,” respectively, pursuant to DFAR Section 227.7202 and FAR Section 12.212, as applicable. Any use, modification, reproduction release, performance, display, or disclosure of the Training Materials and accompanying documentation by the U.S. Government shall be governed solely by the terms of this Agreement and shall be prohibited except to the extent expressly permitted by the terms of this Agreement.

IF YOU DO NOT AGREE WITH THE ABOVE TERMS AND CONDITIONS, DO NOT OPEN OR USE THE TRAINING MATERIALS AND CONTACT LICENSOR FOR INSTRUCTIONS ON RETURN OF THE TRAINING MATERIAL

Contents

Chapter 1: Introduction to IPv4/6 QoS Operation and Troubleshooting

Chapter 1: Introduction to IPv4/6 Quality of Service (QoS) Operation and Troubleshooting introduces the team of authors, consultants, and editors that completed this book and describes the book's purpose. This chapter also provides suggestions for the usage of this written work. This introductory chapter also covers a basic overview of Quality of Service operation and troubleshooting concerns.

About the Authors

Anthony Sequeira, CCIE No. 15626 (R&S), formally began his career in the information technology industry in 1994 with IBM in Tampa, Florida. He quickly formed his own computer consultancy, Computer Solutions, and then discovered his true passion—teaching and writing about Microsoft and Cisco technologies. Anthony is currently pursuing his second CCIE in the area of Security, and is a full-time instructor for the next generation of KnowledgeNet: www.StormWindLive.com. He recently achieved his VMware Certified Professional certification. When Anthony is not writing or lecturing about the latest innovations in networking technologies, you may find him flying a Cessna in the Florida skies.

Terry Vinson, CCIE No. 35457 (R&S), Terry Vinson is a highly experienced training consultant, specializing in documentation development, validation, verification and communications. For the last 10 years, Terry has worked in the private sector as a Senior Technology Consultant and Trainer for several consulting firms in Washington DC, Northern and Central Virginia. In this capacity, he has provided services to Major Metropolitan Health Systems, the Mexican Embassy, and the Executive Office of the President of the United States of America (EOP).

About the Technical Editors

Carl Yost Jr., CCIE No. 30486 (R&S), currently works as a Network Engineer/Director of I.T. for a health care company in Buffalo NY. He has worked in numerous roles in I.T. since 1998. Carl is currently preparing for the CCIE in Security while living with his wife and children in Western New York. When not surrounded by Cisco devices, Carl truly enjoys working with Redhat Linux.

Jason Gooley, CCNP, Jason is a highly motivated network engineer with over 17 years of experience in the communications industry. Based in Chicago, Jason currently manages the network for the nation's most famous next day carpet company. Jason is currently in the process of pursuing his CCIE certification for Routing and Switching while also expanding his knowledge in Unified Communications and Security.

About the Technical Consultants

Scott Morris, CCIEx4, CCDE, JNCIEx2, CISSP, Scott has been one of the most well-known figures in the IT industry for over 25 years. He has fulfilled a number of important roles within both the public and private sectors. As a Certified Cisco Systems Instructor (CCSI) and Juniper Networks Certified Instructor (JNCI), Scott has provided world-renowned CCIE training since 2002. He has delivered courses to a wide variety of audiences including internal training at Cisco Systems.

Vikram Malhi, CCIE #13890 (Voice), CCVP, Cisco IP Telephony Support Specialist, Cisco IP Telephony Operations Specialist, Cisco IP Telephony Design Specialist, Cisco Wireless LAN Design Specialist, with over 14 years of IP Telephony training and consulting experience and a wealth of technical certifications, Vik Malhi has proven that he is one of the top Cisco CCIE Voice Instructors and Consultants in the world! Vik was the first engineer to install CM 3.0 in Europe, has years of AVVID consulting and implementation experience and has been teaching and developing CCIE Voice Lab courses and self-study learning materials for over 4 years. Vik is responsible for updating, supporting and teaching IPexpert's Voice-related products, services and courses. Over the past 4 years Vik has been the cornerstone of IPexpert's

world-renowned CCIE Voice Lab product development and training and has assisted more CCIE Voice engineers in passing the lab than any other Instructor, worldwide!

Marko Milivojevic, CCIE #18427 (R&S SP), CCNP, CCDP, CCIP, Marko, a dual CCIE who has recently passed the CCIE R&S 4.0 lab, has spent 12 years designing and supporting some of Europe's largest service provider networks. He is accredited with designing the largest multi-service internet infrastructure in Iceland. He has been working with IPexpert over the past few years developing several self-study products, and will now be more-heavily involved in product development, product support and classroom training (throughout Europe, Australia and Asia) on full time basis.

About the Editor

Tiffany Pagan began her career in editing in 1997. Throughout her career, she has worked with several private individuals and companies such as Moffitt Cancer Center and Tampa General Hospital. Tiffany is currently working on writing her own series of short stories as well as working as an editor and personal assistant. Tiffany resides in Tampa, Florida with her husband and three beautiful children.

Who Should Read this Book?

This text has two primary audiences. The first audience is for those CCIE candidates that are searching for the most comprehensive and error-free materials available for the operation and troubleshooting of key technologies presented in the various tracks of the CCIE written and practical lab exams. These students should possess a home rack of equipment for CCIE-level command-line practice, they should possess an equipment emulator, or they should rent equipment from a company like www.proctorlabs.com. The authors and technical editors exhaustively tested all of the demonstrations found throughout the text and the important end of chapter Trouble Ticket challenges against all practice rack options described earlier. Where issues arise with popular equipment emulators, the text makes note. This book is the most remarkably thorough and technically accurate book written on the subject of Quality of Service to date.

The book's second audience is those readers that must support Quality of Service technologies in their actual network environments. This book serves as an amazing guide and reference for real-world problem solving within production networks that deploy these specific technologies. In fact, while many courses and texts purport to have certification success as a by-product of a thorough investigation of all protocols, this book actually succeeds in this approach.

How to Use this Book

This book breaks specific Quality of Service technologies down on a chapter-by-chapter basis for a complete and thorough review of this broad set of topics. Each chapter begins with a review of the selected technology. Following this, the text provides an intense examination of the operation of the protocols, including key aspects of troubleshooting for the specific technology. After this, the chapter presents some of the most common issues that can result with a particular technology, and most importantly, details the simple troubleshooting tools and steps that succeed for remediation.

Each chapter concludes with sample troubleshooting scenarios that provide a full walkthrough of a well-designed approach for troubleshooting each major issue. The text provides reference guides for the most popular and powerful **show** and **debug** commands for a specific technology.

Some chapters also contain sample Trouble Tickets on specific technologies. Readers may download initial configurations for these sample Trouble Tickets, or install them in a simple Graphical User Interface (GUI) on www.proctorlabs.com. These sample Trouble Tickets allow students to build confidence and expertise by actually troubleshooting issues in the Quality of Service domain presented in the chapter.

Students are encouraged to follow along on a rack of equipment for every section of every chapter. This really enhances and strengthens the learning process.

An Introduction to IPv4/6 Quality of Service

The modern day network is more and more a network that one can consider a “converged” network. This use of the word converged means that the network no longer features traditional forms of data traffic only. There might be Voice over IP traffic (VoIP), there might be Video traffic, teleconference traffic, all competing for the precious network resources in the enterprise. The more your network becomes “converged” with all of these differing traffic forms, the more it could benefit from the Quality of Service approaches and mechanisms covered in this course.

Note: Prior to converged networks, enemies to a high Quality of Service for traffic forms were handled by mechanisms of the independent network carrying the traffic form. For example, traditional Voice QoS problems were handled in the Public Switched Telephone Network (PSTN) network for the traditional Voice traffic.

It is important to remember that not all traffic behaves the same way. For example, bulk FTP transfers might be very insensitive to jitter (variations in delay), but Voice over IP traffic might be very sensitive to such variations! This makes the design and implementation of QoS solutions much more complex.

For data networks prior to the convergence of more sensitive (often termed fragile) traffic forms, a simple First In First Out (FIFO) approach to congestion management might make sense. Some packet loss might also be acceptable in this network due to bandwidth constraints. The data might certainly have “bursty” traffic patterns, but again, this was not a tremendous issue due to the insensitivity to QoS enemies like jitter and packet drop.

Modern converged networks like those that contain some amount of fragile Voice over IP (VoIP) traffic typically deserve intense Quality of Service scrutiny. This is because VoIP traffic is some of the most sensitive traffic that can course through the veins of the infrastructure. While the VoIP traffic might not be bursty in nature, and it might not require large amounts of bandwidth, it can suffer incredibly through competition with other traffic forms on the converged network. This gives rise to the need for the “managed unfairness” that can be used to describe Quality of Service (QoS).

Common Issues and their Solutions in Modern Networks

The previous section of this chapter mentioned some of the classic issues in the modern converged network. Here is a list of those previously mentioned, and more:

- Bandwidth shortages
- End to end delay (both fixed and variable end to end delays)
- Jitter
- Packet loss

Bandwidth shortages are common for many reasons. First, realize that the overall bandwidth available for a traffic flow is hindered by the weakest link in the full path that the traffic takes. In addition, at key aggregation points in the network design, bandwidth may fall short as many flows contend for a single link. Also, consider the incredible speed mismatches that may exist in the network. A switch might feature several Gigabit per second uplinks, but may only offer Fast Ethernet connections out to client workstations.

This book in its various chapters details solving bandwidth shortages, amongst solving the other issues listed. The text covers many tools and techniques. Understand that if a constant bandwidth shortage is the issue, upgrading the link is the only real answer. Quality of Service mechanisms are not magic. They may be used to solve sporadic issues causing bandwidth shortages, but often times, the only real solution is equipment and link upgrades.

A common QoS category of tools that can assist with bandwidth shortages is called congestion management. As you will learn in ***Chapter 6: Congestion Management***, these tools operate in the software queue of an interface. This is separate and distinct from the hardware queue. Examples of these tools include Modified Deficit Round Robin (MDRR), Weighted Fair Queuing (WFQ), Class-Based Weighted Fair Queuing (CBWFQ), and Low Latency Queuing (LLQ) just to name a few.

Another approach to attacking bandwidth shortages is to reduce either the headers placed on information payloads, or reduce the payloads themselves, or both. This is accomplished with various forms of compression. This book details those in ***Chapter 10: Compression***.

It is important that you realize that when we discuss the bandwidth of a link in Cisco networking, we are referring to the actual speed at which the interface send data. This value is often referred to as Line Rate (LR) or the Available Rate (AR). This might not be that value you see for the interface as configured with the **bandwidth** command. Remember, the **bandwidth** command is not setting the line rate of the interface. The bandwidth command merely reports the speed of the interface to processes running on the router. For example, EIGRP relies upon this command for the value of bandwidth in the calculation of that routing protocol's metric. There are times when you do not want this bandwidth value as reported by the **bandwidth** command to equal the true line rate (true bandwidth) of the interface. For example, you might want to claim the bandwidth of an interface is much greater than it actually is to cause EIGRP to consume more of the link bandwidth for its operations.

What command on a Cisco router actually sets the line rate of an interface? In the case of a Cisco serial interface, the command is **clock rate**. This command should be set for the interface in conjunction with the appropriate line rate as specified by the Internet Service Provider or WAN equipment configuration.

There are many types of delay that modern networks must experience. These include the following:

- Propagation delay – this is delay as a result of the speed of light in the media
- Serialization delay – this is the delay as the result of time it takes to clock all bits onto the wire
- Processing delay – this is a delay for the time spent by a router to take data from the input interface and move it to an output interface
- Packetization delay – this is the delay that occurs as the data is converted to packet form
- Queuing Delay – this is the delay as a result of time spent in the queues of the output interface

This list is certainly lengthy. In fact, at this point you may be impressed with the simple fact that the data ever actually makes it to the destination. Amazingly enough, this lengthy list of delays faced in the modern network is not complete. Other types of delays in the network can be measured – such as codec, compression, and shaping delays.

As mentioned earlier, a particular enemy of Voice over IP (VoIP) traffic is jitter. This is variations in delay. If these variations are somewhat predictable, then Voice engineers can use de-jitter buffers to mitigate problems, but if the jitter variations are very great, major issues can result in the Voice quality perceived by the callers.

Just like with bandwidth shortages, many potential solutions exist for combatting issues caused by delay in modern networks. In fact, just like bandwidth shortages, the only viable solution at times might be to upgrade link(s). Perhaps you can use an advanced queuing strategy in the software queue of the interface like MDRR, WFQ, CBWFQ, or LLQ. Or once again, perhaps compression of the packet headers, packet payloads, or both can be the answer. Or perhaps the solution lies in a clever combination of several of these QoS solutions.

Another issue to contend with is packet loss. A very common reason for this issue in networks is known as tail drop. Tail drop can occur when the hardware queue and the software queue of an interface completely fill. New packets arriving at the interface have nowhere to be queued and are dropped. Less common reasons why packet loss might occur in the network include:

- Input queue drops due to CPU congestion
- There is an Ignore condition as a result of no buffer space on the Cisco router
- There is an Overrun condition as a result of a congested CPU that cannot assign free buffer space
- There is a frame error such as a Cyclical Redundancy Check (CRC) error, or a runt, or a giant

Just like with the other enemies to Quality of Service in the network, packet loss can be a large problem for certain traffic forms, or its effects are not that important. It is probably no surprise that Voice over IP (VoIP) traffic is much more sensitive to packet loss than something like a bulk data transfer.

Once again, the ultimate solution might be link(s) upgrades. But also once again, the use of an advanced queuing strategy in the software queue of the interface might help. Another solution might be to use Weighted Random Early Detection (WRED). This technology randomly drops unimportant packets in an attempt to avoid the hardware and software queues from filling in the first place. ***Chapter 7: Congestion Avoidance*** details this exciting technology in great depth. Finally, traffic shaping or traffic policing might assist by limiting the rate of traffic sent or received. Chapters 8 and 9 cover these technologies in great detail.

QoS Defined

As stated earlier in this chapter, Quality of Service is all about providing managed unfairness to certain forms of traffic in the network compared to others. The real goal is to provide predictable performance (based on defined metrics) for certain classes of traffic. The ultimate solution is to construct a network that is capable of being over-provisioned with various traffic forms. In fact, networks that lack proper Quality of Service mechanisms are often chronically underutilized, but still suffer badly from the enemies we examined such as packet loss, jitter, and dramatic sporadic bandwidth shortages.

Creating a QoS Policy

When setting out to implement Quality of Service in your network environment, there is much careful planning that is required for a successful implementation. First, carefully identify all of the network traffic that flows through the environment and carefully define the traffic requirements of each form. You will quickly discover that you can group many of the classes of traffic into categories thanks to similar traffic requirements. Cisco Systems suggests that you use no more than 11 overall categorizations or classes of traffic for your Quality of Service design. Once this careful work is complete, you can begin to define the QoS policies that meet the class requirements. Obviously, the remainder of this book is designed to assist you with this task.

Examples of various traffic forms include:

- Transmission Control Protocol (TCP) flows that are adaptive in nature vs UDP and ICMP
- User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) flows that can be aggressive in nature
- Bulk transfers like File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP)
- Interactive, fragile traffic flows like Remote Desktop Protocol and Telnet
- Variable bit rate video traffic
- Voice over IP (VoIP)
- Management and control plane traffic

Voice over IP traffic is so fragile, and poor Quality of Service implementations are so noticeable that it deserves special mention here. Voice traffic (for example Real Time Transport Protocol (RTP)) is a benign constant bit rate traffic form that is very sensitive to bandwidth shortages, packet loss, and jitter. Cisco provides guidelines for its proper implementation. For example, total one-way latency should be approximately 150 ms, with total one-way jitter featuring a variation with 30 ms. One-way packet loss

should be less than 1%. Bandwidth requirements are slight and are typically up to 106 kbps per call with 150 bps per call for control traffic. This 150 bps guideline does not include layer 2 overhead.

Note: These guidelines are identical for Videoconferencing traffic forms except it requires higher bandwidth.

Chapter 1 Challenge: Multiple Choice Review

1-1. A critical link in your network infrastructure is at 80% capacity 100% of the time. What is the best approach to solve this problem?

a. Upgrade the link

b. Use an advanced queuing strategy

c. Compress payloads

d. Compress headers

1-2. What is the delay that describes the time spent by a router to take data from the input interface and move it to an output interface?

a. Processing Delay

b. Queuing Delay

c. Propagation Delay

d. Serialization Delay

e. Packetization Delay

1-3. What is the most common reason for packet loss in a network?

a. Ignore

b. Overrun

c. Frame Error

d. Tail Drop

1-4. You junior administrator returns from a Cisco Live conference and indicates that the absolute best solution for your Quality of Service problems is to simply overprovision the network. What should be a valid concern for you with this suggestion?

a. Only your edge devices may participate in this approach

b. Only your core devices are impacted by this approach

c. This solution requires too much administrative overhead

d. This solution actually has the net result of increasing overall network congestion

e. Recurring expenses tend to be higher than with a QoS-enabled network

1-5. Which of the following types of delay is a variable delay?

a. serialization delay

b. processing delay

c. queuing delay

d. propagation delay

Chapter 1 Challenge: Multiple Choice Review Answers

1-1. a

1-2. a

1-3. d

1-4. e

1-5. c

Chapter 2: Overall Quality of Service (QoS) Approaches

This chapter examines the overall approaches that can be taken to provide Quality of Service in an enterprise network. These three approaches are Best Effort (BE), Integrated Services (IntServ), or Differentiated Services (DiffServ). This chapter details each of these approaches for you.

Best Effort (BE)

The Best Effort approach relies heavily upon overprovisioning your bandwidth. This is because with this approach, you are not applying any special QoS tools like congestion management or congestion avoidance tools. In fact, the default congestion management tool used here is First In First Out (FIFO). The very first packets to arrive, are those packets that are first to be sent out. Notice what a problem this can be for Voice packets that might be stuck behind larger less important packets in the event that bandwidth becomes scarce during time of extreme congestion.

In order for this approach to be successful then, you must really focus on this overprovisioning. Unfortunately, when you overprovision in an attempt to avoid any congestion, you tend to have many links that are underutilized. Now you are in the bad situation of paying for bandwidth that you are not consuming.

Obviously, the beauty of Best Effort is its simplicity. It is a very scalable solution that requires little to no QoS expertise for IT staff members. In addition, the Cisco devices are not burdened with any overhead due to Quality of Service tools.

Integrated Services (IntServ)

Another overall approach to the application of Quality of Service in an enterprise is called Integrated Services (IntServ). Under the IntServ approach, applications signal to the network the QoS requirements that they possess. The idea here is that the appropriate resources can be guaranteed to the particular traffic form. This approach is often known as “Hard QoS”. This term makes sense in that these strict reservations can be made under the system.

What is the main protocol that makes the IntServ approach possible? Well, it is the VERY appropriately named Resource Reservation Protocol (RSVP).

While this approach to QoS sounds absolutely ideal, please keep in mind that it is not nearly as scalable as other approaches. All systems in the environment must support RSVP and be configured appropriately in order to honor the reservations. This is often difficult or even impossible to guarantee as you move from enterprise to enterprise.

Also, keep in mind that there is overhead associated with this Hard QoS. Messages and CPU cycles must be generated in order manage the reservations. While this overhead is certainly not all that significant, it still must be accounted for in the design.

This IntServ approach to QoS is detailed in ***Chapter 12: Resource Reservation Protocol (RSVP)***.

Differentiated Services (DiffServ)

The major focus of this book is on the Differentiated Services approach to Quality of Service. That is because this is a proven method that is very scalable. Under this approach, the network recognizes different classes of traffic (that you configure) and provides different levels of QoS to these traffic forms.

The issue with the DiffServ approach is that it is very complex. Thankfully, resources like this book exist so that you can master it.

DiffServ is broken down into the following categories. Notice this list also demonstrated the appropriate chapters of coverage in this text:

- **Classification** – the process of identifying traffic forms in the network; class-maps are used for this, and often times, Network Based Application Recognition (NBAR)
  Chapter 4: Classification
- **Marking** – placing identifying marks on traffic; can be accomplished at layer 2, layer 2.5 (MPLS), and Layer 3
  Chapter 5: Marking
- **Congestion Management** – using "fancy queuing" techniques to avoid problems caused by FIFO
  Chapter 6: Congestion Management
- **Congestion Avoidance** – randomly dropping packets (influenced by priority markings) in order to attempt to avoid congestion
  Chapter 7: Congestion Avoidance
- **Shaping** – buffering traffic to ensure a certain transmit speed
  Chapter 8: Traffic Shaping
- **Policing** – dropping traffic to ensure a certain transmit or receive speed
  Chapter 9: Traffic Policing
- **Link Efficiency** – manipulating traffic through compression or size manipulation in order to ensure acceptable QoS over very slow links
  Chapter 10: Compression
  Chapter 11: Link Fragmentation and Interleaving (LFI)

Note: These mechanisms as they apply to a Cisco Catalyst switch are covered in Chapter 13: Catalyst Quality of Service (QoS).

Chapter 2 Challenge: Multiple Choice Review

2-1. Which overall approach to QoS is considered the least scalable?

a. DiffServ

b. Congestion Management

c. Best Effort

d. IntServ

2-2. Which overall approach to QoS is considered the most complex?

a. DiffServ

b. Congestion Management

c. Best Effort

d. IntServ

2-3. Which DiffServ component buffers traffic in order to ensure a certain transmit speed?

a. Congestion Management

b. Congestion Avoidance

c. Traffic Shaping

d. Traffic Policing

Chapter 2 Challenge: Multiple Choice Review Answers

2-1. d

2-2. a

2-3. c

Chapter 3: Configuration and Monitoring Tools

While there are many graphical user interface (GUI) tools that can assist you in your configuration and monitoring of Quality of Service in your enterprise, this chapter focuses on the Modular Quality of Service Command Line Interface (MQC). This command line approach to the configuration of QoS features is critical. Even if you plan to automate the QoS configurations on your devices with a tool like AutoQoS, you should master the MQC in order to be able to edit and fine-tune the configurations created automatically for you.

Configuration and Monitoring Tools Technology Review

The MQC approach consists of three important steps:

1. The classification of traffic into class maps – ***Chapter 4: Classification*** explores this topic in detail. This chapter will completely explore the class map itself. We create class maps with the **class-map** command.
2. The QoS treatment of the traffic classifications – policy maps are used to configure the actual QoS mechanisms on the traffic that has been classified. The **policy-map** command creates a policy map. These policy maps reference the class maps created in step 1.
3. The application of the policy map to an interface – applying the policy map is accomplished with a service policy. You create a service policy using the **service-policy** command.

Note: In this chapter, our emphasis is on the MQC itself. Please understand that configuration examples will be shown, but those specific QoS configurations are taught in other chapters of this text. Please try and focus on the MQC itself here and do not be distracted by the additional commands.

Class Maps

Class maps consist of a case sensitive name, a series of match commands, and a **match-all** or **match-any** instruction.

Note: The default match all or match any condition is a **match-all**.

As this section will demonstrate shortly, class maps can nest in other class maps.

Let us now review the configuration syntax of these powerful class maps:

```
IPexpert1(config)# class-map [match-all | match-any] cmap_name
IPexpert1(config-cmap)# match condition
IPexpert1(config-cmap)# match not condition
IPexpert1(config-cmap)# match class-map cmap_name
IPexpert1(config-cmap)# match any
IPexpert1(config-cmap)# description my_optional_description
```

Here is an example of a class map configuration.

```
IPexpert1(config)# class-map match-all CM_EXAMPLE
IPexpert1(config-cmap)# match protocol ip
IPexpert1(config-cmap)# match qos-group 3
IPexpert1(config-cmap)# match access-group 110
```

Note: This course will teach about these specific statements, such as **qos-group**, where appropriate in the chapters that follow.

Here is an example of nesting class maps inside of each other:

```
IPexpert1(config)# class-map match-all CM_SAMPLE_CHILD_CM
IPexpert1(config-cmap)# match protocol ip
```

```
IPexpert1(config-cmap)# match qos-group 3
IPexpert1(config-cmap)# exit
IPexpert1(config)# class-map match-any CM_SAMPLE_PARENT_CM
IPexpert1(config-cmap)# match class-map CM_SAMPLE_CHILD_CM
IPexpert1(config-cmap)# match input-interface FastEthernet 0/0
```

How can we verify the class maps that we create at the command line? It is simple thanks to the **show class-map** command.

show class-map [*class_name*] [| {begin | exclude | include } exp]

An interesting feature of Cisco's QoS implementation is a default class map that Cisco creates for us automatically. This default class map acts like a "catch-all". It automatically matches all traffic forms that we do not explicitly match with our user defined class maps. This class map has the name – **class-default**. Should you want to configure a QoS treatment to this catch-all class map, you may reference it in your policy map and apply the appropriate QoS. Below is an example of this class map. Notice there are no quality of service settings configured on this router at all. The router is in a default configuration. Sure enough, there is an existing class map on the device with a match all condition!

```
R4#show class-map
 Class Map match-any class-default (id 0)
   Match any
```

Policy Maps

Remember, the MQC relies upon policy maps to dictate the QoS controls that you want to take on traffic that is defined in the class maps. Just like with a class map, a policy map consists of a case-sensitive name. It also contains references to the traffic classes that we want to configure. On most IOS versions, up to 256 classes can be referenced in a single policy map. Just as we saw with class maps, policy maps may be nested in each other for powerful QoS configurations.

Here is a look at the configuration syntax:

```
IPexpert1(config)# policy-map policy_map_name
IPexpert1(config-pmap)# class {class-name | class-default}
IPexpert1(config-pmap-c)#
```

Notice that much is accomplished in policy map configuration mode. It is in policy map class configuration mode where the actual QoS settings are applied. Here is an example:

```
IPexpert1(config)# class-map CM_SAMPLE_PM
IPexpert1(config-cmap)# match access-group 110
IPexpert1(config-cmap)# class-map CM_SAMPLE2_PM
IPexpert1(config-cmap)# match access-group 112
IPexpert1(config-cmap)# !
IPexpert1(config-cmap)# policy-map PM_SAMPLE
IPexpert1(config-pmap)# class CM_SAMPLE_PM
IPexpert1(config-pmap-c)# bandwidth 100
IPexpert1(config-pmap-c)# class CM_SAMPLE2_PM
IPexpert1(config-pmap-c)# bandwidth 200
```

As stated earlier, policy maps do indeed have the ability to be nested inside other policy maps. When we engage in this nesting behavior, we refer to the policy as a hierarchical policy. This is typical done to configure multiple treatments to QoS. For example – we might want to traffic shape all traffic to 3 MB; and then inside that 3 MB shaped traffic, guarantee Web traffic at least 1 MB of bandwidth.

Remember, we use a service-policy (normally done with interfaces) to assign the QoS policies that we define inside the policy-map.

With the nesting of policy maps, there are some restrictions that you should be aware of. For example:

- the **set** command is not supported on the child policy
- the **priority** command can be used in either the parent or the child policy, but not both
- the **fair-queue** command cannot be used in the parent

Here is an example of nesting policy maps:

```
class-map CM_ALL_TRAFFIC
  match any
!
class-map CM_WEB
  match protocol http
!
policy-map PM_CHILD
  class CM_WEB
    bandwidth 1000
!
policy-map PM_PARENT
  class CM_ALL_TRAFFIC
    shape 3000000
service-policy PM_CHILD
```

In order to verify your policy map, it is very simple thanks to the following **show** commands:

show policy-map

show policy-map interface int_name

Note: While the show policy-map command allows you to see your policy map construction and syntax, this command does not show the effectiveness of the policy map. For example, how many packets are in a traffic class and are actually receiving a particular QoS treatment. For this verification, we have the **show policy-map interface** command.

The Service Policy

Remember, we use a service policy in order to dictate where we apply a QoS policy. The syntax of this command is as follows:

```
IPexpert1(config-if)# service-policy {input | output} pm_name
```

For example:

```
IPexpert1(config-if)# service-policy output PM_TEST
```

Note: A common misconception regarding service policies is the fact they must be applied only to interfaces. As you will see throughout this text, there are indeed other places these important commands may be issued. This includes within policy maps for nesting purposes!

The Operation and Troubleshooting of Configuration Tools

While it is very important to verify the correct construction of your class maps, policy maps, and service policies with the appropriate show commands described in the previous section, one command really dominates when it comes to troubleshooting your MQC and resulting Quality of Service configuration. This command is the **show policy-map interface** command.

The power and importance of this command cannot be emphasized enough. This command allows you to not only confirm the proper configuration of the MQC elements, but it also allows you to see the effectiveness of the QoS mechanisms inside the modular configuration. For example, how many packets are being matched by your class map and are they being given the priority treatment you expect.

Here is an example of this imperative command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      91 packets, 4230 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 91/4230
      bandwidth 35% (540 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        class     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                  pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        0               0/0               0/0              0/0                 20
40  1/10
        1               0/0               0/0              0/0                 22
40  1/10
        2               0/0               0/0              0/0                 24
40  1/10
        3               0/0               0/0              0/0                 26
40  1/10
        4               0/0               0/0              0/0                 28
40  1/10
        5               0/0               0/0              0/0                 30
40  1/10
        6              91/4230            0/0              0/0                 32
40  1/10
        7               0/0               0/0              0/0                 34
40  1/10
```

This example is taken from ***Chapter 7: Congestion Avoidance***. At this point, observe the following that are easily confirmed through the output:

- Our policy map is assigned to the Serial0/2/0 interface of R6 (using a service policy of course)
- The policy map is named AVOIDANCE
- The service policy is an outbound policy
- There is a match-all type class map named TELNET in use
- This class map uses Network Based Application Recognition (NBAR) in order to match Telnet packets; NBAR is detailed in ***Chapter 4: Classification***
- 91 packets are matching this class map

Notice that we can easily see this information and that it only scratches the surface of the powerful information revealed by this command.

Troubleshooting Class Maps

It was once asked, what is in a name? Well, in the case of a class map, very much indeed. Remember, the class map is identified by a name and that this name is case sensitive. While you can add optional descriptions to your class maps, we recommend using a very descriptive, all uppercase name. You might even want to consider a naming convention for your class maps. For example:

```
CM_EMAIL_TRAFFIC
CM_SCAVENGER_TRAFFIC
```

Note: In the CCIE lab exam, you might be asked to use specific names for these objects. Be sure you follow these names exactly in order to achieve full credit for the task completion.

Another important point to keep in mind with your class maps is the fact that they are of a match all or match any type. We recommend always specifying **match-all** or **match-any** keywords during their construction to ensure the appropriate usage.

Here is an example of match all logic in a class map:

```
IPexpert1(config)# class-map match-all CM_EXAMPLE_MATCH_ALL
IPexpert1(config-cmap)# match protocol telnet
IPexpert1(config-cmap)# match access-group 110
```

Notice that in order to be properly categorized in this class map, the traffic must be Telnet, *and* it must match the criteria specified in access list 110.

Here is an example of match any login in a class map:

```
IPexpert1(config)# class-map match-any CM_EXAMPLE_MATCH_ANY
IPexpert1(config-cmap)# match protocol telnet
IPexpert1(config-cmap)# match protocol icmp
```

Here, the traffic will be categorized in this class map if it is Telnet traffic, *or* of it is ICMP traffic.

Notice a major a point of potential trouble with a class map. Examine this example:

```
IPexpert1(config)# class-map match-all CM_EXAMPLE_WILL_NOT_WORK
IPexpert1(config-cmap)# match protocol telnet
IPexpert1(config-cmap)# match protocol icmp
```

This class map will never match traffic. Why? The answer is simple, traffic will never be Telnet and ICMP traffic at the same time!

Troubleshooting Policy Maps

Similar to class maps, ensure the case sensitive name is used properly in your service policy that will apply the policy map. Also, be very careful to use the correct class map names in your policy map. Finally, remember the rules regarding policy map nesting:

- the **set** command is not supported on the child policy
- the **priority** command can be used in either the parent or the child policy, but not both
- the **fair-queue** command cannot be used in the parent

Troubleshooting Service Policies

With service policy troubleshooting, we need to ensure the correct name is referenced for the policy map. Also, of particular importance is the location and direction where the policy map is applied. Ensure you are:

- On the correct router where the policy map needs to be applied
- On the correct interface where the policy map needs to be applied
- Applying the service policy in the correct direction

Fortunately, in almost all cases, the latest Internetwork Operating Systems (IOSs) from Cisco will provide an error message and not allow you to apply a service policy in an appropriate direction for a specific feature. For example, if you attempt to apply traffic shaping in the inbound direction, the service policy application will return an error and not apply. Of major concern, however, is when you are applying a service policy for a policy map that can function in either the inbound or outbound direction. An excellent example would be traffic policing. These topics are covered in detail in chapters 8 and 9, respectively.

Chapter Challenge: Configuration Tools Trouble Tickets

The following section includes a Trouble Ticket associated with configuration tool scenarios. Be sure to load the appropriate initial configuration files in order to perform the ticket.

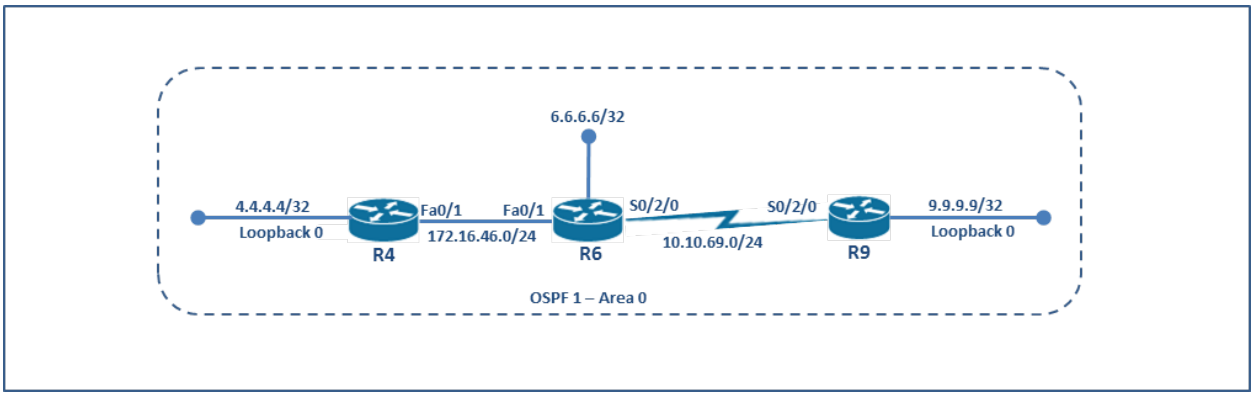

Figure 3-1: A Basic Quality of Service Topology

Trouble Ticket #1

Your supervisor has asked you to configure R4 to produce the output here:

```
R4#show policy-map interface
 FastEthernet0/1

  Service-policy output: PM_EMAIL_WEB

    Class-map: CM_EMAIL (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol smtp
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol pop3
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol imap
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: CM_WEB (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol http
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: class-default (match-any)
      15 packets, 1363 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
R4#
```

Chapter Challenge: Configuration Tools Trouble Ticket Solutions

The following section includes solutions to the Trouble Ticket associated with configuration tool scenarios.

Trouble Ticket #1

Your supervisor has asked you to configure R4 to produce the output here:

```
R4#show policy-map interface
 FastEthernet0/1

  Service-policy output: PM_EMAIL_WEB

    Class-map: CM_EMAIL (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol smtp
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol pop3
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol imap
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: CM_WEB (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol http
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: class-default (match-any)
      15 packets, 1363 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
R4#
```

Step 1 - Fault Verification:

What is the current output of this show command on R4?

```
R4#show policy-map interface

R4#
```

Here we can see there are no policy maps assigned to any interfaces on this device. Are there any class maps or policy maps created at all?

```
R4#show class-map
 Class Map match-any class-default (id 0)
   Match any
```

```
R4#show policy-map

R4#
```

Here we can see just the default class-map of class-default. We must create the appropriate class maps, the policy map, and the service policy in this case.

Step 2 - Fault Remediation:

First, we examine the class maps that we must create based on the show output:

```
Class-map: CM_EMAIL (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol smtp
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol pop3
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol imap
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: CM_WEB (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol http
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: class-default (match-any)
      15 packets, 1363 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Notice that we must create a CM_EMAIL match any class map that matches protocols of SMTP, POP3, or IMAP. We must also create a CM_WEB class map that matches on HTTP traffic. This is a match-any class map as well, although this is not relevant in practice since there is only one match condition. In order to be precise against the instructions, however, we will ensure it is a match-any. Notice also that we have the class-default class, but it is in its default state so there is nothing that we need to configure there.

Here is the configuration of the class maps:

```
R4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#class-map match-any CM_EMAIL
R4(config-cmap)#match protocol smtp
R4(config-cmap)#match protocol pop3
R4(config-cmap)#match protocol imap
```

```
R4(config-cmap)#exit
R4(config)#class-map match-any CM_WEB
R4(config-cmap)#match protocol http
R4(config-cmap)#end
R4#
%SYS-5-CONFIG_I: Configured from console by console
R4#
```

Next, we examine the details of the policy map from the output.

```
Service-policy output: PM_EMAIL_WEB
```

By examining the class maps, we can see that there is no particular QoS treatments applied. In this case, we just need to ensure to name the service policy correctly and reference the appropriate class maps:

```
R4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#policy-map PM_EMAIL_WEB
R4(config-pmap)#class CM_EMAIL
R4(config-pmap-c)#exit
R4(config-pmap)#class CM_WEB
R4(config-pmap-c)#end
R4#
```

Now to examine the service policy details:

```
R4#show policy-map interface
 FastEthernet0/1

  Service-policy output: PM_EMAIL_WEB
```

We can see that the service policy is applied in the outbound direction on the FastEthernet0/1 interface.

Let us now engage in the final configuration:

```
R4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface fastethernet0/1
R4(config-if)#service-policy output PM_EMAIL_WEB
R4(config-if)#end
R4#
%SYS-5-CONFIG_I: Configured from console by console
R4#
```

Step 3 - Remediation Verification:

In this case, verification is very simple. We run the show policy-map interface command and compare the results carefully to the Trouble Ticket output:

```
R4#show policy-map interface
 FastEthernet0/1
```

```
  Service-policy output: PM_EMAIL_WEB

    Class-map: CM_EMAIL (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol smtp
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol pop3
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match: protocol imap
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: CM_WEB (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps
      Match: protocol http
        0 packets, 0 bytes
        5 minute rate 0 bps

    Class-map: class-default (match-any)
      23 packets, 2132 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
R4#
```

Notice that the output matches (with the exception of the packet counts, of course) and we have successfully solved the ticket.

Chapter 4: Classification

We know that classification of traffic is a key element for implementing proper Quality of Service (QoS). In this chapter, we will discuss classification in much detail. We will also examine best practices and tools that are available to assist with this important aspect of QoS.

Classification Technology Review

As we discussed in Chapter 2, a key element in Quality of Service is the classification of different traffic forms in order to provide varying levels of service to these classifications. You have already learned that class maps are your key tool for classification at the command line. You have also already gotten a preview of how Network Based Application Recognition (NBAR) can play a huge role in these class maps. In this chapter, we expand dramatically upon this important tool of NBAR.

Network Based Application Recognition (NBAR)

The amazing Network Based Application Recognition set of tools has two jobs in the network. We can use it to assist with QoS classification, or we can use the protocol to engage in protocol discovery (analysis). You might even be using NBAR for both jobs concurrently in your network.

It will not surprise the reader that the focus of NBAR in this book is for the QoS classification inside the MQC. This is accomplished using the **match protocol** command within a class map. Cisco makes this very easy and powerful for us to implement, as there are pre-defined definitions in the IOS of popular protocols. You can even extend the classification capabilities of NBAR using by downloading custom PDLMs to your router for other automatic traffic classification capabilities.

We stated here that the popular **match protocol** command in a class map is how we use the classification capabilities of NBAR. But how can we configure the other job of NBAR (should we want to). That was the protocol analysis job. It is done this way:

```
IPexpert1(config-if)# ip nbar protocol-discovery
```

Note: This job of NBAR is ***not*** a requirement for matching on protocols in class maps.

In order to view the results of NBAR's protocol analysis, use the following show command:

show ip nbar protocol-discovery

Note: The protocol analysis job of NBAR is actually utilized by AutoQoS for Enterprise. AutoQoS for Enterprise uses the feature in order to discover the types of traffic that are running in the enterprise. AutoQoS then uses this information to set the QoS appropriately given those traffic forms. ***Chapter 14: AutoQoS*** details this for you.

While NBAR is very powerful, there are some points that you should be aware of:

- NBAR requires Cisco Express Forwarding (CEF) in order to function
- NBAR can operate with non-fragmented packets only
- NBAR cannot be used with MPLS
- NBAR is restricted to IP traffic only
- NBAR is not supported on logical interfaces like EtherChannel or dialer interfaces
- NBAR cannot be used on interfaces engaged in tunneling or encryption

You might be curious of some of the NBAR classification capabilities. Here are some of the exciting ways in which we can use NBAR:

- To classify applications that use static TCP and UDP port numbers
- To classify applications that use dynamic TCP and UDP port numbers
- To classify Non-TCP and Non-UDP IP protocols; for example, ICMP, EIGRP, or GRE
- To engage in deep packet inspection– for example Web traffic carrying a certain payload type (.jpg)

As stated earlier, when we use NBAR to assist us with classification, we use the match protocol command in a class map. Here are some examples:

```
class-map CM_NBAR1
 match protocol citrix


class-map CM_NBAR2
match protocol http url /media/sample*


class-map CM_NBAR3
match protocol http mime "*jpeg"


class-map CM_NBAR4
match protocol fasttrack file-transfer "*.mpeg"


class-map CM_NBAR5
match protocol rtp audio
```

Should you need to customize an NBAR definition for a protocol, this is possible. For example, perhaps your Web server is located at port 8080 instead of the default port 80. Here is how you can configure this:

```
IPexpert1(config)# ip nbar port-map protocol-name [tcp | udp] port-number
```

For example:

```
IPexpert1(config)# ip nbar port-map http tcp 80 8080
```

In order to verify your custom port mappings in NBAR, simply use:

show ip nbar port-map

As described earlier, PDLMs allow you to upgrade the built-in definitions of protocols that NBAR is aware of on a Cisco router or switch. Cisco calls PDLMs not already installed in the IOS non-native PDLMs. After downloading these definitions from cisco.com, installing on the router is simple:

```
IPexpert1(config)# ip nbar pdlm flash://directconnect.pdlm
```

In order to verify, use:

show ip nbar pdlm

What if there is a custom protocol in use in your environment for some custom application your enterprise has created? Well, NBAR does allow for the creation of new protocol definitions. Once you create your own protocol, you can easily reference it in **match protocol** commands or you **port-map** syntax.

Here is an example of the creation of a custom protocol in NBAR:

```
ip nbar custom MYCOOLAPP 8 ascii SAMPLE tcp range 2000 2999
```

The Operation and Troubleshooting of Classification

At this point, you realize just how important traffic classification can be. Realize that it is typically performed as close to the network edge as possible. This is what is referred to as the trust point in the network. We attempt to do this, so that the traffic can be quickly marked as it enters the network, and can be provided the appropriate QoS treatment at each and every hop as the traffic proceeds through the network cloud.

Classifying Traffic With NBAR

By far the most common issue with the use of NBAR is engineers not understanding the difference between the protocol discovery aspect of the tool, and using the tool for classification within a class map. These roles are separate and distinct, and one is not needed in order for the other to function.

A very large issue to watch out for with NBAR is the fact that Cisco Express Forwarding (CEF) is required for it to run. While in production environments this is typically not a concern, since NBAR is enabled on almost all Cisco equipment by default, be sure to be careful in exam environments. In exam environments, it I highly likely that required configurations may not be in place.

Classifying Traffic Without NBAR

Remember, there are many other methods you can use in order to classify traffic thanks to the flexibility of options within class maps. These options include, but are not limited to:

- Access lists
- CoS markings (marking traffic is covered in detail in chapter 5)
- IP Precedence markings
- DSCP markings
- A QoS Group markings
- MPLS Traffic Class markings
- Another class-map itself
- The Frame Relay DE bit setting
- The input interface for the packet itself

Examine the example below where traffic is classified based on an access control list:

```
R4#configure terminal
R4(config)#class-map CM_ALIST
R4(config-cmap)#match access-group name AL_WEB
R4(config-cmap)#exit
R4(config)#ip access-list extended AL_WEB
R4(config-ext-nacl)#permit tcp host 10.10.10.10 host 10.20.20.10 eq 80
R4(config-ext-nacl)#end
R4#
```

Here is an example of classifying traffic based on IP Precedence:

```
R4#configure terminal
R4(config)#class-map CM_IPPREC
```

```
R4(config-cmap)#match ip precedence 2 3
R4(config-cmap)#end
R4#
```

And finally, an example of classifying traffic based on IPv6 values:

```
R4#configure terminal
R4(config)#class-map CM_IPV6
R4(config-cmap)#match protocol ipv6
R4(config-cmap)#match precedence 5
R4(config-cmap)#end
R4#
```

Chapter Challenge: Multiple Choice

4-1. What is the purpose of the command **ip nbar protocol-discovery**?

a. To classify traffic in preparation for a QoS treatment
b. To specify which protocols are running on your system
c. To eliminate certain protocols from classification
d. To analyze the traffic passing through an interface

4-2. What feature of QoS does NBAR protocol discovery assist with?

a. Protocol classification for QoS treatment
b. AutoQoS for Enterprise
c. Congestion Management
d. Traffic Policing

4-3. What statement regarding NBAR is not true?

a. NBAR requires Cisco Express Forwarding (CEF) in order to function
b. NBAR can operate with non-fragmented packets only
c. NBAR can be used with MPLS
d. NBAR is not supported on logical interfaces like EtherChannel or dialer interfaces
e. NBAR cannot be used on interfaces engaged in tunneling or encryption

4-4. What statement is used for the protocol classification functionality of NBAR?

a. match traffic
b. match protocol
c. match access-group
d. match traffic-class

4-5. Which of the following options can be used within a class map for classifying traffic?

a. access list
b. QoS Group
c. NBAR
d. Input interface
e. All of these options are correct

Chapter Challenge: Multiple Choice Solutions

4-1. d

4-2. b

4-3. c

4-4. b

4-5. e

Chapter 5: Marking

In order to facilitate the popular Differentiated Services approach to Quality of Service, traffic must first be classified. Once the traffic is classified, it can be marked so that the traffic is easily recognized and treated with the appropriate Quality of Service treatment at each hop in the network infrastructure. In this chapter, all forms of marking that you should master will be explored.

Layer 2 Marking Technology Review

ISL trunks feature a method for marking traffic. There a 4-bit USER code field used to carry a "class of service" CoS marking. Meanwhile, in 802.1Q, a 3-bit PRIORITY field is used to carry this CoS marking. These bits are often referred to as 802.1p since this is the name of the committee that outline their usage.

The possible class of service markings for traffic are detailed in Figure 5-1.

Marking	Binary	Service Level
0	000	Routine
1	001	Priority
2	010	Immediate
3	011	Flash
4	100	Flash Override
5	101	Critical
6	110	Internetwork Control
7	111	Network Control

Figure 5-1: Possible CoS Markings

It is very important to note that Cisco recommends we never assign traffic with a CoS of 6 or 7. These markings are reserved for control traffic created and used by the routers themselves. Voice over IP traffic is typically assigned a CoS marking of 5. In fact, a Cisco VoIP phone marks the voice packets this way automatically as they exit the phone.

In Layer 2 Frame Relay, there are three different bits used in the Frame Relay header for QoS:

FECN – Forward Explicit Congestion Notification - if the frame is being carried in the Service Provider cloud and there is congestion experienced, the DCE (most often a Frame Relay Switch) can set the FECN bit and send the frame forward in the cloud. This notifies the receiver that there was congestion experienced during the transmission.

BECN – Backward Explicit Congestion Notification - if a DTE in the Frame network receives a frame with the FECN bit set, it can respond in a clever way. It can generate a test frame and set the BECN bit. It then sends this frame backward in the cloud to the original sender. This way the sender can be notified of the congestion, and (hopefully), slow down the rate at which it sends traffic.

DE – Discard Eligible - when there is congestion on the line, the network must decide which frames to discard in order to free the line. Discard Eligibility provides the network with a signal to determine which frames to discard. The network will discard frames with a DE value of 1 before discarding other frames.

Note: While ATM is not a topic in the CCIE R&S lab exam, for completion's sake in this discussion of Quality of Service markings, it should be noted that in ATM there is a Cell Loss Priority (CLP) bit. This bit is very comparable to the DE bit in Frame Relay networks.

Layer 2.5 Marking Technology Review

MPLS is often referred to as Layer 2.5. This is because of the placement of the MPLS header between the traditional layer 2 and layer 3 headers. With this said, let us review the marking capabilities at this layer of the OSI model.

The designers of MPLS technology reserved three bits that they called the Experimental Bits. As the name clearly implies, they did not have a clear vision of how these bits would be used. Most vendors immediately began copying the CoS or IP Precedence (Layer 3) bit settings into this field. In fact, this became very much the norm.

As a result, the standards bodies have renamed these bits to Traffic Class and the intent of these three bit options is the communicate the QoS marking.

Layer 3 Marking Technology Review

There is an 8-bit byte in the IP packet header called the Type of Service (ToS) byte. This field was always designed to carry QoS marking information, but it was not completely clear how it would do so.

The first approach to gain any traction was called IP Precedence. This marking strategy uses the first three leftmost bits for marking. Notice this was a common sense approach that is consistent with the CoS marking strategy used at Layer 2.

Figure 5-2 shows the markings used for IP Precedence.

Marking	Binary	Service Level
0	000	Routine
1	001	Priority
2	010	Immediate
3	011	Flash
4	100	Flash Override
5	101	Critical
6	110	Internetwork Control
7	111	Network Control

Figure 5-2: IP Precedence Markings

Once again, like in CoS, the highest possible markings, in this case IP Precedence 6 and 7, should not be used. These are reserved for router control packets and similar "internal" traffic forms.

As networks became more and more complex, carrying more and more different forms of traffic, it quickly became apparent that there needed to be more options for classification.

The standards makers quickly devised a scheme to use more of the bits in the ToS byte for marking traffic at Layer 3. This approach would just need to be backward compatible with IP Precedence. In other words, if Voice over IP traffic is marked for priority under the new approach, it would need to translate to IP Precedence 5 in order to be compatible.

The approach created was the Differentiated Service Code Point (DSCP). This approach utilizes the first 6 high-order bits of the Type of Service (ToS) byte in the IP header. But what about the last two bits of this field. Surely they can be put to good use under this system? These last 2 bits of the ToS byte are for congestion notification purposes. Specifically, these bits are now called the Explicit Congestion Notification (ECN) bits. When these ECN capabilities are successfully negotiated between two IP speakers, an ECN-aware router may set a mark in the IP header instead of dropping a packet in order to

signal impending congestion. The receiver of the packet echoes the congestion indication to the sender, which reduces its transmission rate as though it detected a dropped packet.

Figure 5-3 show the possible uses of the important ToS field in the IP header.

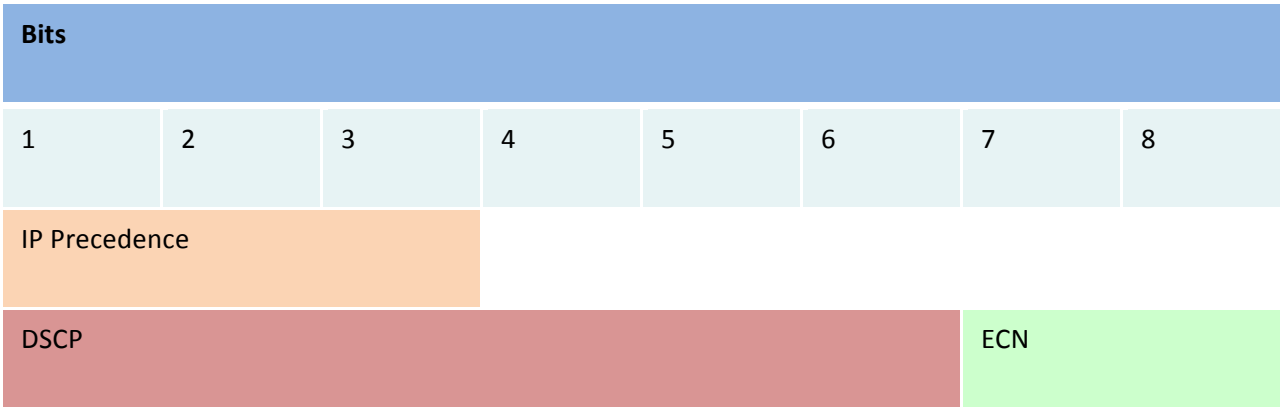

Figure 5-3: The Layer 3 Markings Possible in the ToS Byte of the IP Header

The standards makers went one important step further when they defined these new marking capabilities. They provided guidance in their usage. These suggested usages are called Per Hop Behaviors (PHBs). Each marking receives a name and a suggested usage. Here are some of the PHBs you should be familiar with:

- The Default PHB – this is when the DSCP marking is left at its default; the six DSCP bits are left at 000000
- The Assured Forwarding PHBs – (AF1, AF2, AF3, and AF4) – these markings are compatible with the IP Precedence markings of 1, 2, 3, and 4 respectfully; the DSCP bit settings are 001XXX, 010XXX, 011XXX, and 100XXX
- The Expedited Forwarding PHB - Forwarding (EF) – this marking is used for VoIP; notice that it is compatible with an IP Precedence of 5; the DSCP bit setting is 101110; this has a decimal representation of 46
- The Class Selector PHBs – these are for "pure" compatibility with the earlier IP Precedence; in this approach, only the first three bit settings of the six DSCP bits are utilized; for example, Class Selector 5 is 101000 (DSCP decimal value of 40) which is perfectly compatible with IP Precedence 5

Note: The Class Selector 5 marking of DSCP 40 explains why some Catalyst switches mark CoS 5 traffic with a Layer 3 DSCP value of 40. They are marking the traffic at Layer 3 to be compatible with IP Precedence-only aware routers.

The Assured Forwarding classes do warrant some more explanation. Earlier we state these classes function as follows - the DSCP bit settings are 001XXX, 010XXX, 011XXX, and 100XXX respectfully. What about the usage of the three bits following the three leftmost bits? These bits are used as follows:

dd0 where dd is drop probability

These drop probabilities are 01, 10 , and 11 for LOW, MEDIUM, and HIGH drop probabilities. Figure 5-4 provides a complete breakdown of these classes, therefore:

PHB	Low Drop Preference	Medium Drop Preference	High Drop Preference
Class 1	AF11 (10)	AF12 (12)	AF13 (14)
Class 2	AF21 (18)	AF22 (20)	AF23 (22)
Class 3	AF31 (26)	AF32 (28)	AF33 (32)
Class 4	AF41 (34)	AF42 (36)	AF43 (38)

Figure 5-4: The Assured Forwarding Classes

Note: Remember the Expedited Forwarding (EF) class? The markings were 101110 giving a decimal value of 46. This is the highest priority marking we should provide traffic and it is reserved for Voice over IP. Notice how with the Assured Forwarding code points, the (11) indicates HIGH DROP probability. With the special Expedited Forwarding (EF) class, it means the exact opposite, it means NO DROP.

Figure 5-5 provides a look at some of the various PHBS and their usage.

Classification DSCP	Classification CS	Application Type
DSCP 46 (EF)	CS 5	Voice Bearer
DSCP 32	CS 4	Streaming Video
DSCP 26 (Previously marked as AF31)	CS 3	Voice/Call Signaling
DSCP 8	CS 2	Network Management
DSCP 0	CS 1	Scavenger

Figure 5-5: DSCP PHB Examples

Various other Marking Topics Technology Review

What about traffic marking capabilities in IPv6? The IPv6 header possesses a Traffic Class byte that works just like ToS field used in the IPv4 header for DSCP. Thus, IPv6 is capable of using the exact same DSCP system.

The IPv6 header also adds a 20-bit Flow Label. This permits transit routers to be able to quickly identify a packet as belonging to a particular flow without looking deep in the packet.

A final marking opportunity that should be presented for completeness is the QoS Group marking capability in Cisco routers. This marking is used on the local router only, and the value is not transmitted to other devices. This provides a method of “marking” traffic locally, without manipulating the packet in any way.

The Operation and Troubleshooting of Class-Based Marking

Class-based marking on the Cisco router or switch can be used to mark incoming or outgoing packets.

Note: Whenever you see the terminology *class-based*, you can safely assume this is a reference to the use of the Modular Quality of Service Command Line Interface (MQC).

Keep in mind, when marking traffic with this method of the MQC, you can combine the marking process with any other Quality of Service feature you would like to utilize on the traffic on *output*. When marking traffic on packet *input*, you can combine class-based marking with class-based policing.

It might not surprise you to learn that CEF must be enabled on the interface in order for class-based marking to function properly.

Of all of the markings that you mastered in this chapter, which can be applied to packets using class-based marking in the IOS? The list is certainly impressive:

- IP Precedence
- IP DSCP
- QoS Group
- MPLS Traffic Class
- 802.1Q CoS
- ISL CoS
- Frame Relay DE bit
- ATM CLP Bit

Here is an example of class-based marking in action:

```
R4#configure terminal
R4(config)#access-list 100 permit tcp any any lt 1024
R4(config)#access-list 100 permit tcp any lt 1024 any
R4(config)#class-map CM_WELL_KNOWN
R4(config-cmap)#match access-group 100
R4(config-cmap)#class-map CM_UNKNOWN
R4(config-cmap)#match not class-map CM_WELL_KNOWN
R4(config-cmap)#policy-map PM_DSCP
R4(config-pmap)#class CM_WELL_KNOWN
R4(config-pmap-c)#set dscp AF21
R4(config-pmap-c)#class CM_UNKNOWN
R4(config-pmap-c)#set dscp 0
R4(config-pmap-c)#interface fastethernet 0/1
R4(config-if)#service-policy input PM_DSCP
R4(config-if)#end
R4#
```

Notice the power of the set command when engaging on class-based marking on the Cisco device. Syntax for other marking options includes:

- **set cos *cos-value*** – this option sets the CoS marking; you should note that it is only available on LAN interfaces running 802.1Q or ISL encapsulation
- **set ip precedence *ip_precedence_value*** – this option sets the IP Precedence marking; notice that you can use a number 5, or the name, **priority**; remember, you should use **match protocol ip** or **match protocol ipv6** in order to control marking for IPv4 versus IPv6
- **set [ip] dscp *dscp_value*** – this option sets the DSCP marking; again, you can use the number or the name, for example, 47 or **ef**; the optional **ip** keyword is used to control marking IPv4 packets only versus also marking IPv6 packets

Chapter Challenge: Marking Trouble Tickets

The following section includes a Trouble Ticket associated with marking scenarios. Be sure to load the appropriate initial configuration files in order to perform the ticket.

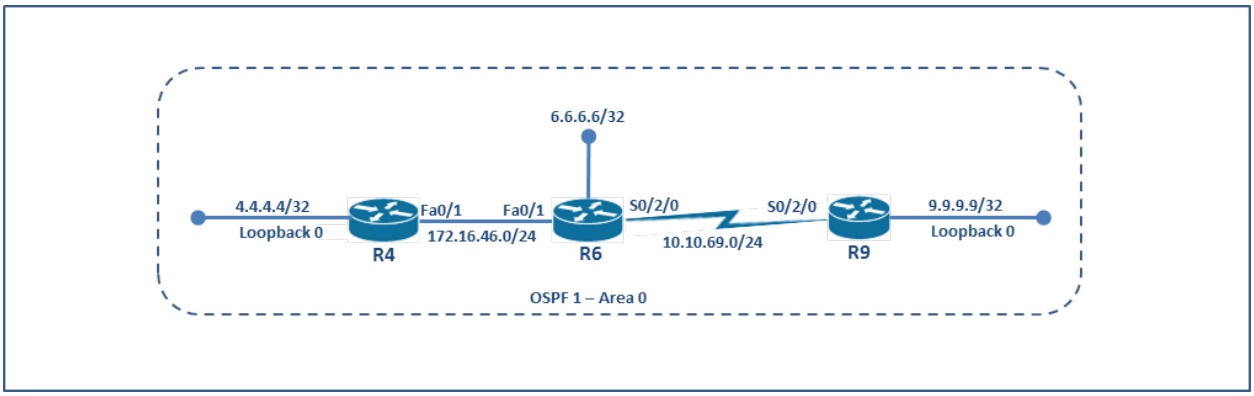

Figure 5-5: A Basic Quality of Service Topology

Trouble Ticket #1

Your supervisor has asked you to ensure that R4 is marking all VoIP packets with the appropriate DSCP value for VoIP packets as it sends these packets to R6. He wants to ensure these markings are backward compatible with IP Precedence-only devices. Ensure R4 is configured to achieve this.

Chapter Challenge: Marking Trouble Ticket Solutions

The following section includes solutions to the Trouble Ticket associated with marking scenarios.

Trouble Ticket #1

Your supervisor has asked you to ensure that R4 is marking all VoIP packets with the appropriate DSCP value for VoIP packets as it sends these packets to R6. He wants to ensure these markings are backward compatible with IP Precedence-only devices. Ensure R4 is configured to achieve this.

Step 1 - Fault Verification:

What is the current configuration for QoS class-based marking on R4, if any?

```
R4#show policy-map interface
 FastEthernet0/1

  Service-policy output: markvoice

    Class-map: voice (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol rtp audio
      QoS Set
        dscp ef
          Packets marked 0

    Class-map: class-default (match-any)
      42 packets, 3714 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
R4#
```

Here we can see there is indeed a **markvoice** policy map assigned in the outbound direction toward R6. This policy is marking voice traffic (using NBAR to identify voice packets). There is a problem, however. The policy is marking DSCP Expedited Forwarding (EF). Technically, this value will not be backward compatible with an IP Precedence-only speaking device. This policy needs to be updated for a DSCP value of Class Selector 5 (CS5).

Step 2 - Fault Remediation:

Here we edit the existing policy map for the correct QoS marking treatment:

```
R4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#policy-map markvoice
R4(config-pmap)#class voice
R4(config-pmap-c)#no set dscp ef
R4(config-pmap-c)#set dscp ?
  <0-63>     Differentiated services codepoint value
  af11       Match packets with AF11 dscp (001010)
  af12       Match packets with AF12 dscp (001100)
  af13       Match packets with AF13 dscp (001110)
```

```
  af21       Match packets with AF21 dscp (010010)
  af22       Match packets with AF22 dscp (010100)
  af23       Match packets with AF23 dscp (010110)
  af31       Match packets with AF31 dscp (011010)
  af32       Match packets with AF32 dscp (011100)
  af33       Match packets with AF33 dscp (011110)
  af41       Match packets with AF41 dscp (100010)
  af42       Match packets with AF42 dscp (100100)
  af43       Match packets with AF43 dscp (100110)
  cos        Set packet DSCP from L2 COS
  cs1        Match packets with CS1(precedence 1) dscp (001000)
  cs2        Match packets with CS2(precedence 2) dscp (010000)
  cs3        Match packets with CS3(precedence 3) dscp (011000)
  cs4        Match packets with CS4(precedence 4) dscp (100000)
  cs5        Match packets with CS5(precedence 5) dscp (101000)
  cs6        Match packets with CS6(precedence 6) dscp (110000)
  cs7        Match packets with CS7(precedence 7) dscp (111000)
  default    Match packets with default dscp (000000)
  ef         Match packets with EF dscp (101110)
  qos-group  Set packet dscp from QoS Group.

R4(config-pmap-c)#set dscp cs5
R4(config-pmap-c)#end
R4#
```

Step 3 - Remediation Verification:

In this case, verification is very simple. We run the show policy-map interface command and ensure we meet the ticket requirement:

```
R4#show policy-map interface
 FastEthernet0/1

  Service-policy output: markvoice

    Class-map: voice (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol rtp audio
      QoS Set
        dscp cs5
          Packets marked 0

    Class-map: class-default (match-any)
      145 packets, 12827 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
R4#
```

Notice the policy is now marking the traffic appropriately.

Chapter 6: Congestion Management

Occasional, sporadic congestion may occur at key points in your network infrastructure. When this happens, you can manage this congestion using a variety of what are termed “fancy queuing” mechanisms. This chapter explores such options.

Congestion Management Technology Review

Before we dive into the operation of congestion management tools at the Cisco command line, we need to review the technologies for this section.

Note: This chapter deals with congestion management tools for the Cisco routers. This book details congestion management tools for the Catalyst switches in great depth in ***Chapter 13: Catalyst Quality of Service (QoS)***.

The Software Queue Versus the Hardware Queue

One of the points that many Cisco QoS texts fail to make clear is that a typical Cisco network device interface possess a ***software queue*** and a ***hardware queue*** for outbound traffic transmissions. When we read that a particular QoS tool will engage and work in times of congestion only, this is a reference to packets overflowing the ***hardware*** queue into the software queue. The software queue is where the magic of Quality of Service happens. In fact, one of the QoS mechanisms called Link Fragmentation and Interleaving (LFI) directly addresses the split between these two queues. LFI ensures that large packets (Jumbograms) do not clog the hardware queue. These large packets are chopped up (fragmented) so that higher priority, smaller packets (like voice) can be interwoven between the fragments in the hardware queue. ***Chapter 11: Link Fragmentation and Interleaving (LFI)*** details this process for you.

The hardware queue is typically referred to as the Transmit Ring or TX-Ring. I forever refer to it as simply HQ for the Hardware Queue. What is the size of the Hardware Queue? This is set by Cisco based on different interfaces of particular speeds and capabilities. Cisco sets it in a manner that is optimized for times of no congestion, and also is of an appropriate size for times of congestion where the software queue can work its magic. Cisco suggests we never manipulate their optimal calculations for the size of this queue. If we manipulate it incorrectly and have many QoS mechanisms in place, we can cause their behavior to miss the objectives they were designed to achieve.

However, we all know the CCIE lab exam might have us perform some very unusual configurations; those that we might never perform in our day job. The type of configuration we could only learn about from the device documentation. So with this said, what is the command on a 2800 series device to set the size of the HQ (Hardware Queue) for the outbound direction? The command is:

tx-ring-limit *x*.

Where x = the number of packets

Other than your ability to manipulate the size of the HQ, you really cannot do much else. It uses a Congestion Mechanism of First In First Out (FIFO) and you are stuck with this Best Effort approach. Thank goodness, we can configure the Software Queue that sits behind this HQ with many other sophisticated QoS tools. For the purpose of this chapter, we will apply a software queuing concept to demonstrate how this alters the configuration of a given interface but it is not necessary at this juncture for us to understand the nature of the software queuing mechanism we will use.

First In First Out (FIFO) Queuing

Remember in ***Chapter 2: Overall Quality of Service (QoS) Approaches*** we discussed the best effort approach to QoS. This approach relies on over provisioning bandwidth, and allowing the First In First Out method of queuing do the job. Recall from the previous section that FIFO is the default queuing method in the hardware queue. With a FIFO best effort approach, you also allow this to be the default method in the software queue.

The obvious danger of this approach occurs when there is a period of extreme congestion on the interface. Packets that you would want to prioritize, such as VoIP packets, might be stuck behind far less important packets in the congested hardware and software queues.

FIFO embodies no concept of priority or classes of traffic and consequently makes no decision about packet priority. There is only one queue, and all packets receive equal treatment. Packets exit an interface in the same order that they arrive.

When FIFO is used, aggressive sources can consume all the bandwidth, while a "bursty" source can simply generate delays in time-sensitive or important traffic in the presence of either aggressive or "bursty" flows, discarding of important traffic may take place due to less important traffic filling the queue.

When no "fancy queuing" strategies are applied, all interfaces except serial interfaces at E1 (2.048 Mbps) and below will use FIFO by default.

FIFO, is the fastest method of our available queuing mechanisms. FIFO is a very effective mechanism for high-speed low congestion links. In the situation of a high speed very little congestion interface, FIFO queuing may be the only queuing you need.

Fair Queuing

Fair queuing is the next form of congestion management we will discuss. In this method of software queuing, flows of traffic are automatically classified and put into different queues without the application of any of the quality of the service tools we discussed in the previous chapters. The important thing to note is that fair queuing works automatically without the need for access-lists or protocol matching. This makes fair queuing extremely easy to configure, but removes the most essential asset we look for in a modern congestion management strategy: granularity.

Fair queuing classifies traffic based on a concept of flows. The Cisco IOS deployment of WFQ classifies packets based on the concept of flows or conversations. A flow, as it pertains to fair queuing, is a stream of packets that share a common set of parameters. Specifically these parameters include ip protocol, source ip address, destination ip address, source port, destination port and the type of service (ToS) value. After automatically identifying individual flows, a router will then quantify each flow based on how aggressive they are regarding bandwidth and prioritize them in such a way that lower volume flows receive preference over higher volume flows.

The important thing to note is that each of these individual flows get placed into their own separate queues dynamically. By default, there are 256 queues available to fair queuing on a given interface. Additionally, where high speed interfaces default to FIFO as their mode of software queuing, slow speed interfaces, those less than E1 speed (2.048 Mbps) default to fair queuing. The command **fair-queue** is all it takes to enable this behavior on a particular interface.

Before moving on it is necessary to discuss how Cisco IOS implements fair queuing.

Weighted Fair Queuing (WFQ)

Have described the basic concept of WFQ in the previous section it is necessary to discuss Cisco's implementation of fair queuing. We will begin with a definition: WFQ is a dynamic scheduling strategy designed to provide a fair allocation of available bandwidth to all network traffic. As discussed in the previous section ***Fair Queuing***, WFQ applies priority, or assigns weights, to traffic in order to classify that traffic into conversations and determine how much bandwidth each conversation gets relative to other conversations. WFQ operates via a flow-based algorithm that schedules interactive traffic to the front of a queue to improve response time while simultaneously **fairly sharing** the remaining bandwidth among high-bandwidth flows. The overall effect is that WFQ gives low-volume traffic priority over high-volume traffic. WFQ will give multiple file transfer operations comparable bandwidth in relation to all the traffic utilizing the link.

In FIFO queuing, traffic leaves the queue in the order it arrives, this takes place without considering bandwidth utilization or delay based on serialization. Therefore, in FIFO queuing, high-volume network applications that generate aggressive packet flows can consume all available bandwidth on a link thus starving out other traffic on a link. WFQ is the answer to this serious limitation found in FIFO queuing.

WFQ provides traffic priority management that will dynamically sort traffic into messages that make up a conversation. WFQ breaks up the train of packets within a conversation to ensure equal distribution of bandwidth between individual conversations and ensure the transfer of that low-volume traffic in a timely fashion.

WFQ classifies traffic into different flows based on packet header addressing, including such characteristics as source and destination network or MAC address, protocol, source and destination port and socket numbers of the session, Frame Relay data-link connection identifier (DLCI) value, and ToS value. There are two categories of flows of WFQ:

- **Low-bandwidth traffic** – This category of traffic will receive preferential treatment over high-bandwidth traffic.
- **High-bandwidth traffic** – This category of traffic will share the transmission service proportionally according to any assigned weights.

This means that low-bandwidth traffic flows, which make up the majority of traffic on a link, will receive preferential service, thus insuring timely delivery of their packets. Additionally, now that low-volume traffic has consumed a portion of the available bandwidth, any high-volume traffic streams will share the remaining capacity proportionally.

How does this preferential treatment take place in WFQ? In WFQ, the algorithm places the individual packets of various conversations into their assigned fair queues before transmission. The preferential treatment in WFQ manifests itself in the order of removal of these packets from their fair queues. The order is determined based on the "virtual time of the delivery" of the last bit of each arriving packet.

This mechanism establishes a method by which new messages for high-bandwidth flows get discarded after the congestive-message threshold (CDT) has been met. However, low-bandwidth flows, which include control-message conversations, will continue to queue data. This means because of the behavior of the WFQ algorithm a given fair queue may occasionally contain more messages than that specified by the assigned threshold number.

WFQ and IP Precedence

WFQ is IP precedence-aware. It can detect higher priority packets marked with precedence and can schedule them faster, providing superior response time for this traffic. Thus, as the precedence increases, WFQ allocates more bandwidth to the conversation during periods of congestion.

WFQ assigns a weight to each flow, which determines the transmit order for queued packets. In this scheme, lower weights get scheduled first. In the Cisco IOS WFQ deployment, IP precedence is a divisor in this weight calculation. Meaning that the calculated weight is divided by the IP precedence value, thus a higher IP Precedence flow will have a lower calculated weight and thus greater priority to be de-queued by the scheduler.

The WFQ scheduler will send a certain number of bytes from each queue. As we described earlier each queue corresponds to a particular flow. The WFQ algorithm will cycle through all flows periodically, at which time it will effectively send a number of bytes equal to the precedence of the flow plus one. This is a ratio used by the algorithm to determine how many bytes per packet to send. However, for the purposes of understanding WFQ, we will use this value as the actual byte count. For instance, traffic with an IP Precedence value of 5 gets a lower weight than traffic with an IP Precedence value of 2, thus it will receive greater priority in transmit order. At its simplest, the calculated weights are inversely proportional to the IP Precedence value.

To determine the bandwidth allocation for each queue, divide the byte count for the flow by the total byte count for all flows. For example, if you have one flow at each precedence level, each flow will get precedence + 1 parts of the link. This means that the following precedence values:

0 + 1 + 2 + 3 + 4 + 5 + 6 + 7

Would become:

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

After we add one to each value.

This new string of values add up to 36. Thus, precedence 0 traffic will get 1/36 of the bandwidth, precedence 1 traffic will get 2/36, and the precedence 7 traffic will get 8/36.

However, if you have 18 precedence 1 flows and one of each of the rest, the total is now:

1 + 2(18) + 3 + 4 + 5 + 6 + 7 + 8 = 70

Precedence 0 traffic will get 1/70, each of the precedence 1 flows will get 2/70, and so on.

As flows are added or ended, the actual allocated bandwidth will continuously change.

The operational process employed by WFQ is illustrated in **Figure 6-1**.

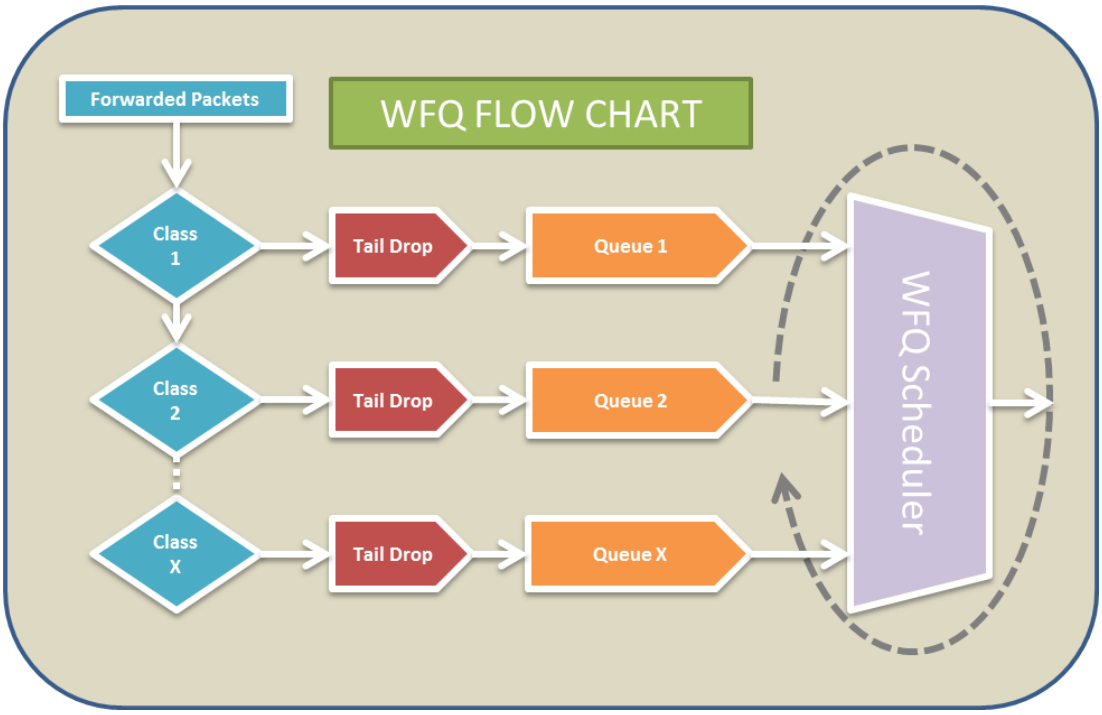

Figure 6-1 : WFQ Queuing and Scheduling Workflow

Class-Based Weighted Fair Queuing (CBWFQ)

In the previous section, we discussed how WFQ gives certain types of traffic preferential treatment when congestion occurs on a slow-speed link. In that section, we reviewed how WFQ does this by automatically detecting flows and preventing any one conversation from starving out others on a link. Additionally, WFQ suffered from limitations in scalability and granularity. These issues result when the WFQ algorithm has to manage very aggressive individual flows, or a very high number of flows. Class-based weighted fair queuing (CBWFQ) overcomes these limitations. CBWFQ enhanced the fair queuing algorithm such that it supports statically defined classes. This fact affords CBWFQ greater control over how traffic will be queued and bandwidth assignment to particular flows.

The advantages of CBWFQ come in the form of scalability, granularity and ease of configuration. Not to mention that it combines all the capabilities of many the legacy congestion management protocols into

one mechanism. CBWFQ also incorporates support for weighted random early detection (WRED). We will review how CBWFQ works with WRED in ***Chapter 7: Congestion Avoidance***. All these factors make CBWFQ possibly one of the most advanced and powerful congestion management mechanisms offered by Cisco IOS.

CBWFQ defined an extension to the standard functionality of WFQ to allow it to support user-defined traffic classes. In CBWFQ, the administrator defines the different traffic classes based on match criteria including protocols, access control lists (ACLs), and input interfaces. Packets satisfying the match criteria for a class constitute the traffic for that class. For each of these classes a single FIFO queue is reserved, and traffic belonging to that particular class is directed to the queue for that class.

Once a class has been created according to its match criteria, various attributes or treatments may be assigned to it. These attributes come in the form of bandwidth, weight, and maximum packet limit values assigned to a particular class. The bandwidth assigned to a class is the guaranteed bandwidth delivered to the class during periods of congestion on link.

To provide even more levels of control, the ability exists to specify the queue limit for a given class, the queue limit defines the maximum number of packets allowed to accumulate in the queue for a given class. Packets belonging to a class are subject to the bandwidth and queue limits that assigned to that class.

After a queue has reached its configured queue limit packets for that class will experience tail drop depending on how class policy is configured.

Tail drop takes place in CBWFQ classes by default unless the process is explicitly configured to use WRED to drop packets as a means of avoiding congestion. We will cover the details regarding WRED in ***Chapter 7: Congestion Avoidance***.

If a default class is configured, with the bandwidth policy-map class configuration command, all unclassified traffic goes into a single FIFO queue where it is treated according to the configured bandwidth. If a default class is configured with the fair queue command, all unclassified traffic is flow classified and given best-effort treatment. If there is no default class, then the traffic that does not match any of the configured classes is flow classified and given best-effort treatment. Once a packet is classified, all of the standard mechanisms that can be used to differentiate service between and for classes can now be applied.

We discussed in the previous section, how flow-based classification is the standard WFQ treatment. That means that packets with the same source IP address, destination IP address, source TCP or UDP port, or destination TCP or UDP port are classified as belonging to the same flow. WFQ then allocates an equal share of bandwidth to each of these flows. Flow-based WFQ is also called fair queuing because all flows are equally weighted.

In CBWFQ this standard treatment is no longer the norm. In CBWFQ, the weight specified for a given class becomes the weight of each packet that meets the match criteria of the class. Packets that arrive

at the output interface are classified according to the match criteria filters that have been user defined, and then each one is assigned the appropriate weight. The weight for a packet belonging to a specific class is derived from the bandwidth that has been assigned to the class; in this sense, the weight for a class in CBWFQ can be considered user-configurable.

After the assignment of a weight for a packet has taken place, that packet is the queued up in the appropriate class queue. CBWFQ uses the weights that were assigned in the initial stages of this process to ensure that the class queue receives fair service.

Configuring a class policy—thus, configuring CBWFQ— follows the model used in the Modular Quality of Service CLI as we discussed in **Chapter 3: Configuration and Monitoring tools**. The CBWFQ MQC implementation process entails three essential processes:

1. **Define the traffic classes to specify the classification policy (class maps)**. This process determines how many types of packets will be differentiated from one another. For this process, it is necessary to configure a class map that specifies the traffic of interest.
2. **Associating policies (policy maps)** —that is to say, the assignment of treatment characteristics to each defined class. This process entails the configuration of policies to be applied to packets belonging to one of the classes previously defined through a class map. For this process, it is necessary to configure a policy map that specifies the actual treatment policy for each traffic class.
3. **Attaching policies to interfaces (service policies).** This process requires an existing policy map, or service policy, to be associated with an interface to apply the particular set of policies in the map to that interface. For this process, it is necessary to configure a service policy that specifies the policy map to be applied to a given interface. Be aware that service policies are directional in nature but those specifying congestion management strategies can only be applied in the outbound direction.

Use of the bandwidth command inside a given policy map is how these CBWFQ treatment is established and they can be quantified using fix bandwidth values, fixed percentage values, or values based on portions of the remaining bandwidth after previous allocations:

```
R6(config)#policy-map SET_DSCP
R6(config-pmap)#class CS4
R6(config-pmap-c)#bandwidth ?
  <8-2000000>  Kilo Bits per second
  percent      % of total Bandwidth
  remaining    The remaining bandwidth
```

The operational process employed by CBWFQ is illustrated in **Figure 6-2**.

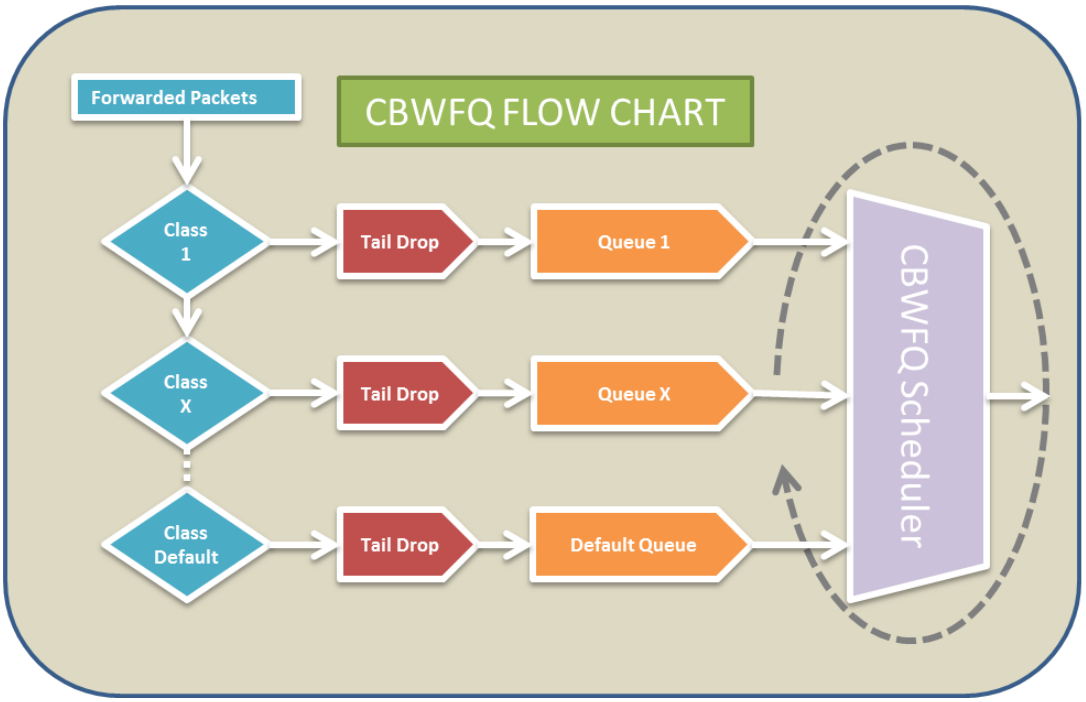

Figure 6-2: CBWFQ Queuing and Scheduling Workflow

Thus far we have discussed the treatment of packets as it applies to bandwidth, but one particular packet type has been absent from our discussions. What happens to packets that are delay or latency sensitive? In legacy Quality of Service mechanisms like Priority Queuing these packets would have been assigned to a queue that offered strict priority treatment to these packets such that these packets would be scheduled and de-queued before any other packets, thus protecting them from latency and delay. This type of treatment is a foreign concept to a protocol like CBWFQ, or at least it was until the advent of Low Latency Queuing.

Low Latency Queuing (LLQ)

The LLQ feature brings strict priority queuing (PQ) to CBWFQ. A strict priority queue allows delay-sensitive traffic to be de-queued before packets in any other queues. Without LLQ, CBWFQ provides WFQ based on defined classes with no strict priority queue available for real-time traffic. CBWFQ allows the definition of traffic classes and then the assignment of treatment characteristics to that class. For example, you can designate the minimum bandwidth delivered to a particular class during link congestion.

For CBWFQ, as we discussed earlier, the weight for a packet belonging to a specific class comes from the bandwidth assigned to the class. Therefore, the bandwidth assigned to the packets of a class determines the order in which packets are transmitted. Keep in mind that in CBWFQ all packets are serviced fairly based on weight; no class of packets may be granted strict priority. This concept presents issues for applications like voice traffic that cannot tolerant delay, and especially variation in delay (jitter). For

voice traffic, latency introduces irregularities in the voice transmission known as jitter, a condition that can make voice communication unintelligible.

The fix for this problem is LLQ. LLQ provides strict priority queuing for traditional CBWFQ, thus reducing jitter in voice communications. LLQ is configured by specifying the priority command under the policy map. LLQ enables the use of a single, strict priority queue within CBWFQ at the class level, allowing traffic belonging to a delay sensitive class to be directed to a CBWFQ strict priority queue. To queue traffic in the strict priority queue, specify the named class within a policy map and then apply the priority command for that class. Classes that have the priority command applied to them are considered priority classes. Within a policy map, you can give one or more classes priority status. When multiple classes within a single policy map are configured as priority classes, all traffic from these classes will be queued to the same, single, strict priority queue.

Strict priority queuing applied inside CBWFQ offers enhanced capabilities over its use in legacy queue mechanisms, for instance, you are no longer limited to a UDP port number to stipulate priority flows because you can configure the priority status for a class within CBWFQ. Instead, all of the valid match criteria used to specify traffic for a class now applies to priority traffic. These methods of specifying traffic for a class include matching on access lists, protocols, and input interfaces. Additionally, within an access list traffic matches can even be based on the DSCP value.

Although it is possible to schedule various types of real-time traffic to the strict priority queue, it is strongly recommend that only voice traffic should be directed to it because voice traffic is well behaved by design, where other types of real-time traffic like video are not. Additionally, voice traffic requires that delay be non-variable in order to avoid jitter and adding other classes of traffic to the strict priority queue apart from voice could confound the steadiness of delay required for successful VOIP communications.

As mentioned previously LLQ is configured using the priority command under a given policy map. The priority can be specified based on a fixed bandwidth per second value or a percentage of the total bandwidth.

```
R6(config)#policy-map SET_DSCP
R6(config-pmap)#class CS4
R6(config-pmap-c)#priority ?
  <8-2000000>  Kilo Bits per second
  percent      % of total bandwidth
  <cr>
```

The operational process employed by Strict Priority Queuing inside CBWFQ is illustrated in **Figure 6-4**.

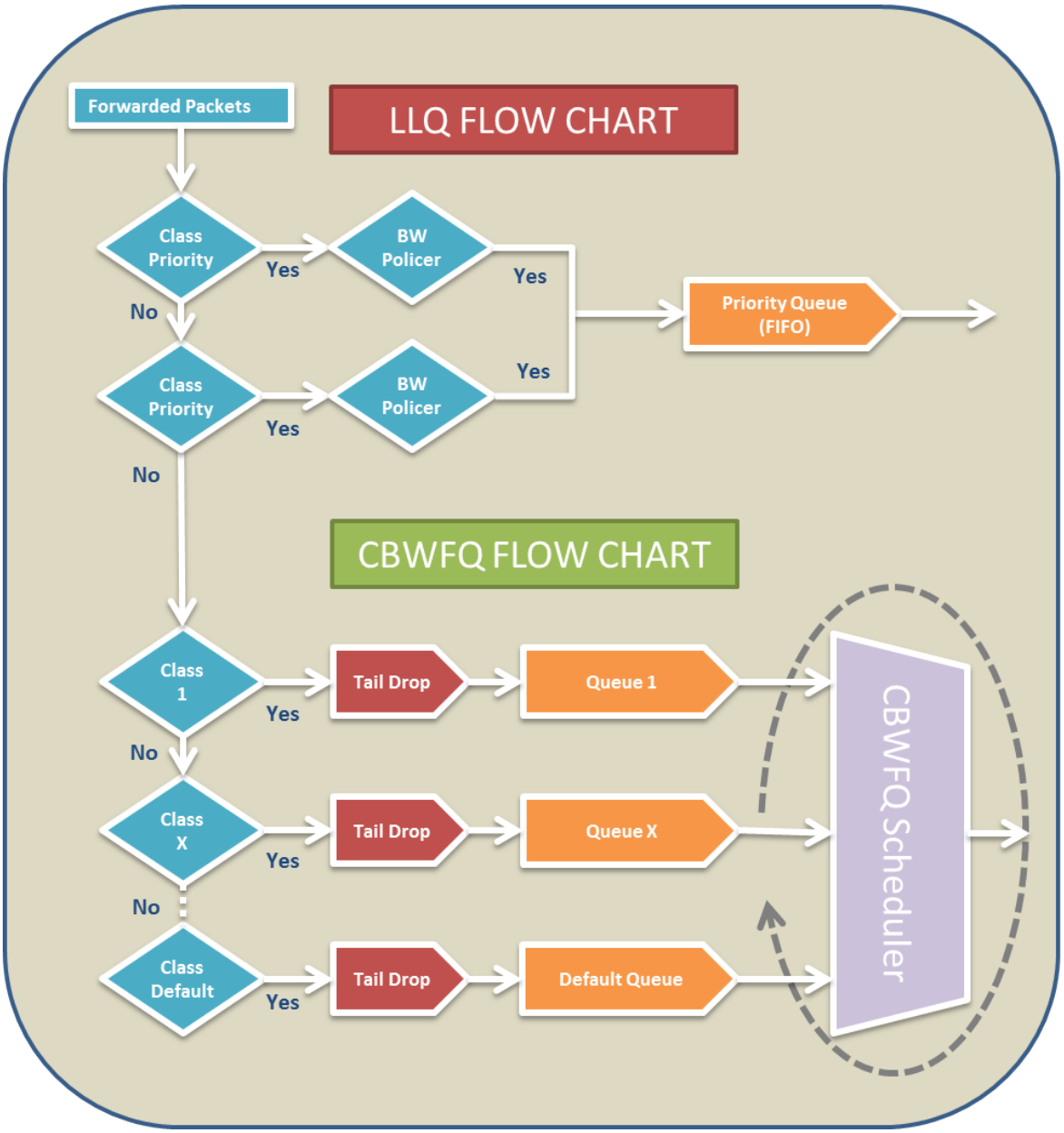

Figure 6-4: LLQ Workflow Process inside CBWFQ structure

The Operation and Troubleshooting of Congestion Management

Now that we have reviewed the technologies of this chapter, it is time to analyze their operation at the Cisco command line.

The Software Queue Versus the Hardware Queue

For exploring the concepts behind the software queue and the hardware queue, we will utilize the following topology:

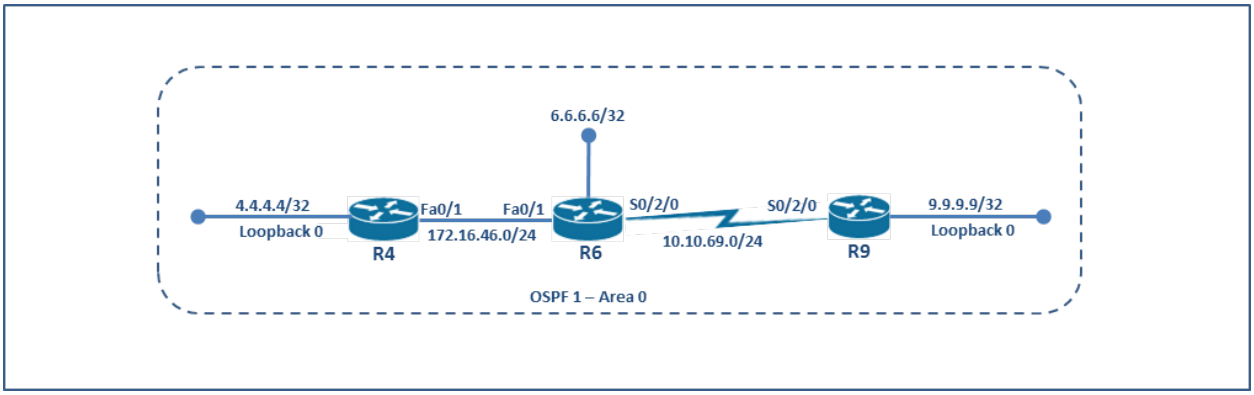

Figure 6-5: A Basic Quality of Service Topology

The predominant commands mentioned in the ***Congestion Management Technology Review*** section of this chapter deal specifically with the values and parameters of the physical hardware queue of a given interface. We will first look at the basic default configuration of the hardware queue itself on a cisco router. This task is best accomplished via the **show controllers** command:

```
R6#show controllers Serial 0/2/0 | inc tx_limit
tx_limited = 1(2), errata19 count1 - 0, count2 - 0
```

The output of this command reveals that the hardware queue on R6 has a maximum size of two packets as indicated by the number in parenthesis. This can be changed by applying the **tx-ring-limit** command under the interface:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#interface Serial0/2/0
R6(config-if)#tx-ring-limit 100
R6(config-if)#end
```

Now we can see that the size of the hardware queue has been modified from 2 to a value of 100.

```
R6#show controllers Serial 0/2/0 | inc tx_limit
tx_limited = 1(100), errata19 count1 - 0, count2 - 0
```

This illustrates the parameters that we can manipulate regarding the hardware queue itself, but this section deals with two concepts; hardware queues and software queues. Where there is a single transmit ring hardware queue it is important to note that there are two software queues. One of these queues is for outbound packets and the other is for inbound packets. We can see these queue by

looking at another interface on R6. Before we make any configuration changes to the FastEthernet0/1 interface, we clearly see the default values associated with the interface by using the **show interface** command:

```
R6#show interfaces FastEthernet0/1 | include queue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Output queue: 0/40 (size/max)
```

We see the two queues clearly identified in the **show** output. We also see the default sizes associated with each queue. These values can be manipulated by using the **hold-queue** command:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#interface FastEthernet0/1
R6(config-if)#hold-queue 50 in
R6(config-if)#hold-queue 50 out
R6(config-if)#end
```

Repeating the show interfaces command will reveal that we have modified the maximum number of packets that can be held by either of the software queues to 50 packets:

```
R6#show interfaces FastEthernet0/1 | include queue
  Input queue: 0/50/0/0 (size/max/drops/flushes); Total output drops: 0
  Output queue: 0/50 (size/max)
```

We need to look now at how the hardware queue and these software queues interoperate with one another. We will observe this behavior by generating a packet stream from R4 destined to the loopback0 interface of R9:

```
R4#ping 9.9.9.9 size 512 repeat 1000000

Type escape sequence to abort.
Sending 1000000, 512-byte ICMP Echos to 9.9.9.9, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! <output omitted>
```

Now we will look at the output on R6 again:

```
R6#show interfaces FastEthernet0/1 | include queue
  Input queue: 0/50/0/0 (size/max/drops/flushes); Total output drops: 0
  Output queue: 0/50 (size/max)
```

Observe that the values for the software queues do not change. In this particular instance, we are looking at the input queue specifically. This is because the hardware queue is not congested. We can

artificially force the hardware queue to experience congestion by changing the size of the queue via the **tx-ring-limit** command:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#
R6(config)#interface FastEthernet0/1
R6(config-if)#tx-ring-limit 15
R6(config-if)#end
```

Now we will be able to see the software queue buffering packets that would normally be dropped:

```
R6#show interfaces Fasterethernet0/1 | include queue
  Input queue: 4/50/0/0 (size/max/drops/flushes); Total output drops: 0
  Output queue: 0/50 (size/max)
```

Observe that the input software queue has “held” 4 packets based on this **show** command output. Specific values will vary over time but the important point to note is that the software queue is absorbing packets that would normally have been dropped. Later in this chapter, we will explore this behavior as it affects and applies to various types of “fancy queuing” technologies.

First In First Out (FIFO) Queuing

FIFO queuing is the default queuing method employed by all interfaces that are considered "fast" interfaces. This includes interfaces operating at speeds in excess of E1 speed, or 2.048 Mbps. Interfaces operating at slower than E1 speeds also default to a basic queuing method but we will cover that in a later section. The following output shows what FIFO queuing looks like on a FastEthernet interface.

```
R4#show interface FastEthernet0/1
FastEthernet0/1 is up, line protocol is up
  Hardware is MV96340 Ethernet, address is 000a.b819.c921 (bia 000a.b819.c921)
  Internet address is 172.16.46.4/24
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, 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: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     2170556 packets input, 942343994 bytes
     Received 23820 broadcasts, 0 runts, 0 giants, 1 throttles
     21 input errors, 0 CRC, 0 frame, 0 overrun, 21 ignored
     0 watchdog
     0 input packets with dribble condition detected
     1824923 packets output, 929860618 bytes, 0 underruns
     0 output errors, 0 collisions, 1 interface resets
     2200 unknown protocol drops
```

```
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
```

The main purpose of a FIFO queue is to manage inbound packets on an interface, placing them in the queue in the order they arrive, and to de-queue them out to the exit interface at a fixed rate that the interface can process. Provided the rate at which packets enter a FIFO queue is slower than the rate the queue can transmit them, then the FIFO will be a virtually "invisible" mechanism to the packets traversing the interfaces it manages. In this scenario, packets will simply flow through the queue as if it did not exist. If the situation changes and packets cannot be de-queued faster than they are queued then the FIFO queue will begin to fill up. Situations exist where it is possible for this software queue to become completely full. In this case, the queue will not allow packets to enter the queue simply because there is no available space in the queue. This situation describes a condition known as "tail" drop. Packets leaving a queue are considered to be at the "head" of the queue where packets will always enter the tail of a queue. When a software queue is full, we will experience "non-discriminatory" drop or "tail" drop. This type of dropping of packets means that packets never enter the queue and are immediately discarded by the router resulting in upper layer protocols that use acknowledgement processes to detect packet loss to begin retransmitting packets. This can compound the problem of a full FIFO queue because now there are significantly more packets to be queued meaning that more packets will be tail dropped more often than not resulting in a condition known as Global TCP Synchronization that can significantly affect the experience of users on the network. In ***Chapter 7: Congestion Avoidance*** we will explore Quality of Service mechanisms that can mitigate this tail drop/global synchronization scenario.

In subsequent sections of this chapter, we will explore how FIFO functions as the default queuing method in some aspects of "fancy" queuing. Until then simply keep in mind, that FIFO is the simplest mode of queuing we can deploy in any situation, and all it does is attempt to prevent tail drop. A FIFO queue offers us no special distinction between packet types, and it is comprised of what could be best described as a single "leaky bucket" that buffers all egress traffic for a given interface. Short of using the **hold-queue** command to specify the actual size of the FIFO queue in packets we have no actual control over its operation at all. In the previous section, ***The Software Queue Versus the Hardware Queue*** we covered manipulating the size of the software queues on a FastEthernet interface. Those software queues we manipulated employed FIFO queuing by default.

Fair Queuing

Fair queuing as discussed in the ***Congestion Management Technology Review*** can be enabled on an interface very simply by applying the **fair queue** command under an interface.

```
R6#show interfaces FastEthernet0/1 | be Queueing
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 1 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     132 packets input, 15068 bytes
     Received 120 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
```

```
     0 watchdog
     0 input packets with dribble condition detected
     111 packets output, 11471 bytes, 0 underruns
     0 output errors, 0 collisions, 1 interface resets
     13 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
```

We see the queuing strategy is FIFO but that can be changed easily.

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#interface FastEthernet0/1
R6(config-if)#fair-queue
R6(config-if)#end
```

Observe the change via the **show interface** command.

```
R6#show interfaces FastEthernet0/1 | be Queueing
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     169 packets input, 19404 bytes
     Received 157 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog
     0 input packets with dribble condition detected
     152 packets output, 15473 bytes, 0 underruns
     0 output errors, 0 collisions, 2 interface resets
     19 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
```

Thus far, we have only looked at the basic configuration behind fair queuing. The dynamic nature of the tool provides only a few ways for us to manipulate its behavior but it must be noted that there are some advanced features that can be enabled. In the ***Congestion Management Technology Review*** section, we discussed how ip precedence can be used to manipulate how packets are released from the queue. WFQ methodology is a leaky bucket mechanism the same as FIFO, but now with weighting applied, the amount of information processed from each individual queue depends on the weight of that queue. We have to note that the actual sequence that the individual queues are serviced by WFQ is not affected by weighting only the amount of information processed within each queue is affected.

As we mentioned earlier there are up to 256 queues supported by the fair-queue command by default this value can be changed simply by specifying the number of queues or conversations desired. Interestingly enough, WFQ also offers us another tool that will allow us to specify the maximum number

of packets allowed into any single dynamic queue. This behavior equates to the equivalent of a collision avoidance mechanism. What happens, when this maximum value is exceeded new arriving packets for a given queue may or may not be dropped. Whether a packet is tail dropped in the scenario depends on a complex mathematical calculation that we will not cover in this section. Instead, we will describe the process by saying that if a packet exists in another queue with a worse set of parameters (referred to as a sequence number) then that packet may be dropped instead. This process serves to mitigate both tail drop and possible global synchronization issues. Collision avoidance mechanisms similar to this will be discussed in detail in ***Chapter 7: Congestion Avoidance***.

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#interface FastEthernet0/1
R6(config-if)#fair-queue 4096 4096 0
R6(config-if)#end
```

This can be verified with the show interface command:

```
R6#show interfaces FastEthernet0/1 | be Queueing
  Queueing strategy: weighted fair
  Output queue: 0/1000/4096/0 (size/max total/threshold/drops)
     Conversations  0/1/4096 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     710 packets input, 78740 bytes
     Received 691 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog
     0 input packets with dribble condition detected
     644 packets output, 63921 bytes, 0 underruns
     0 output errors, 0 collisions, 4 interface resets
     89 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
```

We see now that the max total and max threshold have changed to the maximum values. That leaves the last value we specified of 0. WFQ embraces the concept of Resource Reservation Protocol (RSVP), which we will discuss in depth in ***Chapter 12: Resource Reservation Protocol (RSVP)***. The thing to note here is that WFQ allows us to specify if any of our queues will be designated as RSVP queues. This is required if the possibility exists that individual flows could consume all configured queues. By configuring the number of reserved queues, the router will always keep that many queues available for any path reservations. In our configuration, we did not establish any reserved queues, this does not mean that our configuration will not support RSVP protocol, it just means that if all queues are being used then no resources can be assigned for the RSVP path creation.

Troubleshooting Weighted Fair Queuing

The dynamic nature of WFQ makes the protocol more difficult to troubleshoot than other protocols that rely on static queuing processes. There is no way to control the information entering the router, nor can we manipulate what queue the traffic enters; all this is managed by the WFQ scheduling algorithm. It also has to be noted that there are no weighted fair queuing debugging commands incorporated in the IOS. The most common issues with WFQ will involve dynamic queue assignment problems and high tail drop occurrences.

These issues can best be isolated and identified by using the **show queueing fair** and **show interface** commands:

```
R6#show queueing fair
Current fair queue configuration:

  Interface           Discard    Dynamic  Reserved  Link    Priority
                      threshold  queues   queues    queues  queues
  FastEthernet0/1     4096       4096     0         8       1
  Serial0/1/0         64         256      0         8       1
  Serial0/2/0         64         256      0         8       1
  Serial0/2/1         64         256      0         8       1
```

The dynamics associated with the actual WFQ process can be seen via the **show interface** command when we generate traffic from R9 to R4. Observe the 5 minute input and output rates:

```
R9#ping 4.4.4.4 repeat 10000

Type escape sequence to abort.
Sending 10000, 100-byte ICMP Echos to 4.4.4.4, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! <output omitted>
```

Observe the difference between this output and the previous.

```
R6#show interfaces FastEthernet0/1 | be Queueing
  Queueing strategy: weighted fair
  Output queue: 0/1000/4096/0 (size/max total/threshold/drops)
     Conversations  0/2/4096 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
  5 minute input rate 54000 bits/sec, 56 packets/sec
  5 minute output rate 53000 bits/sec, 56 packets/sec
     21343 packets input, 2427452 bytes
     Received 1223 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog
     0 input packets with dribble condition detected
     21205 packets output, 2400558 bytes, 0 underruns
     0 output errors, 0 collisions, 4 interface resets
     157 unknown protocol drops
```



```
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
```

Class-Based Weighted Fair Queuing (CBWFQ)

CBWFQ expands on the basic WFQ algorithm such that it includes user defined static classes of traffic. These classifications of traffic can be selected based on network protocol; access control lists (ACL), IP precedence, DSCP values or even input interface. In CBWFQ, each statically assigned class of traffic will be queued in a separate queue, and all packets matching that class will be assigned to that specific queue.

Once a class has been created and traffic has been identified that matches the criteria in that class CBWFQ allows those packets to be serviced in a preferential fashion. Each class of traffic thus identified can be thought of as having a specific share of the available bandwidth. We have to be very clear here to identify the fact that this bandwidth guarantee is not a reservation; it is better described as a minimum guarantee of bandwidth that is available during periods of congestion. It is just as important to note that if a class is not using its assigned bandwidth that bandwidth is available to other classes. This describes the functional behavior of CBWFQ where first the algorithm will ensure the minimum bandwidth guaranteed to a specific class, and then provide a equal or fair share of any unassigned bandwidth to all active flows.

Troubleshooting CBWFQ

The problem with troubleshooting CBWFQ is the fact that CBWFQ is not an actual network QoS mechanism in the most basic sense. CBWFQ is instead a queuing mechanism that operates on individual routers, and that is where the troubleshooting has to start when issues surface involving CBWFQ. The most useful tools in our toolkit are show commands. Specifically, show commands that allow us to verify that policies have been correctly configured, and applied to the appropriate interfaces. The paramount thing to determine is whether CBWFQ is working.

```
R6#show policy-map interface
 FastEthernet0/1

  Service-policy input: SET_DSCP

    Class-map: TELNET_3023 (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      QoS Set
        dscp af41
          Packets marked 0

    Class-map: CS4 (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 100
      QoS Set
        dscp cs4
```

```
        Packets marked 0

  Class-map: CS2 (match-all)
    0 packets, 0 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: access-group 1
    QoS Set
      dscp cs2
        Packets marked 0

  Class-map: CS3 (match-all)
    1 packets, 94 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: access-group 2
    QoS Set
      dscp cs3
        Packets marked 1

  Class-map: class-default (match-any)
    0 packets, 0 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: any
```

This output will help us determine whether our CBWFQ application is working or not. In situations where things are not going as planned it will be necessary to see if the defined classes are receiving the correct bandwidth guarantees. In some advanced situations, it may be advisable to test the network with and without the policies applied.

Low Latency Queuing (LLQ)

As mentioned in the ***Congestion Management Technology Review***, low latency enhancements made to the queuing algorithm now allows CBWFQ to send traffic to a strict priority queue. One or more classes can be assigned a priority status within a given policy map. However, when multiple classes within a single policy map are configured with priority, all traffic from these classes will be enqueued to the same single, strict priority queue.

Troubleshooting LLQ

To troubleshoot low latency queuing it is necessary to have a firm grasp of what to expect from the router based on its configuration and how any impacted applications are performing. Should it seem that applications are not receiving the low latency treatment they should be it may be necessary to utilize the **debug priority** command. Any packets that are dropped based on the LLQ configuration will be logged in the output. This command was created for use with other forms of legacy queuing with priority, but it will send notifications via console messages any time packets are dropped as a result of a priority queuing scheme where the low latency queue is overflowing. Additionally, we can see the results of packet drops in the output of the **show interface**:

```
R6#show policy-map interface S0/2/0
 Serial0/2/0

  Service-policy output: PRIORITY
```

```
queue stats for all priority classes:
  Queueing
  queue limit 64 packets
  (queue depth/total drops/no-buffer drops) 0/40770/0
  (pkts output/bytes output) 23882/21828698

Class-map: class-default (match-any)
  64651 packets, 83149215 bytes
  5 minute offered rate 1958000 bps, drop rate 1525000 bps
  Match: any
  Priority: 8 kbps, burst bytes 1500, b/w exceed drops: 40770
```

The output indicates that the 40770 packets have been dropped based on exceeding the 8kbps priority applied to the interface under the class-default class.

Common Issues with Congestion Management Technologies

Troubleshooting congestion management technologies regardless of the specific mechanism employed can best be approached by classifying any failure into one of four broad categories:

Lack of QoS

In the ***Technology Review*** section, this text discussed the different configuration modes and queuing strategies that can be used to enact quality of service policies on Cisco IOS devices to meet various congestion management needs. Based on these previous discussions we have to acknowledge the fact that it is possible that an assigned service policy may not actually take effect on a given interface. More often than not, this is a result of a configuration error on the part of the administrator. The problem associated with the type of scenario however is isolating the area within the configuration where the mistake has been made, a process that must be completed end-to-end along the path the particular class of traffic follows. This means that is necessary to ask a series of leading questions on a hop-by-hop basis through the network. These questions should include, but are not necessarily limited to:

- Are the packets being classified or marked correctly as the enter the network?
- Has the QoS policy been consistently applied end-to-end?
- Is the policy applied to the correct interface based on routing or forwarding path failover?
- Do the interfaces in question show the correct queuing strategy?

Excessive Drops

The purpose of applying Quality of Service initially was to avoid tail drop scenarios. It should come as no surprise that erroneously designed or misconfigured application of these congestion management strategies can exacerbate existing problems or create them where the normally would not exist. In situations where drops either increase as a result of applying fancy queuing schemes or when they "creep" into the network over time it is best to use the Cisco IOS show commands to isolate where and why the drops are taking place. Again, this process will need to be executed on a hop-by-hop basis along the path the particular packets are following to reach their desired destination. During this process it is important to ask the following questions:

- Are there any drops or errors occurring on WAN links?
- Are the drop counters incrementing for WFQ, CBWFQ or LLQ?
- Has the same queuing strategy been applied on all outbound interfaces?
- Are there any speed or duplex mismatches on the Ethernet links?

Induced Latency

In the ***Technology Review*** section, we discussed how the priority command can be used to alleviate latency. It was also mentioned that the possibility exists where more than one classification of traffic can be assigned to our single strict priority queue. In this situation these two (or more) classification will compete for the available bandwidth assigned to the strict priority queue on a FIFO basis. This can generate undesired behavior in the network should one or more of these classifications of traffic be overly aggressive. Keep in mind that it was mentioned that voice traffic was "well behaved" but video

was not "well behaved". Placing these two classifications of traffic in the same strict priority queue could and often will generate the opposite of the desired effect. When voice application specific issues like jitter manifest themselves inside the network after the application of a mechanism such as LLQ this contention between traffic classes could be the cause. However, on very slow links like frame relay, even LLQ may not be enough to prevent jitter or dropped voice communications. In these situations, it may be necessary to employ a mechanism like link fragmentation and interleaving to improve the actual serialization delay of packets onto the physical link because larger packets may get in the way. LFI will be covered in depth in ***Chapter 11: Link Fragmentation and Interleaving (LFI)***.

Incorrect QoS Treatment

This is a scenario where a queuing mechanism has be applied but the desired effect is not taking place. More often than not, this is a situation where the given policy is misconfigured. The most common cause of this type of problem are:

- Incorrect bandwidth statement used to specify the allocated bandwidth. The most common issue associated with this type of configuration mistake is using the wrong unit of measurement.
- Incorrect priority statement used to specify the allocation of bandwidth to the strict priority queue. Much the same problem as that found with the previous bandwidth command.
- Is the policy map calling the correct class map?
- Are the class-maps configured properly?

Chapter Challenge: Congestion Management Trouble Tickets

The following section includes two Trouble Tickets associated with the problematic congestion management scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that R4 is dropping an excessive number of packets out its FastEthernet0/1 interface during periods of congestion. These packet drops are associated with many different traffic flows. You have been instructed to correct this issue and show that R4 reflects the output of the following command:

```
R4#show interfaces FastEthernet0/1 | sec [Q|q]ueue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/256/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
```

Trouble Ticket #2

Your supervisor has brought to your attention that traffic originating from networks attached to R4 are not receiving any CBWFQ treatment as they traverse R6 to reach R9. You are not allowed to alter any policy-map or class-map in the process of remediating this issue.

Chapter Challenge: Congestion Management Trouble Ticket Solutions

The following section includes solutions to the two Trouble Tickets associated with the problematic congestion management scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that R4 is dropping an excessive number of packets out its FastEthernet0/1 interface during periods of congestion. These packet drops are associated with many different traffic flows. You have been instructed to correct this issue and show that R4 reflects the output of the following command:

```
R4#show interfaces FastEthernet0/1 | sec [Q|q]ueue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/256/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
```

Step 1 - Fault Verification:

What is the current output of this show command on R4?

```
R4#show interfaces FastEthernet0/1 | sec [Q|q]ueue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/10/0 (size/max total/threshold/drops)
     Conversations  0/1/16 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
```

This output does not match the desired configuration provided in the exhibit. We can see that the current queuing strategy is “weighted fair”, but the parameters do not match. In the current configuration the output queue will only allow up to 10 packets into a given queue, and we are only allowing there to be up to 16 queues to be created dynamically. The exhibit demonstrates that there should be up to 256 queues with a configured of threshold of 256 packets per queue. This can be corrected by changing the current configuration under FastEthernet0/1:

```
R4#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface FastEthernet0/1
R4(config-if)#fair-queue 256 256
R4(config-if)#end
```

Now we can most quickly verify the configuration by using the same show command to see the new output:

```
R4#show interfaces FastEthernet0/1 | sec [Q|q]ueue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/256/0 (size/max total/threshold/drops)
```

```
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
```

Compared to the desired output:

```
R4#show interfaces FastEthernet0/1 | sec [Q|q]ueue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/256/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
```

As we can see, the parameters now match perfectly.

Trouble Ticket #2

Your supervisor has brought to your attention that traffic originating from networks attached to R4 are not receiving any CBWFQ treatment as they traverse R6 to reach R9. You are not allowed to alter any policy-map or class-map in the process of remediating this issue.

Step 1 - Fault Verification:

What policy should be applied to this traffic?

```
R6#show policy-map interface
 FastEthernet0/1

  Service-policy output: QOS

    Class-map: WWW (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
      queue limit 30 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      bandwidth 60% (60000 kbps)

    Class-map: FTP (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol ftp
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      bandwidth 30% (30000 kbps)
```

```
    Class-map: TELNET (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      bandwidth 5% (5000 kbps)

    Class-map: class-default (match-any)
      392 packets, 35317 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 392/38416
```

We can clearly see a policy exists. But, what interface is the policy being applied to? FastEthernet0/1 points to R4 not R9. The task specifically tells that QOS treatment should be applied to packets going to R9 through R6. In this instance R6 is applying CBWFQ to traffic returning to R4 from R9. This can be corrected by removing the service-policy command from FastEthernet0/1 and applying it to Serial0/2/0.

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#interface FastEthernet0/1
R6(config-if)#no service-policy output QOS

R6(config-if)#interface Serial0/2/0
R6(config-if)#service-policy output QOS
R6(config-if)#end
```

Now what type of queuing is taking place on Serial0/2/0 outbound?

```
R6#show interface Serial0/2/0 | inc strategy
  Queueing strategy: Class-based queueing
```

CBWFQ is being applied outbound on R6 toward R9 now. Be very careful of where service policies are applied, and be mindful of directionality.

Chapter 7: Congestion Avoidance

An interesting area of Quality of Service (QoS) deals with the concept of avoiding congestion before it can even occur at a key interface in the topology. This can be accomplished by randomly dropping low priority packets at a particular defined threshold. This chapter explores congestion avoidance mechanisms in detail.

Congestion Avoidance Technology Review

Before we dive into the operation of congestion management tools at the Cisco command line, we need to review the technologies for this section.

Note: This chapter deals with congestion avoidance tools for Cisco routers. This book details congestion avoidance tools for the Catalyst switches in great depth in ***Chapter 13: Catalyst Quality of Service (QoS)***.

Avoiding Tail Dropped Packets by Avoiding Congestion

In ***Chapter 6: Congestion Management*** we discussed at length what happens as a software or hardware queue fills with packets. We covered the primary issues associated with full queues and how this process can foster a condition known as *global tcp synchronization*. In the previous chapter we discussed a number of very powerful tools created to "manage" congestion of the hardware queue. These mechanisms buffer packets in software based structures we call queues. This method of software queuing packets as they wait to gain admission to the transmit-ring discussed in ***Chapter 6: Congestion Management*** is a normal and logical process. However, what happens when or if these software queues fill up?

The answer to that question is non-discriminatory drop otherwise referred to as tail drop. As a whole, we created the software queue to prevent tail drop from taking place, but in the event that these queues become so full that packets can no longer be admitted into the queue we still manage to encounter tail drop and all the ensuing issues associated with it. It is important to note at this time that there are protocols and applications less sensitive to tail drop. Imagine an application that does not have the ability to retransmit lost information, a successive group of dropped packets associated with a single communication or multiple communications could have devastating effects. Additionally, there are some applications developed specifically to adapt to lost or dropped packets. TCP based applications fall in this category of drop resilient protocols.

TCP based applications have the ability to respond to dropped packets by backing off and transmitting data at a slower rate. Nevertheless, even TCP has its own issues. The first is the mechanism the protocol uses to determine when to backoff and change its transmission rate. We already stated that dropped packets drive this process. It is safe to assume that before TCP will adapt its communication rate the associated queue must be full enough to start the tail drop process on any given interface in the data path. This means that it is possible for the wrong data flow to be instructed to slow down. This happens because tail drop is an entirely non-discriminatory process. The second issue associated with a TCP-based application is the notion of *global synchronization*.

Global synchronization occurs when many different TCP flows all encounter tail drop at the same time. The fact that these individual applications use the same mechanism to manage their flow rate means that they will all backoff at the same time. These applications will then begin to steadily increase their transmission rates until they again encounter packet loss. It has to be noted that these flows are all managed by the same algorithm and because their packets will be dropped simultaneously, they all begin to synchronize their flow management behavior. This creates a wave-like surge of traffic rates that will repeat for as long as there is congestion on the interface or in the associated queue.

This begs the question, "How can we prevent global synchronization and manage tail drop?" The answer is Random Early Detection (RED).

Random Early Detection or the Cisco IOS implementation of RED, known as Weighted Random Early Detection (WRED) is a mechanism that allows preemptively scheduling the dropping of packets before a queue becomes full. Again, we need to revisit the queuing mechanism to better visualize this concept. If the hardware queue is not congested none of these algorithms do anything. However, when the link is congested we now have either statically defined queuing or dynamic queuing taking place based on our configuration. In situations where these software queues become full, we have the tail drop scenario we have been discussing. However, now IOS gives us a tool to change the behavior inside the software queue itself.

Imagine a mechanism where we can thin out the queue before it becomes completely full. By doing this we could at best pick when and which TCP-based applications will backoff, or at least randomize the process such that we prevent global synchronization from taking place. Visualize a queue that can hold 20 packets. Without WRED, this queue could continue to accept packets until the packet count reached its maximum. Once 20 packets are in the queue, packets will be discarded as they try to enter the queue because there is no room for more. WRED introduces a new concept to the software queue called a minimum threshold. This threshold allows us to define a level of packets in the queue at which we will begin a random drop process. This new mechanism offers some very powerful advantages. Primarily the most aggressive flows are going to be the most likely flows dropped. We know this based on the knowledge that these flows will have more packets in the queue and a higher probability of drops occurring and thus being forced to backoff. Secondarily, by randomly dropping packets we are going to avoid global TCP synchronization.

One more element regarding WRED needs discussed and that is the concept of a mark probability denominator (MPD). The mark probability denominator describes how aggressively the WRED process will discard packets. Note that before we reach the defined minimum threshold, no packets will be discarded by the WRED mechanism, but once the threshold has been reached the algorithm governing random drops will initiate. The MPD defines the highest amount of packets dropped by the WRED process and the value will scale as the queue continues to become more full. Therefore, the actual number of packets dropped by WRED will "ramp" up to the maximum value based on a logical progression from the minimum threshold to the maximum size of the queue. It is possible that even with WRED detection enabled that the queue can still fill up faster than packets can be discarded. The outcome of this situation will be tail drop.

"Weighted"-RED

The explanation of the mechanism associated with the Cisco IOS deployment of Random Early Detection above outlines the process and logic governing the mechanism itself. However, there are additional issues that we need to understand about WRED and its operation. Specifically we need to discuss the "weighted" component of WRED. Cisco IOS deploys an algorithm that is very similar to the industry standard RED values, however, WRED takes into account concepts like IP precedence and DSCP to name a few.

WRED will selectively prefer to drop packets with lower QoS markings before those with higher markings. A process accomplished by having lower minimum thresholds established for lower traffic markings. This means that as congestion persists on a link, packets with lower markings will be discarded before more valuable traffic like voice. This mechanism protects latency/loss sensitive traffic at the expense of less valuable traffic so the greater the volume of high priority traffic on the network the less useful this weighting mechanism will be. This fact proves to emphasis the importance of being mindful of how traffic is marked and categorized on the network, if you are too generous with your marking of high priority traffic you can invalidate or degrade the performance of a mechanism like WRED.

WRED Burst Sensitivity

The algorithm employed by WRED was never meant to be instantaneous. The process uses an exponential moving time window to calculate an average queue depth rather than rely on a constant queue depth threshold. In simple language, this means that WRED will not react immediately to bursts of packets in the short term. However, if the queue should remain congested by bursts of traffic over the long term the algorithm will become more aggressive when deciding whether to drop packets or not. This process means that WRED will not punish momentary short bursts of traffic, but it will punish applications that regularly attempt to oversubscribe network resources.

We can adjust this "sensitivity" to bursts of traffic using the exponential-weighting-constant. This command specifies the weighting value used by WRED when it calculates the average queue length. The default value used by Cisco IOS is (9) nine. However, this field can be any number 1-16 inclusive. The lower the exponential-weighting-constant is, the more quickly the average queue depth will change, and by raising the value, you instruct the algorithm to react slower thus making it less sensitive to even long term bursts of traffic.

The Operation and Troubleshooting of Congestion Avoidance

Now that we have reviewed the technologies of this chapter, it is time to analyze their operation at the Cisco command line.

Congestion Avoidance

For exploring the concepts behind congestion avoidance, we will utilize the following topology:

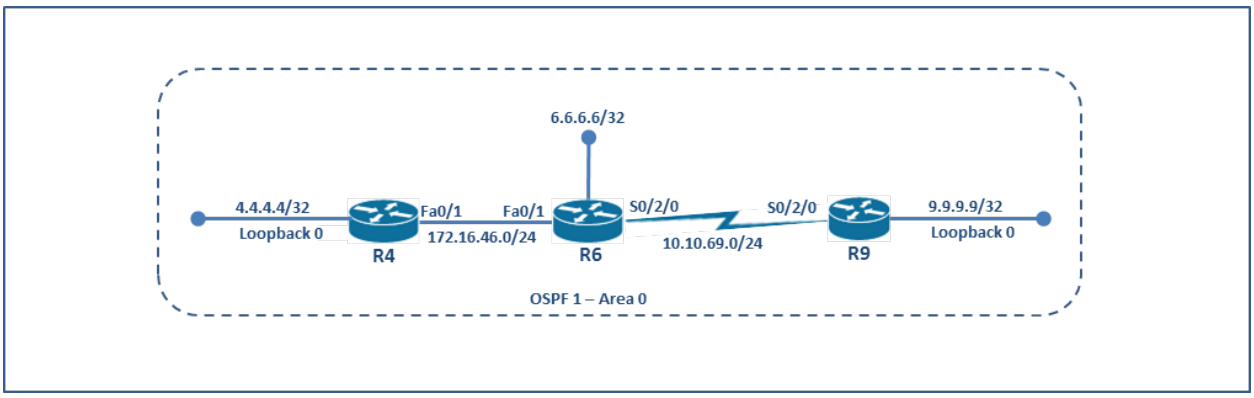

Figure 7-1: A Basic Quality of Service Topology

The Class Based Weighted Fair Queuing deployment of Quality of Service supports three - drop methodologies.

- Tail Drop
- Congestive Discard Threshold (CDT)
- Weighted Random Early Detection (WRED)

Tail drop is the customary default for user-defined classes of traffic, CDT is the normal tool employed by WFQ, and WRED is an option applied under user-defined classes as an alternate behavior to classic tail drop. We discussed the WRED process and the nature of its behavior and deployment characteristics in the ***Congestion Avoidance Technology Review*** section. Now we will take a very close look at the actual command line interface configuration of the process and its operational features. MQC's deployment of WRED requires the three-part process used in all CBWFQ deployments. First, we need a class-map to identify the traffic of interest. Second, there must be a policy map created to define that traffic's treatment with in its queue and finally the application of the policy under an interface using the service-policy command.

For the purpose of this demonstration, we will employ a class-map to match telnet traffic:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#class-map TELNET
R6(config-cmap)#match protocol telnet
R6(config-cmap)#end
```

Now that we have a class of traffic, we will simply enable random early detection:

```
R6(config)#policy-map AVOIDANCE
```

```
R6(config-pmap)#class TELNET
R6(config-pmap-c)#bandwidth percent 35
R6(config-pmap-c)#random-detect
R6(config-pmap-c)#end
```

Lastly, so that we can observe the policy as it applies to an interface we will deploy it on Serial0/2/0 of R6:

```
R6(config)#interface Serial0/2/0
R6(config-if)#service-policy output AVOIDANCE
R6(config-if)#exit
```

Now we can see the default behavior of the command with the **show policy-map interface** command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      bandwidth 35% (540 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        class     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                  pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        0                0/0               0/0              0/0                 20
40  1/10
        1                0/0               0/0              0/0                 22
40  1/10
        2                0/0               0/0              0/0                 24
40  1/10
        3                0/0               0/0              0/0                 26
40  1/10
        4                0/0               0/0              0/0                 28
40  1/10
        5                0/0               0/0              0/0                 30
40  1/10
        6                0/0               0/0              0/0                 32
40  1/10
        7                0/0               0/0              0/0                 34
40  1/10

<output omitted>
```

Observe that the default behavior of the command is to apply ip prec-based WRED with a MPD of 10. Notice that the minimum thresholds get progressively higher as the IP precedence values get higher. Additionally, observe the exponential-weighting-constant defaulted to 9 as we predicted.

Now we need to generate a telnet session from R4 to the loopback of R9 and observe the results of via the **show policy-map interface** command on R6:

```
R4#telnet 9.9.9.9
Trying 9.9.9.9 ... Open


User Access Verification

Password:
R9>en
Password:
R9#
```

Now on R6:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      91 packets, 4230 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 91/4230
      bandwidth 35% (540 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        class     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                  pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        0               0/0               0/0              0/0                 20
40  1/10
        1               0/0               0/0              0/0                 22
40  1/10
        2               0/0               0/0              0/0                 24
40  1/10
        3               0/0               0/0              0/0                 26
40  1/10
        4               0/0               0/0              0/0                 28
40  1/10
        5               0/0               0/0              0/0                 30
40  1/10
```

```
        6               91/4230            0/0               0/0                32
40  1/10
        7                0/0               0/0               0/0                34
40  1/10
```

Notice that the counters for the IP Precedence value of 6 are incrementing. Why would telnet be marked with a value of ip precedence 6? The ip precedence level of 6 assigned by default on all Cisco routers. This can be changed via the ip telnet tos command. We will change the value to an IP precedence of 3:

```
R4(config)#ip telnet tos 60
```

We will reinitiate the telnet session on R4 for R9's loopback 0 interface:

```
R4#telnet 9.9.9.9
Trying 9.9.9.9 ... Open


User Access Verification

Password:
R9>en
Password:
R9#
```

Now we will use the show policy-map interface command to verify the behavior:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      2720 packets, 120158 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 2720/120158
      bandwidth 35% (54 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        class     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                  pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        0                0/0               0/0               0/0                20
40  1/10
        1                0/0               0/0               0/0                22
40  1/10
```

```
        2               0/0               0/0               0/0                24
40  1/10
        3              27/1242            0/0               0/0                26
40  1/10
        4               0/0               0/0               0/0                28
40  1/10
        5               0/0               0/0               0/0                30
40  1/10
        6            2693/118916          0/0               0/0                32
40  1/10
        7               0/0               0/0               0/0                34
40  1/10
<output omitted>
```

Now observe that the ip precedence value for 3 is incrementing rather than 6. The default prec-based behavior can be changed by modifying the existing policy-map. In this instance we will specify dscp-based behavior for the WRED:

```
R6(config)#policy-map AVOIDANCE
R6(config-pmap)#class TELNET
R6(config-pmap-c)#no random-detect
R6(config-pmap-c)#random-detect dscp-based
R6(config-pmap-c)#end
```

Now if we verify the configuration with the **show policy-map interface** command we will see that we are running dscp-base WRED under the policy-map configuration now:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      2722 packets, 120246 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 2722/120246
      bandwidth 35% (54 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob


    Class-map: class-default (match-any)
      772 packets, 55397 bytes
      30 second offered rate 0 bps, drop rate 0 bps
```

```
      Match: any
```

We can test the process by generating a telnet session from R4 to R9:

```
R4#telnet 9.9.9.9
Trying 9.9.9.9 ... Open


User Access Verification

Password:
R9>en
Password:
R9#
```

Now we will verify on R6 using the show policy-map interface command:

```
 R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      2745 packets, 121310 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 2745/121310
      bandwidth 35% (54 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        cs6           23/1064            0/0              0/0                  32
40  1/10
```

Now we need to look at how to manipulate the thresholds and MPD for a single class of traffic:

```
R6(config-pmap-c)#do show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      58 packets, 2666 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
```

```
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 58/2666
      bandwidth 35% (54 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        cs6           36/1646            0/0              0/0                 32
40  1/10
```

Observe that the thresholds for the CS6 traffic class are set to 32 and 40 with a MPD of 10. We will change this via the **random-detect dscp** command under the policy-map:

```
R6(config)#policy-map AVOIDANCE
R6(config-pmap)#class TELNET
R6(config-pmap-c)#random-detect dscp ?
  <0-63>   Differentiated services codepoint value
  af11     Match packets with AF11 dscp (001010)
  af12     Match packets with AF12 dscp (001100)
  af13     Match packets with AF13 dscp (001110)
  af21     Match packets with AF21 dscp (010010)
  af22     Match packets with AF22 dscp (010100)
  af23     Match packets with AF23 dscp (010110)
  af31     Match packets with AF31 dscp (011010)
  af32     Match packets with AF32 dscp (011100)
  af33     Match packets with AF33 dscp (011110)
  af41     Match packets with AF41 dscp (100010)
  af42     Match packets with AF42 dscp (100100)
  af43     Match packets with AF43 dscp (100110)
  cs1      Match packets with CS1(precedence 1) dscp (001000)
  cs2      Match packets with CS2(precedence 2) dscp (010000)
  cs3      Match packets with CS3(precedence 3) dscp (011000)
  cs4      Match packets with CS4(precedence 4) dscp (100000)
  cs5      Match packets with CS5(precedence 5) dscp (101000)
  cs6      Match packets with CS6(precedence 6) dscp (110000)
  cs7      Match packets with CS7(precedence 7) dscp (111000)
  default  Match packets with default dscp (000000)
  ef       Match packets with EF dscp (101110)

R6(config-pmap-c)#random-detect dscp cs6 ?
  <1-4096>  minimum threshold (in packet by default)

R6(config-pmap-c)#random-detect dscp cs6 20 ?
  <1-4096>  maximum threshold (in packet by default)
  bytes     number of bytes
  ms        milliseconds

R6(config-pmap-c)#random-detect dscp cs6 20 40 ?
  <1-65535>  mark probability denominator
  <cr>
```

```
R6(config-pmap-c)#random-detect dscp cs6 20 40 5 ?
  <cr>

R6(config-pmap-c)#random-detect dscp cs6 20 40 5
R6(config-pmap-c)#end
```

Now we will look again at the output of **show policy-map interface** again:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: AVOIDANCE

    Class-map: TELNET (match-all)
      58 packets, 2666 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 58/2666
      bandwidth 35% (54 kbps)
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        cs6           36/1646            0/0              0/0                 20
40  1/5
<output omitted>
```

The values are in accordance with the new specification for the cs6 class. Beware of making changes to the default configurations because they may result in odd or unpredictable behavior for one or all classes of traffic.

Troubleshooting WRED

The principle step in troubleshooting devices running WRED is to understand all the commands that we have demonstrated in this section thus far. A complete understanding of these commands will afford an administrator the ability to rely on their output for the purpose of fault isolation. These commands allow us to monitor the performance of WRED, and we should have a solid understanding of what the output of each looks like in a normal running configuration. Based on the understanding of how the protocol works normally it becomes easier to answer qualifying questions regarding the following common issues associated with WRED:

- Higher Priority classes are dropped at a rate higher than expected.
    - You have changed the default operating parameters and issued a non-default exponential-weighting-constant or random precedence/dscp value.

- Tail drop is occurring on a link even though it is running WRED.
    - This may require converting the random-detection mechanism to the legacy flow-based random early detection under the physical interface for more refined granularity than can be offered by the MQC process in Cisco IOS.
- The traffic of interest is not correctly marked with the proper DSCP or IP Precedence values.
    - Correct any class-map or ensure that all traffic is not being marked with the same QoS marking values. Failure to assign different levels of classification will result in WRED operating in accordance with the standard random early detection thus no preference can be assigned to any one class of traffic.

Chapter Challenge: Congestion Avoidance Sample Trouble Tickets

The following section includes two Trouble Tickets associated with the problematic congestion avoidance scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that the show queueing random-detect command should have the following results on R4. You have been instructed to make the necessary changes to reproduce the following show command output. There are multiple problems associated with this ticket.

```
R4#show queueing random-detect
Current random-detect configuration:
  FastEthernet0/0
    Queueing strategy: random early detection (WRED)
    Random-detect not active on the dialer
    Exp-weight-constant: 9 (1/512)
    Mean queue depth: 0

  class           Random drop      Tail drop    Minimum Maximum  Mark
                   pkts/bytes       pkts/bytes    thresh  thresh  prob
      0                          0/0              0/0            20      40  1/10
      1                          0/0              0/0            22      40  1/10
      2                          0/0              0/0            24      40  1/10
      3                          0/0              0/0            26      40  1/10
      4                          0/0              0/0            28      40  1/10
      5                          0/0              0/0            39      40  1/10
      6                          0/0              0/0            33      40  1/10
      7                          0/0              0/0            35      40  1/10
   rsvp                          0/0              0/0            37      40  1/10
```

Trouble Ticket #2

Your supervisor has brought to your attention that the current deployment of CBWFQ with WRED is not configured properly on R9. You have been instructed to make the necessary changes to the configuration to obtain identical results to the attached exhibit.

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: WRED

    Class-map: class-default (match-any)
      15 packets, 962 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 15/1082
        Exp-weight-constant: 7 (1/128)
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
```

```
                  pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        default        8/494             0/0              0/0                20
40  1/10
        cs6            7/588             0/0              0/0                32
40  1/10
```

Chapter Challenge: Congestion Avoidance Trouble Ticket Solutions

The following section includes two Trouble Tickets associated with the problematic congestion avoidance scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that the show queueing random-detect command should have the following results on R4. You have been instructed to make the necessary changes to reproduce the following show command output. There are multiple problems associated with this ticket.

```
R4#show queueing random-detect
Current random-detect configuration:
  FastEthernet0/1
    Queueing strategy: random early detection (WRED)
    Random-detect not active on the dialer
    Exp-weight-constant: 9 (1/512)
    Mean queue depth: 0

  class          Random drop      Tail drop    Minimum Maximum  Mark
                  pkts/bytes       pkts/bytes    thresh  thresh  prob
     0                          0/0              0/0           20      40  1/10
     1                          0/0              0/0           22      40  1/10
     2                          0/0              0/0           24      40  1/10
     3                          0/0              0/0           26      40  1/10
     4                          0/0              0/0           28      40  1/10
     5                          0/0              0/0           39      40  1/10
     6                          0/0              0/0           33      40  1/10
     7                          0/0              0/0           35      40  1/10
   rsvp                         0/0              0/0           37      40  1/10
```

Step 1 - Fault Verification:

What is the status of the output on R4 as it is currently configured?

```
R4#show queueing random-detect
Current random-detect configuration:
  FastEthernet0/1
    Queueing strategy: random early detection (WRED)
    Random-detect not active on the dialer
    Exp-weight-constant: 9 (1/512)
    Mean queue depth: 0

  class          Random drop      Tail drop    Minimum Maximum  Mark
                  pkts/bytes       pkts/bytes    thresh  thresh  prob
     0                          0/0              0/0           20      40  1/10
     1                          0/0              0/0           22      40  1/10
     2                          0/0              0/0           24      40  1/10
     3                          0/0              0/0           26      40  1/10
     4                          0/0              0/0           28      40  1/10
     5                          0/0              0/0           31      40  1/10
     6                          0/0              0/0           33      40  1/10
     7                          0/0              0/0           35      40  1/10
   rsvp                         0/0              0/0           37      40  1/10
```

We can immediately see that the policy is applied to the FastEthernet0/0 interface. That is one problem. But the task advises us the there are multiple problems associated with this ticket. Based on values that we can manipulate, we can see that everything matches in the two exhibits up until we reach the configured minimum and maximum thresholds for ip precedence 5 traffic. The current values are 31 and 40, and the desired values are 39 and 40. No other issues are visible.

To correct this issue we will need to remove the random-detect config from R4's FastEthernet0/0 interface and apply it to the FastEthernet0/1 interface. At the same time we will manipulate the threshold levels for the precedence 5 traffic.

```
4#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface FastEthernet0/0
R4(config-if)#no random-detect
R4(config-if)#exit


R4(config)#interface FastEthernet0/1
R4(config-if)# random-detect
R4(config-if)# random-detect precedence 5 39 40
R4(config-if)#end
```

No we can verify the correct values by repeating the show command:

```
R4#show queueing random-detect
Current random-detect configuration:
  FastEthernet0/1
    Queueing strategy: random early detection (WRED)
    Random-detect not active on the dialer
    Exp-weight-constant: 9 (1/512)
    Mean queue depth: 0

  class          Random drop      Tail drop    Minimum Maximum  Mark
                  pkts/bytes       pkts/bytes    thresh  thresh  prob
     0                          0/0              0/0           20      40  1/10
     1                          0/0              0/0           22      40  1/10
     2                          0/0              0/0           24      40  1/10
     3                          0/0              0/0           26      40  1/10
     4                          0/0              0/0           28      40  1/10
     5                          0/0              0/0           39      40  1/10
     6                          0/0              0/0           33      40  1/10
     7                          0/0              0/0           35      40  1/10
   rsvp                         0/0              0/0           37      40  1/10
```

We clearly see the treatment is applied to the correct interface and the values for precedence 5 traffic have been successfully adjusted.

Trouble Ticket #2

Your supervisor has brought to your attention that the current deployment of CBWFQ with WRED is not configured properly on R9. You have been instructed to make the necessary changes to the configuration to obtain identical results to the attached exhibit.

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: WRED

    Class-map: class-default (match-any)
      15 packets, 962 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 15/1082
        Exp-weight-constant: 7 (1/128)
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        default        8/494             0/0              0/0                 20
40   1/10
        cs6            7/588             0/0              0/0                 32
40   1/10
```

Step 1 - Fault Verification:

What is the status of the show command on R9 currently?

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: WRED

    Class-map: class-default (match-any)
      371 packets, 23250 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 2/108
        Exp-weight-constant: 9 (1/512)
        Mean queue depth: 0 packets
        class     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob
```

```
        0               1/24            0/0             0/0             20
40  1/10
        1               0/0             0/0             0/0             22
40  1/10
        2               0/0             0/0             0/0             24
40  1/10
        3               0/0             0/0             0/0             26
40  1/10
        4               0/0             0/0             0/0             28
40  1/10
        5               0/0             0/0             0/0             30
40  1/10
        6               1/84            0/0             0/0             32
40  1/10
        7               0/0             0/0             0/0             34
40  1/10
```

We immediately see that the policy is applied to the correct interface but the WRED configuration is currently supporting precedence-based WRED. Additionally, the current configuration is using the default exponential-weighting-constant value of 9, where the exhibit is using a non-default value of 7.

This can be corrected by editing the existing policy-map under the class-default class.

```
R9(config)#policy-map WRED
R9(config-pmap)#class class-default
R9(config-pmap-c)#random-detect dscp-based
random-detect [precedence] must be disabled first.
```

Observe that to change the mode of WRED it is necessary to remove the previous commands.

```
R9(config-pmap-c)#no random-detect
R9(config-pmap-c)#random-detect dscp-based
R9(config-pmap-c)#random-detect exponential-weighting-constant 7
R9(config-pmap-c)#end
```

Don't forget to change the weighting constant, and then verify that the output of the show policy-map interface command now matches the exibit:

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: WRED

    Class-map: class-default (match-any)
      485 packets, 30024 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 40/3006
        Exp-weight-constant: 7 (1/128)
```

```
        Mean queue depth: 0 packets
        dscp     Transmitted       Random drop      Tail drop          Minimum
Maximum     Mark
                 pkts/bytes     pkts/bytes       pkts/bytes          thresh
thresh     prob

        default       21/1410            0/0              0/0                20
40  1/10
        cs6           19/1596            0/0              0/0                32
40  1/10
```

Chapter 8: Traffic Shaping

Need to control the rate at which traffic is being sent from an interface? Traffic shaping can assist. Traffic that is to be sent that is exceeding a certain rate can be buffered and sent later. This chapter explores this important option of the DiffServ approach to Quality of Service.

Traffic Shaping Technology Review

Before we dive into the operation of traffic shaping at the Cisco command line, we need to review the technology for this section.

Note: This chapter deals with traffic shaping tools for Cisco routers.

Traffic Shaping Defined

Traffic shaping as a technology is divided into two distinct concepts: Generic Traffic Shaping (GTS) and Frame Relay Traffic Shaping (FRTS). Each of these variations utilizes the same basic principles, but they employ different implementation methods, interact differently with other quality of service mechanisms and have some terminology differences. However, despite these differences traffic shaping, no matter the type, is a tool used in situations where outbound traffic is required to respect a specific maximum transmission rate.

Generic Traffic Shaping

Generic traffic shaping employs a single token bucket concept to regulate the amount of traffic allowed to exit an interface. GTS can be deployed on an interface-by-interface basis, and offers the ability to further classify network traffic through the use of extended access lists. These ACLs allow the mechanism to manage these different classifications of traffic with different shaping policies. The important thing to note is that a TS mechanism will restrict the outbound traffic flow to a particular speed, while simultaneously buffering burst in excess of the configured rate. This has a direct effect of "smoothing out" or "shaping" the traffic into a flow that conforms to the configured specifications.

GTS uses a concept referred to as a token bucket to shape traffic. There is a careful distinction that has to made here. Previously we discussed a "leaky" bucket that is filled with packets, but now we are talking about a "token" bucket. A token bucket should be thought of as being filled with tokens, not packets. A token in this scenario should be seen as permission for a specific number of bits to be transmitted to the network. A token bucket is also commonly referred to as a "scheduler" and it assigns credits to traffic for transmission. Before a packet can be transmitted, a certain number of tokens need to be removed from the bucket. Token Buckets are refilled at a constant rate and can be configured for different sizes. If a token bucket is full, any newly arriving tokens will be discarded. However, if the bucket is empty, an incoming packet must wait for enough tokens to fill the bucket before it can be de-queued by the scheduler. This means that the maximum burst size is roughly proportional to the overall size of the bucket.

The token bucket analogy relies on five main variables to perform its role: Tc, Bc, Be, CIR, and the Shaped Rate.

- **Tc –** Time interval (in milliseconds) over which the committed burst (Bc) can be sent. Tc = Bc/CIR
- **Bc –** Committed burst size (in bits). This is the amount of traffic that can be sent over an interval Tc
- **CIR –** Committed Information Rate (in bits per second). The rate defined in the traffic contract
- **Shaped Rate –** The rate at which a particular traffic is shaped. It could be same as CIR or higher than CIR.

- **Be – Excess burst size (in bits). This is the number of bits that can be sent beyond Bc**

GTS uses the token bucket concept to send data equaling the burst size over a single time interval. Thus, the CIR is the burst size divided by the time interval. This allows traffic to be throttled down to what the interface and network can manage. Using this model, arbitrary bursts in traffic will be shaped to the configured rate by buffering, rather than being discarded.

Frame Relay Traffic Shaping (FRTS)

Frame relay traffic shaping employs a single token bucket concept to regulate the amount of traffic allowed to exit an interface or specific permanent virtual circuit. Specifically, Frame-Relay congestion control can be done on a DLCI basis. Actual flow control is accomplished at the upper layer protocols. Keep in mind that frame-relay is just a layer 2 protocol and restricted to its capabilities and operations. That means that frame-relay will simply discard packets and only provides notifications during periods of congestion.

In frame-relay, the messages that are sent during these periods of congestion are called congestion-notification messages and they support the protocol's ability to regulate transmission rates. As an example if a given sender is transmitting data faster than the link can support, the frame switch will send a congestion-notification to the sender to slow down. Frame-relay, in the interim will simply discard packets until the sender complies. If a packet is lost, the receiving device has to request a retransmission of that packet, if the associated application supports retransmission, if not the packet is simply lost.

Just like its generic traffic shaping cousin, frame-relay traffic shaping has its own terms and definitions that need to be fully understood before we even attempt to configure it on a device.

- **FECN** – Forward-explicit congestion notification
- **BECN** – Backward-explicit congestion notification
- **DE** – Discard Eligibility
- **CIR** – Committed Information Rate

Both FECN and BECN notifications are represented as a single bit in the frame-relay header. An addition piece of information found in the header is the DE bit, which is responsible for dropping and marking non-important traffic during times of congestion. A frame with the DE bit set has low priority and can be dropped in case of congestion in favor of traffic that is not marked as discard eligible.

CIR represents the agreement about the normal transmission rate over a link and it is defined in terms the volume of data that can be transmitted in a specified window of time. Traditionally, this unit of time is one second.

BECN – indicates to the receiving device that frames sent in the opposite direction may experience congestion.

FECN – indicates to the receiving device that the frame has encountered congestion.

DE – when the DE bit is set in a frame, the packet could be selected for discard. It is possible to manually set the DE bit for certain protocols.

The following values can be configured on an interface DLCI with a map-class command:

- **EIR** is the rate that you can exceed CIR before the ingress port will discard your frames.
- **CIR** – Committed Information Rate is the average rate that you can transmit at measured over a time period Tc without having the ingress port on the Frame network flag your frames as over contract by setting the DE bit. If you have multiple connections, the aggregated total of the CIR for all PVC's should not exceed the physical port speed.
- **Bc** – Committed Burst is the maximum amount of data that a network will transfer in one interval of time (Tc) or in other words is the total number of bits that you can send in period Tc and not have the frames marked with DE.
- **Be** – Burst in excess is the maximum amount of data that the network will transfer above Bc in one interval of time. When transmitting data at excess burst (Be) these frames would either be dropped by the network at ingress or marked discard eligible (DE) and potentially dropped somewhere else in the network if congestion is experienced or you can say it is the total number of bits in addition to Bc that you can send in period Tc that the switch will allow before discarding any excess traffic at the ingress port to the Frame network.
- **MINCIR** – minimum committed information rate is the actual guaranteed rate obtained from service provider in bps. This value should be the minimum rate you should drop to in the event of congestion (dropping below this rate implies you are not getting the bandwidth you are paying for). In certain cases, the **mincir** and **cir** values must be the same. The value of **mincir** is half of the CIR value in bps by default.
- The default Tc is 1/8 sec = 0.125 seconds, and is calculated based on the formula Bc=CIR*Tc

For example, if you have a CIR of 64000 Bits per second and want for voice a Tc of 10 ms or 1/100, then the Bc would be 640 bits. Note that 10 ms is the smallest value possible for a Tc.

Frame-relay traffic shaping can be configured in one of three principle methods.

- Generic Traffic Shaping
- Legacy Frame Relay Traffic Shaping
- Class Based Frame Relay Traffic Shaping

Generic Traffic Shaping

As the name implies, there is nothing different about this version of traffic shaping verses the GTS we discussed already. The only difference is that the shaping command has been applied to a frame relay

enabled interface. The **traffic-shape** command applied in this fashion only allows the Bc, Be and queue depth of the actual shaping queue to be manipulated. The notion of the depth of the shaping queue is important, because this is how traffic will be delayed by the traffic shaping process if necessary. The queue that is created during this process employs standard WFQ as its queuing strategy. This WFQ process will dynamically create queues based on their flow characteristics; however, values like congestive discard thresholds and max number of queues cannot be manually specified.

```
R6(config)#interface Serial0/2/0
R6(config-if)#traffic-shape rate 128000
R6(config-if)#end
```

We can see the results of the configuration via the show traffic-shape command:

```
R6#show traffic-shape

Interface   Se0/2/0
       Access Target     Byte   Sustain   Excess    Interval  Increment Adapt
VC     List   Rate       Limit  bits/int  bits/int  (ms)      (bytes)   Active
-              128000     1984   7936      7936      62        992       -
```

Additionally, we can clearly see the nature of the queue that was created as a result of the traffic-shape operation.

```
R6#show traffic-shape queue
Traffic queued in shaping queue on Serial0/2/0
  Queueing strategy: weighted fair
  Queueing Stats: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/0/16 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 128 kilobits/sec
```

Observe that fair queuing is in effect, as we discussed, and we can see that the available bandwidth made to the queue is the shaping rate we specified.

Legacy Frame-Relay Traffic Shaping

The deployment of legacy frame-relay is implemented via map-class commands. This method, even though it is considered legacy, is still used in many networks. As mentioned, all parameters are initiated under the **map-class frame-relay** command:

```
map-class frame-relay PVC_100
 frame-relay cir 128000
 frame-relay bc 1280
 frame-relay be 0
 frame-relay mincir 64000
 frame-relay adaptive-shaping becn
```

Once the map-class has been created, it is applied under the individual pvc in question:

```
interface Serial 0/2/0
 frame-relay interface-dlci 100
  class PVC_100
```

Before this configuration will take effect it will be necessary to apply the frame-relay traffic-shaping command at the interface level.

```
R6(config)#interface Serial0/2/0
R6(config-if)#no traffic-shape rate 128000
R6(config-if)#frame-relay traffic-shape
R6(config-if)#end

R6#show traffic-shape

Interface   Se0/2/0
       Access Target    Byte   Sustain   Excess    Interval  Increment Adapt
VC     List   Rate      Limit  bits/int  bits/int  (ms)      (bytes)   Active
200           56000     875    7000      0         125       875       -
100           128000    160    1280      0         10        160       BECN
```

Once this command is applied all circuits will be shaped, DLCIs not configured with a **map-class** will receive the default parameters as part of this process as for example all unspecified PVCs will be assigned a CIR of 56Kbps.

Class Based Frame Relay Traffic Shaping

This is the three-phase implementation created as part of MQC as it applies to traffic shaping on frame-relay enabled WAN links. The issue regarding this traffic shaping deployment method is its reliance on terminology that is still fundamentally rooted in the concepts and definitions used in legacy frame-relay traffic shaping. Another critical issue to keep in mind is that all “class-based” policies applied in this fashion must be applied specifically to the class class-default category. As an example to configure treatments specific to frame relay traffic shaping we would apply in a fashion similar to the following:

```
R6(config)#policy-map CBWFQ_FR_SHAPING
R6(config-pmap)#class class-default
R6(config-pmap-c)#shape average 128000
R6(config-pmap-c)#shape adaptive 64000
R6(config-pmap-c)#end
```

Now the important aspect of this strategy is that we would need to create any class-maps identifying any traffic of interests that we would want to provide special treatment to inside the shaping queue that is being created for the class-default class. As an example we will create class maps to identify voice traffic which in our scenario will be marked with a DSCP value of EF:

```
R6(config)#class-map VOICE_TRAFFIC
R6(config-cmap)#match dscp ef
R6(config-cmap)#exit
```

By creating a policy map calling the newly defined class map we can provide strict priority treatment for voice and offer any other traffic treatment using the weighted fair queuing.

```
R6(config)#policy-map PM_CHILD
R6(config-pmap)#class
R6(config-pmap)#class VOICE_TRAFFIC
```

```
R6(config-pmap-c)#priority 64
R6(config-pmap-c)#exit
R6(config-pmap)#class class-default
R6(config-pmap-c)#fair-queue
R6(config-pmap-c)#end
```

With this accomplished we will now “nest” the policy map of child inside the policy-map CBWFQ_FR_SHAPING:

```
R6(config)#policy-map CBWFQ_FR_SHAPING
R6(config-pmap)#class class-default
R6(config-pmap-c)#service-policy PM_CHILD
R6(config-pmap-c)#end
```

Before we go any further, we need to take a look at what we have just done:

```
R6#show policy-map
  Policy Map PM_CHILD
    Class VOICE_TRAFFIC
      priority 64 (kbps)
    Class class-default
      fair-queue

  Policy Map CBWFQ_FR_SHAPING
    Class class-default
      Average Rate Traffic Shaping
      cir 128000 (bps)
      shape adaptive 64000
      service-policy PM_CHILD
```

The policy map named PM_CHILD has been applied inside the class class-default section of the policy map CBWFQ_FR_SHAPING. This means that we are creating a shaping queue for all traffic (by virtue of using the default class) wherever this policy will be applied. Inside that shaping queue, we are going to give packets with a DSCP value of EF priority treatment. Traffic not possessing a DSCP value of EF will be queued up and sent via the weighted fair queuing strategy.

All that is left now is to apply our policy, and herein lies a another significant difference in the way things are done in class-base frame-relay traffic shaping. We are going to turn again to the map-classes to manipulate this process. By creating a map-class and assigning the policy to it:

```
R6(config)#map-class frame-relay CB_FR_SHAPING
R6(config-map-class)#service-policy output CBWFQ_FR_SHAPING
R6(config-map-class)#exit
```

We can strategically apply it to a single DLCI:

```
R6(config)#interface Serial0/2/0
R6(config-if)#frame-relay interface-dlci 100
R6(config-fr-dlci)#class CB_FR_SHAPING
R6(config-fr-dlci)#end
```

We can see the policy is associated with DLCI 100 via the **show policy-map interface** command:

```
R6#show policy-map interface
 Serial0/2/0: DLCI 100 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0

      Service-policy : PM_CHILD

        queue stats for all priority classes:

          queue limit 64 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 0/0

        Class-map: VOICE_TRAFFIC (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


        Class-map: class-default (match-any)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match: any
          Queueing
          queue limit 64 packets
          (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
          (pkts output/bytes output) 0/0
          Fair-queue: per-flow queue limit 16
```

However, it is important to note that we can apply the policy to the entire interface as well:

```
R6(config)#interface Serial0/2/0
R6(config-if)#frame-relay interface-dlci 100
R6(config-fr-dlci)#no class CB_FR_SHAPING
R6(config-fr-dlci)#exit
R6(config-if)#frame-relay class CB_FR_SHAPING
R6(config-if)#end
```

Now we will clearly see that the policy has been applied to both DLCIs on Serial0/2/0 by using the **show policy-map interface** command:

```
R6#sh policy-map interface
 Serial0/2/0: DLCI 100 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0

      Service-policy : PM_CHILD

        queue stats for all priority classes:

          queue limit 64 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 0/0

        Class-map: VOICE_TRAFFIC (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


        Class-map: class-default (match-any)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match: any
          Queueing
          queue limit 64 packets
          (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
          (pkts output/bytes output) 0/0
          Fair-queue: per-flow queue limit 16
 Serial0/2/0: DLCI 200 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
```

```
    lower bound cir 64000,  adapt to fecn 0

  Service-policy : PM_CHILD

    queue stats for all priority classes:

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0

    Class-map: VOICE_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  dscp ef (46)
      Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
      (pkts output/bytes output) 0/0
      Fair-queue: per-flow queue limit 16
```

This offers us scalability and granularity not offered by the more legacy tools we have looked at to date.

The Operation and Troubleshooting of Traffic Shaping

Now that we have reviewed the technologies of this chapter, it is time to analyze their operation at the Cisco command line.

For exploring the concepts behind troubleshooting traffic shaping, we will utilize the following topology:

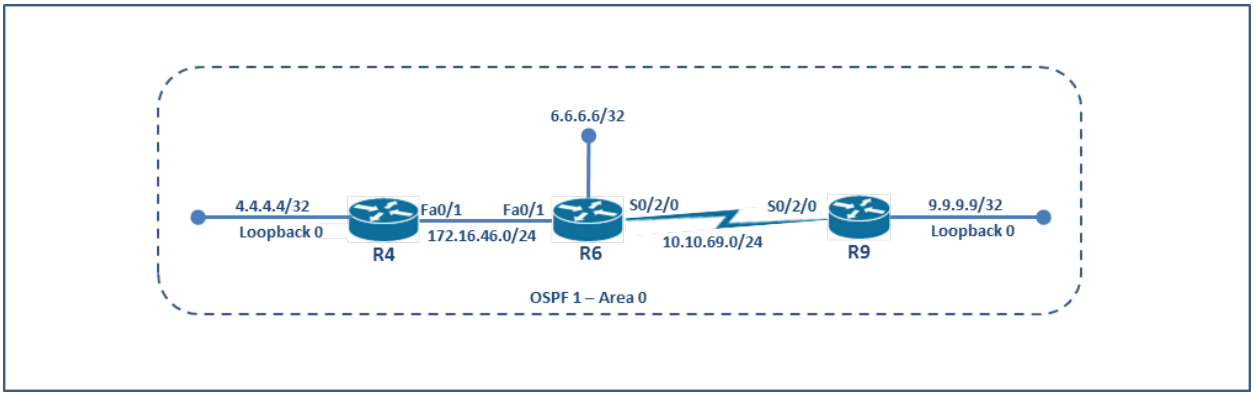

Figure 8-1: A Basic Quality of Service Topology

An IOS router divides a single second into multiple sub-second intervals. In the figure 8-2, 1 second is broken up into 8 periodic intervals equaling 125ms each. Each of these intervals is called a Tc. In traffic shaping, a router can send a burst of traffic equal to the Bc during each of these intervals.

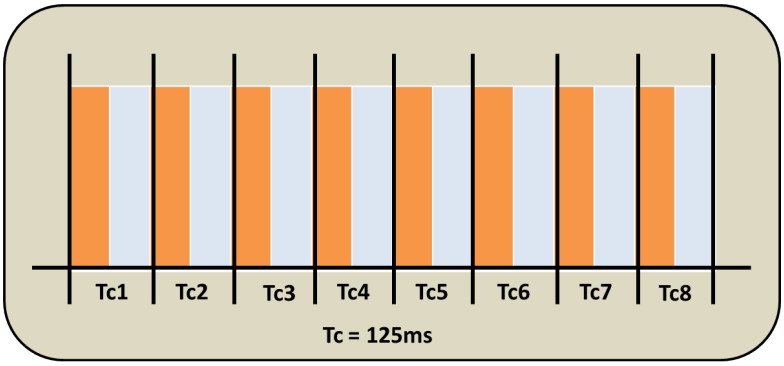

Figure 8-2: 8 period intervals

To better visualize this process imagine a scenario where we are looking to achieve a CIR (contract rate) of 128Kbps. Since we are looking to send 128000 bits in a single second, a router will send 16000 bits (Bc = 128000 * 0.125) during the first interval and stop sending any further traffic until the next interval. This means that if the maximum access link speed where to be 1.544Mbps, it would take the device about 10ms to send 16000 bits. For any remaining time in the first interval, no more traffic will be sent. This is illustrated in **Figure 8-3**.

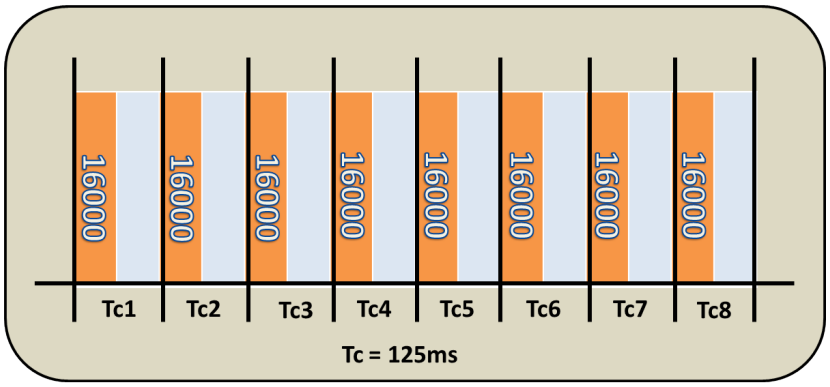

Figure 8-3: 8 period intervals

Traffic Shaping with no Excess Burst

We discussed in the technology review section that traffic shaping involves the concept of a Token Bucket. In the token bucket scenario, each token equates to 1 bit of information that can be sent. The size of the token bucket is defined by the committed burst value (Bc).

In traffic shaping operations there are two primary actions that directly relate to the token bucket and the tokens it contains:

- Token replenishment
- Token Consumption

At the beginning of each Tc interval, the token bucket will be replenished with tokens equal to the value of the Bc. If the token bucket is full or there is not enough room for all the replenished tokens then some or all of the tokens spill out. In a scenario where there is no excess burst (Be) value defined these spilled tokens are discarded and never used. This process is illustrated in **Figure 8-4**.

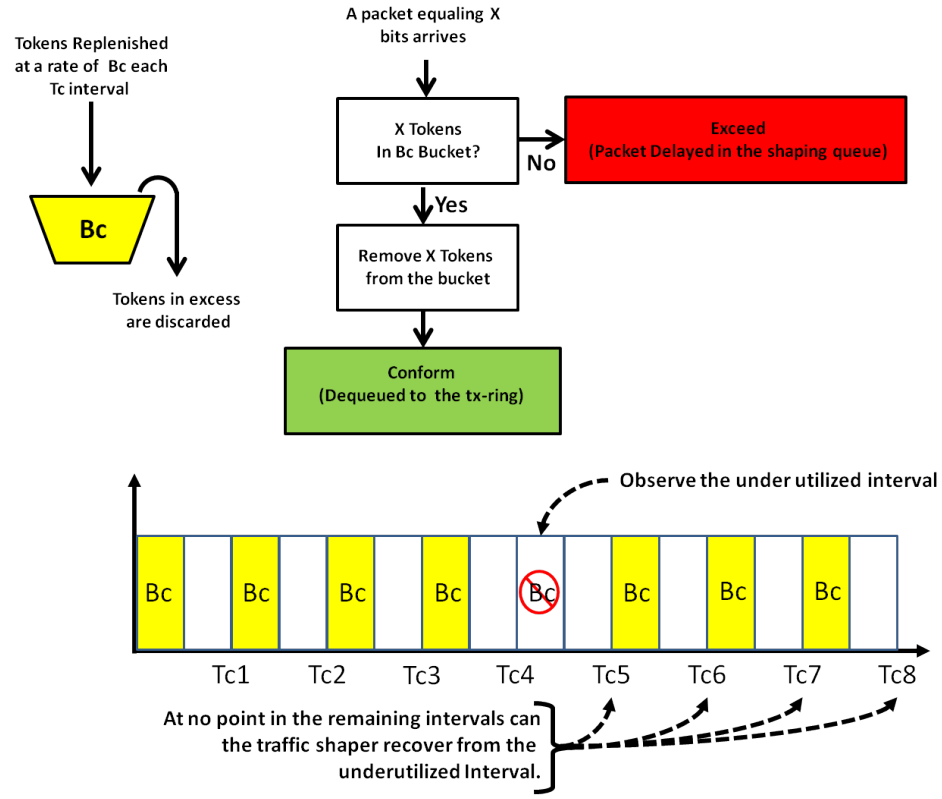

Figure 8-4: Traffic Shaping Without Excess Burst

Each time a packet is sent, the Shaper consumes tokens from the bucket like a toll levied for each packet. If the Shaper tries to send the packet, and there are not enough tokens in the bucket, the Shaper will wait until the next interval when the bucket is replenished.

In Cisco IOS, traffic shaping with no excess burst can be configured using shape average <shaping-rate> command from policy-map configuration mode. The "shaping-rate" could be same as the CIR or slightly higher than CIR but less than access-link rate. This would be deployed on an IOS router like so:

```
access-list 101 permit ip any any dscp cs4
!
class-map CM_SHAPE
match access-group 101
!
policy-map PM_SHAPE
class CM_SHAPE
shape average percent 50
!
interface FastEthernet 0/1
load-interval 30
bandwidth 100
service-policy output PM_SHAPE
```

Cisco recommends only the shaping rate should be specified and thus allowing the shaping algorithm to calculate the best Bc value. We can see the values produced by this process by using the **show policy-map** command:

```
R6#show policy-map interface FastEthernet0/1
 FastEthernet0/1

  Service-policy output: PM_SHAPE

    Class-map: CM_SHAPE (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 101
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 500000, bc 200, be 200
      target shape rate 500000

    Class-map: class-default (match-any)
      12 packets, 1306 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 12/1257
```

Observe that the queue depth/total drops/no-buffer drops values are 0/0/0 respectively. This means that there is no congestion on the link and therefore we are not actively shaping any traffic. This can be tested by creating a stream of traffic from R9 destined to R4's loopback0 interface with a TOS value of 128:

```
R9#ping
Protocol [ip]:
Target IP address: 4.4.4.4
Repeat count [5]: 1000000
Datagram size [100]: 1500
Timeout in seconds [2]: 0
Extended commands [n]: y
Source address or interface:
Type of service [0]: 128
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 1000000, 1500-byte ICMP Echos to 4.4.4.4, timeout is 0 seconds:
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................<output omitted>
```

Now back to R6:

```
R6#show policy-map interface FastEthernet0/1
 FastEthernet0/1

  Service-policy output: PM_SHAPE

    Class-map: CM_SHAPE (match-all)
      1346 packets, 2037844 bytes
      30 second offered rate 125000 bps, drop rate 38000 bps
      Match: access-group 101
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 61/1410/0
      (pkts output/bytes output) 13396/2258776
      shape (average) cir 50000, bc 200, be 200
      target shape rate 50000

    Class-map: class-default (match-any)
      355 packets, 322930 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: any

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 2375/389568
```

We can clearly see shaping taking place.

Traffic Shaping with Excess Burst:

Traffic shaping allows the concept of Excessive Burst (Be) by making the token bucket bigger; now the size of the token bucket is dictated by Bc + Be. Tokens now will be replenished to the bucket at the rate of Bc+Be. Still, at the start of each interval Tc, the Shaper replenishes the bucket with Bc amount of tokens. However now, due to inactivity during some interval, the bucket can accommodate those Bc worth of tokens since its size has been increased. In the next Tc interval, if there are more bits to send, the Shaper can use these Bc+Be tokens to send that amount of bits. In essence, exceed the shaping rate as long as there are extra tokens left in the bucket.

The diagram below shows that since there was no activity during fifth Tc interval, the Shaper used up those extra tokens and sent Bc+Be worth of bits in the sixth interval.

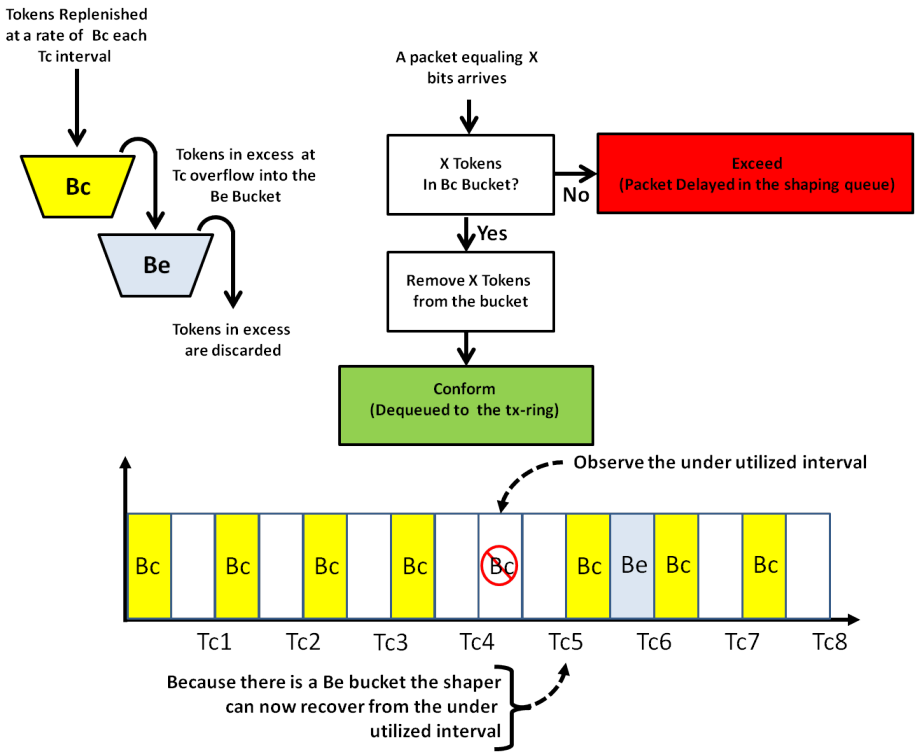

Figure 8-5: Traffic Shaping With Excess Burst

Shaping at Peak Rate

Using shape peak, class based shaping allows Bc+Be bits to be sent every interval even if there has been no period of inactivity. The Shaper replenishes Bc+Be tokens into the bucket.

In Cisco IOS, peak traffic shaping is configured using the **shape peak *<shaping-rate>*** command from the policy-map configuration mode.

The following example shows the configuration of peak traffic shaping. Traffic shaping is applied to outgoing TCP traffic generated with a dscp value of CS4:

```
class-map CM_CS4
match access-group 101
!
policy-map PM_SHAPE_PEAK
class CM_CS4
shape peak percent 50
!
interface FastEthernet 0/1
bandwidth 100
no service-policy output PM_SHAPE
service-policy output PM_SHAPE_PEAK
```

We can observe the effect of this configuration using the show policy-map interface command once more:

```
R6#show policy-map interface FastEthernet 0/1
```

```
FastEthernet0/1

 Service-policy output: PM_SHAPE_PEAK

   Class-map: CM_CS4 (match-all)
     0 packets, 0 bytes
     30 second offered rate 0 bps, drop rate 0 bps
     Match: access-group 101
     Queueing
     queue limit 64 packets
     (queue depth/total drops/no-buffer drops) 0/0/0
     (pkts output/bytes output) 0/0
     shape (peak) cir 50000, bc 200, be 200
     target shape rate 100000

   Class-map: class-default (match-any)
     10 packets, 862 bytes
     30 second offered rate 0 bps, drop rate 0 bps
     Match: any

     queue limit 64 packets
     (queue depth/total drops/no-buffer drops) 0/0/0
     (pkts output/bytes output) 10/1046
```

Observe that we are now setting a target shaped rate of 100,000 which is double the rate in the previous shape average configuration.

Once these concepts are fully understood the actual troubleshooting and verification of traffic shaping becomes an exercise in the judicious and surgical application of verification commands to isolate where deviant behavior or erroneous shaping treatment is taking place. Admittedly, the particular show commands may change based on the type or method of shaping being employed but the output and its interpretation is what sits at the heart of being a good troubleshooter. Whether we are relying on show traffic-shape or the show policy-map interface command, the output will drive the answers we need to the questions what, where and how that surface during troubleshooting and troubleshooting exercises.

Chapter Challenge: Traffic Shaping Trouble Tickets

The following section includes two Trouble Tickets associated with problematic traffic-shaping scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic exiting the Serial0/2/0 interface of R9 is being correctly being shaped but voice traffic is not receiving any preferential treatment as compared to other classes of traffic. You have been instructed to correct this issue. The current deployment of CB-Traffic Shaping must be applied to all virtual circuits assigned to Serial0/2/0 with a single command.

Trouble Ticket #2

Your supervisor has brought to your attention that the current deployment of CB-Traffic shaping on R6 for the interface Serial0/2/0 is incorrect. The policy applied should only be affecting DLCI 200.

Chapter Challenge: Traffic Shaping Trouble Ticket Solutions

The following section includes two Trouble Tickets associated with the problematic congestion avoidance scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic exiting the Serial0/2/0 interface of R9 is being correctly being shaped but voice traffic is not receiving any preferential treatment as compared to other classes of traffic. You have been instructed to correct this issue. The current deployment of CB-Traffic Shaping must be applied to all virtual circuits assigned to Serial0/2/0 with a single command.

Step 1 - Fault Verification:

What is the status of the output on R9 with respect to any applied policy maps?

```
R9#sh policy-map interface
 Serial0/2/0: DLCI 100 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0
 Serial0/2/0: DLCI 200 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0
```

We immediately see that traffic shaping policy is being applied to both DLCI 100 and 200, additionally we observe that the policy is applied under the class-default class. We can see no reference to voice or any other types of traffic in this current deployment. This begs the question, “is there a policy-map in existence on R9 that we can nest into the existing parent policy?”

We kind find the answer to this question by using the **show policy-map** command:

```
R9#show policy-map
  Policy Map PM_CHILD
    Class VOICE_TRAFFIC
      priority 64 (kbps)
    Class class-default
      fair-queue

  Policy Map CBWFQ_FR_SHAPING
    Class class-default
      Average Rate Traffic Shaping
      cir 128000 (bps)
      shape adaptive 64000
```

We can clearly see that a policy map named PM_CHILD does exist but that policy is not being applied to the parent policy of the CBWFQ_FR_SHAPING. We can correct this by using the service-policy command under the class-default class of the parent and reference the child policy.

```
R9#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)#policy-map CBWFQ_FR_SHAPING
R9(config-pmap)#class class-default
R9(config-pmap-c)#service-policy PM_CHILD
R9(config-pmap-c)#end
```

We can see the effects of this configuration by looking again at the **show policy-map interface** command:

```
R9#show policy-map interface
 Serial0/2/0: DLCI 100 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0

      Service-policy : PM_CHILD

        queue stats for all priority classes:

          queue limit 64 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 0/0
```

```
        Class-map: VOICE_TRAFFIC (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


        Class-map: class-default (match-any)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match: any
          Queueing
          queue limit 64 packets
          (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
          (pkts output/bytes output) 0/0
          Fair-queue: per-flow queue limit 16
Serial0/2/0: DLCI 200 -

 Service-policy output: CBWFQ_FR_SHAPING

   Class-map: class-default (match-any)
     0 packets, 0 bytes
     5 minute offered rate 0 bps, drop rate 0 bps
     Match: any
     Queueing
     queue limit 64 packets
     (queue depth/total drops/no-buffer drops) 0/0/0
     (pkts output/bytes output) 0/0
     shape (average) cir 128000, bc 512, be 512
     target shape rate 128000
       lower bound cir 64000,  adapt to fecn 0

     Service-policy : PM_CHILD

       queue stats for all priority classes:

         queue limit 64 packets
         (queue depth/total drops/no-buffer drops) 0/0/0
         (pkts output/bytes output) 0/0

       Class-map: VOICE_TRAFFIC (match-all)
         0 packets, 0 bytes
         5 minute offered rate 0 bps, drop rate 0 bps
         Match:  dscp ef (46)
         Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


       Class-map: class-default (match-any)
         0 packets, 0 bytes
         5 minute offered rate 0 bps, drop rate 0 bps
         Match: any
         Queueing
         queue limit 64 packets
         (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
```

```
          (pkts output/bytes output) 0/0
          Fair-queue: per-flow queue limit 16
```

Now we are both shaping and providing LLQ to voice traffic on both DLCIs. This has successfully remediated trouble ticket 1.

Trouble Ticket #2

Your supervisor has brought to your attention that the current deployment of CB-Traffic shaping on R6 for the interface Serial0/2/0 is incorrect. The policy applied should only be affecting DLCI 200.

Step 1 - Fault Verification:

What is the status of the output on R6 with respect to any applied policy maps?

```
R6#show policy-map interface
 Serial0/2/0: DLCI 100 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0

      Service-policy : PM_CHILD

        queue stats for all priority classes:

          queue limit 64 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 0/0

        Class-map: VOICE_TRAFFIC (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


        Class-map: class-default (match-any)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match: any
          Queueing
          queue limit 64 packets
          (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
```

```
          (pkts output/bytes output) 0/0
          Fair-queue: per-flow queue limit 16
 Serial0/2/0: DLCI 200 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0

      Service-policy : PM_CHILD

        queue stats for all priority classes:

          queue limit 64 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 0/0

        Class-map: VOICE_TRAFFIC (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0


        Class-map: class-default (match-any)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match: any
          Queueing
          queue limit 64 packets
          (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
          (pkts output/bytes output) 0/0
          Fair-queue: per-flow queue limit 16
```

We immediately see that the policy is being applied to all virtual circuits existing on Serial0/2/0, this has to do with the manner used to actually apply the **map-class frame-relay** command. This can be most simply confirmed via the **show interface** command:

```
R6#show run interface Serial0/2/0
Building configuration...

Current configuration : 357 bytes
!
interface Serial0/2/0
 ip address 11.11.69.9 255.255.255.0 secondary
```

```
 ip address 10.10.69.6 255.255.255.0
 encapsulation frame-relay
 no keepalive
 clock rate 2000000
 frame-relay class CB_FR_SHAPING
 frame-relay map ip 11.11.69.9 200 broadcast
 frame-relay map ip 10.10.69.9 100 broadcast
 frame-relay interface-dlci 100
 no frame-relay inverse-arp
end
```

Note that the frame-relay class CB_FR_SHAPING is being applied at the physical interface level, this means that it will affect all PVCs. We were instructed to apply the policy only to DLCI 200. To correct this we must first remove the existing map-class and then apply it under the dlci 200:

```
R6(config)#interface Serial0/2/0
R6(config-if)#no frame-relay class CB_FR_SHAPING
R6(config-if)#frame-relay interface-dlci 200
R6(config-fr-dlci)#class CB_FR_SHAPING
R6(config-fr-dlci)#end
```

Now we can verify if the policy is only being applied to DLCI 200:

```
R6#show policy-map interface
 Serial0/2/0: DLCI 200 -

  Service-policy output: CBWFQ_FR_SHAPING

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      shape (average) cir 128000, bc 512, be 512
      target shape rate 128000
        lower bound cir 64000,  adapt to fecn 0

      Service-policy : PM_CHILD

        queue stats for all priority classes:

          queue limit 64 packets
          (queue depth/total drops/no-buffer drops) 0/0/0
          (pkts output/bytes output) 0/0

        Class-map: VOICE_TRAFFIC (match-all)
          0 packets, 0 bytes
          5 minute offered rate 0 bps, drop rate 0 bps
          Match:  dscp ef (46)
          Priority: 64 kbps, burst bytes 1600, b/w exceed drops: 0
```

```
Class-map: class-default (match-any)
  0 packets, 0 bytes
  5 minute offered rate 0 bps, drop rate 0 bps
  Match: any
  Queueing
  queue limit 64 packets
  (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
  (pkts output/bytes output) 0/0
  Fair-queue: per-flow queue limit 16
```

Now the policy is applied in accordance with the trouble ticket. This verifying that this issue has been correctly remediated.

Chapter 9: Traffic Policing

Traffic policing is very similar to traffic shaping. Except with traffic policing, there is no buffering of excess packets. The packets that are exceeding a defined rate will be dropped, or perhaps sent, but with a new Quality of Service marking that penalizes the packets. This chapter explores this important QoS tool in detail.

Traffic Policing Technology Review

Before we dive into the operation of traffic policing at the Cisco command line, we need to review the technology for this section.

Note: This chapter deals with traffic policing tools for Cisco routers.

Traffic Policing

Traffic policing is a tool, much like traffic shaping, used to enforce a traffic contract. More often-than-not this simply means to ensure that traffic abides by an established CIR. A policer relies on an algorithm that will measure the aggregate byte-rate associated with packets. Based on the outcome of the configured parameters the policer will either transmit, drop, or remark and transmit a packet with a new IP precedence or DSCP marking.

Just like class based traffic shaping, class based policing employs the concept of token buckets; however there a number of variations associated with the concept as it compares to its traffic shaping counterpart. At its most basic level class based policing can be configured to use three primary categories in how it sees a packet in relation to the configured traffic contract.

- **Conforming-** The packet is inside the contract
- **Exceeding-** The packet is outside the contract but within the excess burst capability
- **Violating-** The packet is outside the contract

There are three main classifications of class based policing and each type and method has its own operational processes and interpretation of how to manage traffic. It is important to note that not all modes of CB policers use all three of the categories outline above.

CB Policing: Single-Rate Two-Color Policer

In policing token buckets support to processes:

- **Token replenishment** -- In CB Policing, each token equals the ability to send one byte (instead of a bit in CB Shaping). CB Policing replenished tokens in the bucket in response to a packet arriving at the policing function. Every time a packet is policed, CB policing puts some tokens back into the bucket. The number of tokens put in the bucket is calculated based on the formula:

  (Current_packet_arrival_rate - Previous_packet_arrival_rate) * (Police_rate) / 8

- **Conforms or not** -- CB Policing compares the number of bytes in the packet to the number of tokens in the token bucket.

  If the number, of bytes in the packet, is less than or equal to the number of tokens in the token bucket, the packet is said to ***conform*** to the contract. CB policing removes tokens from the token bucket equal to the number of bytes in the packet and then performs the action that conforms to the contract.

If the number of bytes in the packet is greater than the number of tokens in the token bucket, the packet ***exceeds*** the contract. In this scenario CB policing does not remove the tokens from the bucket and performs the action on packets that exceed the contract (drop or remark to different IP Precedence or IP DSCP value)

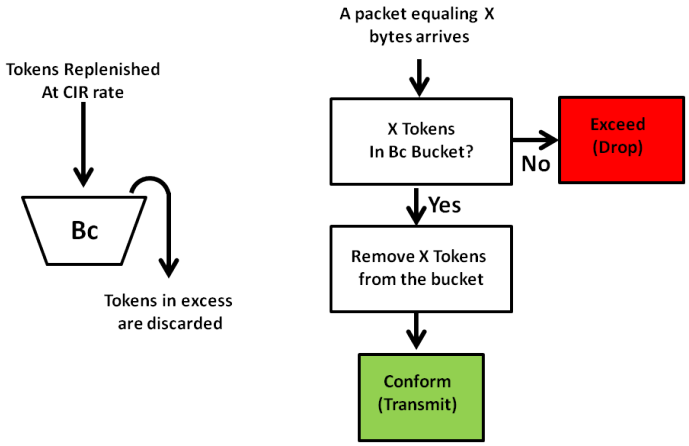

Figure 9-1: Single Bucket, Single Rate Algorithm flowchart

CB Policing: Single-Rate Three-Color Policer

In this policing configuration we have a dual token bucket concept that supports the idea of a Bc (committed burst) and a Be (excess burst). In dual token bucket paradigm (Bc and Be buckets), CB policing characterizes packet into three groups

- **Conform**
- **Exceed**
- **Violate**

CB policing continues to replenish the Bc bucket when a packet arrives. However, any tokens that overflow are captured in the Be bucket. If the Be bucket fills then the tokens are discarded. The number of tokens replenished in the Bc bucket is calculated using the same formula as the single bucket, single rate policer.

The dual token bucket now follows the following algorithm

- If the number of bytes in the packet is less than or equal to the number of tokens in the Bc bucket, the packet ***conforms***. CB policing removes tokens from Bc bucket equal to the number of bytes in the packet and then performs the action that conform to the contract.
- If the packet does not conform, and if the number of bytes in the packet is less than or equal to number of tokens in the Be bucket, the packet ***exceeds***. CB policing removes tokens from Be

bucket equal to the number of bytes in the packet and then performs the action that exceed the contract.

- If the packet neither conforms nor exceeds, the packet ***violates***. CB policing does not remove tokens from any bucket, and perform the action that violates the contract.

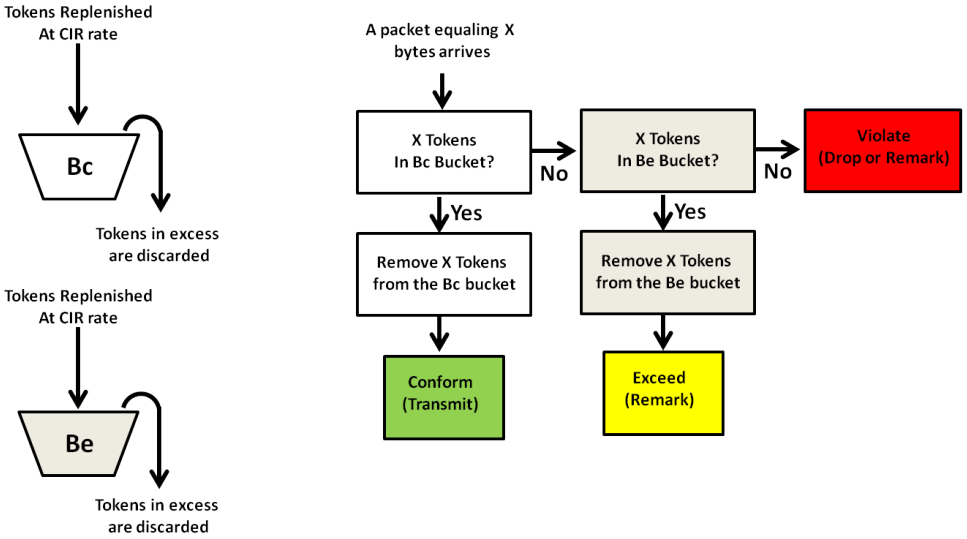

Figure 9-2: Dual Bucket, Single Rate Algorithm flowchart

CB Policing: Dual-Rate Three-Color Policer

Dual token bucket with dual rate has two sustained rates, CIR (Committed Information Rate) and PIR (Peak Information Rate). The CIR conforms to the traffic contract while PIR exceeds CIR. CB Policing replenishes tokens into both, CIR and PIR, buckets when a packet arrives that needs to be policed. The PIR bucket is replenished with tokens directly rather than collecting spilled tokens.

The spending of token algorithm is similar as above but with little difference-

- If the number of bytes in the packet is less than or equal to the tokens in the CIR bucket, the packet ***conforms***. CB policing removes tokens from CIR bucket equal to the number of bytes in the packet, and performs the conform action. ***However, it will also remove the same amount of tokens from the PIR bucket.***

- If the packet does not conform, and if the number of bytes in the packet is less than or equal to number of tokens in the PIR bucket, the packet ***exceeds***. CB policing removes tokens from PIR bucket equal to the number of bytes in the packet and then performs the action that exceed the contract.

- If the packet neither conforms nor exceeds, the packet ***violates***. CB policing does not remove tokens from any bucket, and perform the action that violates the contract.

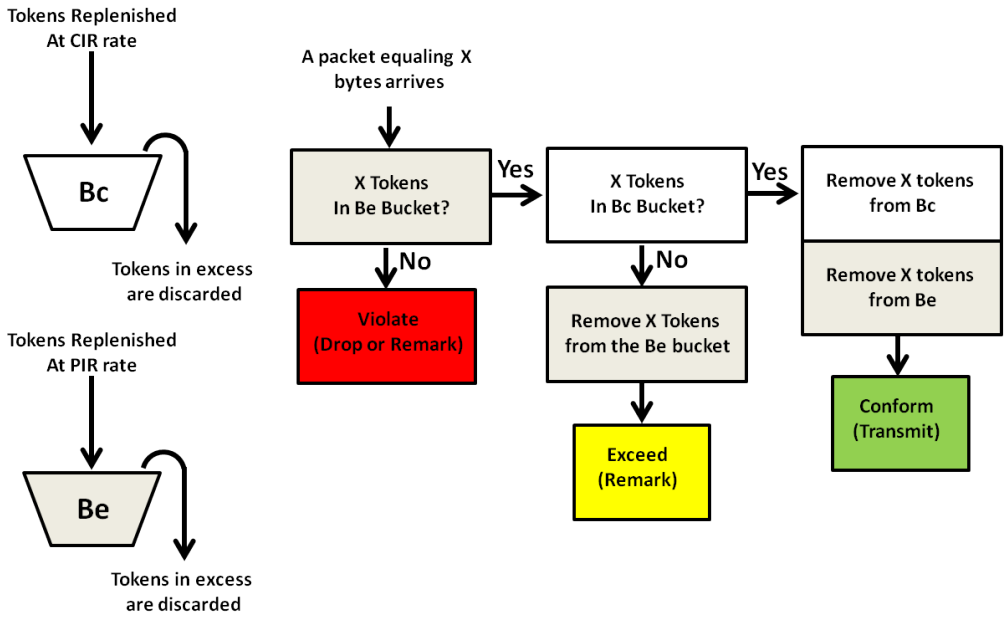

Figure 9-3: Dual Bucket, Dual Rate Algorithm flowchart

The Operation and Troubleshooting of Traffic Shaping

Now that we have reviewed the technologies of this chapter, it is time to analyze their operation at the Cisco command line.

For exploring the concepts behind troubleshooting traffic shaping, we will utilize the following topology:

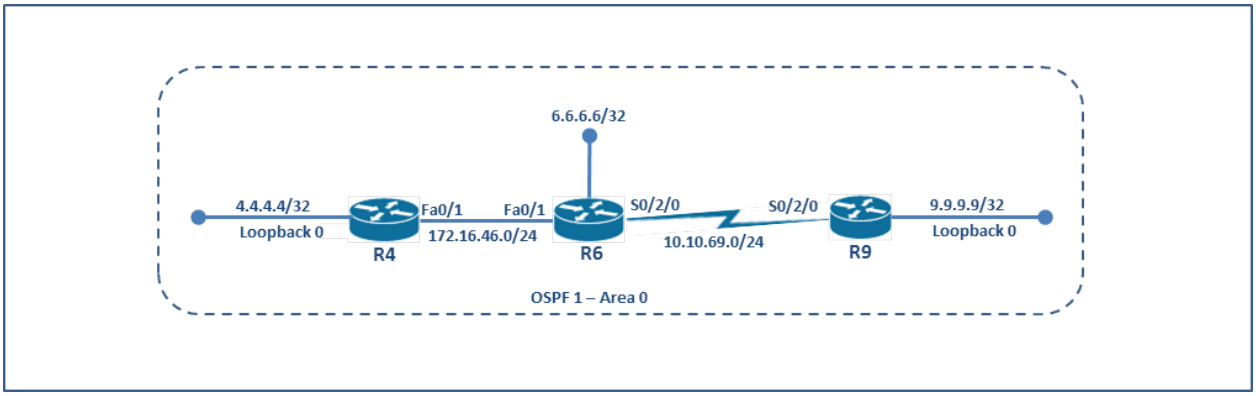

Figure 8-1: A Basic Quality of Service Topology

CB Policing: Single-Rate Two-Color Policer Operation and Troubleshooting

Generally, when it comes to policing this configuration is considered the most basic configuration possible. We will be discussing policing in this section purely from the point of view of CB-Policing and within that application, there are two typical variations of deployment. These two variants only differ in how they treat traffic that is considered to have violated the policing policy. The first method will employ a drop methodology:

```
R6(config)#
%SYS-5-CONFIG_I: Configured from console by console
R6(config)#
R6(config)#class-map POLICE_THIS_TRAFFIC
R6(config-cmap)#match protocol HTTP
R6(config-cmap)#policy-map PM_SINGLE_RATE_2COLOR
R6(config-pmap)#class POLICE_THIS_TRAFFIC
R6(config-pmap-c)#police cir 256000
R6(config-pmap-c-police)#conform-action transmit
R6(config-pmap-c-police)#exceed-action drop
R6(config-pmap-c-police)#exit
R6(config-pmap-c)#exit
R6(config-pmap)#exit
```

Now we apply the policy to the interface:

```
R6(config)#interface Serial0/2/0
R6(config-if)#service-policy input PM_SINGLE_RATE_2COLOR
R6(config-if)#exit
```

This configuration can best be verified via the show policy-map interface command:

```
R6#show policy-map interface
 Serial0/2/0
```

```
  Service-policy input: PM_SINGLE_RATE_2COLOR

    Class-map: POLICE_THIS_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          drop
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Sometimes actually dropping traffic that is violating the policy is not the best method to mitigate problems in the network. This brings us to our second method of how to handle violating traffic. CB-base policing supports the capability to “remark” or “mark down” traffic so that it still has a chance to make it through the end-to-end topology, but remarking the traffic we ensure that it will not receive the same preferential treatment of packets in its class that have not violated the policing policy. This in affect penalizes traffic that misbehaves while protecting traffic that does not. The following example is shows how to configure policing for a Single-Rate, 2-color policer with DSCP markdown:

```
R6#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#policy-map PM_SINGLE_RATE_2COLOR_WITH_MARKDOWN
R6(config-pmap)#class POLICE_THIS_TRAFFIC
R6(config-pmap-c)#police cir 256000
R6(config-pmap-c-police)#conform-action transmit
R6(config-pmap-c-police)#exceed-action set-dscp-transmit AF12
R6(config-pmap-c-police)#exit
```

We will first have to remove the previous configuration:

```
R6(config-pmap-c)#interface Serial0/2/0
R6(config-if)#no service-policy input PM_SINGLE_RATE_2COLOR
R6(config-if)#service-policy input PM_SINGLE_RATE_2COLOR_WITH_MARKDOWN
R6(config-if)#exit
```

Now we can verify the configuration with the **show policy-map interface** command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: PM_SINGLE_RATE_2COLOR_WITH_MARKDOWN

    Class-map: POLICE_THIS_TRAFFIC (match-all)
```

```
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Troubleshooting Single-Rate Two-Color Policers

We have to always keep in mind that traffic shaping will use a token bucket to strictly limit the transmission of packets at an administratively defined rate. Streams of information that exceed this rate will see their traffic being dropped, or their ToS values remarked to a marking that will be more likely dropped further into the network. Obviously, this process could be mismanaged on any given router or any given interface. The show commands demonstrated in this section offer the fastest and most useful tools that we have available to us to isolate these problems. The clear and precise method that we recommend is to verify the configuration on a hop-by-hop basis ensuring uniformity.

CB Policing: Single-Rate Two-Color Policer Operation and Troubleshooting

When a three-color policer is employed, we are leveraging the full capability of policing to offer traffic the best possible chance to successfully navigate the network while utilizing one rate. As we saw in the ***Technology Review*** section of this chapter, the three colors of traffic we are referring to are conforming, exceeding and violating traffic. This categorization of color is assigned to at packet's flow based on which token bucket has enough tokens to allow the packet to be passed. The configuration of this method of policing is as follows and again we are presented with the same two basic actions that we had in the single-rate two-color policer we discusses previously. However, now we can make the choice to drop or remark for three categories of traffic rather than just two. This process is best illustrated as follows:

```
R6#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#policy-map PM_SINGLE_RATE_3COLOR
R6(config-pmap)#class POLICE_THIS_TRAFF
%SYS-5-CONFIG_I: Configured from console by console
R6(config-pmap)#class POLICE_THIS_TRAFFIC
R6(config-pmap-c)#police cir 256000
R6(config-pmap-c-police)#conform-action transmit
R6(config-pmap-c-police)#exceed-action set-dscp-transmit AF12
R6(config-pmap-c-police)#violate-action drop
R6(config-pmap-c-police)#exit
```

We will remove the previous configuration and apply this new policy to Serial0/2/0:

```
R6(config-pmap-c)#interface Serial0/2/0
R6(config-if)#no service-policy input PM_SINGLE_RATE_2COLOR_WITH_MARKDOWN
R6(config-if)#service-policy input PM_SINGLE_RATE_3COLOR
R6(config-if)#exit
```

The **show policy-map interface** command demonstrates the effect of this configuration:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: PM_SINGLE_RATE_3COLOR

    Class-map: POLICE_THIS_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offe red rate 0 bps, drop rate 0 bps
      Match: any
```

Observe the three categories of treatment in this configuration includes transmit, set-dscp-transmit, and lastly drop for traffic that violates the policy. Refer to the flow charts in the technology review section if you are having issues remembering what situations will result in what categories.

Troubleshooting Single-Rate Three-Color Policers

We have to always keep in that even in this scenario were we have “three colors” to be aware of that the mechanism will still use one token bucket to strictly limit the transmission of packets at the administratively defined rate. Streams of information that exceed this rate will see their traffic being dropped, or their ToS values remarked to a marking that will be more likely dropped further into the network, however because there are now three criteria that we can select from the algorithm is even more granular that the previous mechanism. Obviously, no that there are more options, this process is even more likely to suffer from mismanage or deployment on any given router or any given interface. The show commands demonstrated in this section offer the fastest and most useful tools that we have available to us to isolate these problems. The clear and precise method that we recommend is to verify

the configuration on a hop-by-hop basis ensuring uniformity paying careful attention to the order and actions taken at each level of treatment in the policing process.

CB Policing: Dual-Rate Three-Color Policer Operation and Troubleshooting

The dual rate policing operates completely different than either of the other version of policing we have mentioned in this chapter. The process follows the same mechanism of have multiple colors of traffic or categories that follow the same patterns of conforming, exceeding and violating, but now are introducing a process by which the token buckets (2 buckets) are being filled at different rates. This can be observed closely in the ***Technology Review*** section, but in summary the bucket filling the role of Bc is being filled at a rate equal to the CIR, where the token bucket for the Be value is filled at the rate equivalent to the PIR. This process by design allows for sustainable excess burst of traffic that with the forced reliance on credits accumulation. This process does however result in an absolute peak limit with regard to traffic flow, because any traffic that would look to exceed the PIR will be immediately marked as violating and subjected to its prescribed actions.

The application of this type of policing is characteristically different than what we have seen before and would follow a model like follows for most scenarios and deployments:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#policy-map DUAL_RATE_THREE_COLOR
R6(config-pmap)#class class-default
R6(config-pmap-c)#police cir 8000 bc 1000 pir 10000 be 2000
R6(config-pmap-c-police)#conform-action set-dscp-transmit af31
R6(config-pmap-c-police)#exceed-action  set-dscp-transmit af32
R6(config-pmap-c-police)#violate-action set-dscp-transmit af32
```

Just as before the other service-policy must be removed before this one can be applied:

```
R6(config-pmap-c-police)#interface Serial0/2/0
R6(config-if)#no service-policy input PM_SINGLE_RATE_3COLOR
R6(config-if)#service-policy input DUAL_RATE_THREE_COLOR
R6(config-if)#exit
```

This configuration can be verified using the **show policy-map interface** command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: DUAL_RATE_THREE_COLOR

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      police:
          cir 8000 bps, bc 1000 bytes
          pir 10000 bps, be 2000 bytes
        conformed 0 packets, 0 bytes; actions:
```

```
    set-dscp-transmit af31
  exceeded 0 packets, 0 bytes; actions:
    set-dscp-transmit af32
  violated 0 packets, 0 bytes; actions:
    set-dscp-transmit af32
  conformed 0 bps, exceed 0 bps, violate 0 bps
```

The output demonstrates the two rates (cir,pir) and the three colors (conform, exceed and violate).

Troubleshooting Dual-Rate Three-Color Policers

We have to always keep in that in this scenario we have “three colors” to be aware of as well as two rates of token replenishment. The dual rates work together as a single mechanism that allows the process to strictly limit the transmission of packets at the administratively defined rate. Streams of information that exceed this rate will see their traffic being dropped, or their ToS values remarked to a marking that will be more likely dropped further into the network, however because there are now three criteria and two replenishment rates the algorithm has the highest level of granularity available in policing. Obviously, now that there are more options and multiple rates, this process is even more likely to suffer from mismanage or deployment on any given router or any given interface. The show commands demonstrated in this section offer the fastest and most useful tools that we have available to us to isolate these problems. The clear and precise method that we recommend is to verify the configuration on a hop-by-hop basis ensuring uniformity paying careful attention to the order and actions taken at each level of treatment in the policing process, as well as the defined rates utilized in each configuration.

Chapter Challenge: Traffic Policing Trouble Tickets

The following section includes two Trouble Tickets associated with problematic traffic-policing scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic entering the Serial0/2/0 interface of R9 is not receiving the correct CB-policing treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on matches the attached exhibit. There are multiple issues associated with this ticket.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: PM_SINGLE_RATE_3COLOR

    Class-map: POLICE_THIS_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps

    Class-map: class-default (match-any)
```

Trouble Ticket #2

Your supervisor has brought to your attention that traffic entering the Serial0/2/0 interface of R9 is not receiving the correct CB-policing treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on matches the attached exhibit. There are multiple issues associated with this ticket.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: DUAL_RATE_THREE_COLOR

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      police:
          cir 8000 bps, bc 1000 bytes
          pir 10000 bps, be 2000 bytes
        conformed 0 packets, 0 bytes; actions:
          set-dscp-transmit af31
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af32
```

```
violated 0 packets, 0 bytes; actions:
  set-dscp-transmit af32
conformed 0 bps, exceed 0 bps, violate 0 bps
```

Chapter Challenge: Traffic Policing Trouble Ticket Solutions

The following section includes the Trouble Ticket Solutions.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic entering the Serial0/2/0 interface of R9 is not receiving the correct CB-policing treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on matches the attached exhibit. There are multiple issues associated with this ticket.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: PM_SINGLE_RATE_3COLOR

    Class-map: POLICE_THIS_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps

    Class-map: class-default (match-any)
```

Step 1 - Fault Verification:

What is the status of the output on R9 with respect to any applied policing policy?

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: PM_SINGLE_RATE_3COLOR

    Class-map: POLICE_THIS_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          drop
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps
```

```
    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

We immediately see that traffic policing policy has been applied outbound rather than inbound. Additionally, further inspection of the policy reveals that conforming traffic is being dropped. This does does not match the provided exhibit. However this can be corrected by editing the policy-map, and reapplying it to Serial0/2/0 in the inbound direction.

```
R9#config t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)#policy-map PM_SINGLE_RATE_3COLOR
R9(config-pmap)#class POLICE_THIS_TRAFFIC
R9(config-pmap-c)#police cir 256000
R9(config-pmap-c-police)#no conform-action drop
R9(config-pmap-c-police)#conform-action transmit
```

Now we will remove the previous service-policy and apply the corrected one in the proper direction:

```
R9(config-pmap)#interface Serial0/2/0
R9(config-if)#no service-policy output PM_SINGLE_RATE_3COLOR
R9(config-if)#service-policy input PM_SINGLE_RATE_3COLOR
R9(config-if)#end
```

We can verify that the output now matches the exhibit via the **show policy-interface** command:

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy input: PM_SINGLE_RATE_3COLOR

    Class-map: POLICE_THIS_TRAFFIC (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol http
      police:
          cir 256000 bps, bc 8000 bytes, be 1500 bytes
        conformed 0 packets, 0 bytes; actions:
          transmit
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        violated 0 packets, 0 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps, violate 0 bps

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

This has successfully remediated trouble ticket 1.

Trouble Ticket #2

Your supervisor has brought to your attention that traffic entering the Serial0/2/0 interface of R6 is not receiving the correct CB-policing treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on matches the attached exhibit. There are multiple issues associated with this ticket.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: DUAL_RATE_THREE_COLOR

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      police:
          cir 8000 bps, bc 1000 bytes
          pir 10000 bps, be 2000 bytes
        conformed 0 packets, 0 bytes; actions:
          set-dscp-transmit af31
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af32
        violated 0 packets, 0 bytes; actions:
          set-dscp-transmit af32
    conformed 0 bps, exceed 0 bps, violate 0 bps.
```

Step 1 - Fault Verification:

What is the status of the output on R6 with respect to any applied policing policy?

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy input: DUAL_RATE_THREE_COLOR

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      police:
          cir 8000 bps, bc 1000 bytes
          pir 10000 bps, be 2000 bytes
        conformed 0 packets, 0 bytes; actions:
          set-dscp-transmit af11
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af13
        violated 0 packets, 0 bytes; actions:
          set-dscp-transmit af12
        conformed 0 bps, exceed 0 bps, violate 0 bps
```

We immediately see that the policy is not specifying the correct DSCP remarking this can be corrected by editing the policy with the correct values:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#policy-map DUAL_RATE_THREE_COLOR
R6(config-pmap)#class class-default
R6(config-pmap-c)#police cir 8000 bc 1000 pir 10000 be 2000
R6(config-pmap-c-police)#conform-action set-dscp-transmit af31
R6(config-pmap-c-police)#exceed-action set-dscp-transmit af32
R6(config-pmap-c-police)#violate-action set-dscp-transmit af32
R6(config-pmap-c-police)#end
```

This can best be verified via the **show policy-map interface** command:

```
R6#sh policy-map interface
 Serial0/2/0

  Service-policy input: DUAL_RATE_THREE_COLOR

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      police:
          cir 8000 bps, bc 1000 bytes
          pir 10000 bps, be 2000 bytes
        conformed 0 packets, 0 bytes; actions:
          set-dscp-transmit af31
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit af32
        violated 0 packets, 0 bytes; actions:
          set-dscp-transmit af32
        conformed 0 bps, exceed 0 bps, violate 0 bp
```

This verifies that this issue has been correctly remediated.

Chapter 10: Compression

One easy way to save on bandwidth would be to just make the headers of the traffic, or the traffic itself, or both…smaller. This is the job of the various forms and styles of compression available on Cisco routers and switches. This chapter details these various options.

Header Compression Technology Review

Before we dive into the operation of header compression at the Cisco command line, we need to review the technology for this section.

Note: This chapter deals with header compression tools used by the Cisco IOS.

Two Types of Header Compression

There are two types of header compression: TCP header compression and RTP header compression. RTP and TCP IP header compression are mechanisms that compress the IP header in a data packet before the packet is transmitted onto the physical link. Header compression therefore lowers network overhead and thus speeds up the transmission of RTP and TCP packets.

It should be noted that modern Cisco IOS enabled devices provide a related feature called Express RTP/TCP Header Compression. Before this feature was available, if compression of TCP or RTP headers was enabled, compression was performed in the process-switching path. Compression performed in this fashion meant that packets traversing interfaces that had TCP or RTP header compression enabled were queued and passed up the process to be forwarded. This procedure would actually slow down the transmission rate of a packet, and therefore some cisco environments merely decided to fast-switch uncompressed TCP and RTP packets rather than endure the added delay associated with header compression.

However, with this enhanced functionality added to the compression process, now when TCP or RTP header compression is enabled, it occurs by default in either fast-switching or the Cisco Express Forwarding (CEF-switched) processes, depending on which switching method is enabled on the interface. Another enhancement to the process made by CISCO was the decision to increase the number of TCP and RTP header compression connections that are simultaneously manageable by a given device. It must be pointed out, however, that if neither fast-switching nor CEF-switching is enabled, and TCP or RTP header compression is enabled, the process will occur using process-switching and as such the process will still insert delay into the data flow.

The Express RTP and TCP Header Compression feature has the following benefits:

- It reduces network overhead.
- It speeds up transmission of TCP and RTP packets. The faster speed provides a greater benefit on slower links than faster links.

Throughout this chapter, we will be discussing class-based header compression.

The Operation and Troubleshooting of Class-based Header Compression

Now that we have reviewed the technologies of this chapter, it is time to analyze their operation at the Cisco command line.

For exploring the concepts behind troubleshooting header compression, we will utilize the following topology:

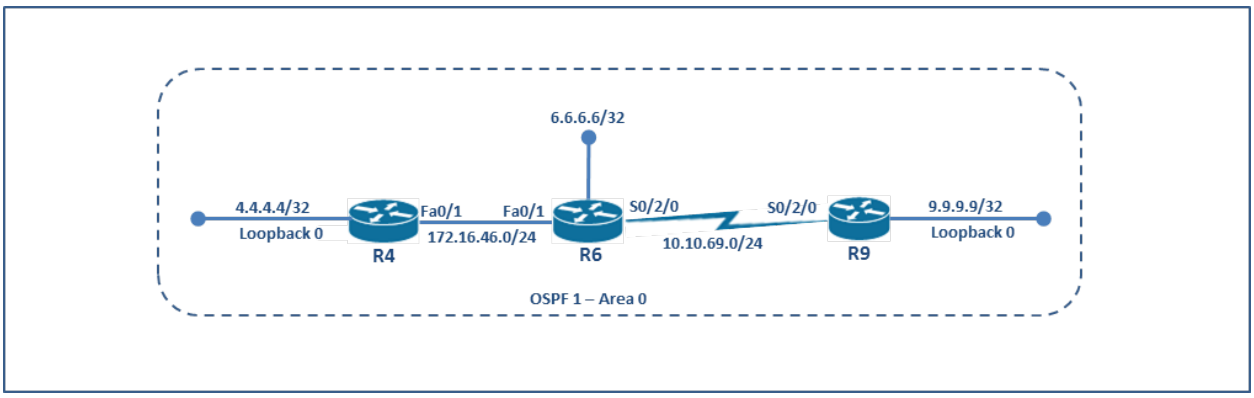

Figure 10-1: A Basic Quality of Service Topology

Header Compress is about Link Efficiency

We need to look at some examples of how much compression (or bandwidth savings) we can achieve using header compression. The IP version 4 header is 20 bytes and when carrying UDP (8 bytes) and RTP (12 bytes, at least), the packet header becomes 40 bytes in length. A header compression scheme usually compresses such headers to 2 – 4 bytes. On an average, considering a few uncompressed packets and a few relatively large packets, we can see more than an 80% savings when we employ compression. In environments where the network is dealing with packetized voice the actual payload size is relatively fixed and normally can be expected to be in the range of 20 – 60 bytes, this header size represents a huge overhead in this situation. The use of header compression in such cases results in significant bandwidth savings. This benefit of this process is clearly visible in **Figure 10-2**.

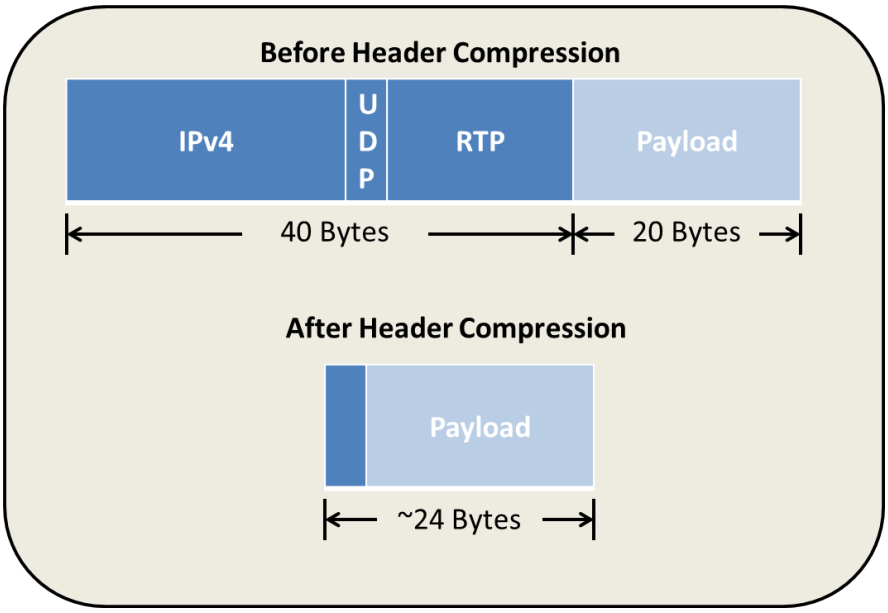

Figure 10-2: Benefits of Compression

How to Configure TCP and RTP Header Compression

Again we will follow the MQC three step process by first identifying what specific class we will be configuring header compress for. This is accomplished with a class-map.

```
R6(config)#class-map TELNET
R6(config-cmap)#match pro
R6(config-cmap)#match protocol telnet
R6(config-cmap)#exit
```

Now we will specify the class we just established in a policy-map and apply the compression strategy:

```
R6(config)#policy-map COMPRESSION
R6(config-pmap)#class TELNET
R6(config-pmap-c)#compression header ip tcp
R6(config-pmap-c)#exit
```

Next, we apply the policy to our interface:

```
R6(config)#interface Serial0/2/0
R6(config-if)#service-policy output COMPRESSION
R6(config-if)#end
```

To verify the configuration we will use the **show policy-map interface** command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
```

```
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Observe that we are using tcp header compression, but what if we want to run both TCP and RTP header compression at the same time. We would need to edit our policy to accomplish this in the following fashion.

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#policy-map COMPRESSION
R6(config-pmap)#class TELNET
R6(config-pmap-c)#no compression header ip tcp
R6(config-pmap-c)#compression header ip
R6(config-pmap-c)#end
```

We can see the immediate effect of this configuration using the **show policy-map interface** command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip
          UDP/RTP (compression on, Cisco, RTP)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps

          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

We can clearly see that we are running both UDP/RTP and TCP header compression inside this policy now.

Troubleshooting TCP and RTP Header Compression

The commands in the previous section will allow you to verify whether header compression is configured and functioning correctly. If, after using the **show** commands listed above, you find that the configuration is not correct or the feature is not functioning as expected, you may want to try these steps:

If the configuration is not the one you intended, complete the following procedures:

- Use the **show running-config** command and analyze the output of the command.
- If the policy map does not appear in the output of the **show running-config** command, enable the **logging console** command.
- Attach the policy map to the interface again.

If the packets are not being matched correctly (for example, the packet counters are not incrementing correctly), complete the following procedures:

- Run the **show policy-map** command and analyze the output of the command.
- Run the **show running-config** command and analyze the output of the command.
- Use the **show policy-map interface** command and analyze the output of the command. Look for the following findings:
    - If a policy map applies queuing, and the packets are matching the correct class, but you see unexpected results, compare the number of the packets in the queue with the number of the packets matched.
    - If the interface is congested, and only a small number of the packets are being matched, check the tuning of the tx-ring, and evaluate whether the queuing is happening on the tx ring. To do this, use the **show controllers** command, and look at the value of the tx count in the output of the command.

Chapter Challenge: Header Compression Trouble Tickets

The following section includes two Trouble Tickets associated with problematic header compression scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic exiting the Serial0/2/0 interface of R6 is not receiving the correct header compression treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on R6 matches the attached exhibit.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Trouble Ticket #2

Your supervisor has brought to your attention that traffic entering the Serial0/2/0 interface of R9 is not receiving the correct header compression treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on matches the attached exhibit.

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
```

```
                  0 bytes saved, 0 bytes sent
                  rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Chapter Challenge: Header Compression Trouble Tickets

The following section includes two Trouble Tickets associated with the problematic header compression scenarios.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic exiting the Serial0/2/0 interface of R6 is not receiving the correct header compression treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on R6 matches the attached exhibit.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Step 1 - Fault Verification:

What is the status of the output on R6 with respect to any applied compression policy?

```
R6#sh policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip rtp
          UDP/RTP (compression on, Cisco, RTP)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps
```

```
    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
```

We immediately see that R6 is using RTP header compression rather than TCP header compression. This can be best corrected by editing the policy-map that is being applied:

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#policy-map COMPRESSION
R6(config-pmap)#class TELNET
R6(config-pmap-c)#no compression header ip rtp
R6(config-pmap-c)#compression header ip tcp
R6(config-pmap-c)#end
```

We can verify that the output now matches the exhibit via the **show policy-interface** command:

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

This has successfully remediated trouble ticket 1.

Trouble Ticket #2

Your supervisor has brought to your attention that traffic entering the Serial0/2/0 interface of R9 is not receiving the correct header compression treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on matches the attached exhibit.

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION
```

```
    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Step 1 - Fault Verification:

What is the status of the output on R9 with respect to any applied compression policy?

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip
          UDP/RTP (compression on, Cisco, RTP)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps

          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

We immediately see that R9 is performing both TCP and RTP header compression and it should only be performing tcp compression per the exhibit. This can be corrected by editing the policy with the correct values:

```
R9#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)#policy-map COMPRESSION
R9(config-pmap)#class TELNET
R9(config-pmap-c)#no compression header ip
R9(config-pmap-c)#compression header ip tcp
R9(config-pmap-c)#end
```

This can best be verified via the **show policy-map interface** command:

```
R9#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps

    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

This verifies that this issue has been correctly remediated.

Chapter 11: Link Fragmentation and Interleaving (LFI)

Think about this scenario your hardware queue and software queues are empty. At that moment, a very large packet arrives. This packet is marked with a very low priority marking, but since the queues are empty, the packet is moved through the software queue immediately, and placed in the software queue. Now, sure enough, as bad luck would have it, a very important packet arrives. This packet is a small Voice over IP (VoIP) packet. The software queue immediately rushes this packet into the hardware queue, but alas, the packet is stuck behind the large frame that just entered that queue. Link fragmentation and interleaving was designed to handle this issue. Enjoy this chapter, which reveals the details of this link conditioning mechanism.

Link Fragmentation and Interleaving Technology Review

Before we dive into the operation of LFI at the Cisco command line, we need to review the technology for this section.

Note: This chapter deals with LFI tools used by the Cisco IOS.

Three "Flavors" of LFI

There are three types of Link Fragmentation and Interleaving that we will need to address in this chapter:

- **Multilink PPP with interleaving** – A Cisco IOS QoS tool that prevents small, delay-sensitive packets from having to wait on longer, delay-insensitive packets to be completely serialized out an interface. To do so, LFI tools fragment larger packets, and then send the delay-sensitive packet after just a portion of the original, longer packet. The key elements include fragmentation, the ability to interleave parts of one packet between fragments and the fact that it requires PPP.

- **FRF.12** – FRF.12 is an Implementation Agreement defined to help support voice and other real-time (delay-sensitive) data on lower-speed links. It accommodates the variation of frame sizes in a manner that allows the mixture of real-time and non-real-time data.

- **FRF.11 Annex C LFI** – This agreement specifies how to transport voice application data over a Frame Relay network. Its focus is to extend the application support of frame relay to include the transport of digital voice payloads, with specific emphasis on the unique needs of voice.

The Operation and Troubleshooting of Link Fragmentation and Interleaving

Now that we have reviewed the technologies of this chapter, it is time to analyze their operation at the Cisco command line.

For exploring the concepts behind troubleshooting LFI, we will utilize the following topology:

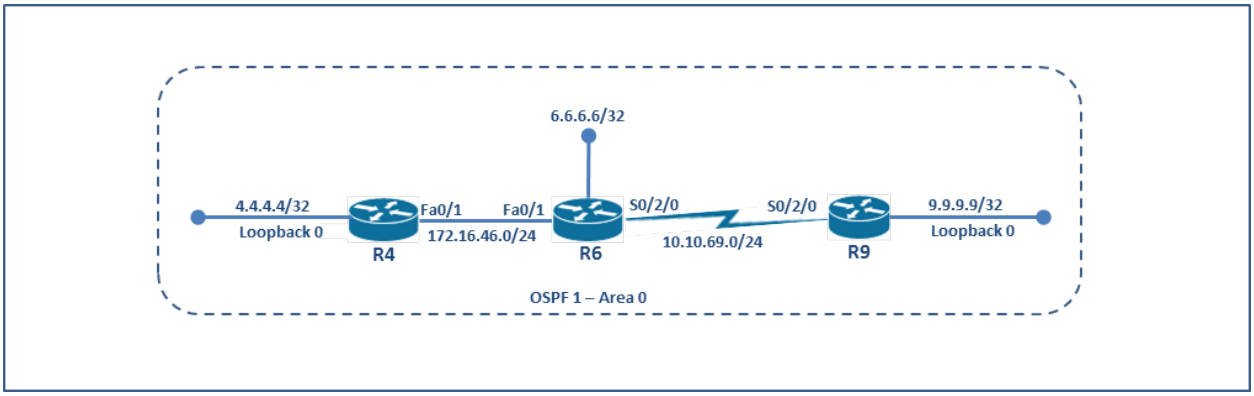

Figure 11-1: A Basic Quality of Service Topology

Multilink PPP with Interleaving

In the Cisco IOS deployment of MLP with interleaving, we follow a very simple three-step process:

- **Step 1** – enable MLP on a PPP interface
  - ppp multilink
- **Step 2** – enable interleaving
  - ppp multilink interleave
- **Step 3** – specify the max fragment size by setting the max desired serialization delay in ms
  - ppp multilink fragment delay

When deployed on a router this configuration will look similar to the example that we will explore on R6 using the MQC process. In this configuration, we are doing more than just configuring LFI, but even with that, we will still follow the three-step process outlined above. We will break down this configuration as we progress through it. First, we need to select a class of traffic to which we want to apply QoS treatment.

```
R6#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R6(config)#class-map VOICE
R6(config-cmap)# match ip dscp ef
R6(config-cmap)#class-map SIGNALING
R6(config-cmap)# match ip dscp cs3
```

In this configuration, we have specified that any packet with a dscp value of EF is classified as voice by the class-map VOICE, and traffic with a dscp value of CS3 is categorized as signaling traffic by the class-map SIGNALING. Now, we will want to assign QoS treatment to traffic matching these classes:

```
R6(config)#policy-map CBWFQ
R6(config-pmap)# class VOICE
R6(config-pmap-c)#  priority 48
```

```
R6(config-pmap-c)# class SIGNALING
R6(config-pmap-c)#  bandwidth 8
R6(config-pmap-c)# class class-default
R6(config-pmap-c)#  fair-queue
```

The policy-map we have created is going to provide 48,000 bits of LLQ throughput to any packets that match the class CM_VOICE, 8,000 bits of guaranteed minimum bandwidth for packets that match the class SIGNALING, and any other traffic will receive weighted fair queuing.

Next we will need to create a Virtual-Template interface for the PPPoFR operation. But first, we need to calculate the fragment size for MLPPP. Since physical port speed is 512Kpbs, and the required serialization delay should not exceed 10ms for voice (remember, fragment size is based on physical port speed!), the fragment size must be set to 512000/8*0.01=640 bytes. How is the fragment size configured with MLPPP? By using command **ppp multilink fragment delay** – however, IOS CLI takes this delay value (in milliseconds) and multiplies it by configured interface (virtual-template) bandwidth (in our case 384Kbps). We can actually change the virtual-template bandwidth to match the physical interface speed, but this would affect the CBWFQ weights! Therefore, we take the virtual-template bandwidth (384Kpbs) and adjust the delay to make sure the fragment size matches the physical interface rate is 512Kpbs. This way, the "effective" delay value would be set to "640*8/384 = 13ms" (Fragment_Size/CIR*8) to accommodate the physical and logical bandwidth differences. (This may be unimportant if our physical port speed does not differ much from our PVC CIR. However, if you have a PVC with a CIR=384Kbps and a port speed of 768Kbps you may want to pay attention to this issue)

```
R6(config)#interface Virtual-Template 1
R6(config-if)# encapsulation ppp
R6(config-if)# ip address 10.10.69.6 255.255.255.0
R6(config-if)# bandwidth 384
R6(config-if)# service-policy output CBWFQ
```

In order to implement the interleaving aspect of this configuration apply the three commands mentioned in the three-step process:

```
R6(config-if)# ppp multilink
R6(config-if)# ppp multilink interleave
R6(config-if)# ppp multilink fragment delay 13
```

Next, we need to configure the PVC shaping settings by using legacy FRTS configuration method as we discussed in **Chapter 8: Traffic Shaping**. Will try to get the Bc set to CIR*10ms.

```
R6(config)#map-class frame-relay SHAPE_384K
R6(config-map-class)#frame-relay cir 384000
R6(config-map-class)#frame-relay mincir 384000
R6(config-map-class)#frame-relay bc 3840
R6(config-map-class)#frame-relay be 0
```

All that remains is to apply the settings to the frame-relay interface and the appropriate PVC's.

```
R6(config)#interface Serial 0/2/0
R6(config-if)# no ip address
```

```
R6(config-if)# frame-relay traffic-shaping
R6(config-if)# frame-relay interface-dlci 100 ppp virtual-template 1
R6(config-fr-dlci)#  class SHAPE_384K
R6(config-fr-dlci)#
%LINK-3-UPDOWN: Interface Virtual-Access2, changed state to up
R6(config-fr-dlci)#end
```

Finally, we need to verify that the configuration is working as expected and that is best accomplished with the **show ppp multilink** command:

```
R6#show ppp multilink
No active bundles
No inactive multilink interfaces
```

This tells us the multilink bundle is not operational. The issue here is that the other end of the circuit must be configured before we can see the multilink bundle become active:

```
R9>en
R9#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)#
R9(config)#
R9(config)#class-map VOICE
R9(config-cmap)# match ip dscp ef
R9(config-cmap)#class-map SIGNALING
R9(config-cmap)# match ip dscp cs3
R9(config-cmap)#!
R9(config-cmap)#policy-map CBWFQ
R9(config-pmap)# class VOICE
R9(config-pmap-c)#  priority 48
R9(config-pmap-c)# class SIGNALING
R9(config-pmap-c)#  bandwidth 8
R9(config-pmap-c)# class class-default
R9(config-pmap-c)#  fair-queue
R9(config-pmap-c)#!
R9(config-pmap-c)#interface Virtual-Template 1
R9(config-if)# encapsulation ppp
R9(config-if)# ip address 10.10.69.9 255.255.255.0
R9(config-if)# bandwidth 384
R9(config-if)# service-policy output CBWFQ
R9(config-if)#!
R9(config-if)# ppp multilink
R9(config-if)# ppp multilink interleave
R9(config-if)# ppp multilink fragment delay 13
R9(config-if)#!
R9(config-if)#map-class frame-relay SHAPE_384K
R9(config-map-class)# frame-relay cir 384000
R9(config-map-class)# frame-relay mincir 384000
R9(config-map-class)# frame-relay bc 3840
R9(config-map-class)# frame-relay be 0
R9(config-map-class)#!
R9(config-map-class)#interface Serial 0/2/0
R9(config-if)# no ip address
```

```
R9(config-if)# frame-relay traffic-shaping
R9(config-if)# frame-relay interface-dlci 100 ppp virtual-template 1
R9(config-fr-dlci)#  class SHAPE_384K
R9(config-fr-dlci)#
%LINK-3-UPDOWN: Interface Virtual-Access2, changed state to up
%LINK-3-UPDOWN: Interface Virtual-Access3, changed state to up
R9(config-fr-dlci)#
%OSPF-5-ADJCHG: Process 1, Nbr 6.6.6.6 on Virtual-Access3 from LOADING to FULL,
Loading Done
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access2, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access3, changed state to up
R9(config-fr-dlci)#end
```

Now with the other end configured, we should see the bundle as up and operational on either R6 or R9.

```
R6#show ppp multilink

Virtual-Access3
  Bundle name: R9
  Remote Endpoint Discriminator: [1] R9
  Local Endpoint Discriminator: [1] R6
  Bundle up for 00:01:27, total bandwidth 384, load 1/255
  Receive buffer limit 12192 bytes, frag timeout 1000 ms
  Interleaving enabled
    0/0 fragments/bytes in reassembly list
    0 lost fragments, 0 reordered
    0/0 discarded fragments/bytes, 0 lost received
    0x14 received sequence, 0x15 sent sequence
  Member links: 1 (max not set, min not set)
    Vi2, since 00:01:27, 624 weight, 614 frag size
No inactive multilink interfaces
```

Next, we need to look at output of the **show policy-map interface** command:

```
R6#show policy-map interface
 Virtual-Template1

  Service-policy output: CBWFQ

    Service policy content is displayed for cloned interfaces only such as vaccess and
sessions
 Virtual-Access3

  Service-policy output: CBWFQ

    queue stats for all priority classes:

      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0

    Class-map: VOICE (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: ip dscp ef (46)
```

```
      Priority: 48 kbps, burst bytes 1500, b/w exceed drops: 0


    Class-map: SIGNALING (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: ip dscp cs3 (24)
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops) 0/0/0
      (pkts output/bytes output) 0/0
      bandwidth 8 kbps

    Class-map: class-default (match-any)
      37 packets, 3471 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
      queue limit 64 packets
      (queue depth/total drops/no-buffer drops/flowdrops) 0/0/0/0
      (pkts output/bytes output) 37/3014
      Fair-queue: per-flow queue limit 16
```

Observe that this output show no mention of LFI, to see that we will need to look at the output of the **show interface virtual-access** command for the most resent virtual-access interface:

```
R6#show interface virtual-access 3 | include MLP|multilink
  Encapsulation PPP, LCP Open, multilink Open
  MLP Bundle vaccess, cloned from Virtual-Template1
```

Lastly, do we have reachability?

```
R6#ping 10.10.69.9 size 1500 repeat 10

Type escape sequence to abort.
Sending 10, 1500-byte ICMP Echos to 10.10.69.9, timeout is 2 seconds:
!!!!!!!!!!
Success rate is 100 percent (10/10), round-trip min/avg/max = 56/56/56 ms
```

This demonstrates that the configuration is operational.

FRF.11 - Voice over Frame Relay Implementation Agreement

This agreement specifies how to transport voice application data over a Frame Relay network. Its focus is to extend the application support of frame relay to include the transport of digital voice payloads, with specific emphasis on the unique needs of voice.

It addresses the following set of requirements:

- Transport of compressed voice within the payload of a frame relay frame
- Support a diverse set of voice compression algorithms

- Effective utilization of low bit-rate frame relay connections
- Multiplexing of up to 255 sub-channels on a single frame relay DLCI
- Support of multiple voice payloads on the same or different sub-channel within a single frame
- Support of data sub-channels on a multiplexed frame relay DLCI

The transport of compressed voice is supported with generalized frame formats, the definition of sub-channels, and the multiplexing of these sub-channels into a single DLCI. These three specific concepts will be the focus of the remainder of this section. Please keep in mind that this is not a definitive list.

Sub-channels

Each Frame Relay PVC (DLCI) represents one logical stream or traffic flow, and it will generally interconnect two logical points in a network to exchange information. The concept of a sub-channel extends this ability allowing the formation of multiple streams within a single PVC. With sub-channels, any given PVC may support up to 255 sub-channels or streams creating up to 255 logical traffic lanes within the connection between two network points.

The content, also called the payload, of an individual PVC is transparent to a Frame Relay network so the implementation of sub-channels remains compatible with existing Frame Relay services. As such, it becomes solely the responsibility of the end-devices to manage the use of sub-channels within the context of the PVCs.

Payloads

Fundamental to the transport of data and voice over sub-channels is the definition of the various payloads. There are two basic types of payloads consisting of "Primary" and "Signaled" payloads. Without going into too many unnecessary details, the Primary payloads include Encoded Voice, Encoded FAX or Voice-band Modem data, and Data frames, while the Signaled payloads include Channel Associated Signaling (CAS), dialed digits, in-band encoded FAX relay, and fault indications.

Each payload entity is referred to as a sub-frame, which consists of a variable length header and the payload itself. Included in the header is the voice/data channel identification (CID) with extension and length indications.

Sub-channel Multiplexing

This may already be clear, but the ability to share a single PVC for the purpose of carrying multiple data flows through a Frame Relay network has a universal benefit, and is not just limited to carrying voice. By placing multiple sub-frame payloads into a single DLCI frame, the end devices may now save on network charges normally incurred with multiple individual PVCs. This helps to increase processing and transport efficiencies.

FRF.12 – Frame Relay Fragmentation

FRF.12 was created by the Frame Relay Forum and addresses the following set of requirements that are not natively supported by frame-relay technology:

- Allow real-time and non-real-time data to share the same Frame Relay link
- Allow the fragmentation of frames of all formats
- Define a fragmentation procedure that can be used by other protocols or IAs, such as FRF.11 (discussed later in this chapter)
- Define structured fragmentation models, such that all models will share a same common fragmentation procedure.

FRF.12 helps to address the impact of delay through a network by way of fragmentation. (Fragmentation, in this context, is the act of splitting a large data frame into smaller frames.) It provides a transmitting Frame Relay device with the ability to fragment long frames into a sequence of shorter frames that are reassembled into the original frame by the receiving device. Additionally, each of the smaller pieces can be transmitted separately through the network allowing better control over delay and delay variation.

Fragmentation also allows the end-station to interleave delay-sensitive traffic from one stream with fragments of a longer frame in another stream onto the same physical link thus improving the delay experienced by each circuit.

FRF.12 requires frame-relay traffic shaping to operate and was developed to support legacy technologies like Voice Over Frame-Relay (VOFR). FRF.12 is deployed using the **map-class** utility in the following fashion:

```
R9#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)# map-class frame-relay FRF_12
R9(config-map-class)#  frame-relay fragment 56
R9(config-map-class)#  exit
R9(config)#  interface Serial0/2/0
R9(config-if)#  frame-relay class FRF_12
R9(config-if)#exit
```

To verify we rely on the **show frame-relay fragment** command.

```
R9#show frame-relay fragment
interface               dlci frag-type  size in-frag    out-frag   dropped-frag
R9#
```

Interestingly enough we do not see any output from this command, and the reason for this again is due to the legacy nature of this deployment. In order for the map-class application to operate on Serial0/2/0 it is necessary to enable frame-relay traffic-shaping:

```
R9#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)#interface Serial0/2/0
R9(config-if)#frame-relay traffic-shaping
R9(config-if)#end
```

Now will repeat the previous show command:

```
R9#show frame-relay fragment
interface                dlci frag-type  size in-frag    out-frag   dropped-frag
Se0/2/0                  100  end-to-end 56   0          0          0
Se0/2/0                  200  end-to-end 56   0          0          0
```

Chapter Challenge: LFI Trouble Ticket

The following section includes one Trouble Ticket associated with a problematic LFI scenario.

Trouble Ticket #1

Your supervisor has brought to your attention that multilink bundle with LFI configured on R6 is not operational. You have been instructed to perform any necessary configuration to bring the multilink bundle up in such a fashion that the output of the show multilink ppp command on R6 matches the attached exhibit.

```
R6#show ppp multilink

Virtual-Access3
  Bundle name: R9
  Remote Endpoint Discriminator: [1] R9
  Local Endpoint Discriminator: [1] R6
  Bundle up for 00:01:27, total bandwidth 384, load 1/255
  Receive buffer limit 12192 bytes, frag timeout 1000 ms
  Interleaving enabled
    0/0 fragments/bytes in reassembly list
    0 lost fragments, 0 reordered
    0/0 discarded fragments/bytes, 0 lost received
    0x14 received sequence, 0x15 sent sequence
  Member links: 1 (max not set, min not set)
    Vi2, since 00:01:27, 624 weight, 614 frag size
No inactive multilink interfaces
```

Chapter Challenge: LFI Trouble Ticket Solution

The following section includes the solution the Trouble Ticket associated with the problematic LFI scenario.

Trouble Ticket #1

Your supervisor has brought to your attention that traffic exiting the Serial0/2/0 interface of R6 is not receiving the correct header compression treatment. You have been instructed to take any steps necessary to ensure that the output of the show policy interface command on R6 matches the attached exhibit.

```
R6#show policy-map interface
 Serial0/2/0

  Service-policy output: COMPRESSION

    Class-map: TELNET (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol telnet
      compress:
          header ip tcp
          TCP (compression on, VJ)
            Sent:    0 total, 0 compressed,
                     0 bytes saved, 0 bytes sent
                     rate 0 bps


    Class-map: class-default (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
```

Step 1 - Fault Verification:

What is the status of the output on R6 with respect to **show ppp multilink** command?

```
R6#show ppp multilink
No active bundles
No inactive multilink interfaces
```

This verifies that the MLP is not operational. Is the configuration applied correctly?

```
R6#show run int virtual-template 1
Building configuration...

Current configuration : 190 bytes
!
interface Virtual-Template1
 bandwidth 384
 ip address 10.10.69.6 255.255.255.0
 ppp multilink
```

```
 ppp multilink interleave
 ppp multilink fragment delay 13
 service-policy output CBWFQ
end
```

Everything seems correct. Is the template applied to the interface for PPPoFR operation?

```
R6#sh run interface Serial0/2/0
Building configuration...

Current configuration : 340 bytes
!
interface Serial0/2/0
 no ip address
 encapsulation frame-relay
 no keepalive
 no fair-queue
 clock rate 2000000
 frame-relay traffic-shaping
 frame-relay map ip 10.10.69.9 100 broadcast
 frame-relay map ip 11.11.69.9 200 broadcast
 frame-relay interface-dlci 100 ppp Virtual-Template1
  class SHAPE_384K
 no frame-relay inverse-arp
end
```

Everything looks correct on R6 but what about R9?

```
R9#sh ppp multilink
No active bundles
No inactive multilink interfaces
R9#
```

Is there a virtual-link?

```
R9#show interface virtual-template 1
                              ^
% Invalid input detected at '^' marker.
```

Is there a policy?

```
R9#show policy-map

R9#
```

There are no multilink ppp or interleaving configurations on R9. We will have to create them.

```
R9#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R9(config)#
R9(config)#class-map VOICE
R9(config-cmap)# match ip dscp ef
R9(config-cmap)#class-map SIGNALING
```

```
R9(config-cmap)# match ip dscp cs3
R9(config-cmap)#!
R9(config-cmap)#policy-map CBWFQ
R9(config-pmap)# class VOICE
R9(config-pmap-c)#  priority 48
R9(config-pmap-c)# class SIGNALING
R9(config-pmap-c)#  bandwidth 8
R9(config-pmap-c)# class class-default
R9(config-pmap-c)#  fair-queue
R9(config-pmap-c)#!
R9(config-pmap-c)#interface Virtual-Template 1
R9(config-if)# encapsulation ppp
R9(config-if)# ip address 10.10.69.9 255.255.255.0
R9(config-if)# bandwidth 384
R9(config-if)# service-policy output CBWFQ
R9(config-if)#!
R9(config-if)# ppp multilink
R9(config-if)# ppp multilink interleave
R9(config-if)# ppp multilink fragment delay 13
R9(config-if)#!
R9(config-if)#map-class frame-relay SHAPE_384K
R9(config-map-class)# frame-relay cir 384000
R9(config-map-class)# frame-relay mincir 384000
R9(config-map-class)# frame-relay bc 3840
R9(config-map-class)# frame-relay be 0
R9(config-map-class)#!
R9(config-map-class)#interface Serial 0/2/0
R9(config-if)# no ip address
R9(config-if)# frame-relay traffic-shaping
R9(config-if)# frame-relay interface-dlci 100 ppp virtual-template 1
R9(config-fr-dlci)#  class SHAPE_384K
R9(config-fr-dlci)#
%LINK-3-UPDOWN: Interface Virtual-Access2, changed state to up
%LINK-3-UPDOWN: Interface Virtual-Access3, changed state to up
R9(config-fr-dlci)#end
```

Now we will repeat the show ppp multilink verification on R6.

```
R6#show ppp multilink

Virtual-Access3
  Bundle name: R9
  Remote Endpoint Discriminator: [1] R9
  Local Endpoint Discriminator: [1] R6
  Bundle up for 00:00:57, total bandwidth 384, load 1/255
  Receive buffer limit 12192 bytes, frag timeout 1000 ms
  Interleaving enabled
    0/0 fragments/bytes in reassembly list
    0 lost fragments, 0 reordered
    0/0 discarded fragments/bytes, 0 lost received
    0x11 received sequence, 0x13 sent sequence
  Member links: 1 (max not set, min not set)
    Vi2, since 00:00:57, 624 weight, 614 frag size
No inactive multilink interfaces
```

This output matches the exhibit thus demonstrating that the trouble ticket has been successfully remediated.

Chapter 12: Resource Reservation Protocol (RSVP)

"Hard QoS" is a reference to the Integrated Services approach to Quality of Service. A key element in this overall approach is the Resource Reservation Protocol (RSVP). Using this protocol, we can attempt to "reserve" resources in the network to produce the appropriate Quality of Service treatment.

Resource Reservation Protocol (RSVP) Technology Review

Before we dive into the operation of RSVP at the Cisco command line, we need to review the technology for this section.

Note: This chapter deals with RSVP tools used by the Cisco IOS.

RSVP Defined

Thus far, in our discussions in this book we have looked at the aspects, deployment and operations of Quality of Service mechanisms that fall into the differentiated services model. We have reached a turning point in our studies at in this chapter because RSVP is considered to be part of the Integrated Services model of QoS. The InterServ model is built upon the idea that applications or end stations will reserve resources across the network in an effort to guarantee the optimal level of service that can be obtained for their data flows. We should note that RSVP operates independently from IntServ, but at the same time offers complementary enhancements to it. This means that RSVP does not provide quality treatment to applications it merely manages the level of treatment in the data path by making reservation requests.

RSVP makes reservation requests by using various combinations of messages or signals. So at its simplest RSVP can be best described as a signaling protocol that makes resource reservations for client applications in an effort to guarantee a certain level of preferential treatment. RSVP is a considered to be a signaling protocol because the reservations it supports are negotiated by communication between the end-points of the data path in question. This signaling is an out-of-band signaling protocol. RSVP packets do not transmit bulk data; they coexist on the network with other packets and work to reserve resources for IP packets or, more accurately, the IP packets that make up the distinct flows needing specialized treatment. These reserved flows are known as sessions.

A given session's packets will always have the same destination address, IP protocol ID and destination port. RSVP will attempt to reserve resources; typically, these resources are bandwidth allocation, processor cycles, or queuing priority. Clients rely on RSVP to request a guarantee of QoS across the network. Routers participate in RSVP by allocating resources to particular flows, or denying resources if none is available, and by forwarding RSVP messages to other devices.

RSVP Operation

We have discussed the notion behind what RSVP is and what it does, now it is time to look closer at how it the mechanism behind its operation. We will look at the specifics of how to configure RSVP later in this chapter, but for now, we will concern ourselves with the strategy it follows to actually reserve resources.

RSPV Messages

RSVP most notably takes place between two clients in a point-to-point situation, or between multiple senders and multiple recipients in multicast environments. It is even possible for RSVP to negotiate a multipoint to single-point transmission. No matter the method, the RSVP session startup process reserves resources in a unidirectional fashion. It is necessary for the session startup process to be

performed twice, once in each direction to guarantee uniform treatment for a given flow. This process is managed through a combination of RSVP message types.

- **Path Messages (PATH)** – sent along the unicast or multicast route by the routing protocol; used to store path state in each node; path state is used to route reservation request messages in reverse direction
- **Reservation-request Messages (RESV)** – sent by receiver hosts toward the senders
- **Error and confirmation messages** - Three error and confirmation message types exist.
  - path error messages (Patherr)
  - reservation request error message (Resverr)
  - reservation request acknowledgment messages (Resvconf).
- **Teardown messages** – after a session is initiated, it is maintained on the routers as a "soft state." With a soft state session, the path connecting two end stations can be renegotiated without consultation with the end stations. This contrasts with a circuit-switched network, where the connection between two end stations is a hard connection, and when a failure occurs, the connection is broken.

Enabling RSVP

Because RSVP works with different queuing strategies that require RSVP to be configured on the output interfaces of routers, RSVP must be enabled on a per interface basis. This is accomplished with the **ip rsvp bandwidth** command.

```
R4#configure t
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface FastEthernet0/1
R4(config-if)#ip rsvp bandwidth 128 56
R4(config-if)#exit
R4(config)#end
```

To verify the configuration of the command simply use the **show ip rsvp interface** command:

```
R4#show ip rsvp interface
interface    allocated  i/f max  flow max sub max
Fa0/1        0          128K     56K      0
```

Troubleshooting RSVP

There are two types of services that can be guaranteed. To provide controlled-load service to applications, RSVP uses WRED on the output interface following the downstream direction of the reservation. To provide guaranteed-rate service, RSVP uses WFQ on the output interface following the downstream direction of the reservation. WRED helps to emulate a zero-load network by avoiding congestion. WFQ guarantees a predefined packet rate and lowers the possibility of packet drops fore applications by assigning a given RSVP session priority over other flows. It is important to remember that RSVP works in one direction only (simplex) with respect to a given packet flow. If a full duplex reservation is needed, RSVP reservations are required in each direction respectively. In other words reservations for an RSVP session are enforced on the output interfaces of the routers following the

direction of the session stream. The router will keep a stateful database of all reservations, and WFQ, WRED, or both QoS treatments are configured at the interface level to facilitate the reservation.

Enabling RSVP is much easier to configure that it is to understand the mechanics of its operation. However, even though it is simple to enable, do not be tempted to enable it carelessly on all interfaces in your router network. It takes planning to ensure that RSVP reservations do not run rob bandwidth from other applications.

Chapter 12 Challenge: Multiple Choice

12-1. Which of the following is not a message type found in RSVP?

a. PATH

b. PATH-RESV

c. Teardown

d. Error

12-2. RSVP uses what mechanism to prevent the occurrence of congestion in the RSVP data path?

a. Congestion Avoidance

b. WFQ

c. CB-Policing

d. WRED

e. FRAD

12-3. RSVP Path creation is what type of process?

a. Unidirectional

b. Full Mesh

c. Bidirectional

d. Stateless

Chapter 12 Challenge: Multiple Choice Answers

12-1. b

12-2. d

12-3. a

Chapter 13: Catalyst Quality of Service (QoS)

Cisco Systems uses a variety of approaches to Quality of Service across their different switch operating systems and hardware platforms. While this book will focus on Catalyst 3560 QoS mechanisms, it is very important that you consult the documentation for your particular Catalyst switch to ensure mastery of the mechanisms that are possible.

Catalyst QoS Technology Review

While you can leverage your knowledge of QoS mechanisms learned throughout this text in this chapter, there are many interesting caveats of configuring QoS on the Catalyst switches. In fact, in this chapter, you will learn some entirely new QoS mechanisms

Classification and Marking

Remember, QoS is disabled by default on 3560 switches. In order to enable it, we use the global configuration command:

```
mls qos
```

Classification of traffic (and subsequent policy decisions) can be accomplished at the port level, or the VLAN level. In order to carry out the classification at the VLAN level, use the following interface command:

```
mls qos vlan-based
```

You should note that this command needs to be applied to all of the interfaces of a particular VLAN. Here is an example, perhaps we have a policy that does classification named **MYCOOLPOLICY**. This policy map is applied to the VLAN interface as follows:

```
interface vlan 101
  service-policy input MYCOOLPOLICY
```

In order for this policy to correctly apply to the ports of the VLAN (101 in this case), we need the following command in the appropriate ports:

```
interface range fa0/3 – 6
 mls qos vlan-based
```

Methods for Classifying Traffic on the Switch

What options are available for marking on the Catalyst switch? There are plenty. For example, at the Data Link Layer (Layer 2) there are:

- MAC Access Control List
- CoS values

At the Network Layer (Layer 3), you can leverage:

- Layer 3 Access Control List
- DSCP values
- IP Precedence values

Setting the Default CoS Value of a Port

The default CoS value for a port is set to 0. In order to set a new value use the interface configuration command:

```
mls qos cos value
```

If you are interested in overriding any existing CoS marking for an incoming frame, you can follow the above command up with this powerful command:

```
mls qos cos override
```

Trusting Incoming Markings

Should you want to trust markings on frames or packets that are entering your Catalyst switch, use the interface command:

```
mls qos trust  marking
```
Possible options here include:

- CoS
- DSCP
- IP Precedence
- Device – this option allows you to specify a network device such as a Cisco IP Phone

Cisco Discovery Protocol would be required for detection of the Cisco IP Phone. Also, remember that you still need to indicate what markings that you are going to trust from that phone. Here is an example:

```
mls qos trust  dscp
mls qos trust  device cisco-phone
```

QoS Marking Using Maps

Once we decide to trust a marking, the switch can remark frames or packets using a mapping table. For example, let us examine the default CoS to DSCP mapping table on the switch:

```
show mls qos maps cos-dscp
```
Notice that should a trusted CoS marking arrive with a value of 2, the switch will add a DSCP marking to the packet of 16.

There are many mapping tables available on the switch:

- cos-dscp
- cos-input-q
- cos-output-q

- dscp-cos
- dscp-input-q
- dscp-mutation
- dscp-output-q
- ip-prec-dscp
- policed-dscp

In order to customize any of the default maps, simply input the new mapped values in global configuration mode. Here is an example:

```
mls qos map cos-dscp 0 8 16 24 32 46 48 56
```

Note: The mapping table shown above is actual a best practice example for VoIP environments.

Policing

As we learned earlier in this course, policing sets a "speed limit" for traffic that is entering or exiting the Catalyst switch (in the case of this chapter). Traffic that is not exceeding the speed limit is termed the *conforming* traffic. Traffic that is exceeding the speed limit is termed the *exceeding* traffic.

Thanks to flexibility in policing configurations on the Catalyst switch, packets can be:

- Transmitted
- Dropped
- Transmitted and Remarked

Remember, on a Catalyst switch, these options are configurable at the switch port level, or the VLAN level.

When configuring traffic policing at the switch port, this QoS treatment can be based on a single class of traffic, or multiple classes of traffic. In the case of multiple traffic classes, this QoS tool is officially termed an aggregate policer.

Here is an example of policing based on a single class of traffic:

```
CAT2#
CAT2#configure terminal
CAT2(config)#access-list 100 permit udp any any range 16384 32767
CAT2(config)#class-map CM_VOICE
CAT2(config-cmap)#match access-group 100
CAT2(config-cmap)#exit
CAT2(config)#policy-map PM_POLICE
CAT2(config-pmap)#class CM_VOICE
CAT2(config-pmap-c)#police 512000 8000 exceed-action drop
CAT2(config-pmap-c)#exit
CAT2(config-pmap)#exit
CAT2(config)#interface fa0/10
CAT2(config-if)#service-policy input PM_POLICE
```

```
CAT2(config-if)#end
CAT2#
```

Notice that this configuration polices Voice traffic to a rate of 512 Kbps. It uses a burst size of 8000 bytes, and drops traffic that is in excess of this rate. This policer is a applied to the Fa0/10 port in the inbound direction.

Note: When entering QoS configurations, always consider using context sensitive help (?) in order to confirm the measurement value (for example, bits versus bytes). Notice with the example above, the policer on the Catalyst switch defaults to bits per second.

In the case of aggregate policer, you create an aggregate policing rule – this is accomplished with the following global configuration command:

```
mls qos aggregate-police
```

This command specifies the rate and the burst value, as well as the policing action.

Under policy-map class configuration mode, you reference the aggregate policing rule that you created using the following command:

```
police aggregate
```

Here is an example of this configuration:

```
CAT2#configure terminal
CAT2(config)#ip access-list standard AL_ANY_IP
CAT2(config-std-nacl)# permit any
CAT2(config-std-nacl)#mac access-list extended AL_ANY_NONIP
CAT2(config-ext-macl)# permit any any 0x0 0xFFFF
CAT2(config-ext-macl)#class-map CM_IP
CAT2(config-cmap)# match access-group name AL_ANY_IP
CAT2(config-cmap)#class-map CM_NONIP
CAT2(config-cmap)# match access-group name AL_ANY_NONIP
CAT2(config-cmap)#aggregate-policer POLICE_AGG 256000 32000 exceed-action drop
CAT2(config)#policy-map PM_AGGREGATE_LIMIT
CAT2(config-pmap)# class CM_IP
CAT2(config-pmap-c)#  police aggregate POLICE_AGG
CAT2(config-pmap-c)# class CM_NONIP
CAT2(config-pmap-c)#  police aggregate POLICE_AGG
CAT2(config-pmap-c)#exit
CAT2(config-pmap)#exit
CAT2(config)#interface fa0/11
CAT2(config-if)#service-policy input PM_AGGREGATE_LIMIT
CAT2(config-if)#end
CAT2#
```

In order to configure policing on a Switched Virtual Interface (SVI or VLAN interface), here is a sample configuration:

```
CAT2(config)#int range fa0/1 – 5
CAT2(config-if-range)#mls qos vlan-based
CAT2(config-if-range)#exit
CAT2(config)#access-list 100 permit udp any any range 16384 32767
CAT2(config)#class-map RTP
CAT2(config-cmap)#match access-group 100
CAT2(config-cmap)#exit
CAT2(config)#class-map PORTS
CAT2(config-cmap)#match input-interface fa0/1 - fa0/5
CAT2(config-cmap)#exit
CAT2(config)#policy-map PORT
CAT2(config-pmap)#class PORTS
CAT2(config-pmap-c)#police 256000 8000 exceed-action drop
CAT2(config-pmap-c)#exit
CAT2(config-pmap)#exit
CAT2(config)#policy-map VLAN
CAT2(config-pmap)#class RTP
CAT2(config-pmap-c)#set dscp 46
CAT2(config-pmap-c)#service-policy PORT
CAT2(config-pmap-c)#exit
CAT2(config-pmap)#exit
CAT2(config)#int vlan 100
CAT2(config-if)#service-policy input VLAN
CAT2(config-if)#end
CAT2#
```

Notice that this is another example of nesting policy maps that is required for a particular Quality of Service treatment.

Congestion Management and Congestion Avoidance

With the 3560 switch from Cisco, there are two input queues and four output queues. The congestion management tool available for these queues is called Shaped Round Robin (SRR).

For the input (also known as ingress) queues, there are two total queues. Queue number 2, be default, is the priority queue and has 10 percent of the interface's bandwidth assigned to it.

There are 4 output (also known as egress) queues. There is no priority queuing enabled by default. Should you desire priority queuing in the egress direction, then you may enable priority queuing for Queue 1.

Here in this example, we will take the priority input queue and move it to Queue number 1. We will assign it 25% of the interface's bandwidth:

```
CAT2(config)#mls qos srr-queue input priority-queue 1 bandwidth 25
```

To enable egress priority queuing (remember, this is not enabled by default), we must enter interface configuration mode to enable the feature:

```
CAT2(config)#int range fa0/1 – 5
CAT2(config-if-range)#priority-queue out
```

Remember, Queue 1 is effected by the configuration above on egress. It now has priority queuing enabled.

In order to further manipulate the queuing behavior on the ports, it is critical that you understand the congestion avoidance mechanism on the Catalyst 3560. This mechanism is termed Weighted Tail Drop (WTD). With this tool, different CoS values can have various drop thresholds. For example, threshold 1 can be defined as 25% of queue capacity and can be mapped to CoS 0 through 2. Threshold 2 can be defined at 50% of the queue capacity and can be mapped to CoS 3 through 4. Threshold 3 can be defined at 100% of the queue capacity and can be mapped to CoS 5 through 7. How this functions is as the queue starts filling to capacity, the Catalyst switch starts denying admittance to certain packets with specific CoS values. Notice that in our example, the queue must be filled to capacity before VoIP packets are denied entrance. This is, of course, how we would want the switch to behave. Remember, when the queue is completely full, we unfortunately experience what is termed Tail Drop.

Because there are so many different parameters that can be set for a port, Cisco allows the creation of a Queue Set. The Queue Set is a set of output queuing and weighted tail drop parameters that govern the QoS behavior of the port. Queue Sets allow for the following settings to be applied in a macro-like format:

- WTD thresholds
- Guaranteed buffer availability
- Maximum memory allocation
- Buffer allocation for all output queues

By default, all ports on the 3560 belong to Queue Set 1. You can assign a port to a second queue set using the following command:

```
CAT2(config-if)#queue-set qset-id
```

Notice from the configurable parameters for the queue set, you can configure many potential settings. Notice, for example, that each output queue of a port is allocated a specific amount of memory. You can specify what percentage of a queue's allocated memory is guaranteed. You can also specify the maximum percentage of a queue's allocated memory that a queue can have.

Note: When specifying the maximum percentage of a queue's allocated memory that a queue can have, you can specify more than 100 percent. How is this possible? Memory can be taken from the common memory pool.

Creating a Queue Set and assigning the various parameters is done in global configuration mode. Here is the syntax:

```
CAT2(config)# mls qos queue-set output qset-id threshold queue-id drop-threshold1
drop-threshold2 reserved-threshold maximum-threshold
```

In order to size the queues differently, use the following global configuration command:

```
CAT2(config)# mls qos queue-set output qset-id buffers allocation1 allocation2
allocation3 allocation4
```

Here is an example of the use of these powerful queue sets:

```
CAT2(config)#mls qos queue-set output 2 buffers 50 25 10 15
CAT2(config)#mls qos queue-set output 2 threshold 2 33 66 100 200
CAT2(config)#interface fa0/11
CAT2(config-if)#queue-set 2
```

Notice in this example, we create a Queue Set 2. This Queue Set is applied to the fa0/11 port. For Queue Set 2, the following buffer space allocations of the port are made:

- 50 percent of a port's buffer space is allocated for Queue 1
- 25 percent is allocated for Queue 2
- 10 percent is allocated for Queue 3
- 15 percent is allocated for Queue 4

For Queue Set 2, output queue 2 (of 4) has its first drop threshold set at 33 percent. The second drop threshold is set at 66 percent. 100 percent of Queue 2's allocated buffer space is guaranteed to be available, if needed. If Queue 2 needs more buffer space, it can borrow from a port's unused buffer space, up to a maximum of 200 percent of Queue 2's buffer allocation.

How do we map CoS (or DSCP) values into these queues? The syntax is shown here:

```
CAT2(config)# mls qos srr-queue output [cos-map | dscp-map] queue queue-id threshold
threshold-id qos-marking-1 … qos-marking-8}
```

Here is an example:

```
CAT2(config)#mls qos srr-queue output cos-map queue 1 threshold 1 0 1
CAT2(config)#mls qos srr-queue output cos-map queue 1 threshold 2 2 3
CAT2(config)#mls qos srr-queue output cos-map queue 2 threshold 1 4
CAT2(config)#mls qos srr-queue output cos-map queue 3 threshold 2 5
CAT2(config)#mls qos srr-queue output cos-map queue 4 threshold 2 6 7
```

Similar to the configuration of our output queues, we can set the buffer allocation for our input queues with the following command:

```
CAT2(config)# mls qos srr-queue input buffers percentage1 percentage2
```

Note: Remember, there are no Queue Sets to assist with the configuration of inbound (ingress) queues.

Shaped Round Robin

As stated earlier, Shaped Round Robin (SRR) is the congestion management method that is at work when it comes to emptying the queues on the 3560 Catalyst switch. It feature two modes of operation:

- Shaped – the shaped method is only available on egress queues; utilizing this approach, a queue receives a reservation that is a portion of a port's bandwidth, and the queue can receive no more
- Shared – the shared method is available on ingress and egress queues; utilizing this approach, a queue is guaranteed a portion of a port's bandwidth, but is not limited to that guaranteed amount

The following command is used to configure *shared* mode. Notice this command assigns relative queue weights to an interface's four output queues:

```
CAT2(config-if)# srr-queue bandwidth share weight1 weight 2 weight3 weight4
```

Note: When using this command, the *relative* weights to not have to total 100. Please keep in mind, however, that selecting values that do total 100 makes it easier to determine the bandwidth available to each queue.

Remember, *shaped* mode applies a bandwidth restriction (i.e. policing) for a queue. Please notice that the weight configured is not the *relative* weight, as it was for the shared mode. In this case, the inverse of the weight (1/weight) determines the shaped bandwidth for a queue. Here is the command to configure shaped mode:

```
CAT2(config-if)# srr-queue bandwidth shape weight1 weight 2 weight3 weight4
```

Note: Should you configure a queue for both shared and shaped mode, the shaped mode configuration is applied.

Here is a *shared* mode example:

```
CAT2(config)#int fa0/11
CAT2(config-if)#speed 1000
CAT2(config-if)#srr-queue bandwidth share 10 25 35 50
```

How much bandwidth is available for Queue 1? It is [10/(10+25+35+50)] * 1000 Mbps = 83.3 Mbps

Here is a *shaped* mode example:

```
CAT2(config)#int fa0/11
CAT2(config-if)#speed 1000
CAT2(config-if)#srr-queue bandwidth shape 30 0 0 0
```

How much bandwidth is available for Queue 1? It is 1/30 * 1000 Mbps = 33.3 Mbps; notice there is no limit applied to the other queues.

Finally, understand that you can limit the maximum amount of an interface's bandwidth that can be used for outgoing traffic. By default, there is no limit (therefore a weight of 100). The command to change this default is shown here:

```
CAT2(config-if)# srr-queue bandwidth limit weight
```

Note: The authors would like to thank Kevin Wallace of http://www.1examamonth.com for his assistance with this chapter. Be sure to visit this excellent site today for amazingly valuable certification resources, including an excellent video on Catalyst QoS from which this chapter was drawn.

Chapter Challenge: Multiple Choice

13-1. What are two options for the configuration of some Catalyst QoS features? Choose two.

a. Port-based
b. Switch-based
c. Backplane-based
d. VLAN-based

13-2. Which mode of SRR applies a bandwidth restriction (i.e. policing) for a queue?

a. Shaped
b. Shared
c. Smoothed
d. Static

13-3. What must you also do when you use the command `mls qos trust device cisco-phone`?

a. You must indicate how the phone will be detected
b. You must specify the model of phone
c. You must specify what markings are trusted
d. You must enable a mutation map

Chapter Challenge: Multiple Choice Answers

13-1. a, d

13-2. a

13-3. c

Chapter 14: AutoQoS

Cisco realizes just how difficult and complex appropriate Quality of Service configurations throughout the network can be. As a result, they offer a “macro” that can configure appropriate QoS settings automatically. When AutoQos was first developed by Cisco, it was targeted at Voice over IP environments. Cisco has built upon this with AutoQoS for Enterprise. Using the protocol discovery capabilities of NBAR, AutoQoS discovers running protocols and attempts to configure the QoS in the network appropriately.

AutoQoS Technology Review

As amazing as it might seem, AutoQoS provides a "single command" approach to the complex QoS configurations required for a complete QoS solution.

AutoQoS is not totally unique for Cisco, Cisco utilizes a similar approach when it comes to automatically securing a router. Auto Secure allows the automatic "hardening" of a Cisco router with just a single command.

Engineers will often scoff at the thought of AutoQoS. Why have the router auto configure these settings when the engineer is certainly capable of making the. It is certainly worth noting, that even if you feel confident in your abilities to configure QoS, you could consider AutoQoS in the interest of saving time and ensuring consistent configurations. Remember, that one can always tune the automatic configurations that AutoQoS produces.

AutoQoS is carried out on a Catalyst switch thanks to an interface level command. The command (on a 3560) is as follows:

```
auto qos voip [cisco-phone | cisco-softphone | trust]
```

Notice the optional keywords. These keywords permit you to specify what packets will be trusted as far as QoS markings are concerned. In our example here, we need to ensure we only trust markings coming from a genuine Cisco IP Phone, so we need the **cisco-phone** keyword. How does our switch know the packets are coming from a genuine Cisco VoIP end user device? Well, Cisco Discovery Protocol (CDP) is relied upon for this determination.

Note that **the cisco-softphone** keyword enacts trust for voice packets sent from a Cisco phone running in software on a PC. Finally, the **trust** keyword trusts markings for VoIP packets coming from another switch or router.

Should you want to see what this feature did to your device and your port, you can issue the **show auto qos** command.

The Operation and Troubleshooting of AutoQoS

By far, the biggest area of trouble for AutoQoS is the fact that there are indeed requirements that must be met for its successful operation. This is often overlooked by engineers. After all, Cisco promised us just a single command, correct?

The requirements for the successful implementation of AutoQoS are as follows:

- No service policy can already be attached to the interface where you plan to execute AutoQoS; this of course can be confirmed with the command **show policy map interface**
- Should you desire to receive the SNMP trap information that the AutoQoS configuration will generate, you must configure the SNMP server information manually; while AutoQoS is remarkable, it cannot configure supplemental servers that it is not aware of
- AutoQoS supports the following interfaces:
    - Serial (PPP or HDLC)
    - Frame Relay Point-to-Point Subinterfaces
    - ATM Point-to-Point Subinterfaces with Bandwidth less than 768 kbps
- An IP address must be assigned to the interface
- In the case of slow speed Serial interfaces, AutoQoS must be both configured at each end of the link
- The **bandwidth** command should be assigned to the interface; if you change it later, Auto QoS will not dynamically adjust to the new bandwidth that you have configured
- For Frame-Relay configurations, no existing map class can be assigned to the DLCI
- For Frame-Relay configurations, AutoQoS fragmentation is using a delay of 10 ms and a minimum fragment size of 60 bytes; however, with G.711 on low-speed links, this fragment size could be smaller than the G.711 VoIP packet - to solve this issue either change the fragment size to the required value or change the size of the G.711 VoIP packet to a smaller value
- Cisco Express Forwarding must be configured on the interface
- For an ATM PVC, configure VBR or CBR
- For signaling protocols, Auto QoS is built to identify:
    - H.323
    - H.225 (unicast only)
    - Session Initiation Protocol (SIP)
    - "Skinny" gateway protocol
    - Media Gateway Control Protocol (MGCP)

Here is an example of AutoQoS in action:

```
Cat4(config-if)#auto qos voip trust
Cat4(config-if)#
```

And of course, much more remarkable is the result:

```
Cat4#show run
```

```
Building configuration...


Current configuration : 6536 bytes
!
OUTPUT OMITTED
!
hostname Cat4
!
OUTPUT OMITTED
!
!
OUTPUT OMITTED
!
!
!
mls qos map cos-dscp 0 8 16 24 32 46 48 56
mls qos srr-queue input bandwidth 90 10
mls qos srr-queue input threshold 1 8 16
mls qos srr-queue input threshold 2 34 66
mls qos srr-queue input buffers 67 33
mls qos srr-queue input cos-map queue 1 threshold 2  1
mls qos srr-queue input cos-map queue 1 threshold 3  0
mls qos srr-queue input cos-map queue 2 threshold 1  2
mls qos srr-queue input cos-map queue 2 threshold 2  4 6 7
mls qos srr-queue input cos-map queue 2 threshold 3  3 5
mls qos srr-queue input dscp-map queue 1 threshold 2  9 10 11 12 13 14 15
mls qos srr-queue input dscp-map queue 1 threshold 3  0 1 2 3 4 5 6 7
mls qos srr-queue input dscp-map queue 1 threshold 3  32
mls qos srr-queue input dscp-map queue 2 threshold 1  16 17 18 19 20 21 22 23
mls qos srr-queue input dscp-map queue 2 threshold 2  33 34 35 36 37 38 39 48
```

```
mls qos srr-queue input dscp-map queue 2 threshold 2  49 50 51 52 53 54 55 56
mls qos srr-queue input dscp-map queue 2 threshold 2  57 58 59 60 61 62 63
mls qos srr-queue input dscp-map queue 2 threshold 3  24 25 26 27 28 29 30 31
mls qos srr-queue input dscp-map queue 2 threshold 3  40 41 42 43 44 45 46 47
mls qos srr-queue output cos-map queue 1 threshold 3  5
mls qos srr-queue output cos-map queue 2 threshold 3  3 6 7
mls qos srr-queue output cos-map queue 3 threshold 3  2 4
mls qos srr-queue output cos-map queue 4 threshold 2  1
mls qos srr-queue output cos-map queue 4 threshold 3  0
mls qos srr-queue output dscp-map queue 1 threshold 3  40 41 42 43 44 45 46 47
mls qos srr-queue output dscp-map queue 2 threshold 3  24 25 26 27 28 29 30 31
mls qos srr-queue output dscp-map queue 2 threshold 3  48 49 50 51 52 53 54 55
mls qos srr-queue output dscp-map queue 2 threshold 3  56 57 58 59 60 61 62 63
mls qos srr-queue output dscp-map queue 3 threshold 3  16 17 18 19 20 21 22 23
mls qos srr-queue output dscp-map queue 3 threshold 3  32 33 34 35 36 37 38 39
mls qos srr-queue output dscp-map queue 4 threshold 1  8
mls qos srr-queue output dscp-map queue 4 threshold 2  9 10 11 12 13 14 15
mls qos srr-queue output dscp-map queue 4 threshold 3  0 1 2 3 4 5 6 7
mls qos queue-set output 1 threshold 1 138 138 92 138
mls qos queue-set output 1 threshold 2 138 138 92 400
mls qos queue-set output 1 threshold 3 36 77 100 318
mls qos queue-set output 1 threshold 4 20 50 67 400
mls qos queue-set output 2 threshold 1 149 149 100 149
mls qos queue-set output 2 threshold 2 118 118 100 235
mls qos queue-set output 2 threshold 3 41 68 100 272
mls qos queue-set output 2 threshold 4 42 72 100 242
mls qos queue-set output 1 buffers 10 10 26 54
mls qos queue-set output 2 buffers 16 6 17 61
mls qos
!
```

```
OUTPUT OMITTED
!
!
OUTPUT OMITTED
!
!
!
!
!
!
OUTPUT OMITTED
!
OUTPUT OMITTED
!
!
!
!
OUTPUT OMITTED
!
interface FastEthernet0/11
 switchport mode access
 srr-queue bandwidth share 10 10 60 20
 priority-queue out
 mls qos trust cos
 auto qos voip trust
 spanning-tree portfast
!
OUTPUT OMITTED
!
OUTPUT OMITTED
```

```
!
OUTPUT OMITTED
end


Cat4#
```

AutoQoS for Enterprise

With the introduction of AutoQoS for Enterprise, Cisco makes a big leap forward with the feature. Thanks to this technology, AutoQoS is just not for Voice over IP environments anymore.

This approach to AutoQoS consists of two interrelated phases:

Phase 1 – the Discovery Phase – in this phase, NBAR-based protocol discovery detects applications and performs statistical analysis of the amount of traffic that constitutes the different applications

Phase 2 – the Installation Phase – in this phase, the Cisco device generates the configurations based on the information calculated in the Discovery Phase

Should you wish to learn more about AutoQoS for Enterprise, visit:

http://www.cisco.com/en/US/docs/ios-xml/ios/qos_auto/configuration/12-4t/qos-auto-ent.html

Chapter Challenge: Multiple Choice

14-1. What will happen to your AutoQoS configuration should you change the bandwidth as reported by the **bandwidth** command after you have already executed the Auto QoS feature?

e. The AutoQoS configuration will automatically adjust configurations to reflect the new specified bandwidth
f. The AutoQoS configuration will remain the same
g. The AutoQoS configuration will be removed
h. None of these options are correct

14-2. Which of the following is not a valid requirement for the use of AutoQoS?

e. No service policy can already be attached to the interface
f. The interface must be an SVI
g. An IP address must be assigned to the interface
h. The **bandwidth** command should be configured

Chapter Challenge: Multiple Choice Answers

14-1. b

14-2. b