IPQoS Administration Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816–4094–10
September 2002

020618@4333

Contents

Tables

Figures

Examples

Preface

The *IPQoS Administration Guide* explains how to provide differentiated services on a network through use of the IPQoS feature in the Solaris™ operating environment. IPQoS enables you to provide different levels of service to network customers and to manage network traffic.

Who Should Use This Book

This module assumes that you are a very experienced system administrator with an extensive knowledge of TCP/IP concepts. You might be responsible for, or be familiar with, router administration for your network. You should also be familiar with your site's network topology and corporate policies on network usage, and, possibly, on network security.

How This Book Is Organized

The *IPQoS Administration Guide* contains the following chapters:

Chapter 1 provides basic information about the IPQoS feature and the diffserv architecture on which IPQoS is based.

Chapter 2 contains tasks for planning the topology of an IPQoS-aware network. The chapter also contains planning tasks for creating a quality-of-service policy for a prospective IPQoS system.

Chapter 3 contains tasks for building an IPQoS configuration file that is based on the quality-of-service policy.

Chapter 4 contains tasks for maintaining and tracking IPQoS.

Chapter 5 contains tasks for configuring the IPQoS flow accounting and displaying IPQoS statistics with the `kstat` command.

Chapter 6 contains in-depth information about the IPQoS modules and the IPQoS configuration file.

Related Books

The following books discuss the differentiated services architecture:

- Ferguson, Paul and Geoff Huston. *Quality of Service*. John Wiley & Sons, Inc., 1998.
- Kilkki, Kalevi. *Differentiated Services for the Internet*. Macmillan Technical Publishing, 1999.

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Introducing IPQoS (Overview)

The IP quality-of-service (IPQoS) feature enables you to prioritize, control, and gather accounting statistics. Using IPQoS, you can provide consistent levels of service to users of your network, and manage traffic to avoid network congestion.

The following is a list of topics in this chapter:

IPQoS Basics

IPQoS enables the Differentiated Services (diffserv) architecture that is defined by the Differentiated Services Working Group of the Internet Engineering Task Force (IETF). In the Solaris™ 9, 9/02 operating environment, IPQoS is implemented at the IP level of the TCP/IP protocol stack.

What Are Differentiated Services?

By enabling IPQoS, you can provide different levels of network service for selected customers and selected applications. These *differentiated services* can be based on a structure of service levels that your company offers to its customers. You can also provide differentiated services that are based on the priorities set for applications or users on your network.

Providing quality of service involves the following activities:

- Delegating levels of service to different groups, such as customers or departments in an enterprise
- Prioritizing network services that are given to particular groups or applications
- Discovering and eliminating areas of network bottlenecks and other forms of congestion
- Monitoring network performance and providing performance statistics
- Regulating bandwidth to and from network resources

IPQoS Features

IPQoS has the following features:

- Command-line tool for configuring the QoS policy
- Classifier that selects actions, which are based on filters that configure the QoS policy of your organization
- Metering module that measures network traffic, in compliance with the diffserv model
- Service differentiation that is based on the ability to mark a packet's IP header with forwarding information
- Flow-accounting module that gathers statistics for traffic flows
- Statistics gathering for traffic classes, through the UNIX® `kstat` command
- Support for SPARC™ architecture
- Support for IPv4 and IPv6 addressing
- Interoperability with IPsec
- Support for 802.1 D user priority markings for virtual local area networks (VLANs)

Where to Get More Information About Quality of Service

You can find information on differentiated services and quality of service from print and online sources.

Books

For more information on quality-of-service theory and practice, refer to the following books:

- Ferguson, Paul and Geoff Huston. *Quality of Service*. John Wiley & Sons, Inc., 1998.
- Kilkki, Kalevi. *Differentiated Services for the Internet*. Macmillan Technical Publishing, 1999.

Requests for Comments (RFCs)

IPQoS conforms to the specifications that are described in the following RFCs and Internet drafts:

- RFC 2474, *Definition of the Differentiated Services Field*, which describes an enhancement to the ToS or DS fields of the IPv4 and IPv6 packet headers to support differentiated services.
- RFC 2475, *An Architecture for Differentiated Services*, which provides a detailed description of the organization and modules of the diffserv architecture.
- RFC 2597, *Assured Forwarding PHB Group*, which describes how the assured forwarding (AH) per-hop behavior works.
- RFC 2598, *An Expedited Forwarding PHB*, which describes how the expedited forwarding (EF) per-hop behavior works.
- Internet-Draft, *An Informal Management Model for Diffserv Routers*, which presents a model for implementing the diffserv architecture on routers.

Web Sites

The Differentiated Services Working Group of the IETF maintains a Web site with links to diffserv Internet drafts at `http://www.ietf.org/html.charters/diffserv-charter.html`.

Router manufacturers such as Cisco Systems and Juniper Networks provide information on their corporate Web sites that describes how differentiated services are implemented on their products.

Man Pages

The IPQoS distribution includes the following man pages.

- `ipqosconf`(1m), which is the command for setting up the IPQoS configuration file
- `ipqos`(7ipp), which describes the IPQoS implementation of the diffserv architectural model
- `ipgpc`(7ipp), which describes the IPQoS implementation of a diffserv classifier
- `tokenmt`(7ipp), which describes the IPQoS `tokenmt` meter module
- `tswtclmt`(7ipp), which describes the IPQoS `tswtclmt` meter module
- `dscpmk`(7ipp), which describes the DSCP marker module
- `dlcosmk`(7ipp), which describes the IPQoS 802.1D user priority marker module
- `flowacct`(7ipp), which describes the IPQoS flow-accounting module
- `acctadm`(1m), which is the command that configures the Solaris extended accounting facilities and now includes IPQoS extensions

Providing Quality of Service With IPQoS

IPQoS features enable Internet service providers (ISPs) and application service providers (ASPs) to offer different levels of network service to customers. These same features enable individual companies and educational institutions to prioritize services for internal organizations or for major applications.

Implementing Service-Level Agreements

If your organization is an ISP or ASP, you can base your IPQoS configuration on the *service-level agreement* (SLA) that your company offers to its customers. In an SLA, a service provider guarantees to a customer a certain level of network service that is based on a price structure. For example, a premium-priced SLA might ensure that the customer receives highest priority for all types of network traffic 24 hours per day. Conversely, a medium-priced SLA might guarantee that the customer receives high priority for email only during business hours and medium priority for all other traffic 24 hours a day.

Assuring Quality of Service for an Individual Organization

If your organization is an enterprise or an institution, you can also provide quality-of-service features for your network. You can guarantee that traffic from a particular group or from a certain application is assured a higher or lower degree of service.

Introducing the Quality-of-Service Policy

You implement quality of service by defining a *quality-of-service (QoS) policy*. The QoS policy defines various network attributes, such as customers' or applications' priorities, and actions for handling different categories of traffic. You implement your organization's QoS policy in an IPQoS configuration file. This file configures the IPQoS modules that reside in the Solaris 9, 9/02 kernel. A host with an applied IPQoS policy is considered an *IPQoS-enabled system*.

Your QoS policy typically defines the following:

- Discrete groups of network traffic that are called *classes of service*.
- Metrics for regulating the amount of network traffic for each class. These metrics govern the traffic-measuring process that is called *metering*.

- Actions that an IPQoS system and a diffserv router must apply to a packet flow. This action is called a *per-hop behavior* (PHB).
- Any statistics gathering that your organization requires for a class of service, for example, traffic that is generated by a customer or particular application.

When packets pass to your network, the IPQoS-enabled system evaluates the packet headers. The action that the IPQoS system takes is determined by your QoS policy.

Tasks for designing the QoS policy are in "Planning the Quality-of-Service Policy" on page 35.

Improving Network Efficiency With IPQoS

IPQoS contains features that can help you make network performance more efficient as you implement quality of service. When computer networks expand, the need also increases for managing network traffic that is generated by increasing numbers of users and more powerful processors. Some symptoms of an overused network include lost data and traffic congestion that results in slow response times.

In the past, system administrators handled network traffic problems by adding more bandwidth. Often the level of traffic on the links varied widely. With IPQoS, you can manage traffic on the existing network and help assess where, and whether, expansion is necessary.

For example, for an enterprise or institution, you must maintain an efficient network to avoid traffic bottlenecks. You must also assure that a group or application does not consume more than its allotted bandwidth. Moreover, for an ISP or ASP, you must manage network performance to ensure that customers receive the amount of network service for which they have paid.

What Is Bandwidth and How Does It Affect Network Traffic?

You can use IPQoS to regulate network *bandwidth*, the maximum amount of data that a fully used network link or device can transfer. Your QoS policy should prioritize the use of bandwidth to provide quality of service to customers or users. The IPQoS metering modules enable you to measure and control bandwidth allocation among the various traffic classes on an IPQoS-enabled host.

Before you can effectively manage traffic on your network, you must answer these questions about bandwidth usage:

- What are the traffic problem areas for your local network?
- What must you do to achieve optimum use of available bandwidth?
- What are your site's critical applications, which must be given highest priority?
- Which applications are sensitive to congestion?
- What are your less-critical applications, which can be given a lower priority?

Using Classes of Service to Prioritize Traffic

To implement quality of service, you analyze network traffic to determine any broad groupings into which the traffic can be divided. Then you organize the various groupings into classes of service that have individual characteristics and priorities. These classes form the basic categories on which you base the QoS policy for your organization. The classes of service represent the traffic groups that you want to control.

For example, a provider might offer platinum, gold, silver, and bronze levels of service, available at a sliding price structure. A platinum SLA might guarantee top priority to incoming traffic that is destined for a web site that the ISP hosts for the customer. Thus, incoming traffic to the customer's web site could be one traffic class.

For an enterprise, you could create classes of service that are based on department requirements or based on the preponderance of a particular application in the network traffic. Here are a few examples of traffic classes for an enterprise:

- Popular applications such as email and outgoing FTP to a particular server, each of which could constitute a class. Because employees constantly use these applications, your QoS policy might guarantee them a small amount of bandwidth and a lower priority.
- An order-entry database that needs to run 24 hours a day. Depending on the importance of the database application to the enterprise, you might give the database a large amount of bandwidth and a high priority.
- A department that performs critical or sensitive work, such as the payroll department. The importance of the department to the organization would determine the priority and amount of bandwidth you would give to such a department.
- Incoming calls to a company's external Web site. You might give this class a moderate amount of bandwidth that runs at low priority.

Differentiated Services Model

IPQoS includes the following modules, which are part of the *differentiated services (diffserv)* architecture that is defined in RFC 2475.

- Classifier
- Meter
- Marker

IPQoS adds the following enhancements to the diffserv model:

- Flow-accounting module
- 802.1D datagram marker

This section introduces the diffserv modules as they are used by IPQoS. You need to know about these modules, their names, and their uses to set up the QoS policy. For detailed information about each module, refer to "IPQoS Architecture and the diffserv Model" on page 95.

Classifier (`ipgpc`) Overview

In the diffserv model, the classifier selects packets from a network traffic *flow*—a group of packets with identical information in the following IP header fields.

- Source address
- Destination address
- Source port
- Destination port
- Protocol number

In IPQoS, these fields are referred to as the *5–tuple*.

The IPQoS classifier module is named `ipgpc`. `ipgpc` arranges traffic flows into classes that are based on characteristics you configure in the IPQoS configuration file.

For detailed information about `ipgpc`, refer to "Classifier Module" on page 95.

Classes

A *class* is a group of network flows that share similar characteristics. For example, an ISP might define classes to represent the different service levels that are offered to customers. An ASP might define SLAs that give different levels of service to various applications. For an ASP's QoS policy, a class might include outgoing FTP traffic that is bound for a particular destination IP address. Outgoing traffic from a company's external Web site might also be defined as a class.

Grouping traffic into classes is a major part of planning your QoS policy. When you create classes by using the `ipqosconf` utility, you are actually configuring the `ipgpc` classifier.

For information on how to define classes, see "How to Define the Classes for Your QoS Policy" on page 38.

Filters

Filters are sets of rules that contain parameters which are called *selectors*. Each filter must point to a class. IPQoS matches packets against the selectors of each filter to determine if the packet belongs to the filter's class. You can filter on a packet by using a variety of selectors, for example the IPQoS 5–tuple and other common parameters:

- Source and destination addresses
- Source and destination port numbers
- Protocol numbers
- User IDs
- Project IDs
- Differentiated services codepoint (DSCP)
- Interface index

For example, a simple filter might include the destination port with the value of 80. The `ipgpc` classifier then selects all packets that are bound for destination port 80 (HTTP) and handles the packets as directed in the QoS policy.

For information on creating filters, see "How to Define Filters in the QoS Policy" on page 40.

Meter (`tokenmt` and `tswtclmt`) Overview

In the diffserv model, the *meter* tracks the transmission rate of traffic flows on a per-class basis. The meter evaluates how much the actual rate of the flow conforms to the configured rates in order to determine the appropriate outcome. Based on the traffic flow's outcome, the meter selects a subsequent action. Subsequent actions might include to send the packet to another action or to return the packet to the network without further processing.

The IPQoS meters determine whether a network flow conforms to the transmission rate that is defined for its class in the QoS policy. IPQoS includes two metering modules:

- `tokenmt`, which uses a two-token bucket metering scheme
- `tswtclmt`, which uses a time-sliding window metering scheme

Both metering modules recognize three outcomes: red, yellow, and green. You define the actions to be taken for each outcome in the parameters `red action_name`, `yellow action_name`, and `green action_name`.

In addition, you can configure `tokenmt` to be color aware. A color-aware metering instance uses the packet's size, DSCP, traffic rate, and configured parameters to determine the outcome. The meter uses the DSCP to map the packet to a green, yellow or red outcome.

For information on defining parameters for the meters, refer to "How to Plan Flow Control" on page 42.

Marker (`dscpmk` and `dlcosmk`) Overview

In the diffserv model, the *marker* marks a packet with a value that reflects a forwarding behavior. *Marking* is the process of placing a value in the packet's header to indicate how to forward the packet to the network. IPQoS contains two marker modules:

- `dscpmk`, which marks the DS field in an IP packet header with a numeric value called the *DS codepoint*, or *DSCP*. A diffserv-aware router can then use the DS codepoint to apply the appropriate forwarding behavior to the packet.
- `dlcosmk`, which marks the virtual local area network (VLAN) tag of an Ethernet frame header with a numeric value called the *user priority*. The user priority indicates the *class of service (CoS)*, which defines the appropriate forwarding behavior to be applied to the datagram.

  `dlcosmk` is an IPQoS addition that is not part of the diffserv model, as designed.

For information on implementing a marker strategy for the QoS policy, see "How to Plan Forwarding Behavior" on page 44.

Flow Accounting (`flowacct`) Overview

IPQoS adds the `flowacct` accounting module to the diffserv model. You can use `flowacct` to take statistics on traffic flows, and bill customers in agreement with their SLAs. Flow accounting is also useful for capacity planning and system monitoring.

`flowacct` works with the `acctadm` command to create an accounting log file. A basic log includes the IPQoS 5–tuple and two additional attributes, as shown in the following list:

- Source address
- Source port
- Destination address
- Destination port
- Protocol number
- Number of packets
- Number of bytes

You can also gather statistics on other attributes, as described in "Recording Information About Flows" on page 90 and the `flowacct`(7ipp) and `acctadm`(1m) man pages.

For information on planning a flow-accounting strategy, see “How to Plan for Flow Accounting” on page 47.

Example—How Traffic Flows Through the IPQoS Modules

The next figure shows a path that incoming traffic might take as it traverses some of the IPQoS modules.

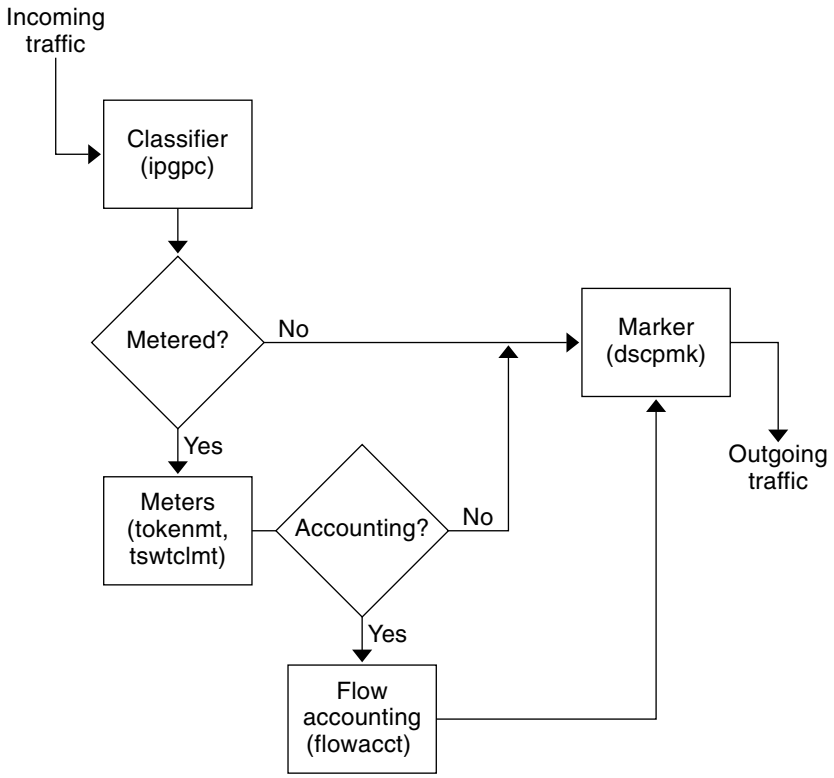

FIGURE 1–1 Traffic Flow Through the IPQoS Implementation of the Diffserv Model

The figure illustrates a common traffic flow sequence¸ on an IPQoS-enabled machine:

1. The classifier selects from the packet stream all packets that match the filtering criteria in the system’s QoS policy.
2. The selected packets are then evaluated for the next action to be taken on them.
3. The classifier sends to the marker any traffic that does not require flow control.
4. Traffic to be flow-controlled is sent to the meter.

5. The meter enforces the configured rate and assigns a traffic conformance value to the flow-controlled packets.
6. The flow-controlled packets are then evaluated to determine if any packets require accounting.
7. The meter sends to the marker any traffic that does not require flow accounting.
8. The flow-accounting module gathers statistics on received packets. The module then sends the packets to the marker.
9. The marker assigns a DS codepoint to the packet header. This DSCP indicates the per-hop behavior that a diffserv-aware system must apply to the packet.

Traffic Forwarding on an IPQoS-Enabled Network

This section introduces the elements that are involved in forwarding packets on an IPQoS-enabled network. An IPQoS-enabled system handles any packets on the network stream with the system's IP address as the destination. The IPQoS system then applies its QoS policy to the packet to establish differentiated services.

DS Codepoint (DSCP)

The DS codepoint defines in the packet header the action that any diffserv-aware system should take on a marked packet. The diffserv architecture defines a set of DS codepoints and corresponding actions, or *forwarding behaviors*, for the IPQoS-enabled system and router to use. The IPQoS-enabled system marks the precedence bits of the DS field in the packet header with the DSCP. When a router receives a packet with a DSCP value, the router applies the forwarding behavior associated with that DSCP as the packet is released onto the network.

Note – The `dlcosmk` meter does not use the DSCP. Rather, `dclosmk` marks Ethernet frame headers with a CoS value. If you plan to configure IPQoS on a network that uses VLAN devices, refer to "Marker Module" on page 101.

Per-Hop Behaviors

In diffserv terminology, the forwarding behavior that is assigned to a DSCP is called the *per-hop behavior (PHB).*The PHB defines the forwarding precedence a marked packet receives in relation to other traffic on the diffserv-aware system. This precedence ultimately determines whether the IPQoS-enabled system or diffserv

router forwards or drops the marked packet. For a forwarded packet, each diffserv router that the packet encounters en route to its destination applies the same PHB. The exception is if another diffserv system changes the DSCP. For more information on PHBs, refer to "Using the `dscpmk` Marker for Forwarding Packets" on page 101.

The goal of a PHB is to provide a specified amount of network resources to a class of traffic on the contiguous network. You can achieve this goal in the QoS policy by defining DS codepoints that indicate the precedence levels for traffic classes when they leave the IPQoS-enabled system. Precedences range from high-precedence/low-drop probability to low-precedence/high-drop probability.

For example, your QoS policy can assign to one class of traffic a DSCP that guarantees a low-drop PHB. This traffic class then receives a low-drop precedence PHB from any diffserv-aware router, which guarantees bandwidth to packets of this class. You can add to the QoS policy other DSCPs that assign varying levels of precedence to other traffic classes. The lower-precedence packets are given bandwidth by diffserv systems in agreement with the priorities that are indicated in the packets' DSCPs.

IPQoS supports two types of forwarding behaviors, which are defined in the diffserv architecture, expedited forwarding and assured forwarding.

Expedited Forwarding (EF)

The *expedited forwarding (EF)* per-hop behavior assures that any traffic class with EF's related DSCP is given highest priority and is not queued. EF provides low loss, latency, and jitter. The recommended DS codepoint for EF is 101110. A packet that is marked with 101110 receives guaranteed low-drop precedence as the packet traverses diffserv-aware networks en route to its destination. Use the EF DSCP when assigning priority to customers or applications with a premium SLA.

Assured Forwarding (AF)

The *assured forwarding (AF)* per-hop behavior provides four different forwarding classes that you can assign to a packet. Each forwarding class provides three drop precedences, as shown in Table 6–2.

The various AF codepoints provide the ability to assign different levels of services to customers and applications. In the QoS policy you can prioritize traffic and services on your network when you plan the QoS policy. You can then assign different AF levels to the prioritized traffic.

Packet Forwarding in a Diffserv Environment

The following figure shows part of an intranet at a company with a partially diffserv-enabled environment. In this scenario, all hosts on networks 10.10.0.0 and 10.14.0.0 are IPQoS-enabled, and the local routers on both networks are diffserv-aware. However, the interim networks are not configured for diffserv.

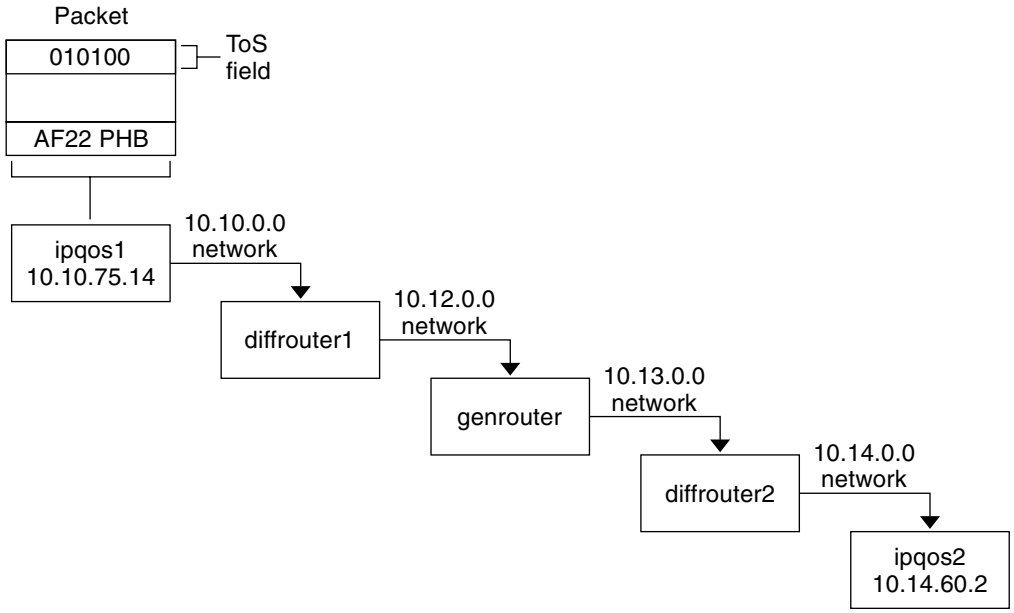

FIGURE 1–2 Packet Forwarding Across Diffserv-Aware Network Hops

The next steps trace the flow of the packet that is shown in the previous figure. The steps begin with the progress of a packet that originates at host `ipqos1` and continues through several hops to `ipqos2`.

1. The user on `ipqos1` runs the `ftp` command to access host `ipqos2`, three hops away.
2. `ipqos-1` applies its QoS policy to the resulting packet flow and successfully classifies the `ftp` traffic.

   The system administrator has created a class for all outgoing `ftp` traffic that originates on the local network 10.10.0.0. Traffic for the `ftp` class is assigned the AF22 per-hop behavior: class two, medium-drop precedence. A traffic flow rate of 2Mbits per second is configured for the `ftp` class.
3. `ipqos-1` meters the `ftp` flow to determine if it exceeds the committed rate of 2 Mbits per second.
4. The marker on `ipqos1` marks the DS fields in the outgoing `ftp` packets with the 010100 DSCP, corresponding to the AF22 PHB.
5. The router `diffrouter1` receives the `ftp` packets and checks the DSCP. If `diffrouter1` is congested, it drops the packets that are marked with AF22.
6. `ftp` traffic is forwarded to the next hop in agreement with the per-hop behavior that is configured for AF22 in `diffrouter1`'s files.

7. The `ftp` traffic traverses network 10.12.0.0 to `genrouter`, which is not diffserv-aware. As a result, the traffic receives “best-effort” forwarding behavior.
8. `genrouter` passes the `ftp` traffic to network 10.13.0.0, where the traffic is received by `diffrouter2`.
9. `diffrouter2` is diffserv-aware. Therefore, the router forwards the `ftp` packets to the network in agreement with the PHB that is defined in the router policy for AF22 packets.
10. `ipqos2` receives the `ftp` traffic and prompts the user on `ipqos1` for a user name and password.

CHAPTER 2

Planning for an IPQoS-Enabled Network (Tasks)

You can configure IPQoS on any system that runs the Solaris 9, 9/02 operating environment. The IPQoS system then works along with diffserv-aware routers to provide differentiated services and traffic management on an intranet.

This chapter contains planning tasks for adding IPQoS-enabled systems onto a diffserv-aware network. The following topics are covered.

General IPQoS Configuration Planning (Task Map)

Implementing differentiated services, including IPQoS, on a network requires extensive planning. You must consider not only the position and function of each IPQoS-enabled system, but also the systems' relationship to the router on the local network. The following table lists the major planning tasks for implementing IPQoS on your network.

TABLE 2–1 IPQoS Configuration Planning (Task Map)

Task	Description	For Instructions
1. Plan a diffserv network topology that incorporates IPQoS-enabled systems.	Learn about the various diffserv network topologies and determine the best solution for your site.	"Planning the Diffserv Network Topology" on page 32.

TABLE 2–1 IPQoS Configuration Planning (Task Map) *(Continued)*

Task	Description	For Instructions
2. Plan the different types of services to be offered by the IPQoS systems.	Organize the types of services that the network provides into service-level agreements.	"Planning the Quality-of-Service Policy" on page 35.
3. Plan the QoS policy for each IPQoS system.	Decide on the classes, metering, and accounting features that are needed to implement each SLA.	"Planning the Quality-of-Service Policy" on page 35.
4. If applicable, plan the policy for the diffserv router.	Decide any scheduling and queuing policies for the diffserv router that is used with the IPQoS systems.	Refer to router documentation for queuing and scheduling policies.

Planning the Diffserv Network Topology

To provide differentiated services for your network, you need at least one IPQoS-enabled system and a diffserv-aware router. You can expand this basic scenario in a variety of ways, as explained in this section.

Hardware Strategies for the Diffserv Network

Typically, customers run IPQoS on servers and server consolidations, such as the Sun Enterprise™ 10000 server. Conversely, you can also run IPQoS on desktop systems such as UltraSPARC systems, depending on the needs of your network. The following list describes possible systems for IPQoS configuration.

- Solaris systems that offer various services, such as Web servers and database servers
- Applications servers that offer email, FTP, or other popular network applications
- Web cache servers or proxy servers
- Network of IPQoS-enabled server farms that are managed by diffserv-aware load balancers
- Firewalls that manage traffic for a single heterogeneous network
- IPQoS systems that are part of a virtual LAN

You might introduce IPQoS systems into a network topology with already functioning diffserv-aware routers. If your router does not currently offer diffserv, consider the diffserv solutions that are offered by Cisco Systems, Juniper Networks, and other router manufacturers. If the local router does not implement diffserv, then the router passes marked packets on to the next hop without evaluating the marks.

IPQoS Network Topologies

This section contains diagrams that illustrate IPQoS strategies for various network needs.

IPQoS on Individual Hosts

The following figure shows a single network of IPQoS-enabled systems.

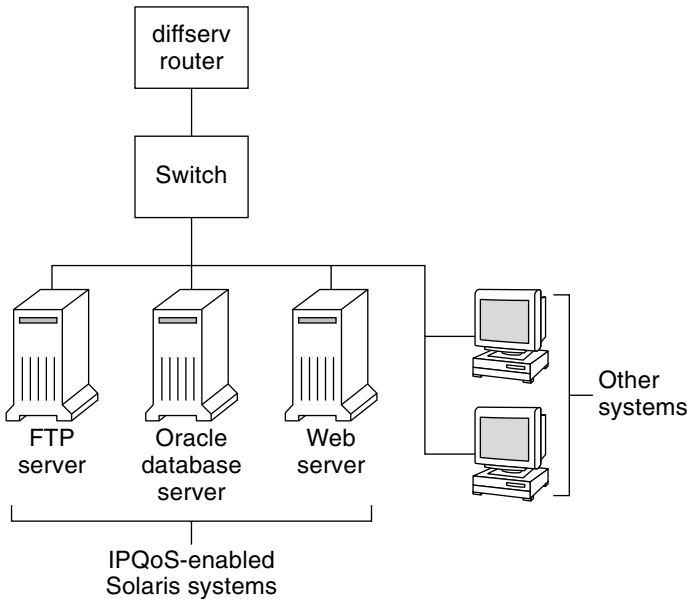

FIGURE 2–1 IPQoS Systems on a Network Segment

The network that is shown in the previous figure is but one segment of a corporate intranet. By enabling IPQoS on the application servers and web servers, you can control the rate at which each IPQoS system releases outgoing traffic onto the network stream. If you make the router diffserv-aware, you can further control incoming and outgoing traffic.

The examples in this guide use the IPQoS on an individual host scenario. For the example topology that is used throughout the guide, see Figure 2–4.

IPQoS on a Network of Server Farms

The following figure shows a network with several heterogeneous server farms.

In such a topology, the router is diffserv-aware, and therefore able to queue and rate both incoming and outgoing traffic. The load balancer is also diffserv-aware, and the server farms are IPQoS-enabled. The load balancer can provide additional filtering beyond the router by using selectors such as userID and projectID, which are included in the application data.

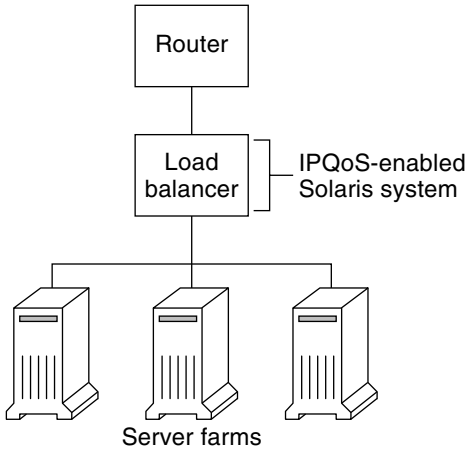

FIGURE 2–2 Network of IPQoS-Enabled Server Farms

This scenario provides flow control and traffic forwarding to manage congestion on the local network. The topology also prevents outgoing traffic from the server farms from overloading other portions of the intranet.

IPQoS on a Firewall

The following figure shows a segment of a corporate network that is secured from other segments by a firewall.

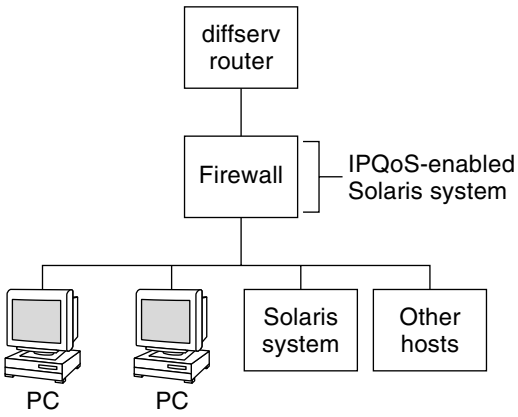

FIGURE 2–3 Network Protected by an IPQoS-Enabled Firewall

In this scenario, traffic flows into a diffserv-aware router where it is filtered and queued. All incoming traffic that is forwarded by the router then travels into the IPQoS-enabled firewall. In order to use IPQoS, the firewall must not bypass the IP forwarding stack.

The firewall's security policy determines whether incoming traffic is permitted to enter or depart the internal network. The QoS policy controls the service levels for incoming traffic that has passed the firewall. Depending on the QoS policy, outgoing traffic can also be marked with a forwarding behavior.

Planning the Quality-of-Service Policy

When you plan the quality-of-service policy, you must review, classify, and then prioritize the services that your network provides. You must also assess the amount of available bandwidth to determine the rate at which each traffic class is released onto the network.

QoS Policy Planning Aids

Gather information for planning the QoS policy in a format that includes the information that you need for the IPQoS configuration file. For example, you can use the following template to list the major information to be used in the IPQoS configuration file.

TABLE 2–2 QoS Organizational Template

Class	Priority	Filters	Selectors	Rate	Forwarding?	Accounting?
Class 1	1	Filter 1 Filter 3	Selector 1 Selector 2	Meter rates, depending on meter type	Marker drop precedence	Requires flow-accounting statistics
		Filter 2	Selector 1 Selector 2			
Class 2	2	Filter 1	Selector 1 Selector 2	Meter rates, depending on meter type	Marker drop precedence	Requires flow-accounting statistics
		Filter 2	Selector 1 Selector 2			

You can divide each major category to further define the QoS policy. The subsequent sections explain how to obtain information for the categories that are shown in the template.

The next task map lists the major tasks for planning a QoS policy.

TABLE 2–3 QoS Policy Planning (Task Map)

Task	Description	For Instructions
1. Design your network topology to support IPQoS.	Identify the hosts and routers on your network to provide differentiated services.	"How to Prepare a Network for IPQoS" on page 37
2. Define the classes into which services on your network must be divided.	Examine the types of services and SLAs that are offered by your site, and determine the discrete traffic classes into which these services fall.	"How to Define the Classes for Your QoS Policy" on page 38
3. Define filters for the classes.	Determine the best ways of separating traffic of a particular class from the network traffic flow.	"How to Define Filters in the QoS Policy" on page 40

TABLE 2–3 QoS Policy Planning (Task Map) *(Continued)*

Task	Description	For Instructions
4. Define flow-control rates for measuring traffic as it leaves the IPQoS system.	Determine acceptable flow rates for each class of traffic.	"How to Plan Flow Control" on page 42
5. Define DS codepoints or user priority values to be used in the QoS policy.	Plan a scheme to determine the forwarding behavior that is assigned to a traffic flow when the flow is handled by the router or switch.	"How to Plan Forwarding Behavior" on page 44
6. If applicable, set up a statistics-monitoring plan for traffic flows on the network.	Evaluate the traffic classes to determine which traffic flows must be monitored for accounting or statistical purposes.	"How to Plan for Flow Accounting" on page 47

Note – The rest of this section explains how to plan the QoS policy of an IPQoS-enabled system. To plan the QoS policy for the diffserv router, refer to router documentation and the router manufacturers' Web sites.

▼ How to Prepare a Network for IPQoS

The following procedure lists general planning tasks to do before you create the QoS policy.

1. **Review your network topology and plan a strategy that uses IPQoS systems and diffserv routers.**

   For topology examples, see "Planning the Diffserv Network Topology" on page 32.

2. **Identify the hosts in the topology that require IPQoS or that might become good candidates for IPQoS service.**

3. **Determine which IPQoS-enabled systems could use the same QoS policy.**

   For example, if you plan to enable IPQoS on all hosts on the network, identify any hosts that could use the same QoS policy. Each IPQoS-enabled system must have a local QoS policy, which is implemented in its IPQoS configuration file. However, you can create one IPQoS configuration file to be used by a range of systems. You can then copy the configuration file to every system with the same QoS requirements.

4. **Review and perform any planning tasks that are required by the diffserv router on your network.**

   Refer to the router documentation and router manufacturer's Web site for details.

▼ How to Define the Classes for Your QoS Policy

The first step in defining the QoS policy is organizing traffic flows into classes. You do not need to create classes for every type of traffic on a diffserv network. Moreover, depending on your network topology, you might have to create different QoS policies for each IPQoS-enabled system.

Note – For an overview of classes, see "Classes" on page 23.

The next procedure assumes that you have determined which systems on your network are to be IPQoS-enabled, as identified in "How to Prepare a Network for IPQoS" on page 37.

1. **Create a table for organizing the QoS policy.**

   For suggestions, refer to Table 2–2.

2. **Perform the remaining steps for every QoS policy that is on your network.**

3. **Define the classes to be used in the QoS policy.**

   The following questions are a guideline for analyzing network traffic for possible class definitions.

   - **Does your company offer service-level agreements to customers?**

     If yes, then evaluate the relative priority levels of the SLAs that your company offers to customers. The same applications might be offered to customers who are guaranteed different priority levels.

     For example, your company might offer web site hosting to each customer, which indicates that you need to define a class for each customer web site. Moreover, one SLA might provide a premium web site as one service level, while another SLA offers a "best-effort" personal web site to discount customers. This factor indicates not only different web site classes but also potentially different per-hop behaviors that are assigned to the web site classes.

   - **Does the IPQoS system offer popular applications that might need flow control?**

     You can improve network performance by enabling IPQoS on servers that offer popular applications that generate a lot of traffic. Common examples are electronic mail, network news, and FTP. Consider creating separate classes for incoming and outgoing traffic for each service type, where applicable. For example, you might create a `mail-in` class and a `mail-out` class to the QoS policy for a mail server.

- **Does your network run certain applications that require highest-priority forwarding behaviors?**

  Any critical applications that require the highest-priority forwarding behaviors must be given special treatment by the router. Typical examples are streaming video and streaming audio.

  Define incoming classes and outgoing classes for these high-priority applications. Then add the classes to the QoS policies of both the IPQoS-enabled system that serves the applications and the diffserv router.

- **Does your network experience traffic flows that must be controlled because they consume large amounts of bandwidth?**

  Use `netstat`, `snoop`, and other network monitoring utilities to discover the types of traffic that are causing problems on the network. Review the classes you have created thus far, and then create new classes for any undefined problem traffic category. If you have already defined classes for a category of problem traffic, then define rates for the meter to control the problem traffic.

  Create classes for the problem traffic on every IPQoS-enabled system on the network. Each IPQoS system can then handle any problem traffic it receives by limiting the rate at which the traffic flow is released onto the network. Be sure also to define these problem classes in the QoS policy on the diffserv router. The router can then queue and schedule the problem flows as configured in its QoS policy.

- **Do you need to obtain statistics on certain types of traffic?**

  A quick review of an SLA can indicate which types of customer traffic require accounting. If your site does offer SLAs, you probably have already created classes for traffic that requires accounting. You might also define classes to add statistics taking to traffic flows that you are monitoring or to which you are restricting access for security purposes.

4. **List the classes you have defined in the organizational table.**

5. **Assign a priority level to each class.**

   For example, have priority level 1 represent the highest-priority class, and assign descending-level priorities to the remaining classes. The priority level you assign is for organizational purposes only and is not actually used by IPQoS. Moreover, you can assign the same priority to more than one class, if appropriate for your QoS policy.

   For information about the importance of prioritizing classes, refer to the next section.

6. **When you finish defining classes, you next define filters for each class, as explained in "How to Define Filters in the QoS Policy" on page 40.**

Prioritizing the Classes

As you create classes, it quickly becomes apparent which classes have highest priority, medium priority, and best-effort priority. Prioritizing the classes becomes particularly important when you assign per-hop behaviors to outgoing traffic, as explained in "How to Plan Forwarding Behavior" on page 44.

In addition to assigning a PHB to a class, you can also define a priority selector in a filter for the class. The priority selector is active on the IPQoS-enabled host only. Suppose several classes with equal rates and identical DSCPs sometimes compete for bandwidth as they leave the IPQoS system. The priority selector in each class can further order the level of service that is given to the otherwise identically valued classes.

▼ How to Define Filters in the QoS Policy

You create filters to identify packet flows as members of a particular class. Each filter contains selectors, which define the criteria for evaluating a packet flow. The IPQoS-enabled system then uses the criteria in the selectors to extract packets from a traffic flow and associate them with a class. (For an introduction to filters, see "Filters" on page 24.)

Before you can perform the next steps, you should have completed the procedure "How to Define the Classes for Your QoS Policy" on page 38.

1. **Create at least one filter for each class in the QoS organizational table that you created in "How to Define the Classes for Your QoS Policy" on page 38.**

   Consider creating separate filters for incoming and outgoing traffic for each class, where applicable. For example, add an `ftp-in` filter and an `ftp-out` filter to the QoS policy of an IPQoS-enabled FTP server. Then you can define an appropriate `direction` selector in addition to the basic selectors.

2. **Define at least one selector for each filter in a class.**

   The following table lists the most commonly used selectors. The first five selectors represent the IPQoS 5–tuple, which the IPQoS system uses to identify packets as members of a flow. For a complete list of selectors, see Table 6–1.

Note – Be judicious in your choice of selectors. Use only as many selectors as you need to extract packets for a class. The more selectors you define, the greater the impact on IPQoS performance.

TABLE 2–4 Common IPQoS Selectors

Name	Definition
`saddr`	Source address.
`daddr`	Destination address.
`sport`	Source port number. You can use a well-known port number, as defined in `/etc/services`, or user-defined port number.

TABLE 2–4 Common IPQoS Selectors *(Continued)*

Name	Definition
`dport`	Destination port number.
`protocol`	IP protocol number or protocol name that is assigned to the traffic flow type in `/etc/protocols`.
`ip_version`	Addressing style to use. Use either V4 or V6. V4 is the default.
`dsfield`	Contents of the DS field, that is, the DS codepoint. Use this selector for extracting incoming packets that are already marked with a particular DSCP.
`priority`	Priority level that is assigned to the class. For more information, see "Prioritizing the Classes" on page 39.
`user`	Either the UNIX userID or user name that is used when the upper-level application is executed.
`projid`	Project ID that is used when the upper-level application is executed.
`direction`	Direction of traffic flow. Value is either LOCAL_IN, LOCAL_OUT, FWD_IN, or FWD_OUT.

Use the template that was introduced in Table 2–2 to fill in filters for the classes you defined.

Class	Priority	Filters	Selectors
ftp-traffic	4	ftp-out	`saddr 10.190.17.44`
			`daddr 10.100.10.53`
			`sport 21`
			`direction LOCAL_OUT`

Where to Go From Here

Task	For Information
Define a flow-control scheme	"How to Plan Flow Control" on page 42
Define forwarding behaviors for flows as they return to the network stream	"How to Plan Forwarding Behavior" on page 44
Plan for flow accounting of certain types of traffic	"How to Plan for Flow Accounting" on page 47

Task	For Information
Add more classes to the QoS policy	"How to Define the Classes for Your QoS Policy" on page 38
Add more filters to the QoS policy	"How to Define Filters in the QoS Policy" on page 40

▼ How to Plan Flow Control

Flow control involves measuring traffic flow for a class and then releasing packets onto the network at a defined rate. When you plan flow control, you define parameters to be used by the IPQoS metering module. The meter determines the rate at which traffic is released onto the network. For an introduction to the meter, see "Meter (`tokenmt` and `tswtclmt`) Overview" on page 24.

The next procedure assumes that you have defined filters and selectors, as described in "How to Define Filters in the QoS Policy" on page 40.

1. **Determine the maximum bandwidth for your network.**
2. **Review any SLAs that are supported on your network to identify customers and the type of service that is guaranteed to each customer.**

   Because the SLA guarantees a certain level of service to a customer, you might need to meter certain traffic classes that are generated by the customer.
3. **Review the list of classes that you created in "How to Define the Classes for Your QoS Policy" on page 38.**

   Determine if any classes other than those that are associated with SLAs need to be metered.

   Suppose the IPQoS system runs an application that generates a high level of traffic. After you classify the application's traffic, meter the flows to control the rate at which the packets of the flow return to the network.

   Note – Not all classes need to be metered. Remember this guideline as you review your list of classes.
4. **Refine your list of classes to be metered by determining which filters in the class select traffic that needs flow control.**

   Classes that have more than one filter might require metering for only one filter. Suppose you define filters for incoming and outgoing traffic of a certain class. You might conclude that only traffic in one of the directions requires flow control.
5. **Choose a meter module for each class to be flow controlled.**

   Add the module name to the meter column in your organizational table.

6. **Add the rates for each class to be metered to the organizational table.**

   If you use the `tokenmt` module, you need to define the following rates in bits per second.

   - Committed rate
   - Peak rate

   If sufficient to meter a particular class, you can define only the committed rate and committed burst for `tokenmt`.

   If needed, you can also define the following rates.

   - **Committed burst**
   - **Peak burst**

   For a complete definition of `tokenmt` rates, refer to "Configuring `tokenmt` as a Two-Rate Meter" on page 99. You can also find more detailed information in the `tokenmt`(7ipp) man page.

   If you use the `tswtclmt` module, you need to define the following rates in bits per second.

   - Committed rate
   - Peak rate

   You can also define the window size in milliseconds. These rates are defined in "`tswtclmt` Metering Module" on page 100 and in the `twstclmt`(7ipp) man page.

7. **Add traffic conformance outcomes for the metered traffic.**

   The outcomes for both metering modules are green, red, and yellow. Add to your QoS organizational table the traffic conformance outcomes that apply to the rates you define. Outcomes for the meters are fully explained in "Meter Module" on page 98.

   You need to determine what action should be taken on traffic that conforms, or does not conform, to the committed rate. Often this action is to mark the packet header with a per-hop behavior, but not always. One acceptable action for green-level traffic could be to continue processing while traffic flows do not exceed the committed rate. Another action could be to drop packets of the class if flows exceed peak rate.

   The next table shows meter entries for a class of email traffic. The network on which the IPQoS system is located has a total bandwidth of 100 Mbps, or 100000000 bits per second. The QoS policy assigns a low priority and best-effort forwarding behavior for the email class.

TABLE 2–5 Example QoS Policy With Meters Defined

Class	Priority	Filters	Selectors	Rate
`email`	8	`mail_in`	`daddr 10.50.50.5`	
			`dport imap`	
			`direction LOCAL_IN`	

TABLE 2–5 Example QoS Policy With Meters Defined *(Continued)*

Class	Priority	Filters	Selectors	Rate
		`mail_out`	`saddr` 10.50.50.5	meter=`tokenmt`
			`sport imap`	committed rate=5000000
			`direction LOCAL_OUT`	committed burst =5000000
				peak rate =10000000
				peak burst=1000000
				green precedence=continue processing
				yellow precedence=mark yellow PHB
				red precedence=drop

Where to Go From Here

Task	For Information
Define forwarding behaviors for flows as they return to the network stream	"How to Plan Forwarding Behavior" on page 44
Plan for flow accounting of certain types of traffic	"How to Plan for Flow Accounting" on page 47
Add more classes to the QoS policy	"How to Define the Classes for Your QoS Policy" on page 38
Add more filters to the QoS policy	"How to Define Filters in the QoS Policy" on page 40
Define another flow-control scheme	"How to Plan Flow Control" on page 42
Create an IPQoS configuration file	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56

▼ How to Plan Forwarding Behavior

Forwarding behavior determines the priority and drop precedence of traffic flows that are about to be forwarded onto the network. You can choose two major forwarding behaviors: prioritizing the flows of a class in relationship to other traffic classes or dropping the flows entirely.

The diffserv model uses the marker to assign the chosen forwarding behavior to traffic flows. IPQoS offers the following marker modules.

- `dscpmk`, which you use to mark the DS field of an IP packet with a DS codepoint.
- `dlcosmk`, which you use to mark the VLAN tag of a datagram with a class of service (CoS) value.

Note – The suggestions in this section refer specifically to IP packets. If your IPQoS system includes a VLAN device, you can use the `dlcosmk` marker to mark forwarding behaviors for datagrams. For more information, refer to "Using the `dlcosmk` Marker With VLAN Devices" on page 103.

To prioritize IP traffic, you need to assign a DS codepoint to each packet. The `dscpmk` marker marks the DS field of the packet with the DS codepoint. You choose the DS codepoint for a class from a group of well-known codepoints that are associated with the forwarding behavior type. These well-known codepoints are 46 (101110) for the EF PHB and a range of codepoints for the AF PHB. For overview information on DS codepoints and forwarding, refer to "Traffic Forwarding on an IPQoS-Enabled Network" on page 27.

The next steps assume that you have defined classes and filters for the QoS policy. Though you often use the meter with the marker to control traffic, you can use the marker alone to define a forwarding behavior.

1. **Review the classes that you have created thus far and the priorities that you have assigned to them.**

   Not all traffic classes need to be marked.

2. **Assign the EF per-hop behavior to the class with the highest priority.**

   The EF PHB guarantees that packets with the EF DS codepoint 46 (101110) are released onto the network before packets that are marked with any AF PHBs. Use the EF PHB for your highest-priority traffic. For more information about EF, refer to "Expedited Forwarding (EF) PHB" on page 102.

3. **Assign forwarding behaviors to classes that have traffic to be metered.**

   Traffic is generally metered for the following reasons:

   - An SLA guarantees packets of this class greater or lesser service when the network is heavily used.
   - A class with a lower priority might have a tendency to flood the network.

   You use the marker with the meter to provide differentiated services and bandwidth management to these classes. For example, the following table shows a portion of a QoS policy that defines a class for a popular games application that generates a high level of traffic.

Class	Priority	Filters	Selectors	Rate	Forwarding?
games_app	9	games_in	sport 6080		
		games_out	dport 6081	meter=tokenmt committed rate=5000000 committed burst =5000000 peak rate =10000000 peak burst=15000000 green precedence=continue processing yellow precedence=mark yellow PHB red precedence=drop	green =AF31 yellow=AF42 red=drop

The forwarding behaviors assign low-priority DS codepoints to games_app traffic that conforms to its committed rate or is below the peak rate. When games_app traffic exceeds peak rate, the QoS policy indicates that packets from games_app are to be dropped. A list of all AF codepoints is in table Table 6–2.

4. **Assign DS codepoints to the remaining classes in agreement with the priorities that you have assigned to them.**

Where to Go From Here

Task	For Information
Plan for flow accounting of certain types of traffic	"How to Plan for Flow Accounting" on page 47
Add more classes to the QoS policy	"How to Define the Classes for Your QoS Policy" on page 38
Add more filters to the QoS policy	"How to Define Filters in the QoS Policy" on page 40
Define a flow-control scheme	"How to Plan Flow Control" on page 42

Task	For Information
Define additional forwarding behaviors for flows as they return to the network stream	"How to Plan Forwarding Behavior" on page 44
Create an IPQoS configuration file	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56

▼ How to Plan for Flow Accounting

You use the IPQoS `flowacct` module to keep track of traffic flows for billing or network management purposes. Use the following procedure to determine if your QoS policy should include flow accounting.

1. **Does your company offer SLAs to customers?**

   If the answer is yes, then you should use flow accounting. Review the SLAs to determine what types of network traffic your company wants to bill customers to use. Then review your QoS policy to determine which classes select traffic to be billed.

2. **Are there applications that might need monitoring or testing to avoid network problems?**

   If the answer is yes, consider using flow accounting to observe the behavior of these applications. Review your QoS policy to determine the classes you have assigned to traffic that requires monitoring.

3. **Mark Y in the flow-accounting column for each class that requires flow accounting in your QoS planning template.**

Where to Go From Here?

Task	For Information
Add more classes to the QoS policy	"How to Define the Classes for Your QoS Policy" on page 38
Add more filters to the QoS policy	"How to Define Filters in the QoS Policy" on page 40
Define a flow-control scheme	"How to Plan Flow Control" on page 42
Define forwarding behaviors for flows as they return to the network stream	"How to Plan Forwarding Behavior" on page 44
Plan for additional flow accounting of certain types of traffic	"How to Plan for Flow Accounting" on page 47

Task	For Information
Create the IPQoS configuration file	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56

Introducing the IPQoS Configuration Example

Tasks in the remaining chapters of the guide use the example IPQoS configuration that is introduced in this section. The example shows the differentiated services solution that is implemented on the public intranet of a fictitious service provider, which is called BigISP. BigISP offers services to two types of users: large companies that reach BigISP through leased lines, and individuals who dial in from modems to BigISP.

Example—IPQoS Topology

The following figure shows the network topology that is used for BigISP's public intranet.

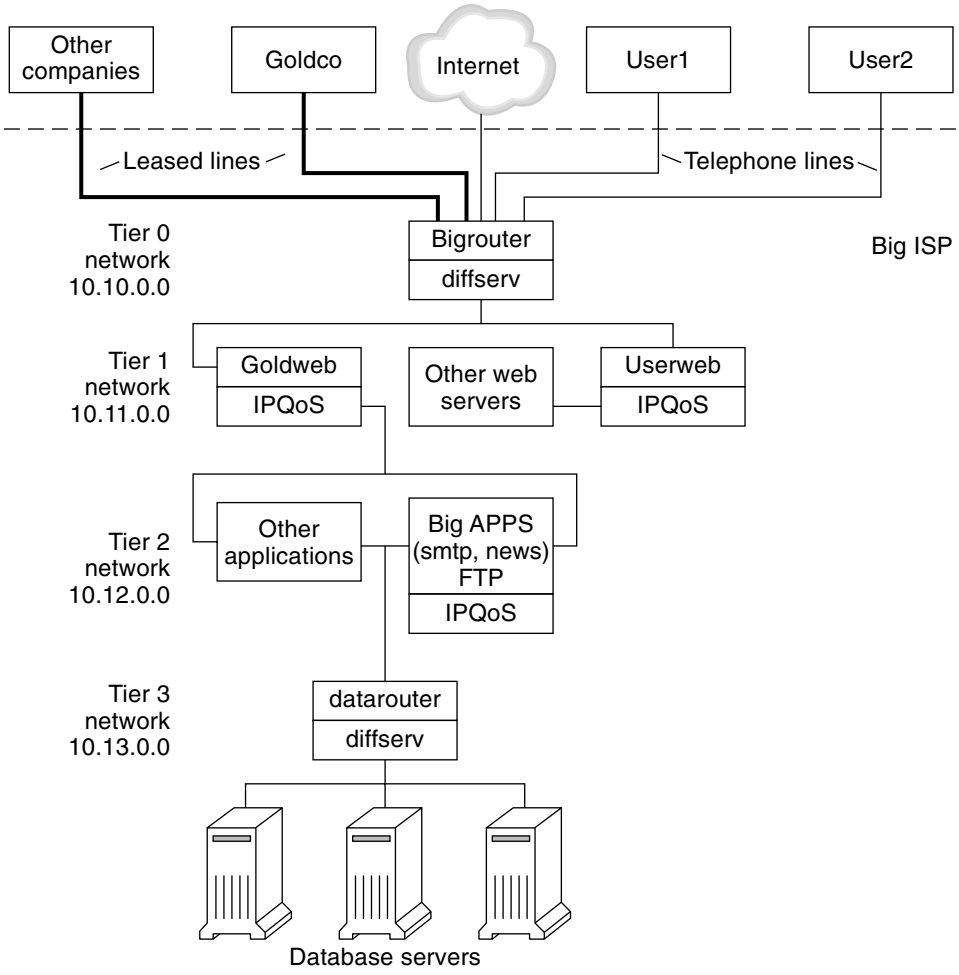

FIGURE 2–4 IPQoS Example Topology

BigISP has implemented four tiers in its public intranet, as shown in the previous figure.

- **Tier 0** – Network 10.10.0.0 includes a large diffserv router that is called `Bigrouter`, which has both external and internal interfaces. Several companies, including a large organization called Goldco, have rented leased-line services that terminate at `Bigrouter`. Tier 0 also handles individual customers who call over telephone lines or ISDN.
- **Tier 1** – Network 10.11.0.0 provides web services. The `Goldweb` server hosts the web site which was purchased by Goldco as part of the premium service that it has purchased from BigISP. The server `Userweb` hosts small web sites that were purchased by individual customers. Both `Goldweb` and `Userweb` are IPQoS

enabled.

- **Tier 2** – Network 10.12.0.0 provides applications for all customers to use. `BigApps`, one of the application servers, is IPQoS-enabled. `BigApps` provides SMTP, News, and FTP services.
- **Tier 3** – Network 10.13.0.0 houses large database servers. Access to Tier 3 is controlled by `datarouter`, a diffserv router.

CHAPTER 3

Creating the IPQoS Configuration File (Tasks)

This chapter shows how to create IPQoS configuration files and use the `ipqosconf` utility. Topics that are covered in the chapter include the following.

- "Defining a QoS Policy in the IPQoS Configuration File (Task Map)" on page 51
- "Tools for Creating a QoS Policy" on page 52
- "Creating IPQoS Configuration Files for Web Servers" on page 53
- "Creating an IPQoS Configuration File for an Application Server" on page 68
- "Providing Differentiated Services on a Router" on page 79

The text assumes that you have a complete QoS policy, and are ready to use this policy as the basis for the IPQoS configuration file. For instructions on QoS policy planning, refer to "Planning the Quality-of-Service Policy" on page 35.

Defining a QoS Policy in the IPQoS Configuration File (Task Map)

The next table lists the general tasks for creating an IPQoS configuration file.

TABLE 3–1 Creating an IPQoS Configuration File (Task Map)

Task	Description	Instructions
1. Plan your IPQoS-enabled network configuration.	Decide which systems on the local network should become IPQoS-enabled.	"How to Prepare a Network for IPQoS" on page 37

TABLE 3–1 Creating an IPQoS Configuration File (Task Map) *(Continued)*

Task	Description	Instructions
2. Plan the QoS policy for IPQoS systems on your network.	Identify traffic flows as distinct classes of service and determine which flows require traffic management.	"Planning the Quality-of-Service Policy" on page 35
3. Begin the IPQoS configuration file and define its first action.	Create the IPQoS file, invoke the IP classifier, and define a class for processing.	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56
4. Create filters for a class.	Add the filters that govern which traffic is selected and organized into a class.	"How to Define Filters in the IPQoS Configuration File" on page 58
5. Add more classes and filters to the IPQoS configuration file.	Create more classes and filters to be processed by the IP classifier.	"How to Create an IPQoS Configuration File for a Best-Effort Web Server" on page 65
6. Add an action statement with parameters that configure the metering modules.	If the QoS policy calls for flow control, assign flow-control rates and conformance levels to the meter.	"How to Configure Flow Control in the IPQoS Configuration File" on page 75
7. Add an action statement with parameters that configure the marker.	If the QoS policy calls for differentiated forwarding behaviors, define how traffic classes are to be forwarded.	"How to Define Traffic Forwarding in the IPQoS Configuration File" on page 60
8. Add an action statement with parameters that configure the flow-accounting module.	If the QoS policy calls for statistics taking on traffic flows, define how these accounting statistics are to be taken.	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63
9. Apply the IPQoS configuration file.	Add the content of a specified IPQoS configuration file into the appropriate kernel modules.	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82
10. Configure forwarding behaviors in the router files.	If any IPQoS configuration files on the network define forwarding behaviors, add the resulting DSCPs to the appropriate scheduling files on the router.	"How to Configure a Router on an IPQoS-Enabled Network" on page 79

Tools for Creating a QoS Policy

The QoS policy for your network resides in the IPQoS configuration file. You create this configuration file with a text editor and provide the file as an argument to `ipqosconf`, the IPQoS configuration utility. When you instruct `ipqosconf` to apply the policy that is defined in your configuration file, the policy is written into the kernel

IPQoS system. For detailed information about the `ipqosconf` command, refer to the `ipqosconf`(1m) man page. For instructions on the use of `ipqosconf`, refer to "How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82.

IPQoS Configuration File

An IPQoS configuration file consists of a tree of *action statements* that implement the QoS policy that you defined in "Planning the Quality-of-Service Policy" on page 35. The IPQoS configuration file configures the IPQoS modules. Each action statement contains a set of *classes*, *filters*, or *parameters* to be processed by the module that is called in the action statement.

For the complete syntax of the IPQoS configuration file, refer to Example 6–3 and the `ipqosconf`(1m) man page.

Configuring the IPQoS Sample Topology

The tasks in this chapter explain how to create IPQoS configuration files for three IPQoS-enabled systems. These systems are part of the network topology of the company BigISP, which was introduced in Figure 2–4.

- `Goldweb` – A web server that hosts web sites for customers who have purchased premium-level SLAs.
- `Userweb` – A less-powerful web server that hosts personal web sites for home users who have purchased "best-effort" SLAs.
- `BigAPPS` – An application server that serves mail, network news, and FTP to both gold-level and best-effort customers.

The three configuration files illustrate the most common IPQoS configurations. You might use them as templates for your own IPQoS implementation.

Creating IPQoS Configuration Files for Web Servers

This section introduces IPQoS configuration file creation by showing how to create a configuration file for a premium web server. The section then shows how to configure a completely different level of service in another configuration file for a server that hosts personal web sites. Both servers are part of the network example that is shown in Figure 2–4.

The following configuration file defines IPQoS activities for the `Goldweb` server, which hosts the web site for Goldco, the company that has purchased a premium SLA.

EXAMPLE 3–1 Sample IPQoS Configuration File for a Premium Web Server

```
fmt_version 1.0

action {
    module ipgpc
    name ipgpc.classify
    params {
        global_stats TRUE
    }
    class {
        name goldweb
        next_action markAF11
        enable_stats FALSE
    }
    class {
        name video
        next_action markEF
        enable_stats FALSE
    }
    filter {
        name webout
        sport 80
        direction LOCAL_OUT
        class goldweb
    }
    filter {
        name videoout
        sport videosrv
        direction LOCAL_OUT
        class video
    }
}
action {
    module dscpmk
    name markAF11
    params {
        global_stats FALSE
        dscp_map{0-63:10}
        next_action continue
    }
}
action {
    module dscpmk
    name markEF
    params {
        global_stats TRUE
        dscp_map{0-63:46}
        next_action acct
    }
}
action {
    module flowacct
    name acct
    params {
```

EXAMPLE 3–1 Sample IPQoS Configuration File for a Premium Web Server *(Continued)*

```
        enable_stats TRUE
        timer 10000
        timeout 10000
        max_limit 2048
    }
}
```

The following configuration file defines IPQoS activities on `Userweb`, which hosts web sites for individuals with low-priced, or *best-effort*, SLAs. This SLA level guarantees the best service that can be delivered to best-effort customers after the IPQoS system handles traffic from customers with more expensive SLAs.

EXAMPLE 3–2 Sample Configuration for a Best-Effort Web Server

```
fmt_version 1.0

action {
    module ipgpc
    name ipgpc.classify
    params {
        global_stats TRUE
    }
    class {
        name Userweb
        next_action markAF12
        enable_stats FALSE
    }
    filter {
        name webout
        sport 80
        direction LOCAL_OUT
        class Userweb
   }
}
action {
    module dscpmk
    name markAF12
    params {
        global_stats FALSE
        dscp_map{0-63:12}
        next_action continue
    }
}
```

▼ How to Begin the IPQoS Configuration File and Define Traffic Classes

You can create your first IPQoS configuration file in whatever directory is easiest for you to maintain and use. The tasks in this chapter use the directory `/var/ipqos` as the location for IPQoS configuration files. The next procedure builds the initial segment of the IPQoS configuration file that is introduced in Example 3–1.

Note – As you create the IPQoS configuration file, be very careful to start and end each action statement and clause with curly braces ({ }). For an example of the use of braces, see Example 3–1.

---

1. **Log in to the premium web server, and create a new IPQoS configuration file with a `.qos` extension.**

   Every IPQoS configuration file must start with the version number `fmt_version 1.0` as its first uncommented line.

2. **Follow the opening parameter with the initial action statement, which configures the generic IP classifier `ipgpc`.**

   This initial action begins the tree of action statements that compose the IPQoS configuration file. For example, the `/var/ipqos/Goldweb.qos` file begins with the initial action statement to call the `ipgpc` classifier.

```
fmt_version 1.0

action {
    module ipgpc
    name ipgpc.classify
```

Entry	Description
`fmt_version 1.0`	Begins the IPQoS configuration file
`action {` `module ipgpc`	Configures the `ipgpc` classifier as the first action in the configuration file
`name ipgpc.classify`	Defines the name of the classifier action statement, which must always be `ipgpc.classify`

   For detailed syntactical information about `action` statements, refer to "action Statement" on page 109 and the`ipqosconf`(1M) man page.

3. **Add a `params` clause with the statistics parameter `global_stats`.**

```
params {
          global_stats TRUE
   }
```

The parameter `global_stats TRUE` in the `ipgpc.classify` action enables statistics taking for that action. `global_stats TRUE` also enables per-class statistics taking wherever a class clause definition specifies `enable_stats TRUE`.

Turning on statistics impacts performance. You might want to take statistics on a new IPQoS configuration file to verify that IPQoS works properly. Later, you can turn off statistics collection by changing the argument to `global_stats` to `FALSE`.

Global statistics are but one type of parameter you can define in a `params` clause. For syntactical and other details about `params` clauses, refer to "Params Clause" on page 111 and the `ipqosconf` man page.

4. **Define a class that identifies traffic that is bound for the premium server.**

```
class {
        name goldweb
        next_action markAF11
        enable_stats FALSE
    }
```

The previous statement is called a *class clause*. A class clause has the following contents.

Entry	Description
`name goldweb`	Creates the class `goldweb` to identify traffic that is bound for the Goldweb server.
`next_action markF11`	Instructs the `ipgpc` module to pass packets of the `goldweb` class to the `markAF11` action statement. The `markAF11` action statement calls the `dscpmk` marker.
`enable_stats FALSE`	Enables statistics taking for the `goldweb` class. However, because the value of `enable_stats` is `FALSE`, statistics for this class are not turned on.

For detailed information about the syntax of the class clause, see "Class Clause" on page 110 and the `ipqosconf`(1M) man page.

5. **Define a class that identifies an application that must have highest-priority forwarding.**

```
 class {
        name video
        next_action markEF
        enable_stats FALSE
    }
```

Entry	Description
name video	Creates the class `video` to identify streaming video traffic that is outgoing from Goldweb.
next_action markEF	Instructs the `ipgpc` module to pass packets of the `video` class to the `markEF` action statement after `ipgpc` completes processing. The `markEF` action statement calls the `dscpmk` marker.
enable_stats FALSE	Enables statistics taking for the `video` class. However, because the value of `enable_stats` is `FALSE`, statistics for this class are not turned on.

Where to Go From Here

Task	For Information
Define filters for the class you just created	"How to Define Filters in the IPQoS Configuration File" on page 58
Create another class clause for the configuration file	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56

▼ How to Define Filters in the IPQoS Configuration File

The next procedure shows how to define filters for a class in the IPQoS configuration file. The procedure assumes that you have already begun the file and have defined classes. The steps continue building the `/var/ipqos/Goldweb.qos` file that is introduced in "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.

Note – As you create the IPQoS configuration file, be very careful to start and end each class clause and filter clause with curly braces ({ }). For an example of the use of braces, use Example 3–1.

1. **Open the IPQoS configuration file, and locate the end of the last class that you defined.**

   For example, on the IPQoS-enabled server `Goldweb` you would start after the following class clause in `/var/ipqos/Goldweb.qos`.

```
class {
        name video
        next_action markEF
```

```
    enable_stats FALSE
}
```

2. **Define a filter clause to select outgoing traffic from the IPQoS system.**

```
filter {
    name webout
    sport 80
    direction LOCAL_OUT
    class goldweb
}
```

Entry	Description
name webout	Gives the name `webout` to the filter
sport 80	Selects traffic with a source port of 80, the well-known port for HTTP (Web) traffic
direction LOCAL_OUT	Further selects traffic that is outgoing from the local system
class goldweb	Identifies the class to which the filter belongs, in this instance, class `goldweb`

For syntactical and detailed information about the filter clause in the IPQoS configuration file, refer to "Filter Clause" on page 111.

3. **Define a filter clause to select streaming video traffic on the IPQoS system.**

```
filter {
    name videoout
    sport videosrv
    direction LOCAL_OUT
    class video
}
```

Entry	Description
name videoout	Gives the name `videoout` to the filter
sport videosrv	Selects traffic with a source port of `videosrv`, a previously defined port for the streaming video application on this system
direction LOCAL_OUT	Further selects traffic that is outgoing from the local system
class video	Identifies the class to which the filter belongs, in this instance, class `video`

Where to Go From Here

Task	For Information
Define forwarding behaviors for the marker modules	"How to Define Traffic Forwarding in the IPQoS Configuration File" on page 60
Define flow-control parameters for the metering modules	"How to Configure Flow Control in the IPQoS Configuration File" on page 75
Activate the IPQoS configuration file	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82
Define additional filters	"How to Define Filters in the IPQoS Configuration File" on page 58
Create classes for traffic flows from applications	"How to Configure the IPQoS Configuration File for an Applications Server" on page 70

▼ How to Define Traffic Forwarding in the IPQoS Configuration File

The next procedure shows how to define traffic forwarding by adding per-hop behaviors for a class into the IPQoS configuration file. The procedure assumes that you have an existing IPQoS configuration file with already-defined classes and filters. The steps continue building the `/var/ipqos/Goldweb.qos` file from Example 3–1.

Note – The procedure shows how to configure traffic forwarding by using the `dscpmk` marker module. For information about traffic forwarding on VLAN systems by using the `dlclosmk` marker, refer to "Using the `dlcosmk` Marker With VLAN Devices" on page 103.

---

1. **Open the IPQoS configuration file, and locate the end of the last filter you defined.**

   For example, on the IPQoS-enabled server Goldweb, you would start after the following filter clause in `/var/ipqos/Goldweb.qos`.

```
filter {
        name videoout
        sport videosrv
        direction LOCAL_OUT
        class video
    }
}
```

   Note that this filter is at the end of the `ipgpc` classifier action statement. Therefore, you need a closing brace to terminate the filter and a second closing brace to terminate the action statement.

2. **Invoke the marker with the following action statement.**

```
action {
    module dscpmk
    name markAF11
```

Entry	Description
module dscpmk	Calls the marker module `dscpmk`
name markAF11	Gives the name `markAF11` to the action statement

The previously defined class `goldweb` includes a `next_action markAF11` statement. This statement sends traffic flows to the `markAF11` action statement after the classifier concludes processing.

3. **Define actions for the marker to take on the traffic flow.**

```
    params {
        global_stats FALSE
        dscp_map{0-63:10}
        next_action continue
    }
}
```

Entry	Description
global_stats FALSE	Enables statistics taking for the `markAF11` marker action statement. However, because the value of `enable_stats` is `FALSE`, statistics are not turned on.
dscp_map{0-63:10}	Assigns a DS codepoint of `10` to the packet headers of the traffic class `goldweb`, which is currently being processed by the marker.
next_action continue	Indicates that no further processing is required on packets of the traffic class goldweb, and that these packets can return to the network stream.

The DS codepoint 10 instructs the marker to set all entries in the `dscp` map to the decimal value 10 (binary 001010). This codepoint indicates that packets of the `goldweb` traffic class are subject to the AF11 per-hop behavior. AF11 guarantees that all packets with DS codepoint 10 receive a low-drop, high-priority service. Thus, outgoing traffic for premium customers on Goldweb is given the highest priority available for the Assured Forwarding PHB. For a table of possible DS codepoints for AF, refer to Table 6–2.

4. **Start another marker action statement.**

```
action {
    module dscpmk
    name markEF
```

Entry	Description
module dscpmk	Calls the marker module `dscpmk`
name markEF	Gives the name `markEF` to the action statement

5. **Define actions for the marker to take on the traffic flow.**

```
    params {
        global_stats TRUE
        dscp_map{0-63:46}
        next_action acct
    }
}
```

Entry	Description
global_stats TRUE	Enables statistics taking on class `video`, which selects streaming video packets.
dscp_map{0-63:46}	Assigns a DS codepoint of `46` to the packet headers of the traffic class `video`, which is currently being processed by the marker.
next_action acct	Instructs the `dscpmk` module to pass packets of the `video` class to the `acct` action statement after `dscpmk` completes processing. The `acct` action statement invokes the `flowacct` module.

The DS codepoint 46 instructs the `dscpmk` module to set all entries in the `dscp` map to the decimal value 46 (binary 101110) in the DS field. This codepoint indicates that packets of the `video` traffic class are subject to the EF per-hop behavior.

Note – The recommended codepoint for EF is 46 (binary 101110). Other DS codepoints assign AF PHBs to a packet.

The EF PHB guarantees that packets with the DS codepoint of 46 are given the highest precedence by IPQoS and diffserv-aware systems. Streaming applications require highest-priority service, which is the rationale behind assigning them EF PHBs in the QoS policy. For more details about the expedited forwarding PHB, refer to “Expedited Forwarding (EF) PHB” on page 102.

6. **Add the DS codepoints that you have just created to the appropriate files on the diffserv router. For more information, refer to "How to Configure a Router on an IPQoS-Enabled Network" on page 79.**

Where to Go From Here

Task	For Information
Start gathering flow-accounting statistics on traffic flows	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63
Define forwarding behaviors for the marker modules	"How to Define Traffic Forwarding in the IPQoS Configuration File" on page 60
Define flow-control parameters for the metering modules	"How to Configure Flow Control in the IPQoS Configuration File" on page 75
Activate the IPQoS configuration file	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82
Define additional filters	"How to Define Filters in the IPQoS Configuration File" on page 58
Create classes for traffic flows from applications	"How to Configure the IPQoS Configuration File for an Applications Server" on page 70

▼ How to Enable Accounting for a Class in the IPQoS Configuration File

The next procedure shows how to enable accounting on a traffic class in the IPQoS configuration file. The procedure assumes that you have an existing IPQoS configuration file with already-defined classes, filters, metering actions, if appropriate, and marking actions, if appropriate. The steps continue building the `/var/ipqos/Goldweb.qos` file from Example 3–1.

The procedure shows how to define flow accounting for the `video` class, which is introduced in "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56. This class selects streaming video traffic, which must be billed as part of a premium customer's SLA.

1. **Open the IPQoS configuration file, and locate the end of the last action statement you defined.**

   For example, on the IPQoS-enabled server Goldweb, you would start after the following `markEF` action statement in `/var/ipqos/Goldweb.qos`.

```
action {
    module dscpmk
    name markEF
```

```
    params {
        global_stats TRUE
        dscp_map{0-63:46}
        next_action acct
    }
}
```

2. **Begin an action statement that calls flow accounting.**

```
action {
    module flowacct
    name acct
```

Entry	Description
module flowacct	Invokes the flow-accounting module `flowacct`
name acct	Gives the name `acct` to the action statement

3. **Define a params clause to control accounting on the traffic class.**

```
params {
        global_stats TRUE
        timer 10000
        timeout 10000
        max_limit 2048
        next_action continue
    }
}
```

Entry	Description
global_stats TRUE	Enables statistics taking on class `video`, which selects streaming video packets.
timer 10000	Specifies the duration of the interval, in milliseconds, when the flow table is scanned for timed out flows. In this parameter, that interval is 10000 milliseconds.
timeout 10000	Specifies the minimum interval time-out value. A flow "times out" when packets for the flow are not seen during a time-out interval. In this parameter, packets time out after 10000 milliseconds.
max_limit 2048	Sets the maximum number of active flow records in the flow table for this action instance.
next_action continue	Indicates that no further processing is required on packets of the traffic class `video`, and that these packets can return to the network stream.

The `flowacct` module gathers statistical information on packet flows of a particular class until a specified `timeout` value is reached.

Where to Go From Here

Task	For Information
Configure per-hop behaviors on a router	"How to Configure a Router on an IPQoS-Enabled Network" on page 79
Activate the IPQoS configuration file	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82
Create classes for traffic flows from applications	"How to Configure the IPQoS Configuration File for an Applications Server" on page 70

▼ How to Create an IPQoS Configuration File for a Best-Effort Web Server

The IPQoS configuration file for a best-effort web server differs slightly from an IPQoS configuration file for a premium web server. The following procedure illustrates the similarities and differences between configuration files for the varying levels of web service. As an example, the procedure uses the configuration file from Example 3–2.

1. **Log in to the best-effort web server.**
2. **Create a new IPQoS configuration file with a `.qos` extension.**

```
fmtversion 1.0

action {
    module ipgpc
    name ipgpc.classify
    params {
        global_stats TRUE
   }
```

The `/var/ipqos/userweb.qos` file must begin with the partial action statement to invoke the `ipgpc` classifier. In addition, the action statement also has a params clause to turn on statistics taking. For an explanation of this action statement, see "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.

3. **Define a class that identifies traffic that is bound for the best-effort web server.**

```
class {
        name userweb
        next_action markAF12
        enable_stats FALSE
```

```
}
```

Entry	Description
name userweb	Creates a class that is called `userweb` for forwarding web traffic from users.
next_action markF12	Instructs the `ipgpc` module to pass packets of the `userweb` class to the `markAF12` action statement after `ipgpc` completes processing. The `markAF12` action statement invokes the `dscpmk` marker.
enable_stats FALSE	Enables statistics taking for the `userweb` class. However, because the value of `enable_stats` is `FALSE`, statistics for this class are not turned on.

For an explanation of the class clause task, see "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.

4. **Define a filter clause to select traffic flows for the `userweb` class.**

```
    filter {
        name webout
        sport 80
        direction LOCAL_OUT
        class userweb
    }
}
```

Entry	Description
name webout	Gives the name `webout` to the filter
sport 80	Selects traffic with a source port of 80, the well-known port for HTTP (Web) traffic
direction LOCAL_OUT	Further selects traffic that is outgoing from the local system
class userweb	Identifies the class to which the filter belongs, in this instance, class `userweb`

For an explanation of the filter clause task, see "How to Define Filters in the IPQoS Configuration File" on page 58.

5. **Begin the action statement to invoke the `dscpmk` marker.**

```
action {
    module dscpmk
    name markAF12
```

Entry	Description
module dscpmk	Invokes the marker module `dscpmk`
name markAF11	Gives the name `markAF12` to the action statement

The previously defined class `userweb` includes a `next_action markAF12` statement. This statement sends traffic flows to the `markAF12` action statement after the classifier concludes processing.

6. **Define parameters for the marker to use for processing the traffic flow.**

```
    params {
        global_stats FALSE
        dscp_map{0-63:12}
        next_action continue
    }
}
```

Entry	Description
global_stats FALSE	Enables statistics taking for the `markAF12` marker action statement. However, because the value of `enable_stats` is `FALSE`, statistics are not turned on.
dscp_map{0-63:12}	Assigns a DS codepoint of `12` to the packet headers of the traffic class `userweb`, which is currently being processed by the marker.
next_action continue	Indicates that no further processing is required on packets of the traffic class `goldweb`, and that these packets can return to the network stream.

The DS codepoint 12 instructs the marker to set all entries in the `dscp` map to the decimal value 12 (binary 001100). This codepoint indicates that packets of the `userweb` traffic class are subject to the AF12 per-hop behavior. AF12 guarantees that all packets with DS codepoint 12 in the DS field receive a medium-drop, high-priority service.

When you complete the IPQoS configuration file, apply the configuration, as described in "How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82.

Where to Go From Here

Task	For Information
Add classes and other configuration for traffic flows from applications	"How to Configure the IPQoS Configuration File for an Applications Server" on page 70
Configure per-hop behaviors on a router	"How to Configure a Router on an IPQoS-Enabled Network" on page 79
Activate your IPQoS configuration file	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82

Creating an IPQoS Configuration File for an Application Server

This section explains how to create a configuration file for an applications server that provides a number of major applications to both internal and external customers. The procedure uses as its example the BigAPPs server from Figure 2–4.

The following configuration file defines IPQoS activities for the BigAPPs server, which hosts FTP, electronic mail (SMTP), and network news (NNTP) for customers.

EXAMPLE 3–3 Sample Configuration for an Application Server

```
fmt_version 1.0

action {
    module ipgpc
    name ipgpc.classify
    params {
        global_stats TRUE
    }
    class {
        name smtp
        enable_stats FALSE
        next_action markAF13
    }
    class {
        name news
        next_action markAF21
    }
    class {
        name ftp
        next_action meterftp
    }
```

EXAMPLE 3–3 Sample Configuration for an Application Server *(Continued)*

```
    filter {
        name smtpout
        sport smtp
        class smtp
    }
    filter {
        name newsout
        sport nntp
        class news
    }
    filter {
        name ftpout
        sport ftp
        class ftp

    filter {
        name ftpdata
        sport ftp-data
        class ftp
    }
}
action {
    module dscpmk
    name markAF13
    params {
        global_stats FALSE
        dscp_map{0-63:14}
        next_action continue
    }
}
action {
    module dscpmk
    name markAF21
    params {
        global_stats FALSE
        dscp_map{0-63:18}
        next_action continue
    }
}
action {
    module tokenmt
    name meterftp
    params {
        committed_rate 50000000
        committed_burst 50000000
        red_action markAF31
        green_action markAF22
        global_stats TRUE
    }
}
action {
    module dscpmk
    name markAF31
```

EXAMPLE 3–3 Sample Configuration for an Application Server *(Continued)*

```
    params {
        global_stats TRUE
        dscp_map{0-63:26}
        next_action continue
    }
}
action {
    module dscpmk
    name markAF22
    params {
        global_stats TRUE
        dscp_map{0-63:20}
        next_action continue
    }
}
```

▼ How to Configure the IPQoS Configuration File for an Applications Server

1. **Log in to the IPQoS-enabled application server, and create a new IPQoS configuration file with a `.qos` extension.**

   For example, you would create the `/var/ipqos/BigAPPS.qos` file for the applications server. Begin with the following required phrases to start the action statement that invokes the `ipgpc` classifier.

   ```
   fmtversion 1.0

   action {
       module ipgpc
       name ipgpc.classify
       params {
           global_stats TRUE
       {
   ```

   For an explanation of the opening action statement, refer to "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.

2. **Create classes to select traffic from three applications on the BigAPPs server.**

   Add the class definitions after the opening action statement.

   ```
       class {
           name smtp
           enable_stats FALSE
           next_action markAF13
       }
       class {
           name news
           next_action markAF21
       }
   ```

```
class {
    name ftp
    enable_stats TRUE
    next_action meterftp
}
```

Entry	Description
name smtp	Creates a class that is called `smtp`, which includes email traffic flows to be handled by the SMTP application.
enable_stats FALSE	Enables statistics taking for the `smtp` class. However, because the value of `enable_stats` is `FALSE`, statistics for this class are not turned on.
next_action markAF13	Instructs the `ipgpc` module to pass packets of the `smtp` class to the `markAF13` action statement after `ipgpc` completes processing.
name news	Creates a class that is called `news`, which includes network news traffic flows to be handled by the NNTP application.
next_action markAF21	Instructs the `ipgpc` module to pass packets of the `news` class to the `markAF21` action statement after `ipgpc` completes processing.
name ftp	Creates a class that is called `ftp`, which handles outgoing traffic that is handled by the FTP application.
enable_stats TRUE	Enables statistics taking for the `ftp` class.
next_action meterftp	Instructs the `ipgpc` module to pass packets of the `ftp` class to the `meterftp` action statement after `ipgpc` completes processing.

For more information about defining classes, refer to "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.

3. **Define filter clauses to select traffic of the previously defined classes.**

```
filter {
    name smtpout
    sport smtp
    class smtp
}
filter {
    name newsout
    sport nntp
    class news
```

```
    }
        filter {
        name ftpout
        sport ftp
        class ftp
    }
        filter {
        name ftpdata
        sport ftp-data
        class ftp
    }
}
```

Entry	Description
name smtpout	Gives the name `smtpout` to the filter
sport smtp	Selects traffic with a source port of 25, the well-known port for the `sendmail` (SMTP) application
class smtp	Identifies the class to which the filter belongs, in this instance, class `smtp`
name newsout	Gives the name `newsout` to the filter
sport nntp	Selects traffic with a source port name of `nntp`, the well-known port name for the network news application
class news	Identifies the class to which the filter belongs, in this instance, class `news`
name ftpout	Gives the name `ftpout` to the filter
sport ftp	Selects control data with a source port of 21, the well-known port number for FTP traffic
name ftpdata	Gives the name `ftpdata` to the filter
sport ftp-data	Selects traffic with a source port of 20, the well-known port number for FTP data traffic
class ftp	Identifies the class to which the `ftpout` and `ftpdata` filters belong, in this instance `ftp`

For more information about defining filters, refer to "How to Define Filters in the IPQoS Configuration File" on page 58.

Where to Go From Here

Task	For Information
Define filters	"How to Define Filters in the IPQoS Configuration File" on page 58
Define forwarding behaviors for application traffic	"How to Configure Forwarding for Application Traffic in the IPQoS Configuration File" on page 73
Configure flow control by using the metering modules	"How to Configure Flow Control in the IPQoS Configuration File" on page 75
Configure flow accounting	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63

▼ How to Configure Forwarding for Application Traffic in the IPQoS Configuration File

The next procedure shows how to configure forwarding for application traffic. In the procedure, you define per-hop behaviors for application traffic classes that might have lower precedence than other traffic on a network. The procedure assumes that you have an existing IPQoS configuration file with already-defined classes and filters for the applications to be marked. The steps continue building the `/var/ipqos/BigAPPs.qos` file in Example 3–3.

1. **Open the IPQoS configuration file you have created for the applications server.**

   Locate the end of the last filter clause. In the `/var/ipqos/BigAPPs.qos` file, the last filter is the following:

```
 filter {
        name ftpdata
        sport ftp-data
        class ftp
    }
}
```

2. **Invoke the marker as follows:**

```
action {
    module dscpmk
    name markAF13
```

Entry	Description
module dscpmk	Invokes the marker module `dscpmk`
name markAF13	Gives the name `markAF13` to the action statement

3. **Define the per-hop behavior to be marked on electronic mail traffic flows.**

```
        params {
            global_stats FALSE
            dscp_map{0-63:14}
            next_action continue
        }
    }
```

Entry	Description
global_stats FALSE	Enables statistics taking for the `markAF13` marker action statement. However, because the value of `enable_stats` is `FALSE`, statistics are not turned on.
dscp_map{0-63:14}	Assigns a DS codepoint of `14` to the packet headers of the traffic class `smtp`, which is currently being processed by the marker.
next_action continue	Indicates that no further processing is required on packets of the traffic class `smtp`. These packets can then return to the network stream.

The DS codepoint 14 tells the marker to set all entries in the `dscp` map to the decimal value 14 (binary 001110). This value sets the AF13 per-hop behavior and marks packets of the `smtp` traffic class with the DS codepoint 14 in the DS field.

AF13 assigns all packets with a DS codepoint of 14 to a high-drop precedence. However, because AF13 also assures a Class 1 priority, the router still guarantees outgoing email traffic a high priority in its queue. For a table of possible AF codepoints, refer to Table 6–2.

4. **Add a marker action statement to define a per-hop behavior for network news traffic:**

```
action {
    module dscpmk
    name markAF21
    params {
        global_stats FALSE
        dscp_map{0-63:18}
        next_action continue
    }
}
```

The next table explains parameters that have not yet been defined in this procedure.

Entry	Description
name markAF21	Gives the name markAF21 to the action statement
dscp_map{0-63:18}	Assigns a DS codepoint of 18 to the packet headers of the traffic class nntp, which is currently being processed by the marker

The DS codepoint 18 tells the marker to set all entries in the dscp map to the decimal value 18 (binary 010010). This value sets the AF21 per-hop behavior and marks packets of the news traffic class with the DS codepoint 18 in the DS field.

AF21 assures that all packets with a DS codepoint of 18 receive a low-drop precedence, but with only Class 2 priority. Thus, the possibility of network news traffic being dropped is low, but the router gives a higher forwarding probability to traffic classes with a Class 1 mark.

Where to Go From Here

Task	For Information
Add configuration information for web servers	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56
Configure flow control by using the metering modules	"How to Configure Flow Control in the IPQoS Configuration File" on page 75
Configure flow accounting	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63
Configure forwarding behaviors on a router	"How to Configure a Router on an IPQoS-Enabled Network" on page 79
Activate the IPQoS configuration file	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82

▼ How to Configure Flow Control in the IPQoS Configuration File

To control the rate at which a particular traffic flow is released onto the network, you must define parameters for the meter. You can use either of the two metering modules, tokenmt or tswtclmt, in the IPQoS configuration file.

The next procedure continues to build the IPQoS configuration file for the application server in Example 3–3. In the procedure, you configure not only the meter but also two marker actions that are called within the meter action statement.

1. **Open the IPQoS configuration file you have created for the applications server.**

   The remaining steps assume that you have already defined a class and a filter for the application to be flow-controlled. In the `/var/ipqos/BigAPPs.qos` file, you begin after the following marker action:

```
action {
    module dscpmk
    name markAF21
    params {
        global_stats FALSE
        dscp_map{0-63:18}
        next_action continue
    }
}
```

2. **Create a meter action statement to flow-control traffic of the `ftp` class:**

```
action {
    module tokenmt
    name meterftp
```

Entry	Definition
`module tokenmt`	Invokes the `tokenmt` meter
`name meterftp`	Gives the name `meterftp` to the action statement

3. **Add parameters to configure the meter's rate:**

```
params {
       committed_rate 50000000
       committed_burst 50000000
```

Entry	Description
`committed_rate 50000000`	Assigns a transmission rate of 5,000,0000 bits-per-second to traffic of the `ftp` class
`committed_burst 50000000`	Commits a burst size of 50,000,000 bits to traffic of the `ftp` class

   For an explanation of `tokenmt` parameters, refer to "Configuring `tokenmt` as a Two-Rate Meter" on page 99.

4. **Add parameters to configure traffic conformance precedences:**

```
    red_action markAF31
    green_action markAF22
    global_stats TRUE
```

```
    }
}
```

Entry	Description
red_action markAF31	Indicates that when the traffic flow of the `ftp` class becomes nonconformant, that is, exceeds the committed rate, packets are sent to the `markAF31` marker action statement
green_action markAF22	Indicates that when traffic flows of class `ftp` conform to the committed rate, packets are sent to the `markAF22` action statement
global_stats TRUE	Enables metering statistics for the `ftp` class

For more information about traffic conformance, see "Meter Module" on page 98.

5. **Add a marker action statement to assign a per-hop behavior to nonconformant traffic flows of class `ftp`.**

```
action {
    module dscpmk
    name markAF31
    params {
        global_stats TRUE
        dscp_map{0-63:26}
        next_action continue
    }
}
```

Entry	Description
module dscpmk	Invokes the marker module `dscpmk`.
name markAF31	Gives the name `markAF31` to the action statement.
global_stats TRUE	Enables statistics for the `ftp` class.
dscp_map{0-63:26}	Assigns a DS codepoint of `26` to the packet headers of the traffic class `ftp` whenever this traffic exceeds the committed rate.
next_action continue	Indicates that no further processing is required on packets of the traffic class `ftp`. Then these packets can return to the network stream.

The DS codepoint 26 instructs the marker to set all entries in the `dscp` map to the decimal value 26 (binary 011010). This value sets the AF31 per-hop behavior and marks packets of the `ftp` traffic class with the DS codepoint 26 in the DS field.

AF31 assures that all packets with a DS codepoint of 26 receive a low-drop precedence, but with only Class 3 priority. Therefore, the possibility of nonconformant FTP traffic being dropped is low. However, the router gives a higher forwarding probability to traffic classes with a Class 1 or Class 2 low-drop precedence mark or better. For a table of possible AF codepoints, refer to Table 6–2.

6. **Add a marker action statement to assign a per-hop behavior to traffic flows of class `ftp` that conform to the committed rate.**

```
action {
    module dscpmk
    name markAF22
    params {
        global_stats TRUE
        dscp_map{0-63:20}
        next_action continue
    }
}
```

The next table contains parameters that are not defined in the previous step.

Entry	Description
`name markAF22`	Gives the name `markAF22` to the `marker` action
`dscp_map{0-63:20}`	Assigns a DS codepoint of `20` to the packet headers of the traffic class `ftp` whenever `ftp` traffic conforms to its configured rate

The DS codepoint 20 tells the marker to set all entries in the `dscp` map to the decimal value 20 (binary 010100). This value sets the AF22 per-hop behavior and marks packets of the `ftp` traffic class with the DS codepoint 20 in the DS field.

AF22 assures that all packets with a DS codepoint of 20 receive a medium-drop precedence with Class 2 priority. Therefore, conformant FTP traffic is assured a medium-drop precedence among flows that are simultaneously released by the IPQoS system. However, the router gives a higher forwarding priority to traffic classes with a Class 1 medium-drop precedence mark or higher. For a table of possible AF codepoints, refer to Table 6–2.

7. **Add the DS codepoints that you have created for the application server to the appropriate files on the diffserv router. For more information, refer to "How to Configure a Router on an IPQoS-Enabled Network" on page 79.**

Where to Go From Here

Task	For Information
Activate the IPQoS configuration file	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82
Add configuration information for web servers	"How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56
Configure flow accounting	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63
Configure forwarding behaviors on a router	"How to Configure a Router on an IPQoS-Enabled Network" on page 79

Providing Differentiated Services on a Router

To provide true differentiated services, you must include a diffserv-aware router in your network topology, as described in "Hardware Strategies for the Diffserv Network" on page 32. The actual steps for configuring diffserv on a router and updating that router's files are outside the scope of this document.

This section gives general steps for coordinating the forwarding information among various IPQoS-enabled systems on the network and the diffserv router. The next procedure assumes that you have already configured the IPQoS systems on your network by performing the previous tasks in this chapter.

▼ How to Configure a Router on an IPQoS-Enabled Network

The next procedure uses as its example the topology in Figure 2–4.

1. **Review the configuration files for all IPQoS-enabled systems on your network.**
2. **Identify each codepoint that is used in the various policies.**

   List the codepoints, and the systems and classes, to which the codepoints apply. The table can illustrate areas where you might have used the same codepoint. This practice is acceptable, but you should provide other criteria in the IPQoS configuration file, such as a `precedence` selector, to be used to determine the precedence of identically marked classes.

For example, for the sample network that is used in the procedures of this chapter, you might construct the following codepoint table.

TABLE 3–2 Per-Hop Behaviors Configured for a Sample Network

System	Class	PHB	DS Codepoint
Goldweb	`video`	EF	46 (101110)
" "	`goldweb`	AF11	10 (001010)
Userweb	`webout`	AF12	12 (001100)
BigAPPs	smtp	AF13	14 (001110)
"	news	AF18	18 (010010)
"	ftp conformant traffic	AF22	20 (010100)
"	ftp nonconformant traffic	AF31	26 (011010)

3. **Add the codepoints from your network's IPQoS configuration files to the appropriate files on the diffserv router.**

   The codepoints you supply should help to configure the router's diffserv scheduling mechanism. Refer to the router manufacturers' documentation and web sites for instructions.

CHAPTER 4

Starting Up and Maintaining IPQoS (Tasks)

This chapter contains tasks for activating an IPQoS configuration file and for logging IPQoS-related events. The following topics are covered:

Administering IPQoS (Task Map)

This section lists the set of tasks for starting and maintaining IPQoS on a Solaris system. Before you use the tasks, you must have a completed IPQoS configuration file, as described in "Defining a QoS Policy in the IPQoS Configuration File (Task Map)" on page 51.

TABLE 4–1 Configuring and Maintaining IPQoS (Task Map)

Task	Description	For Instructions
1. Configure IPQoS on a system.	Use the `ipqosconf` utility to activate the IPQoS configuration file on a system.	"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82
2. Make the Solaris startup scripts apply the debugged IPQoS configuration file after each system boot.	Ensure that the IPQoS configuration is applied each time the system reboots.	"How to Ensure That the IPQoS Configuration Is Applied After Each Reboot" on page 83

TABLE 4–1 Configuring and Maintaining IPQoS (Task Map) *(Continued)*

Task	Description	For Instructions
3. Enable `syslog` logging for IPQoS.	Add an entry to enable `syslog` logging of IPQoS messages.	"How to Enable Logging of IPQoS Messages During Booting" on page 84
4. Fix IPQoS problems as they arise.	Troubleshoot IPQoS problems by using error messages.	Refer to the error messages in Table 4–2

Activating an IPQoS Configuration

You activate and otherwise manipulate the IPQoS configuration by using the `ipqosconf` command.

▼ How to Apply a New Configuration to the IPQoS Kernel Modules

You use the `ipqosconf` tool to read the IPQoS configuration file and configure the IPQoS modules in the UNIX kernel. The next procedure uses as an example the file `/var/ipqos/Goldweb.qos`, which is created in "Creating IPQoS Configuration Files for Web Servers" on page 53. For detailed information, refer to the `ipqosconf`(1m) man page.

1. **Become superuser on the IPQoS-enabled system.**

2. **Apply the new configuration.**

   ```
   # /usr/sbin/ipqosconf -a /var/ipqos/Goldweb.qos
   ```

   `ipqosconf` writes the information in the specified IPQoS configuration file into the IPQoS modules in the Solaris kernel. In the previous example, the contents of `/var/ipqos/Goldweb.qos` are applied to the current Solaris kernel.

   Note – When you apply an IPQoS configuration file with the `-a` option, the actions in the file are active for the current session only.

3. **Test and debug the new IPQoS configuration.**

   Use UNIX utilities to track IPQoS behavior and gather statistics on your IPQoS implementation. Thus, you can determine if the configuration operates as you expected.

Where to Go From Here

Task	For Instructions
View statistics on how the IPQoS modules are working	"Gathering Statistical Information" on page 93
Log `ipqosconf` messages	"Enabling syslog Logging for IPQoS Messages" on page 83
Ensure that the current IPQoS configuration is applied after each boot	"How to Ensure That the IPQoS Configuration Is Applied After Each Reboot" on page 83

▼ How to Ensure That the IPQoS Configuration Is Applied After Each Reboot

You must explicitly make an IPQoS configuration persistent across reboots. Otherwise, the current configuration applies only until the system reboots. When IPQoS works correctly on a system, do the following to make the configuration persistent across reboots.

1. **Log in as superuser to the IPQoS-enabled system.**
2. **Test for the existence of an IPQoS configuration in the kernel modules:**

   ```
   # ipqosconf -l
   ```

   If a configuration already exists, `ipqosconf` displays it on the screen. If you do not receive output, apply the configuration, as explained in"How to Apply a New Configuration to the IPQoS Kernel Modules" on page 82.
3. **Ensure that the existing IPQoS configuration is applied every time the IPQoS system reboots.**

   ```
   # /usr/sbin/ipqosconf -c
   ```

   The `-c` option causes the current IPQoS configuration to be represented in the boot-time configuration file `/etc/inet/ipqosinit.conf`.

Enabling syslog Logging for IPQoS Messages

To record IPQoS boot-time messages, you need to modify the `/etc/syslog.conf` file as shown in the next procedure.

▼ How to Enable Logging of IPQoS Messages During Booting

1. **Become superuser on the IPQoS-enabled machine.**
2. **Open the** `/etc/syslog.conf` **file.**
3. **Add the following text as the final entry in the file.**

   Use tabs, rather than spaces, between the columns.

   ```
   user.info                    /var/adm/messages
   ```

   This entry logs all boot-time messages that are generated by IPQoS into the `/var/adm/messages` file.
4. **Reboot the system to apply the messages.**

Examples—IPQoS Output From `/var/adm/messages`

When you view `/var/adm/messages` after system reboot, your output might contain IPQoS logging messages similar to the following.

```
May 14 10:44:33 ipqos-14 ipqosconf: [ID 815575 user.info]
 New configuration applied.
May 14 10:44:46 ipqos-14 ipqosconf: [ID 469457 user.info]
Current configuration saved to init file.
May 14 10:44:55 ipqos-14 ipqosconf: [ID 435810 user.info]
Configuration flushed.
```

You might also see IPQoS error messages similar to the following in your IPQoS system's `/var/adm/messages` file.

```
May 14 10:56:47 ipqos-14 ipqosconf: [ID 123217 user.error]
 Missing/Invalid config file fmt_version.
May 14 10:58:19 ipqos-14 ipqosconf: [ID 671991 user.error]
No ipgpc action defined.
```

For a description of the previous error messages, see Table 4–2.

Using IPQoS Error Messages

This section contains a table of error messages that are generated by IPQoS and their possible solutions.

TABLE 4–2 IPQoS Error Messages

Error Message	Description	Solution
`Undefined action in parameter` *parameter-name's* `action` *action-name*	In the IPQoS configuration file, the action name that you specified in *parameter-name* does not exist in the configuration file.	Create the action or refer to a different, existing action in the parameter.
`action` *action-name* `involved in cycle`	In the IPQoS configuration file, *action-name* is part of a cycle of actions, which is not allowed by IPQoS.	Determine the action cycle and remove one of the cyclical references from the IPQoS configuration file.
`Action` *action-name* `isn't referenced by any other actions`	A non-`ipgpc` action definition is not referenced by any other defined actions in the IPQoS configuration, which is not allowed by IPQoS.	Remove the unreferenced action or make another action reference the currently unreferenced action.
`Missing/Invalid config file fmt_version`	The format of the configuration file is not specified as the first entry of the file, which is required by IPQoS.	Add the format version, as explained in "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.
`Unsupported config file format version`	The format version that is specified in the configuration file is not supported by IPQoS.	Change the format version to `fmt_version 1.0`, which is required to run the Solaris 9 9/02 version of IPQoS.
`No ipgpc action defined.`	You did not define an action for the `ipgpc` classifier in the configuration file, which is an IPQoS requirement.	Define an action for `ipgpc`, as shown in "How to Begin the IPQoS Configuration File and Define Traffic Classes" on page 56.
`Can't commit a null configuration`	When you ran `ipqosconf -c` to commit a configuration, that configuration was empty, which IPQoS does not allow.	Be sure to apply a configuration file before you attempt to commit a configuration.
`Invalid CIDR mask on line` *line_number*	In the configuration file, you used a CIDR mask as part of the IP address that is out of the valid range for IP addresses.	Change the mask value to be in the range of 1–32 for IPv4 and 1–128 for IPv6.

TABLE 4–2 IPQoS Error Messages *(Continued)*

Error Message	Description	Solution
`Address masks aren't allowed for host names line` *line_number*	In the configuration file, you defined a CIDR mask for a host name, which is not allowed in IPQoS.	Remove the mask or change the host name to an IP address.
`Invalid module name line` *line_number*	In the configuration file, the module name you specified in an action statement is invalid.	Check the spelling of the module name. For a list of IPQoS modules, refer to Table 6–5.
`ipgpc action has incorrect name line` *line_number*	The name that you gave to the `ipgpc` action in the configuration file is not the required `ipgpc.classify`.	Rename the action `ipgpc.classify`.
`Second parameter clause not supported line` *line_number*	In the configuration file, you specified two parameter clauses for a single action, which IPQoS does not allow.	Combine all parameters for the action into a single parameters clause.
`Duplicate named action`	In the configuration file, you gave the same name to two actions.	Rename or remove one of the actions.
`Duplicate named filter/class in action` *action_name*	You gave the same name to two filters or two classes in the same action, which is not allowed in the IPQoS configuration file.	Rename or remove one of the filters or classes.
`Undefined class in filter` *filter_name* `in action` *action_name*	In the configuration file, the filter references a class that is not defined in the action.	Create the class, or change the filter reference to an already existing class.
`Undefined action in class` *class_name* `action` *action_name*	The class refers to an action that is not defined in the configuration file.	Create the action, or change the reference to an already existing action.
`Invalid parameters for action` *action_name*	In the configuration file, one of the parameters is invalid.	For the module that is called by the named action, refer to the module entry in "IPQoS Architecture and the diffserv Model" on page 95. Alternatively, you can refer to the `ipqosconf`(1M) man page.
`Mandatory parameter missing for action` *action_name*	You have not defined a required parameter for an action in the configuration file.	For the module that is called by the named action, refer to the module entry in "IPQoS Architecture and the diffserv Model" on page 95. Alternatively, you can refer to the `ipqosconf`(1M) man page.

TABLE 4–2 IPQoS Error Messages *(Continued)*

Error Message	Description	Solution
`Max number of classes reached in ipgpc`	You specified more classes than are allowed in the `ipgpc` action of the IPQoS configuration file. The maximum number is 10007.	Review the configuration file and remove unneeded classes. Alternatively, you can raise the maximum number of classes by adding to the `/etc/system` file the entry `ipgpc_max_classes` *class_number*.
`Max number of filters reached in action ipgpc`	You specified more filters than are allowed in the `ipgpc` action of the IPQoS configuration file. The maximum number is 10007.	Review the configuration file and remove unneeded filters. Alternatively, you can raise the maximum number of filters by adding to the `/etc/system` file the entry `ipgpc_max_filters` *class_number*.
`Invalid/missing parameters for filter` *filter_name* `in action ipgpc.`	In the configuration file, filter *filter_name* has an invalid or missing parameter.	Refer to the `ipqosconf`(1m) man page for the list of valid parameters.
`Name not allowed to start with '!', line` *line_number*	You began an action, filter, or class name with an exclamation mark (!), which is not allowed in the IPQoS file.	Remove the exclamation mark or rename the action, class, or filter.
`Name exceeds the maximum name length line` *line_number*	You defined a name for an action, class, or filter in the configuration file that exceeds the maximum length of 23 characters.	Give a shorter name to the action, class, or filter.
`Array declaration line` *line_number* `is invalid`	In the configuration file, the array declaration for the parameter on line *line_number* is invalid.	For the correct syntax of the array declaration that is called by the action statement with the invalid array, refer to "IPQoS Architecture and the diffserv Model" on page 95. Alternatively, refer to the `ipqosconf`(1m) man page.
`Quoted string exceeds line,` *line_number*	The string does not have the terminating quotation marks on the same line, which is required in the configuration file.	Make sure that the quoted string begins and ends on the same line in the configuration file.
`Invalid value, line` *line_number*	The value that is given on *line_number* of the configuration file is not supported for the parameter.	For the acceptable values for the module that is called by the action statement, refer to module description in "IPQoS Architecture and the diffserv Model" on page 95. Alternatively, you can refer to the `ipqosconf`(1m) man page.

TABLE 4–2 IPQoS Error Messages *(Continued)*

Error Message	Description	Solution
`Unrecognized value, line` *line_number*	The value on *line_number* of the configuration file is not a supported enumeration value for its parameter.	Check that the enumeration value is correct for the parameter. For a the description of the module that is called by the action statement with the unrecognized line number, refer to "IPQoS Architecture and the diffserv Model" on page 95. Alternatively, you can refer to the `ipqosconf`(1m) man page.
`Malformed value list line` *line_number*	The enumeration that is specified on *line_number* of the configuration file does not conform to the specification syntax.	For correct syntax of the module that is called by the action with the malformed value list, refer to the module description in "IPQoS Architecture and the diffserv Model" on page 95. Alternatively, you can refer to the `ipqosconf`(1m) man page.
`Duplicate parameter line` *line_number*	A duplicate parameter was specified on *line_number*, which is not allowed in the configuration file.	Remove one of the duplicate parameters.
`Invalid action name line` *line_number*	You gave the action on *line_number* of the configuration file a name that uses the predefined name "continue" or "drop."	Rename the action so that it does not use a predefined name.
`Failed to resolve src/dst host name for filter at line` *line_number*`, ignoring filter`	`ipqosconf` could not resolve the source or destination address that was defined for the given filter in the configuration file. Therefore, the filter is ignored.	If the filter is important, try applying the configuration at a later time.
`Incompatible address version line` *line_number*	The IP version of the address on *line_number* is incompatible with the version of a previously specified IP address or `ip_version` parameter in the configuration file.	Change the two conflicting entries to be compatible.
`Action at line` *line_number* `has the same name as currently installed action, but is for a different module`	You tried to change the module of an action that already exists in the system's IPQoS configuration, which is not allowed.	Flush the current configuration before you apply the new configuration.

CHAPTER 5

Using Flow Accounting and Statistics Gathering (Tasks)

This chapter explains how to obtain accounting and statistical information on traffic that is handled by an IPQoS system. The following topics are discussed:

- "Recording Information About Flows" on page 90
- "Gathering Statistical Information" on page 93

Setting Up Flow Accounting (Task Map)

The following table lists the generic tasks for obtaining information about traffic flows by using the `flowacct` module.

TABLE 5–1 Configuring Flow Accounting (Task Map)

Task	Description	For Instructions
1. Create a file to contain accounting information for traffic flows.	Use the `acctadm` command to create a file that holds the results of processing by `flowacct`.	"How to Create a File for Flow-Accounting Data" on page 90
2. Define `flowacct` parameters in the IPQoS configuration file.	Define values for the `timer`, `timeout`, and `max_limit` parameters.	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63
3. View the contents of the file.	View an example program that can be used to create a utility to read accounting records from a file.	"How to Get Instructions for Viewing a Flow-Accounting File" on page 92

Recording Information About Flows

You use the IPQoS `flowacct` module to collect information about traffic flows, such as the source and destination addresses, amount of packets in a flow, and similar data. The process of accumulating and recording information about flows is called *flow accounting*.

The results of flow accounting on traffic of a particular class are recorded in a table of *flow records*. Each flow record consists of a series of attributes. These attributes contain data about traffic flows of a particular class over an interval of time. For a list of the `flowacct` attributes, refer to Table 6–4.

Flow accounting is particularly useful for billing clients as is defined in their service-level agreements (SLAs). You can also use flow accounting to obtain flow statistics for critical applications. This section contains tasks for using `flowacct` with the Solaris extended accounting facility to obtain data on traffic flows.

The following information is contained in sources outside this chapter:

- For instructions on creating an action statement for `flowacct` in the IPQoS configuration file, refer to "How to Configure Flow Control in the IPQoS Configuration File" on page 75.
- To learn how `flowacct` works, refer to "Classifier Module" on page 95.
- For technical information, refer to the `flowacct`(7ipp) man page.

▼ How to Create a File for Flow-Accounting Data

Before you add a `flowacct` action to the IPQoS configuration file, you must create a file for flow records from the `flowacct` module. You use the `acctadm` command for this purpose. `acctadm` can record either basic attributes or extended attributes in the file. All `flowacct` attributes are listed in Table 6–4. For detailed information about `acctadm`, refer to `acctadm`(1m).

1. **Log in as superuser to the IPQoS-enabled system.**
2. **Create a basic flow-accounting file.**

   The following example shows how to create a basic flow-account file for the premium web server that is configured in Example 3–1.

   ```
   # /usr/sbin/acctadm -e basic -f /var/ipqos/goldweb/account.info flow
   ```

Statement	Definition
`acctadm -e`	Invokes `acctadm` with the `-e` option. The `-e` option enables the arguments that follow.
`basic`	States that only data for the eight basic `flowacct` attributes are to be recorded in the file.
`/var/ipqos/goldweb/account.info`	Specifies the fully qualified path name of the file to hold the flow records from `flowacct`.
`flow`	Tells `acctadm` to enable flow accounting.

3. **View information about flow accounting on the IPQoS system by typing `acctadm` without arguments.**

   `acctadm` generates the following output:

```
Task accounting: inactive
       Task accounting file: none
     Tracked task resources: none
   Untracked task resources: extended
         Process accounting: inactive
    Process accounting file: none
  Tracked process resources: none
Untracked process resources: extended,host,mstate
            Flow accounting: active
       Flow accounting file: /var/ipqos/goldweb/account.info
     Tracked flow resources: basic
   Untracked flow resources: dsfield,ctime,lseen,projid,uid
```

   All but the last four entries are for use with the Solaris 9 Resource Manager feature. The next table explains the entries that are specific to IPQoS.

Entry	Description
`Flow accounting: active`	Indicates that flow accounting is turned on
`Flow accounting file: /var/ipqos/goldweb/account.info`	Gives the name of the current flow-accounting file
`Tracked flow resources: basic`	Indicates that only the basic flow attributes are tracked
`Untracked flow resources: dsfield,ctime,lseen,projid,uid`	Lists the `flowacct` attributes that are not tracked in the file

4. **(Optional) Add the extended attributes to the accounting file as follows:**

```
# acctadm -e extended -f /var/ipqos/goldweb/account.info flow
```

5. **(Optional) Return to recording only the basic attributes in the accounting file.**

```
# acctadm -d extended -e basic -f /var/ipqos/goldweb/account.info
```

The `-d` option disables extended accounting.

Where to Go Next

Task	For Instructions
Define `flowacct` parameters in the IPQoS configuration file	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 63
Print out the data in the file that was created with `acctadm`	"How to Get Instructions for Viewing a Flow-Accounting File" on page 92

▼ How to Get Instructions for Viewing a Flow-Accounting File

You must create a script to display the contents of the flow-accounting file that was created by `acctadm`. You can use as the basis for your script a demonstration script for resource management tasks and processing accounting. The following task shows how to get information about the demonstration script.

Before you can use the next procedure, you must have created a file to hold flow records, as described in "How to Create a File for Flow-Accounting Data" on page 90. You also must have added a `flowacct` action and parameters to the IPQoS configuration file so that traffic classes are tracked by `flowacct`.

The next task introduces the `libexacct` programmatic interface and `exdump` utility to provide output for an `acctadm` file for viewing processes and tasks. For technical information, refer to the `libexacct`(3LIB) man page.

1. **Become superuser on the IPQoS system and access the directory `/usr/demo/libexacct`.**

   This directory contains a Makefile and the `exdump.c` script.

2. **Perform the instructions to build `exdump` as is explained in the README.**

3. **View the data in the flow-accounting file, as is explained in the README.**

Gathering Statistical Information

You can use the `kstat` command to generate statistical information from the IPQoS modules. Use the following syntax.

`/bin/kstat -m` *ipqos-module-name*
You can specify any valid IPQoS module name, as shown in Table 6–5. For example, to view statistics that are generated by the `dscpmk` marker, you use the following form of `kstat`.

`/bin/kstat -m dscpmk`
For technical details, refer to the `kstat`(1M) man page.

Example—`kstat` Statistics for IPQoS

Here is an example of possible results from running `kstat` to obtain statistics about the flowacct module.

```
# kstat -m flowacct
module: flowacct                        instance: 3
name:   Flowacct statistics             class:    flacct
        bytes_in_tbl                    84
        crtime                          345728.504106363
        epackets                        0
        flows_in_tbl                    1
        nbytes                          84
        npackets                        1
        snaptime                        345774.031843301
        usedmem                         256
```

Entry	Description
`class: flacct`	Gives the name of the class to which the traffic flows belong, in this instance `flacct`.
`bytes_in_tbl`	Total number of bytes in the flow table, that is, the sum in bytes of all the flow records that currently reside in the flow table. The total number of bytes for this flow table is 84. If no flows are in the table, the value for `bytes_in_tbl` is 0.
`crtime`	The last time this `kstat` was created.
`epackets`	Number of packets that resulted in an error during processing, in this instance 0.

Entry	Description
`flows_in_tbl`	Number of flow records in the flow table, which in this instance is 1. When no records are in the table, the value for `flows_in_tbl` is 0.
`nbytes`	Total number of bytes that are seen by this `flowacct` action instance, which is 84 in the example. The value includes bytes that are currently in the flow table and bytes that have timed out—are no longer in the flow table.
`npackets`	Total number of packets that are seen by this `flowacct` action instance, which is 1 in the example. `npackets` includes packets that are currently in the flow table and those that have timed out—are no longer in the flow table.
`usedmem`	Memory in bytes in use by the flow table that is maintained by this `flowacct` instance. The `usedmem` value is 256 in the example. The value for `usedmem` is 0 when the flow table does not have any flow records.

CHAPTER 6

IPQoS in Depth (Reference)

This chapter contains reference materials that provide in-depth details about the following IPQoS topics:

For an overview, refer to Chapter 1. For planning information, refer to Chapter 2. For procedures for configuring IPQoS, refer to Chapter 3.

IPQoS Architecture and the diffserv Model

This section describes the IPQoS architecture and how it implements the differentiated services (diffserv) model that is defined in RFC 2475, *An Architecture for Differentiated Services*. The following elements of the diffserv model are included in IPQoS:

- Classifier
- Meter
- Marker

In addition, IPQoS includes the flow-accounting module and the `dlcosmk` marker for use with VLAN devices.

Classifier Module

In the diffserv model, the *classifier* is responsible for organizing selected traffic flows into groups on which to apply different service levels. The classifiers that are defined in RFC 2475 were originally designed for boundary routers. By contrast, the IPQoS

classifier `ipgpc` is designed to handle traffic flows on hosts internal to the local network. Therefore, a network with both IPQoS systems and a diffserv router can provide a greater degree of differentiated services. For a technical description of `ipgpc`, refer to the `ipgpc`(7ipp) man page.

The `ipgpc` classifier does the following:

1. Selects traffic flows that meet the criteria that are specified in the IPQoS configuration file on the IPQoS-enabled system

   The QoS policy defines various criteria that must be present in packet headers. These criteria are called *selectors*. The `ipgpc` classifier compares these selectors against the headers of packets that are received by the IPQoS system. `ipgpc` then selects all matching packets.
2. Separates the packet flows into *classes*, network traffic with the same characteristics, as defined in the IPQoS configuration file
3. Examines the value in the packet's differentiated service (DS) field for the presence of a differentiated services (DS) codepoint

   The presence of the DS codepoint, also known as the DSCP, indicates whether the incoming traffic has been marked by the sender with a forwarding behavior.
4. Determines what further action is specified in the IPQoS configuration file for packets of a particular class
5. Passes the packets to the next IPQoS module that is specified in the IPQoS configuration file, or returns the packets to the network stream

For an overview of the classifier, refer to "Classifier (`ipgpc`) Overview" on page 23. For information on invoking the classifier in the IPQoS configuration file, refer to "IPQoS Configuration File" on page 108.

Selectors

`ipgpc` supports a variety of selectors that you can use in the filter clause of the IPQoS configuration file. When you define a filter, always use the minimum number of selectors that are needed to successfully retrieve traffic of a particular class. The amount of filters you define can impact IPQoS performance.

The next table lists the selectors available for `ipgpc`.

TABLE 6–1 Filter Selectors for the IPQoS Classifier

Selector	Argument(s)	Information Selected
`saddr`	IP address number.	Source address.
`daddr`	IP address number.	Destination address.

TABLE 6–1 Filter Selectors for the IPQoS Classifier *(Continued)*

Selector	Argument(s)	Information Selected
`sport`	Either a port number or service name, as defined in `/etc/services`.	Source port from which a traffic class originated.
`dport`	Either a port number or service name, as defined in `/etc/services`.	Destination port to which a traffic class is bound.
`protocol`	Either a protocol number or protocol name, as defined in `/etc/protocols`.	Protocol to be used by this traffic class.
`dsfield`	DS codepoint. Default is zero (0).	DS codepoint, which defines any forwarding behavior to be applied to the packet.
`if_name`	Interface name.	Interface to be used for either incoming or outgoing traffic of a particular class.
`if_groupname`	Interface group name.	Interface group to be used for either incoming or outgoing traffic of a particular class.
`user`	Number of the UNIX userID or user name to be selected. If no userID or user name is on the packet, the default –1 is used.	UserID that is supplied to an application.
`projid`	Number of the project ID to be selected.	Project ID that is supplied to an application.
`priority`	Priority number. Lowest priority is 0.	Priority that is given to packets of this class. Priority is used to order the importance of filters for the same class.
`direction`	Argument can be one of the following:	Direction of packet flow on the IPQoS machine.
	`LOCAL_IN`	Input traffic local to the IPQoS system.
	`LOCAL_OUT`	Output traffic local to the IPQoS system.
	`FWD_IN`	Input traffic to be forwarded.
	`FWD_OUT`	Output traffic to be forwarded.
	0	Wildcard that represents either `LOCAL_IN` and `LOCAL_OUT`, or `FORWARD_IN` and `FORWARD_OUT`.
`precedence`	Precedence value. Highest precedence is 0.	Precedence is used to order filters with the same priority.

TABLE 6–1 Filter Selectors for the IPQoS Classifier *(Continued)*

Selector	Argument(s)	Information Selected
`ip_version`	`v4` or `v6`	Addressing scheme that is used by the packets, either IPv4 or IPv6.

Meter Module

The *meter* tracks the transmission rate of flows on a per-packet basis and determines whether the packet conforms to the configured parameters. The meter module determines the next action for a packet from a set of actions that depend on packet size, configured parameters, and flow rate.

The meter consists of two metering modules, `tokenmt` and `tswtclmt`, which you configure in the IPQoS configuration file. You can configure either module or both modules for a class.

When you configure a metering module, you can define two parameters for rate:

- *committed rate* – Defines the acceptable transmission rate in bits per second for packets of a particular class
- *peak rate* – Defines the maximum transmission rate in bits per second that is allowable for packets of a particular class

A metering action on a packet can result in one of the following three outcomes:

- *green* – The packet causes the flow to remain within its committed rate.
- *yellow* – The packet causes the flow to exceed its committed rate but not its peak rate.
- *red* – The packet causes the flow to exceed its peak rate.

You can configure each outcome with different actions in the IPQoS configuration file. Committed rate and peak rate are explained in the next section.

tokenmt Metering Module

The `tokenmt` module uses *token buckets* to measure the transmission rate of a flow. You can configure `tokenmt` to operate as a single-rate or two-rate meter. A `tokenmt` action instance maintains two token buckets that determine whether the traffic flow conforms to configured parameters.

The `tokenmt`(7ipp) man page explains how IPQoS implements the token meter paradigm. You can find more general information about token buckets in Kalevi Kilkki's *Differentiated Services for the Internet* and on a number of Web sites.

Configuration parameters for `tokenmt` are as follows:

- `committed_rate` – Specifies the committed rate of the flow in bits per second.
- `committed_burst` – Specifies the committed burst size in bits. The `committed_burst` parameter defines how many outgoing packets of a particular class can pass onto the network at the committed rate.
- `peak_rate` – Specifies the peak rate in bits per second.
- `peak_burst` – Specifies the peak or excess burst size in bits. The `peak burst` parameter grants to a traffic class a peak-burst size that exceeds the committed rate.
- `color_aware` – Turns on color-aware mode for `tokenmt`.
- `color_map` – Defines an integer array that maps DSCP values to green, yellow, or red.

Configuring `tokenmt` *as a Single-Rate Meter*

To configure `tokenmt` as a single-rate meter, do not specify a `peak_rate` parameter for `tokenmt` in the IPQoS configuration file. To configure a single-rate `tokenmt` instance to have a red, green, or yellow outcome, you must specify the `peak_burst` parameter. If you do not use the `peak_burst` parameter, you can configure `tokenmt` to have only a red or green outcome. For an example of a single-rate `tokenmt` with two outcomes, see Example 3–3.

When `tokenmt` operates as a single-rate meter, the `peak_burst` parameter is actually the excess burst size. `committed_rate` and either `committed_burst` or `peak_burst` must be nonzero positive integers.

Configuring `tokenmt` *as a Two-Rate Meter*

To configure `tokenmt` as a two-rate meter, specify a `peak_rate` parameter for the `tokenmt` action in the IPQoS configuration file. A two-rate `tokenmt` always has the three outcomes, red, yellow, and green. The `committed_rate`, `committed_burst`, and `peak_burst` parameters must all be non-zero positive integers.

Configuring `tokenmt` *to Be Color Aware*

To configure a two-rate `tokenmt` to be color aware, you must add parameters to specifically add "color awareness." The following is an example action statement that configures color-aware `tokenmt`.

EXAMPLE 6–1 Color-Aware `tokenmt` Action for the IPQoS Configuration File

```
action {
    module tokenmt
    name meter1
    params {
          committed_rate 4000000
          peak_rate 8000000
```

EXAMPLE 6–1 Color-Aware `tokenmt` Action for the IPQoS Configuration File *(Continued)*

```
            committed_burst 4000000
            peak_burst 8000000
            global_stats true
            red_action_name continue
            yellow_action_name continue
            green_action_name continue
            color_aware true
            color_map {0-20,22:GREEN;21,23-42:RED;43-63:YELLOW}
    }
}
```

You turn on color awareness by setting the `color_aware` parameter to true. As a color-aware meter, `tokenmt` assumes that the packet has already been marked as red, yellow, or green by a previous `tokenmt` action. Color-aware `tokenmt` evaluates a packet by using the DS codepoint in the packet header in addition to the parameters for a two-rate meter.

The `color_map` parameter contains an array into which the DSCP in the packet header is mapped. Consider the following `color_map` array:

```
color_map {0-20,22:GREEN;21,23-42:RED;43-63:YELLOW}
```

Packets with a DSCP of 0–20 and 22 are mapped to green. Packets with a DSCP of 21 and 23–42 are mapped to red. Packets with a DSCP of 43–63 are mapped to yellow. `tokenmt` maintains a default color map, but you can change it as needed by using the `color_map` parameters.

In the *color*`_action_name` parameters, you can specify `continue` to complete processing of the packet. Or you can add an argument to send the packet to a marker action, for example, `yellow_action_name mark22`.

`tswtclmt` Metering Module

The `tswtclmt` metering module estimates average bandwidth for a traffic class by using a time-based rate estimator. `tswtclmt` always operates as a three-outcome meter. The rate estimator provides an estimate of the flow's arrival rate. This rate should approximate the running average bandwidth of the traffic stream over a specific period or time, its *window*. The rate estimation algorithm is taken from RFC 2859, *A Time Sliding Window Three Colour Marker*.

You use the following parameters to configure `tswtclmt`:

- `committed_rate` – Specifies the committed rate in bits per second
- `peak_rate` – Specifies the peak rate in bits per second
- `window` – Defines the time window, in milliseconds over which history of average bandwidth is kept

For technical details on `tswtclmt`, refer to the `tswtclmt`(7ipp) man page. For general information on rate shapers similar to `tswtclmt`, see RFC 2963, *A Rate Adaptive Shaper for Differentiated Services*.

Marker Module

IPQoS includes two marker modules, `dscpmk` and `dlcosmk`. This section contains information for using both markers. Normally, you should use `dscpmk` because `dlcosmk` is only available for IPQoS systems with VLAN devices.

For technical information about `dscpmk`, refer to the `dscpmk`(7ipp) man page. For technical information about `dlcosmk`, refer to the `dlcosmk`(7ipp) man page.

Using the `dscpmk` Marker for Forwarding Packets

The marker receives traffic flows after they are processed by the classifier or metering modules. The marker marks the traffic with a forwarding behavior, which is the action to be taken on the flows after they leave the IPQoS system. Forwarding behavior to be taken on a traffic class is defined in the per-hop behavior (PHB). The PHB assigns a priority to a traffic class, which indicates the precedence flows of that class have in relation to other traffic classes. PHBs only govern forwarding behaviors on the IPQoS system's contiguous network. For more information on PHBs, refer to "Per-Hop Behaviors" on page 27.

Packet forwarding is the process of sending traffic of a particular class to its next destination on a network. For a host, such as an IPQoS system, a packet is forwarded from the host to the local network stream. For a diffserv router, a packet is forwarded from the local network to the router's next hop.

The marker marks the DS field in the packet header with a well-known forwarding behavior that is defined in the IPQoS configuration file. Thereafter, the IPQoS system and subsequent diffserv-aware systems forward the traffic as indicated in the DS field until the mark changes. To assign a PHB, the IPQoS system marks the DS field of the packet header with a value that is called the differentiated services (DS) codepoint, or DSCP. The diffserv architecture defines two types of forwarding behaviors, EF and AF, which use differing DS codepoints. For overview information about DS codepoints, refer to "DS Codepoint (DSCP)" on page 27.

The IPQoS system reads the DS codepoint for the traffic flow and evaluates the flow's precedence in relation to other outgoing traffic flows. The IPQoS system then prioritizes all concurrent traffic flows and releases each flow onto the network by its priority.

The diffserv router receives the outgoing traffic flows and reads the DS field in the packet headers. The DS codepoint enables the router to prioritize and schedule the concurrent traffic flows and forward each flow by the priority that is indicated by the PHB. Note that the PHB cannot apply beyond the boundary router of the network unless diffserv-aware systems on subsequent hops also recognize the same PHB.

Expedited Forwarding (EF) PHB

Expedited forwarding (EF) guarantees that any packets that are marked with the recommended EF codepoint 46 (101110) receive the best treatment available on release to the network. EF forwarding is often compared to a leased line. Packets with the 46 (101110) codepoint are guaranteed preferential treatment by all diffserv routers en route to the packets' destination. For technical information about EF, refer to RFC 2598, *An Expedited Forwarding PHB*.

Assured Forwarding (AF) PHB

Assured forwarding (AF) provides four different classes of forwarding behaviors that you can specify to the marker. The next table shows the classes, the three drop precedences that are provided with each class, and the recommended DCSPs that are associated with each precedence. Each DSCP is represented by its AF value, its value in hexadecimal, and its value in binary.

TABLE 6–2 Assured Forwarding Codepoints

	Class 1	Class 2	Class 3	Class 4
Low-Drop Precedence	AF11 = 10 (001010)	AF21 = 18 (010010)	AF31 = 26 (011010)	AF41 = 34 (100010)
Medium-Drop Precedence	AF12 = 12 (001100)	AF22 = 20 (010100)	AF32 = 28 (011100)	AF42 = 36 (100100)
High-Drop Precedence	AF13 = 14 (001110)	AF23 = 010110	AF33 = 30 (011110)	AF43 = 38 (100110)

Any diffserv-aware system can use the AF codepoint as a guide for providing differentiated forwarding behaviors to different classes of traffic.

For example, suppose your QoS policy assigns DSCPs of AF31 and AF13 to two different traffic classes. When packets that are marked AF31 (011010) leave the IPQoS system, they receive lower forwarding probability than the packets with AF13 (001110).

When these packets reach a diffserv router, the router evaluates the packets' codepoints along with DS codepoints of other traffic in the queue. The router then forwards or drops packets, depending on the available bandwidth and the priorities that are assigned by the packets' DS codepoints. Note that packets that are marked with the EF PHB are guaranteed bandwidth over packets that are marked with the various AF PHBs.

Coordinate packet marking between any IPQoS systems on your network and the diffserv router to ensure that packets are forwarded as expected. For example, suppose IPQoS systems on your network marks packets with AF21 (010010), AF13 (001110), AF43 (100110), and EF (101110) codepoints. You then need to add the AF21, AF13, AF43, and EF DS codepoints to the appropriate file on the diffserv router.

For a technical explanation of the AF codepoint table, refer to RFC 2597. Router manufacturers Cisco Systems and Juniper Networks have detailed information about setting the AF PHB in their Web sites. You can use this information to define AF PHBs for IPQoS systems as well as routers. Additionally, router manufacturers' documentation contains instructions for setting DS codepoints on their equipment.

Supplying a DS Codepoint to the Marker

The DS codepoint is 6 bits in length. The DS field is 1 byte long. When you define a DS codepoint in the IPQoS configuration file, the marker marks the first 6 significant bits of the packet header with the DS codepoint. The remaining 2 least-significant bits are unused.

To define a DS codepoint, you use the following parameter within a marker action statement:

`dscp_map{0-63:`*DS_codepoint*`}`

The `dscp_map` parameter is a 64-element array, which you populate with the *DS codepoint* (DSCP) value. `dscp_map` is used to map incoming DSCPs to outgoing DSCPs that are applied by the `dscpmk` marker.

You must specify the DSCP value to `dscp_map` in hexadecimal notation. For example, you must translate the EF codepoint of 101110 into the hexadecimal value 46, which results in `dscp_map{0-63:46}`. For AF codepoints, you must translate the various codepoints that are shown in Table 6–2 to hexadecimal for use with `dscp_map`.

Using the `dlcosmk` Marker With VLAN Devices

The `dlcosmk` marker module marks a forwarding behavior in the MAC header of a datagram. You can use `dlcosmk` only on an IPQoS system with a VLAN interface.

`dlcosmk` adds four bytes, which are known as the *VLAN tag*, to the MAC header. The VLAN tag includes a 3-bit user priority value, which is defined by the IEEE 801.D standard. Diffserv-aware switches that understand VLAN can read the user priority field in a datagram. The 801.D user priority values implement the class of service (CoS) marks, which are well known and understood by commercial switches.

You can use the user priority values in `dlcosmk` marker action by defining the class of service marks that are listed in the next table.

TABLE 6–3 801.D User Priority Values

Class of Service	Definition
0	Best effort
1	Background
2	Spare
3	Excellent effort
4	Controlled load
5	Video less than 100ms latency
6	Video less than 10ms latency
7	Network control

For more information on `dlcosmk`, refer to the `dlcosmk`(7ipp) man page.

IPQoS Configuration for Systems With VLAN Devices

This section introduces a simple network scenario that shows how to implement IPQoS on systems with VLAN devices. The scenario includes two IPQoS systems, `machine1` and `machine2`, that are connected by a switch. The VLAN device on `machine1` has the IP address 10.10.8.1. The VLAN device on `machine2` has the IP address 10.10.8.3.

The following IPQoS configuration file for `machine1` shows a simple solution for marking traffic through the switch to `machine2`.

EXAMPLE 6–2 IPQoS Configuration File for a System With a VLAN Device

```
fmt_version 1.0
action {
        module ipgpc
          name ipgpc.classify

        filter {
                name myfilter2
                daddr 10.10.8.3
```

EXAMPLE 6–2 IPQoS Configuration File for a System With a VLAN Device *(Continued)*

```
                class myclass
        }

        class {
                name myclass
                next_action mark4
        }
}

action {
        name mark4
        module dlcosmk
        params {
                cos 4
                next_action continue
        global_stats true
        }
}
```

In this configuration, all traffic from `machine1` that is destined for the VLAN device on `machine2` is passed to the `dlcosmk` marker. The `mark4` marker action instructs `dlcosmk` to add a VLAN mark to datagrams of class `myclass` with a CoS of 4. The 4 user priority value indicates that the switch between the two machines should give controlled load forwarding to `myclass` traffic flows from `machine1`.

`flowacct` Module

The IPQoS `flowacct` module records information about traffic flows, a process that is referred to as *flow accounting*. The results of flow accounting are data that can be used for billing customers or for evaluating the amount of traffic to a particular class.

Flow accounting is optional. `flowacct` is typically the final module that metered or marked traffic flows might encounter before release onto the network stream. For an illustration of `flowacct`'s position in the diffserv model, see Figure 1–1. For detailed technical information about `flowacct`, refer to the `flowacct`(7ipp) man page.

To enable flow accounting, you need to use the Solaris `exacct` accounting facility and the `acctadm` command, as well as `flowacct`. For the overall steps in setting up flow accounting, refer to Table 5–1.

`flowacct` Parameters

`flowacct` gathers information about flows in a *flow table* that is composed of *flow records*. Each entry in the table contains one flow record. You cannot display a flow-account table.

In the IPQoS configuration file, you define the following `flowacct` parameters to measure flow records and to write them to the table:

- **timer** – Defines an interval, in milliseconds, when timed-out flows are removed from the flow table and written to the file that is created by `acctadm`
- **timeout** – Defines a interval, in milliseconds, that specifies how long a packet flow must be inactive before it times out

---

Note – You can configure `timer` and `timeout` to have different values.

- **max_limit** – Places an upper limit on the number of flow records that can be stored in the table

For an example of how `flowacct` parameters are used in the IPQoS configuration file, refer to "How to Configure Flow Control in the IPQoS Configuration File" on page 75.

Flow Record Table

The `flowacct` module maintains a flow table that records all packet flows that are seen by a `flowacct` instance. A flow is identified by the following parameters, which comprise the `flowacct` 8–tuple.

- Source address
- Destination address
- Source port
- Destination port
- DSCP
- User ID
- Project ID
- Protocol

If all the parameters of the 8–tuple for a flow remain the same, the flow table contains only one entry. The `max_limit` parameter determines the number of entries that a flow table can contain.

The flow table is scanned at the interval that is specified in the IPQoS configuration file for the `timer` parameter. The default is 15 seconds. A flow "times out" when its packets are not seen by the IPQoS system for at least the `timeout` interval in the IPQoS configuration file. The default time-out interval is 60 seconds. Entries that have timed out are then written to the accounting file that is created with the `acctadm` command.

`flowacct` Records

A `flowacct` record contains the following attributes.

TABLE 6–4 Attributes of a `flowacct` Record

Attribute Name	Attribute Contents	Type
`src -addr-`*address_type*	Source address of the originator. *address_type* is either v4 for IPv4 or v6 for IPv6, as specified in the IPQoS configuration file.	Basic
`dest_ addr_`*address_type*	Destination address for the packets. *address_type* is either v4 for IPv4 or v6 for IPv6, as specified in the IPQoS configuration file.	Basic
`src- port`	Source port from which the flow originated.	Basic
`dest-port`	Destination port number to which this flow is bound.	Basic
`protocol`	Protocol number for the flow.	Basic
`total-packets`	Number of packets in the flow.	Basic
`total-bytes`	Number of bytes in the flow.	Basic
action_name	Name of the `flowacct` action that recorded this flow.	Basic
`creation_time`	First time that a packet is seen for the flow by `flowacct`.	Extended only
`last_seen`	Last time that a packet of the flow was seen.	Extended only
`diffserv-field`	DS codepoint in the outgoing packet headers of the flow.	Extended only
`user`	Either a UNIX UserID or user name, which is obtained from the application.	Extended only
`projid`	Project ID, which is obtained from the application.	Extended only

Using `acctadm` with the `flowacct` Module

You use the `acctadm` command to create a file in which to store the various flow records that are generated by `flowacct`. `acctadm` works in conjunction with the extended accounting facility. For technical information about `acctadm`, refer to the `acctadm`(1M) man page.

`flowacct` observes flows and fills its table with flow records. `flowacct` then evaluates its parameters and attributes in the interval that is specified by `timer`. When a packet is not seen for at least the `last_seen` plus `timeout` values, the packet times out. All timed-out entries are deleted from the flow table. They are then written to the accounting file each time the interval that is specified in the `timer` parameter elapses.

To invoke `acctadm` for use with the `flowacct` module, use the following syntax:

```
acctadm -e type -f filename flow
```

`acctadm -e`	Invokes `acctadm` with the `-e` option. The -e indicates that a resource list follows.
type	Specifies the attributes to be gathered. *file-type* must be replaced by either `basic` or `extended`. For a list of attributes in each file type, refer to Table 6–4.
`-f` *file_name*	Creates the file *file_name* to hold the flow records.
`flow`	Indicates that `acctadm` is to be run with IPQoS.

IPQoS Configuration File

This section contains full details about the parts of the IPQoS configuration file. The IPQoS boot-time activated policy is stored in the file `/etc/inet/ipqosinit.conf`. Although you can edit this file, the best practice for a new IPQoS system is to create a configuration file with a different name. Tasks for applying and debugging an IPQoS configuration are in Chapter 4.

The syntax of the IPQoS configuration file is shown in the next example. The example uses the following conventions:

- `computer-style type` - Syntactical information that is provided to explain the parts of the configuration file. You do not type any text that appears in computer style type.
- **`bold type`** - Literal text that you must type in the IPQoS configuration file. For example, you must always begin the IPQoS configuration file with **`fmt_version`**.
- *italics type* - Variable text that you replace with descriptive information about your configuration. For example, you must always replace *action_name* or *module_name* with information that pertains to your configuration.

EXAMPLE 6–3 Syntax of the IPQoS Configuration File

```
file_format_version ::= fmt_version version

action_clause ::= action {
      name action_name
      module module_name
      params_clause | ""
      cf_clauses
}
action_name ::= string
```

EXAMPLE 6–3 Syntax of the IPQoS Configuration File *(Continued)*

```
module_name ::= ipgpc | dlcosmk | dscpmk | tswtclmt | tokenmt | flowacct

params_clause ::= params {
     parameters
     params_stats |    ""
     }
parameters ::=     prm_name_value parameters |   ""
prm_name_value ::= param_name param_value

params_stats ::= global_stats boolean

cf_clauses ::= class_clause cf_clauses |
               filter_clause cf_clauses | ""

class_clause ::= class {
     name class_name
     next_action next_action_name
     class_stats | ""
                 }
class_name  ::= string
next_action_name  ::= string
class_stats ::= enable_stats boolean
boolean ::= TRUE | FALSE

filter_clause ::= filter {
                name filter_name
                class class_name
                parameters
                }
filter_name ::= string
```

The remaining text describes each major part of the IPQoS configuration file.

action Statement

You use `action` statements to invoke the various IPQoS modules that are described in "IPQoS Architecture and the diffserv Model" on page 95.

When you begin the IPQoS configuration file, you must always begin with the version number. Then, you must add the following action to invoke the classifier:

```
fmt_version 1.0

action {
    module ipgpc
    name ipgpc.classify
}
```

Follow the classifier action statement with a `params` clause or a `class` clause.

Use the following syntax for all other action statements:

```
action {
name action_name
module module_name
params_clause | ""
cf_clauses
}
```

Statement	Definition
`name` *action_name*	Assigns a name to the action
`module` *module_name*	Identifies the IPQoS module to be invoked, which must be one of the modules in Table 6–5
params_clause	Can be parameters for the classifier to process, such as global statistics or the next action to process
cf_clauses	A set of zero or more `class` clauses or `filter` clauses

Module Definitions

The module definition indicates which module is to process the parameters in the action statement. The IPQoS configuration file can include the following modules.

TABLE 6–5 IPQoS Modules

Module Name	Definition
`ipgpc`	IP classifier
`dscpmk`	Marker to be used to create DS codepoints in IP packets
`dlcosmk`	Marker to be used with VLAN devices
`tokenmt`	Token bucket meter
`tswtclmt`	Time-sliding window meter
`flowacct`	Flow-accounting module

Class Clause

You define a *class clause* for each class of traffic.

Use this syntax to define the remaining classes in the IPQoS configuration.

```
class {
        name class_name
```

```
        next_action next_action_name
}
```

To enable statistics taking on a particular class, you must first enable global statistic in the `ipgpc.classify` action statement. For more information, refer to "action Statement" on page 109.

Use the `enable_stats TRUE` statement whenever you want to turn on statistics for a class. If you do not need to gather statistics for a class, you can specify `enable_stats FALSE`. Alternatively, you can eliminate the `enable_stats` statement.

Traffic on an IPQoS-enabled network that you do not specifically define is relegated to the *default class*.

Filter Clause

Filters are made up of selectors that group traffic flows into classes. These selectors specifically define the criteria to be applied to traffic of the class that was created in the class clause. If a packet matches all selectors of the highest-priority filter, the packet is considered to be a member of the filter's class. For a complete list of selectors that you can use with the `ipgpc` classifier, refer to Table 6–1.

You define filters in the IPQoS configuration file by using a *filter clause*, which has the following syntax:

```
filter {
        name filter_name
        class class_name
        parameters (selectors)
        }
```

Params Clause

The `params` clause contains processing instructions for the module that is defined in the action statement. Use the following syntax for the `params` clause:

```
params {
            parameters
            params_stats |  ""
        }
```

In the `params` clause, you use parameters that are applicable to the module.

The *params_stats* value in the `params` clause is either `global_stats TRUE` or `global_stats FALSE`. The `global_stats TRUE` instruction turns on UNIX-style statistics for the action statement where global statistics is invoked. You can view the statistics by using the `kstat` command. You must enable action statement statistics before you can enable per-class statistics.

ipqosconf Configuration Utility

You use the `ipqosconf` utility to read the IPQoS configuration file and configure IPQoS modules in the UNIX kernel. `ipqosconf` can perform the following actions:

- Apply the configuration file to the IPQoS kernel modules (`ipqosconf -a`) *filename*
- List the IPQoS configuration file currently resident in the kernel (`ipqosconf -l`)
- Ensure that the current IPQoS configuration is read and applied each time the machine reboots (`ipqosconf -c`)
- Flush the current IPQoS kernel modules (`ipqosconf -f`)

For technical information, refer to the `ipqosconf`(1M) man page.

Glossary

class A group of network flows that share similar characteristics. You define classes in the IPQoS configuration file.

diffserv model Internet Engineering Task Force architectural standard for implementing differentiated services on IP networks. The major modules are classifier, meter, marker, scheduler, and dropper. IPQoS implements the classifier, meter, and marker modules. The diffserv model is described in RFC 2475, *An Architecture for Differentiated Services*.

DS codepoint (DSCP) A 6–bit value that, when included in the DS field of an IP header, indicates how a packet must be forwarded.

filter A set of rules that define the characteristics of a class in the IPQoS configuration file. The IPQoS system selects for processing any traffic flows that conform to the filters in its IPQoS configuration file.

flow accounting The process of accumulating and recording information about traffic flows. You establish flow accounting by defining parameters for the `flowacct` module in the IPQoS configuration file.

IPQoS Software feature in Solaris 9, 9/02 that provides an implementation of the diffserv standard, plus flow accounting and 802.1 D marking for virtual LANs. Using IPQoS, you can provide different levels of network services to customers and applications, as defined in the IPQoS configuration file.

marker 1. A module in the diffserv architecture and IPQoS that marks the DS field of an IP packet with a value that indicates how the packet is to be forwarded. In the IPQoS implementation, the marker module is `dscpmk`.

	2. A module in the IPQoS implementation, which marks the virtual LAN tag of an Ethernet datagram with a user priority value. The user priority value indicates how datagrams are to be forwarded on a network with VLAN devices. This module is called `dlcosmk`.
meter	A module in the diffserv architecture that measures the rate of traffic flow for a particular class. The IPQoS implementation includes two meters, `tokenmt` and `tswtclmt`.
outcome	Action to take as a result of metering traffic. The IPQoS meters have three outcomes, red, yellow, and green, which you define in the IPQoS configuration file.
per-hop behavior (PHB)	A priority that is assigned to a traffic class. The PHB indicates the precedence which flows of that class have in relation to other traffic classes.
selector	Element that specifically defines the criteria to be applied to packets of a particular class in order to select that traffic from the network stream. You define selectors in the filter clause of the IPQoS configuration file.
user-priority	A 3-bit value that implements class of service marks, which define how Ethernet datagrams are forwarded on a network of VLAN devices.
virtual LAN (VLAN) device	Network interfaces that provide traffic forwarding at the Ethernet (data link) level of the IP protocol stack.

Index

E

F

H

I

K

M

N

P

Q

R

S

T

U

V

W