Managing IP Quality of Service in Oracle® Solaris 11.2

Part No: E36826
July 2014

Contents

Using This Documentation 7

1 Introducing IPQoS 9
IPQoS Basics 9
What Are Differentiated Services? 9
IPQoS Features 10
Where to Get More Information 10
Providing Quality of Service With IPQoS 11
Implementing Service-Level Agreements 11
Assuring Quality of Service for an Individual Organization 11
Introducing the Quality-of-Service Policy 12
Improving Network Efficiency With IPQoS 12
How Bandwidth Affects Network Traffic 13
Using Classes of Service to Prioritize Traffic 13
Differentiated Services Model 15
Classifier (`ipgpc`) Overview 15
Meter (`tokenmt` and `tswtclmt`) Overview 16
Marker (`dscpmk` and `dlcosmk`) Overview 17
Flow Accounting (`flowacct`) Overview 17
How Traffic Flows Through the IPQoS Modules 18
Traffic Forwarding on an IPQoS-Enabled Network 19
DS Codepoint 19
Per-Hop Behaviors 19

2 Planning for an IPQoS-Enabled Network 23
General IPQoS Configuration Planning Task Map 23
Planning the Diffserv Network Topology 24
Hardware Strategies for the Diffserv Network 24
IPQoS Network Topologies 24
Planning the Quality-of-Service Policy 27

QoS Policy Planning Aids 27
QoS Policy Planning Task Map 28
Preparing a Network for IPQOS 29
Defining the Classes for Your QoS Policy 29
Defining Filters 30
▼ How to Define Filters in the QoS Policy 31
Planning Flow Control 32
Planning Forwarding Behavior 34
▼ How to Plan Forwarding Behavior 35
Planning for Flow Accounting 36
Introducing the IPQoS Configuration Example 36
IPQoS Topology 37

3 Creating the IPQoS Configuration File Tasks 39
Defining a QoS Policy Task Map 39
Tools for Creating a QoS Policy 40
Basic IPQoS Configuration File 41
Creating IPQoS Configuration Files for Web Servers 41
▼ How to Create the IPQoS Configuration File and Define Traffic Classes 43
▼ How to Define Filters in the IPQoS Configuration File 46
▼ How to Define Traffic Forwarding in the IPQoS Configuration File 48
▼ How to Enable Accounting for a Class in the IPQoS Configuration File 51
▼ How to Create an IPQoS Configuration File for a Best-Effort Web Server 53
Creating an IPQoS Configuration File for an Application Server 56
▼ How to Configure the IPQoS Configuration File for an Application Server 58
▼ How to Configure Forwarding for Application Traffic in the IPQoS Configuration File 61
▼ How to Configure Flow Control in the IPQoS Configuration File 63
Providing Differentiated Services on a Router 66

4 Starting and Maintaining IPQoS Tasks 69
Administering IPQoS 69
▼ How to Add the `ipqos` Package 69
▼ How to Start the `ipqos` Service 70
▼ How to Enable Logging of IPQoS Messages During Booting 71
Troubleshooting IPQoS Error Messages 72

5 Using Flow Accounting and Statistics Gathering Tasks 77
Recording Information About Traffic Flows 77
▼ How to Create a File for Flow-Accounting Data 78
Gathering Statistical Information 80

6 IPQoS in Detail Reference 83
IPQoS Architecture and the Diffserv Model 83
Classifier Module 84
Meter Module 86
Marker Module 88
`flowacct` Module 93
IPQoS Configuration File 95
`action` Statement 96
Module Definitions 97
`class` Clause 97
`filter` Clause 98
`params` Clause 98

Index 101

Using This Documentation

- **Overview** – Describes how to configure the IPQos service
- **Audience** – Technicians, system administrators, and authorized service providers
- **Required knowledge** –Some experience working with Oracle Solaris is useful

Product Documentation Library

Late-breaking information and known issues for this product are included in the documentation library at http://www.oracle.com/pls/topic/lookup?ctx=E36784.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Introducing IPQoS

IP Quality of Service (IPQoS) enables you to prioritize, control, and gather accounting statistics. Using IPQoS, you can provide consistent levels of service to users of your network. You can also manage traffic to avoid network congestion.

This chapter covers the following topics:

- "IPQoS Basics" on page 9
- "Providing Quality of Service With IPQoS" on page 11
- "Improving Network Efficiency With IPQoS" on page 12
- "Differentiated Services Model" on page 15
- "Traffic Forwarding on an IPQoS-Enabled Network" on page 19

Note - The IPQoS facility might be removed in a future release. You are encouraged instead to use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see "Managing Network Virtualization and Network Resources in Oracle Solaris 11.2 ".

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 Oracle Solaris, 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. The different levels of service are collectively referred to as *differentiated services*. The differentiated services that you provide to customers can be based on a structure of service levels that your company offers to its customers. You can also provide differentiated services based on the priorities that are set for applications or users on your network.

Providing quality of service (QoS) 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:

- 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 and x86 architecture
- Support for IPv4 and IPv6 addressing
- Interoperability with IP Security Architecture (IPsec)
- Support for 802.1D user-priority markings for virtual local area networks (VLANs)

Where to Get More Information

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

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

- RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers (http://www.ietf.org/rfc/rfc2474.txt?number=2474) – Describes an enhancement to the type of service (ToS) field or DS fields of the IPv4 and IPv6 packet headers to support differentiated services
- RFC 2475, An Architecture for Differentiated Services (http://www.ietf.org/rfc/rfc2475.txt?number=2475) – Provides a detailed description of the organization and modules of the Diffserv architecture

IPQoS documentation includes the following man pages:

- `ipqosconf`(1M) - Describes the command for setting up the IPQoS configuration file
- `ipqos`(7ipp) – Describes the IPQoS implementation of the Diffserv architectural model
- `ipgpc`(7ipp) – Describes the IPQoS implementation of a Diffserv classifier
- `tokenmt`(7ipp) – Describes the IPQoS `tokenmt` meter
- `tswtclmt`(7ipp) – Describes the IPQoS `tswtclmt` meter
- `dscpmk`(7ipp) – Describes the DSCP marker module
- `dlcosmk`(7ipp) – Describes the IPQoS 802.1D user-priority marker module
- `flowacct`(7ipp)– Describes the IPQoS flow-accounting module
- `acctadm`(1M) – Describes the command that configures the Oracle Solaris extended accounting facilities and 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 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. All other traffic would receive medium priority 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 Oracle Solaris 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*.
- An action that an IPQoS system and a Diffserv router must apply to a packet flow. This type of 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 described in "Planning the Quality-of-Service Policy" on page 27.

Improving Network Efficiency With IPQoS

IPQoS contains features that can help you make network performance more efficient as you implement quality of service. For example, for an enterprise or institution, you must maintain an efficient network to avoid traffic bottlenecks. You must also ensure that a group or application does not consume more than its allotted bandwidth. For an ISP or ASP, you must manage network performance to ensure that customers receive their paid-for level of network service.

Some symptoms of an overused network include lost data and traffic congestion. Both symptoms result in slow response times. In the past, system administrators handled network traffic problems by adding more bandwidth, which is the maximum amount of data that a fully used network link or device can transfer. However, the level of traffic on the links often varied widely. With IPQoS, you can manage traffic on the existing network and help assess where, and whether, expansion is necessary.

How Bandwidth Affects Network Traffic

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.

Consider the following questions when determining how to effectively manage traffic on your network:

- 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 that must be given highest priority?
- Which applications are sensitive to congestion?
- What are your less critical applications that 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 with individual characteristics and individual priorities. Classes of service form the basic categories on which you base the QoS policy for your organization and represent the traffic groups that you want to control.

When analyzing network traffic, consider the following guidelines:

- **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. One SLA might provide a premium web site as one service level. Another SLA might offer 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 offering popular applications that generate excessive 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 for the QoS policy for a mail server.
- **Does your network run certain applications that require highest-priority forwarding behaviors?**

Any critical applications that require highest-priority forwarding behaviors must receive highest priority in the router's queue. 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 the flows 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 that 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 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 enable statistics gathering on traffic flows that you are monitoring. You could also create classes for traffic to which you restrict access for security reasons.

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.

For an enterprise, you might create classes of service such as the following examples:

- Popular applications such as email and outgoing FTP to a particular server, either of which could constitute a class. Because employees constantly use these applications, your QoS policy might guarantee email and outgoing FTP 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 work 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 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. For detailed information about each module, refer to "IPQoS Architecture and the Diffserv Model" on page 83.

Classifier (`ipgpc`) Overview

In the Diffserv model, the *classifier* selects packets from a network traffic flow. A *traffic flow* consists of 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, `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 84.

IPQoS Classes

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 about how to define classes, see "Defining the Classes for Your QoS Policy" on page 29.

IPQoS Filters

Filters are sets of rules that contain parameters called *selectors*. Each filter must point to a class. IPQoS matches packets against the selectors of each filter to determine whether 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 address and destination addresses
- Source port and destination port
- 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 about creating filters, see "How to Define Filters in the QoS Policy" on page 31.

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 to determine the appropriate outcome. Based on the traffic flow's outcome, the meter selects a subsequent action, such as sending the packet to another action or returning 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` – Uses a two-token bucket metering scheme
- `tswtclmt` – 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's outcome to green, yellow, or red.

For information about defining parameters for the IPQoS meters, refer to "How to Plan Flow Control" on page 32.

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` – Marks the DS field in an IP packet header with a numeric value that is called the *Differentiated Services codepoint*, or *DSCP*. A Diffserv-aware router can then use the DS codepoint to apply the appropriate forwarding behavior to the packet.
- `dlcosmk` – Marks the virtual local area network (VLAN) tag of an Ethernet frame header with a numeric value that is 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 IETF Diffserv model.

For information about implementing a marker strategy for the QoS policy, see "Planning Forwarding Behavior" on page 34.

Flow Accounting (`flowacct`) Overview

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

The `flowacct` module works with the `acctadm` command to create an accounting log file. A basic log includes the IPQoS 5-tuple and two additional attributes:

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

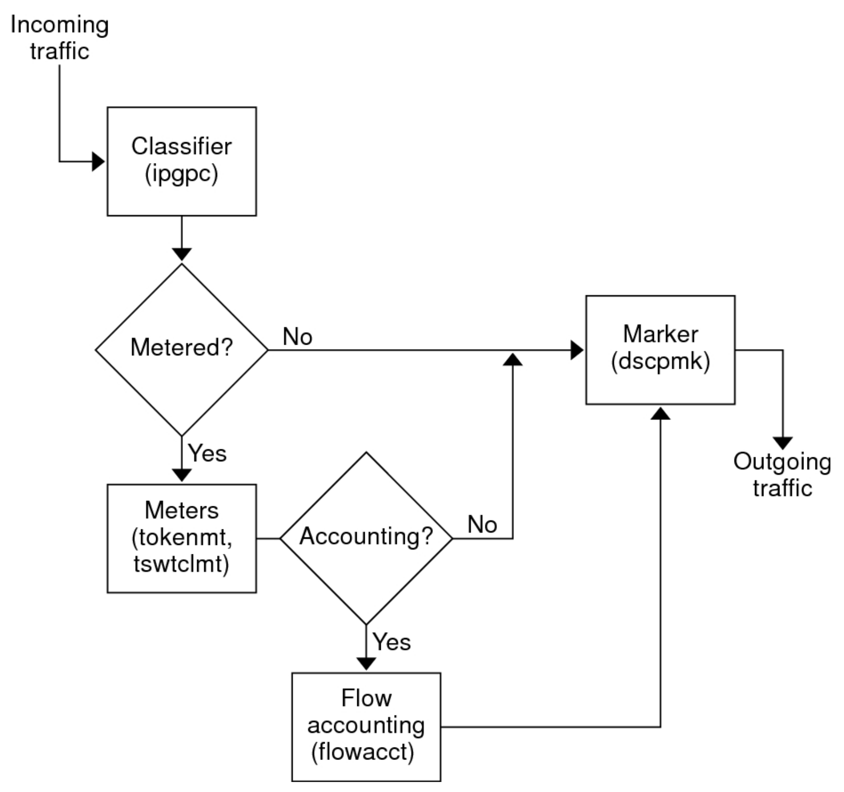
You can also gather statistics on other attributes, as described in "Recording Information About Traffic Flows" on page 77, and in the `flowacct`(7ipp) and `acctadm`(1M) man pages.

For information about planning a flow-accounting strategy, see "Planning for Flow Accounting" on page 36.

How Traffic Flows Through the IPQoS Modules

The following figure shows a path that incoming traffic might take through some of the IPQoS modules.

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



This 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.

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. Then, the meter 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

The DS codepoint (DSCP) 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 for the IPQoS-enabled system and Diffserv router to use. The Diffserv architecture also defines *forwarding behaviors*, which correspond to the DSCPs. 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 that is associated with that DSCP. The packet is then released onto the network.

Note - The `dlcosmk` marker does not use the DSCP. Instead, `dlcosmk` 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 88.

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 that 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 unless another Diffserv system has changed the DSCP. For more information about PHBs, refer to "Using the `dscpmk` Marker for Forwarding Packets" on page 89.

The goal of a PHB is to provide a specified amount of network resources to a class of traffic on the contiguous network. In the QoS policy, you define DSCPs that indicate the precedence levels for traffic classes when the traffic flows leave the IPQoS-enabled system. Precedences can 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 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 (EF) and assured forwarding (AF).

- Expedited forwarding

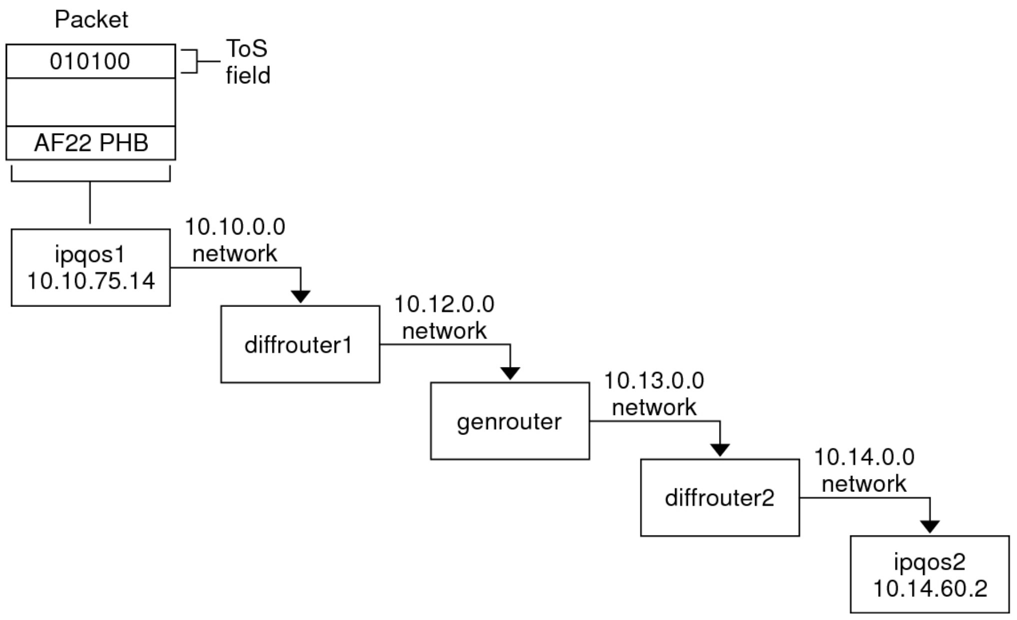
  This per-hop behavior assures that any traffic class with EFs related DSCP is given highest priority. Traffic with an EF DSCP is not queued. EF provides low loss, latency, and jitter. The recommended DSCP for EF is 101110. A packet that is marked with 101110 receives guaranteed low-drop precedence as 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

  This per-hop behavior provides four different forwarding classes that you can assign to a packet. Every forwarding class provides three drop precedences: low-drop, medium-drop, and high-drop. For more detailed information, see Table 6-2. The various AF codepoints provide the ability to assign different levels of service to customers and applications.

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 flow of the packet that is shown in this figure begins with the progress of a packet that originates at host `ipqos1`. The steps then continue through several hops to host `ipqos2`.

1. The user on `ipqos1` runs the `ftp` command to access host `ipqos2`, which is three hops away.
2. `ipqos1` applies its QoS policy to the resulting packet flow. `ipqos1` then 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 2Mb/sec is configured for the `ftp` class.
3. `ipqos-1` meters the `ftp` flow to determine whether the flow exceeds the committed rate of 2 Mbit/sec.
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. `diffrouter1` then checks the DSCP. If `diffrouter1` is congested, packets that are marked with AF22 are dropped.
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. `ipqos2` then prompts the user on `ipqos1` for a user name and password.

♦ ♦ ♦ CHAPTER 2

Planning for an IPQoS-Enabled Network

You can configure IPQoS on any system that runs Oracle Solaris The IPQoS system then works with Diffserv-aware routers to provide differentiated services and traffic management on an intranet.

This chapter contains planning information for adding IPQoS-enabled systems onto a Diffserv-aware network:

- “General IPQoS Configuration Planning Task Map” on page 23
- “Planning the Diffserv Network Topology” on page 24
- “Planning the Quality-of-Service Policy” on page 27
- “QoS Policy Planning Task Map” on page 28
- “Introducing the IPQoS Configuration Example” on page 36

Note - The IPQoS facility might be removed in a future release. Users are encouraged to instead use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see “Managing Network Virtualization and Network Resources in Oracle Solaris 11.2 ”.

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 each system's relationship to the router on the local network. The following task map lists the major planning tasks for implementing IPQoS on your network and links to procedures to complete the tasks.

Task	Description	For Instructions
1. Plan a Diffserv network topology that incorporates IPQoS-enabled systems.	Choose from the various Diffserv network topologies to determine the best solution for your site.	“Planning the Diffserv Network Topology” on page 24.

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 (SLAs).	"Planning the Quality-of-Service Policy" on page 27.
3. Plan the QoS policy for each IPQo S system.	Decide on the classes, metering, and accounting features that are needed to implement each SLA.	"Planning the Quality-of-Service Policy" on page 27.
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. Conversely, you can also run IPQoS on desktop systems, depending on the needs of your network.

The following list describes possible systems for an IPQoS configuration:

- Oracle Solaris systems that offer various services, such as web servers and database servers
- Application 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 local area network (LAN)

You might introduce IPQoS systems into a network topology with already functioning Diffserv-aware routers. 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

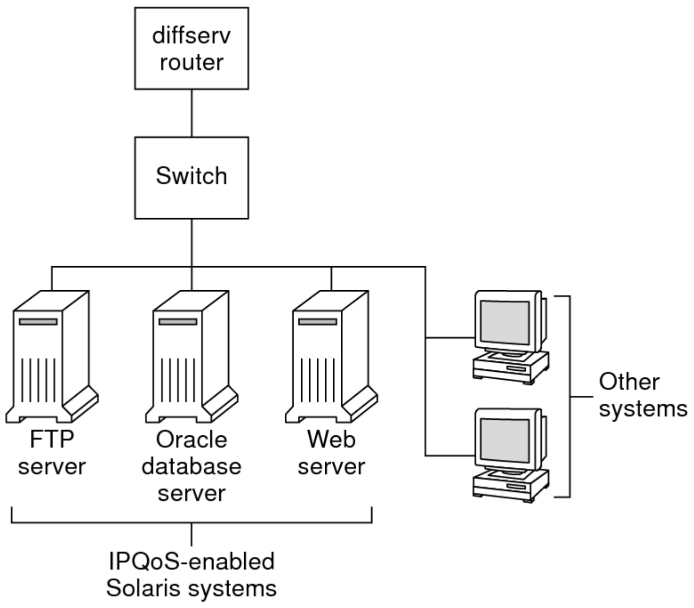
This section illustrates IPQoS strategies for various network needs.

IPQoS on Individual Hosts

The following figure shows a single network of IPQoS-enabled systems. This network is only 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. If you make the router Diffserv aware, you can further control incoming and outgoing traffic.

The examples in this guide use this scenario.

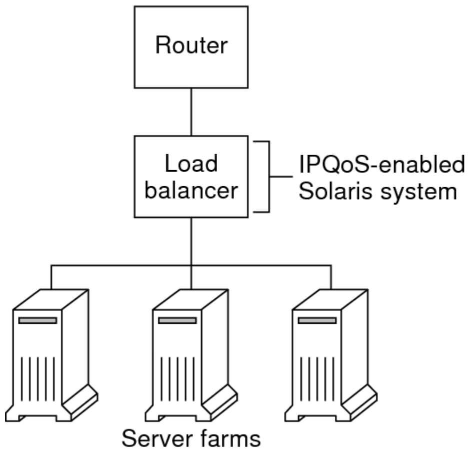
FIGURE 2-1 IPQoS Systems on a Network Segment



IPQoS on a Network of Server Farms

The following figure shows a network with several heterogeneous server farms. In this setup, 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 user ID and project ID. These selectors are included in the application data.

FIGURE 2-2 Network of IPQoS-Enabled Server Farms

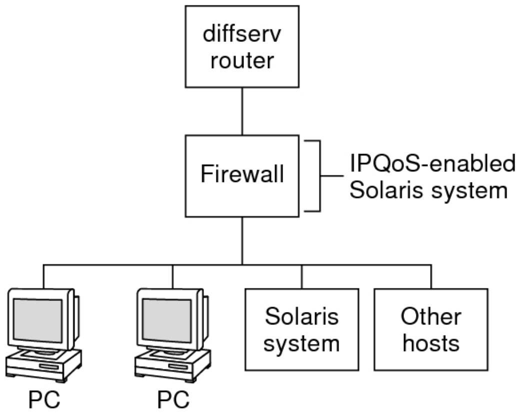


This scenario provides flow control and traffic forwarding to manage congestion on the local network. This scenario also prevents outgoing traffic from the server farms from overloading other portions of the intranet.

IPQoS on a Firewall

The following figure shows a segment of a corporate network that is secured from other segments by a firewall. In this scenario, traffic flows into a Diffserv-aware router where the packets are filtered and queued. All incoming traffic that is forwarded by the router then travels into the IPQoS-enabled firewall. To use IPQoS, the firewall must not bypass the IP forwarding stack.

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



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 (QoS) 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 needed for the IPQoS configuration file. For example, you can use the following template to list the major categories of information to be used in the IPQoS configuration file.

TABLE 2-1 QoS Planning Template

Class	Priority	Filter	Selector	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

Class	Priority	Filter	Selector	Rate	Forwarding	Accounting
Class 1	1	Filter 2	Selector 1 Selector 2	N/A	N/A	N/A
Class 2	2	Filter 1	Selector 1 Selector 2	Meter rates, depending on meter type	Marker drop precedence	Requires flow-accounting statistics
Class 2	2	Filter 2	Selector 1 Selector 2	N/A	N/A	N/A

You can divide each major category to further define the QoS policy. Subsequent sections explain how to obtain information for the categories in the template.

QoS Policy Planning Task Map

The following task map lists the major tasks for planning a QoS policy and links to the instructions to perform each task.

Task	Description	For Instructions
1. Design your network topology to support IPQoS.	Identify the hosts and routers on your network to provide differentiated services.	"Preparing a Network for IPQOS" on page 29
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.	"Defining the Classes for Your QoS Policy" on page 29
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 31
4. Define flow-control rates for measuring traffic as packets leave the IPQoS system.	Determine acceptable flow rates for each class of traffic.	"How to Plan Flow Control" on page 32
5. Define DSCPs 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.	"Planning Forwarding Behavior" on page 34
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.	"Planning for Flow Accounting" on page 36

Note - 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 the router documentation and the router manufacturer's web site.

Preparing a Network for IPQOS

The general planning tasks to do before you create the QoS policy are as follows:

1. Review your network topology. Then, plan a strategy that uses IPQoS systems and Diffserv routers. For topology examples, see "Planning the Diffserv Network Topology" on page 24.
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 policy requirements.
4. Review and perform any planning tasks that are required by the Diffserv router on your network. Refer to the router documentation and the router manufacturer's web site for details.

Defining 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 a different QoS policy for each IPQoS-enabled system. For an overview of classes, see "IPQoS Classes" on page 15.

Before defining classes, you should determine which systems on your network are to be IPQoS-enabled, as identified in "Preparing a Network for IPQOS" on page 29.

1. Create a QoS planning table for organizing the QoS policy information, as shown in Table 2-1.
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.

   For guidelines for analyzing network traffic for possible class definitions, see "Using Classes of Service to Prioritize Traffic" on page 13 .
4. List the classes in the QoS planning table.

5. Assign a priority level to each class.

   For example, use priority level 1 to represent the highest-priority class, and assign descending-level priorities to the remaining classes. This priority level is for organizational purposes only. You can assign the same priority to more than one class if appropriate for your QoS policy.

   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.

   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 31.

Defining Filters

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 uses the criteria in the selectors to extract packets from a traffic flow. The IPQoS system then associates the packets with a class. For an introduction to filters, see "IPQoS Filters" on page 16 on page 16.

Use only as many selectors as you need to extract packets for a class. The more selectors that you define, the greater the impact on IPQoS performance.

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.

TABLE 2-2 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 a user-defined port number.
`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, either IPv4 (the default) or IPv6.
`dsfield`	Contents of the DS field, that is, the DSCP. Use this selector for extracting incoming packets that are already marked with a particular DSCP.

Name	Definition
`priority`	Priority level that is assigned to the class. For more information, see "Defining the Classes for Your QoS Policy" on page 29.
`user`	Either the UNIX user ID or the 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. Acceptable values are either `LOCAL_IN`, `LOCAL_OUT`, `FWD_IN`, or `FWD_OUT`.

▼ How to Define Filters in the QoS Policy

Before You Begin Before you can define filters, you need to define the classes for your QoS policy. For more information, see "Defining the Classes for Your QoS Policy" on page 29 on page 29.

1. **Create at least one filter for each class in your QoS planning table.**

   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. You then can define an appropriate `direction` selector in addition to the basic selectors.

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

   Use the QoS planning table to track the filters for the classes you defined.

Example 2-1 Defining Filters for FTP Traffic

The following example shows how you would define a filter for outgoing FTP traffic.

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

Planning 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 modules. The meters determine the rate at which traffic is released onto the network. For an introduction to the metering modules, see "Meter (`tokenmt` and `tswtclmt`) Overview" on page 16.

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.

▼ How to Plan Flow Control

Before You Begin Before planning flow control, you should have defined filters and selectors, as described in "How to Define Filters in the QoS Policy" on page 31.

1. **Determine the maximum bandwidth for your network.**

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

   To guarantee a certain level of service, you might need to meter certain traffic classes that are generated by the customer.

3. **Review your list of classes to determine whether any classes other than those classes that are associated with SLAs need to be metered.**

   For example, 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.

   ---

4. **Determine which filters in each class select traffic that needs flow control. Then, refine your list of classes that require metering.**

Classes that have more than one filter might require metering for only one filter. For example, if you define filters for incoming and outgoing traffic of a certain class, you might conclude that only traffic in one direction requires flow control.

5. **Choose a meter module for each class to be flow controlled and add the module name to the meter column in your QOS planning table.**

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

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

   - Committed rate
   - Peak rate

   If these rates are sufficient to meter a particular class, you can define only the committed rate and the 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 87. 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 88 and in the `tswtclmt`(7ipp) man page.

7. **Add traffic conformance outcomes for the metered traffic to the planning table.**

   The outcomes for both metering modules are green, red, and yellow. Outcomes for the meters are fully explained in "Meter Module" on page 86.

   You need to determine what action should be taken on traffic that conforms, or does not conform, to the committed rate. Often, but not always, this action is to mark the packet header with a per-hop behavior. 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.

Example 2-2 Defining Meters

The following example shows meter entries for a class of email traffic. The network on which the IPQoS system is located has a total bandwidth of 100 Mbits/sec, or 10000000 bits per second. The QoS policy assigns a low priority to the email class. This class also receives best-effort forwarding behavior.

Class	Priority	Filter	Selector	Rate

`email`	8	`mail_in`	`daddr10.50.50.5`
			`dport imap`
			`direction LOCAL_IN`	

`email`	8	`mail_out`	`saddr10.50.50.5`
			`sport imap`
			`direction LOCAL_ OUT`
				meter=`tokenmt`
				committed rate=5000000
				committed burst =5000000
				peak rate =10000000
				peak burst=1000000
				green precedence=continue processing
				yellow precedence=mark yellow PHB
				red precedence=drop

Planning Forwarding Behavior

Forwarding behavior determines the priority and drop precedence of traffic flows that are about to be forwarded to 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.

Note that 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 91.

To prioritize IP traffic, you need to assign a DSCP to each packet. The `dscpmk` marker marks the DS field of the packet with the DSCP. You choose the DSCP 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 DSCP and forwarding, refer to “Traffic Forwarding on an IPQoS-Enabled Network” on page 19.

▼ How to Plan Forwarding Behavior

Before You Begin Before determining forwarding behavior, you should 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 each class.**

   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 DSCP 46 (101110) are released onto the network before packets 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 89.

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

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

Example 2-3 QoS Policy for a Games Application

The following shows a portion of a QoS policy. This policy defines a class for a popular games application that generates a high level of traffic.

Class	Priority	Filter	Selector	Rate	Forwarding?
`games_app`	9	`games_in`	`sport 6080`	N/A	N/A
`games_app`	9	`games_out`	`dport 6081`	meter=`tokenmt` committed rate= 5000000 committed burst = 5000000 peak rate = 10000000 peak burst= 15000000	green =AF31 yellow=AF42 red=drop

Class	Priority	Filter	Selector	Rate	Forwarding?
				green precedence= continue processing yellow precedence= mark yellow PHB red precedence= drop	

The forwarding behaviors assign low-priority DSCPs to `games_app` traffic that conforms to its committed rate or is under the peak rate. When `games_app` traffic exceeds peak rate, the QoS policy indicates that packets from `games_app` are to be dropped. All AF codepoints are listed in Table 6-2.

Planning for Flow Accounting

You use the IPQoS `flowacct` module to track traffic flows for billing or network management purposes. Your QoS policy should include flow accounting in the following circumstances:

- Your company offers SLAs to customers.

  Review the SLAs to determine what types of network traffic your company wants to bill customers for. Then, review your QoS policy to determine which classes select traffic to be billed.
- You have applications that might need monitoring or testing to avoid network problems.

  Consider using flow accounting to observe the behavior of these applications. Review your QoS policy to determine the classes that you have assigned to traffic that requires monitoring.

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

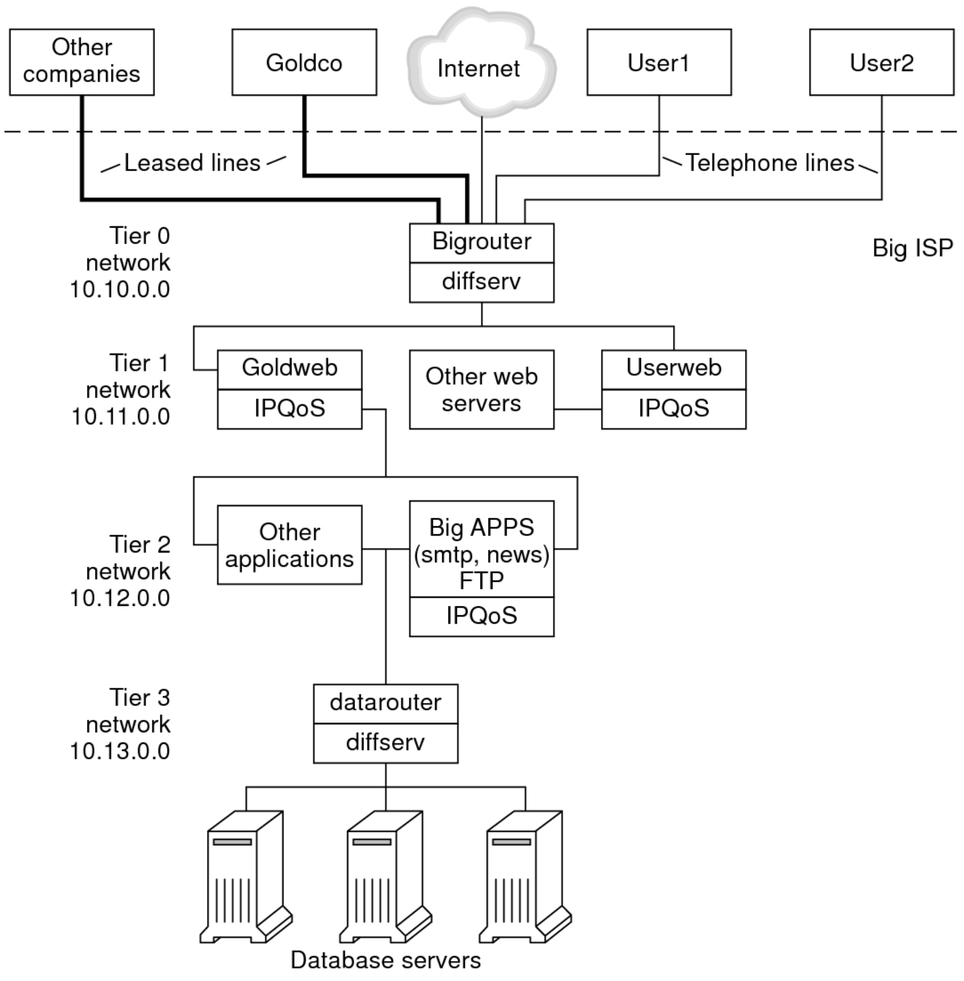
Introducing the IPQoS Configuration Example

This section introduces the example IPQOS configuration that is used in the tasks in the remaining chapters of the guide. The example shows the differentiated services solution on the public intranet of BigISP, a fictitious service provider. BigISP offers services to large companies that reach BigISP through leased lines. Individuals who dial in from modems can also buy services from BigISP.

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 these four tiers in its public intranet:

- **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 that is 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 Goldco 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 the IPQoS configuration file, `/etc/inet/ipqosinit.conf`. It covers the following topics:

- "Defining a QoS Policy Task Map" on page 39
- "Tools for Creating a QoS Policy" on page 40
- "Creating IPQoS Configuration Files for Web Servers" on page 41
- "Creating an IPQoS Configuration File for an Application Server" on page 56
- "Providing Differentiated Services on a Router" on page 66

This chapter assumes that you have defined a complete QoS policy and are ready to use it as the basis for the IPQoS configuration file. For instructions on QoS policy planning, refer to "Planning the Quality-of-Service Policy" on page 27.

Note - The IPQoS facility might be removed in a future release. Instead, use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see "Managing Network Virtualization and Network Resources in Oracle Solaris 11.2 ".

Defining a QoS Policy Task Map

This task map lists the general tasks for creating an IPQoS configuration file and the links to the sections that describe the steps to perform the tasks.

Task	Description	For Instructions
1. Plan your IPQoS-enabled network configuration.	Decide which systems on the local network should become IPQoS enabled.	"Preparing a Network for IPQOS" on page 29
2. Plan the QoS policy for IPQoS systems on your network.	Identify traffic flows as distinct classes of service. Then, determine	"Planning the Quality-of-Service Policy" on page 27

Task	Description	For Instructions
	which flows require traffic management.	
3. Create 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 Create the IPQoS Configuration File and Define Traffic Classes" on page 43
4. Add the filters that govern which traffic is selected and organized into a class.	Create filters for a class.	"How to Define Filters in the IPQoS Configuration File" on page 46
5. Create more classes and filters to be processed by the IP classifier.	Add more classes and filters to the IPQoS configuration file.	"How to Create an IPQoS Configuration File for a Best-Effort Web Server" on page 53
6. If the QoS policy calls for flow control, assign flow-control rates and conformance levels to the meter.	Add an `action` statement with parameters that configure the metering modules.	"How to Configure Flow Control in the IPQoS Configuration File" on page 63
7. If the QoS policy calls for differentiated forwarding behaviors, define how traffic classes are to be forwarded.	Add an `action` statement with parameters that configure the marker.	"How to Define Traffic Forwarding in the IPQoS Configuration File" on page 48
8. If the QoS policy calls for statistics gathering on traffic flows, define how accounting statistics are to be gathered.	Add an `action` statement with parameters that configure the flow-accounting module.	"How to Enable Accounting for a Class in the IPQoS Configuration File" on page 51
9. Add the content of a specified IPQoS configuration file into the appropriate kernel modules.	Apply the IPQoS configuration file.	"How to Start the `ipqos` Service" on page 70
10. If any IPQoS configuration files on the network define forwarding behaviors, add the resulting DSCPs to the appropriate scheduling files on the router.	Configure forwarding behaviors in the router files.	"Providing Differentiated Services on a Router" on page 66

Tools for Creating a QoS Policy

The QoS policy for your network resides in the IPQoS configuration file, `/etc/inet/ipqosinit.conf`. You create this configuration file with a text editor. Then, you start the `svc:/network/ipqos` service which runs the `ipqosconf` command.The policy is then written into the kernel IPQoS system. For more information about the `ipqosconf` command, refer to the `ipqosconf`(1M) man page. Example 6-3 also shows the complete syntax of the IPQoS configuration file.

Basic IPQoS Configuration File

The 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 27. 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.

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

These three sample configuration files illustrate the most common IPQoS configurations. You might use the sample files that are shown in the next section as templates for your own IPQoS implementation.

Creating IPQoS Configuration Files for Web Servers

This section introduces the IPQoS configuration file by showing how to create a configuration for a premium web server. It 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. This server 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 {
        enable_stats TRUE
        timer 10000
        timeout 10000
        max_limit 2048
    }
}
```

The following configuration file defines IPQoS activities on `Userweb`. This server hosts web sites for individuals with low-priced or "best effort" SLAs. This level of service 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 Create the IPQoS Configuration File and Define Traffic Classes

The IPQoS configuration file must be copied to `/etc/inet/ipqosinit.conf` when you are ready to employ it. If you are starting with a new installation it may be easier to edit your draft configuration file in the place that it will be used. This 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. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Log in to the premium web server.**

3. **Edit `/etc/inet/ipqosinit.conf`.**

4. **As the first uncommented line, insert the version number `fmt_version 1.0`.**

   Every IPQoS configuration file must start with this line.

5. **Insert 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 configuration file begins with the initial `action` statement to call the `ipgpc` classifier.

```
fmt_version 1.0

action {
    module ipgpc
    name ipgpc.classify
```

`fmt_version 1.0`	Begins the IPQoS configuration file.
`action {`	Begins the action statement.
`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 96 and the `ipqosconf`(1M) man page.

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

```
params {
          global_stats TRUE
   }
```

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

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

Global statistics are only 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 98 and the `ipqosconf`(1M) man page.

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

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

This statement is called a *class clause*. This `class` clause has the following contents.

`name goldweb`	Creates the class `goldweb` to identify traffic that is bound for the `Goldweb` server.
next_action markAF11	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 disabled.

For detailed information about the syntax of the `class` clause, see "`class` Clause" on page 97 and the `ipqosconf`(1M) man page.

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

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

`name video`	Creates the class `video` to identify streaming video traffic that is outgoing from the `Goldweb` server.
`next_action markEF`	Instructs the `ipgpc` module to pass packets of the `video` class to the `markEF` statement after `ipgpc` completes processing. The `markEF` statement calls the `dscpmk` marker.
`enable_stats FALSE`	Enables statistics collection for the `video` class. However, because the value of `enable_stats` is `FALSE`, statistics collection for this class is disabled.

9. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

   - **If you are done making changes, start the `ipqos` service.**

     See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

   - **If you want to continue making changes in the IPQoS configuration file, choose another task.**

     See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

▼ How to Define Filters in the IPQoS Configuration File

Before You Begin The procedure assumes that you have already started file creation and have defined classes. It continues building the IPQoS configuration file that is created in "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43.

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

---

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **If the IPQoS configuration file is not open, open it.**

3. **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:

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

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

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

`name webout`	Assigns the name `webout` to the filter.
`sport 80`	Selects traffic with a source port of 80, the usual 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 detailed information about the `filter` clause in the IPQoS configuration file, refer to "`filter` Clause" on page 98.

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

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

`name videoout`	Assigns 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`.

6. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

- **If you are done making changes, start the `ipqos` service.**

  See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

- **If you want to continue making changes in the IPQoS configuration file, choose another task.**

  See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

▼ How to Define Traffic Forwarding in the IPQoS Configuration File

This procedure shows how to define traffic forwarding by adding per-hop behaviors for a class into the IPQoS configuration file.

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 91.

Before You Begin The procedure assumes that you have an existing IPQoS configuration file with defined classes and defined filters. It continues building the IPQoS configuration file from Example 3-1.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **If the IPQoS configuration file is not already open, open it.**

3. **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 the configuration file:

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

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

4. **Invoke the marker with an `action` statement.**

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

`module dscpmk`	Calls the marker module `dscpmk`.
`name markAF11`	Assigns 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.

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

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

`global_stats FALSE`	Enables statistics collection for the `markAF11` marker `action` statement. However, because the value of `enable_stats` is `FALSE`, statistics are not collected.
`dscp_map{0–63:10}`	Assigns a DSCP 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 DSCP of `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 the DSCP of `10` receive a low-drop, high-priority service. Thus, outgoing traffic for premium customers on `Goldweb` is given the highest priority that is available for the Assured Forwarding (AF) PHB. For a table of possible DSCPs for AF, refer to Table 6-2.

6. **Start another marker `action` statement.**

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

`module dscpmk`	Calls the marker module `dscpmk`.

`name markEF`	Assigns the name `markEF` to the `action` statement.

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

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

`global_stats TRUE`	Enables statistics collection on class `video`, which selects streaming video packets.
`dscp_map{0–63:46}`	Assigns a DSCP 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 class `video` to the `acct` `action` statement after `dscpmk` completes processing. The `acct` `action` statement invokes the `flowacct` module.

The DSCP of `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 Expedited Forwarding (EF) per-hop behavior.

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

The EF PHB guarantees that packets with the DSCP 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 to streaming applications the EF PHBs in the QoS policy. For more details about the expedited forwarding PHB, refer to "Expedited Forwarding (EF) PHB" on page 89.

8. **Add the DSCPs that you have just created to the appropriate files on the Diffserv router.**

   For more information, refer to "Providing Differentiated Services on a Router" on page 66.

9. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

   - **If you are done making changes, start the `ipqos` service.**

See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

- **If you want to continue making changes in the IPQoS configuration file, choose another task.**

  See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

Next Steps

- To start gathering flow-accounting statistics on traffic flows, refer to "How to Enable Accounting for a Class in the IPQoS Configuration File" on page 51.
- To define forwarding behaviors for the marker modules, refer to "How to Define Traffic Forwarding in the IPQoS Configuration File" on page 48.
- To define flow-control parameters for the metering modules, refer to "How to Configure Flow Control in the IPQoS Configuration File" on page 63.
- To activate the IPQoS configuration file, refer to "How to Start the `ipqos` Service" on page 70.
- To define additional filters, refer to "How to Define Filters in the IPQoS Configuration File" on page 46.
- To create classes for traffic flows from applications, refer to "How to Configure the IPQoS Configuration File for an Application Server" on page 58.

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

This procedure shows how to enable accounting on a traffic class in the IPQoS configuration file. It defines flow accounting for the `video` class, which is introduced in "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43. This class selects streaming video traffic, which must be billed as part of a premium customer's SLA.

Before You Begin The procedure assumes that you have an existing IPQoS configuration file with defined classes, filters, metering actions, if appropriate, and marking actions, if appropriate. It continues building the IPQOS configuration file from Example 3-1.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **If the IPQoS configuration file is not already open, open it.**

3. **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 the configuration file, `/etc/inet/ipqosinit.conf`.

```
action {
    module dscpmk
    name markEF
    params {
        global_stats TRUE
        dscp_map{0-63:46}
        next_action acct
    }
}
```

4. **Begin an `action` statement that calls flow accounting.**

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

`module flowacct`	Invokes the flow-accounting module `flowacct`.
`name acct`	Assigns the name `acct` to the `action` statement

5. **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
    }
}
```

`global_stats TRUE`	Enables statistics collection on the 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 timeout value. A flow “times out” when packets for the flow are not seen during a timeout 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.

6. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

   - **If you are done making changes, start the `ipqos` service.**

     See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

   - **If you want to continue making changes in the IPQoS configuration file, choose another task.**

     See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

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. This procedure uses the configuration file from Example 3-2.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Log in to the best-effort web server.**

3. **Create a new IPQoS configuration file with a `.qos` extension.**

```
fmt_version 1.0
action {
    module ipgpc
    name ipgpc.classify
    params {
        global_stats TRUE
   }
```

The file must begin with the partial `action` statement to invoke the `ipgpc` classifier. In addition, the `action` statement also has a `params` clause to enable statistics collection. For an explanation of this `action` statement, see "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43.

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

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

`name userweb`	Creates a class that is called `userweb` for forwarding web traffic from users.
`next_action markAF1`	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 collection for the `userweb` class. However, because the value of `enable_stats` is `FALSE`, statistics collection for this class does not occur.

For an explanation of the `class` clause task, see "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43.

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

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

`name webout`	Assigns the name `webout` to the filter.
`sport 80`	Selects traffic with a source port of 80, the usual 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 46.

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

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

`module dscpmk`	Invokes the marker module `dscpmk`.
`name markAF12`	Assigns 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.

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

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

`global_stats FALSE`	Enables statistics collection for the `markAF12` marker `action` statement. However, because the value of `enable_stats` is `FALSE`, statistics collection does not occur.
`dscp_map{0–63:12}`	Assigns a DSCP 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 `userweb`, and that these packets can return to the network stream.

The DSCP of `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 the DSCP of `12` in the DS field receive a medium-drop, high-priority service.

8. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

- **If you are done making changes, start the `ipqos` service.**

  See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

- **If you want to continue making changes in the IPQoS configuration file, choose another task.**

  See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

Creating an IPQoS Configuration File for an Application Server

This section explains how to create a configuration file for an application server that provides major applications to 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. This server hosts FTP, electronic mail (SMTP), and network news (NNTP) for customers.

EXAMPLE 3-3 Sample IPQoS Configuration File 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
    }
    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_name AF31
        green_action_name markAF22
        global_stats TRUE
    }
}
action {
    module dscpmk
    name markAF31
    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 Application Server

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Log in to the IPQoS-enabled application server.**

3. **Create a new IPQoS configuration file with a `.qos` extension.**

4. **Insert the following required phrases to start the `action` statement that invokes the `ipgpc` classifier:**

   ```
   fmt_version 1.0

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

   For an explanation of the opening `action` statement, refer to "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43.

5. **Add class definitions to select traffic from three applications on the BigAPPS server.**

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

```
        enable_stats TRUE
        next_action meterftp
    }
```

`name smtp`	Creates a class called `smtp` that includes email traffic flows to be handled by the SMTP application
`enable_stats FALSE`	Enables statistics collection for the `smtp` class. However, because the value of `enable_stats` is `FALSE`, statistics for this class are not taken.
`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 called `news` that 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 called `ftp` that handles outgoing traffic that is handled by the FTP application.
`enable_stats TRUE`	Enables statistics collection 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 Create the IPQoS Configuration File and Define Traffic Classes" on page 43.

6. **Define `filter` clauses to select traffic of the classes defined in Step 2.**

```
   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
    }
}
```

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

7. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

   - **If you are done making changes, start the `ipqos` service.**

     See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

   - **If you want to continue making changes in the IPQoS configuration file, choose another task.**

See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

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

This 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 continues building the IPQoS configuration file in Example 3-3.

Before You Begin The procedure assumes that you have an existing IPQoS configuration file with defined classes and defined filters for the applications to be marked.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Open `/etc/inet/ipqosinit.conf`, and locate the end of the last `filter` clause.**

   In `/etc/inet/ipqosinit.conf`, the last filter is the following:

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

3. **Invoke the marker.**

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

`module dscpmk`	Invokes the marker module `dscpmk`.
`name markAF13`	Assigns the name `markAF13` to the `action` statement.

4. **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
    }
```

```
}
```

`global_stats FALSE`	Enables statistics collection for the `markAF13` marker `action` statement. However, because the value of `enable_stats` is `FALSE`, statistics are not collected.
`dscp_map{0–63:14}`	Assigns a DSCP 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 DSCP of `14` tells the marker to set all entries in the `dscp` map to the decimal value 14 (binary 001110). The DSCP of `14` sets the AF13 per-hop behavior. The marker marks packets of the `smtp` traffic class with the DSCP of `14` in the DS field.

AF13 assigns all packets with a DSCP 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. Lists of possible AF codepoints, refer to Table 6-2.

5. **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
    }
}
```

`name markAF21`	Assigns the name `markAF21` to the `action` statement.
`dscp_map{0–63:18}`	Assigns a DSCP of `18` to the packet headers of the traffic class `nntp`, which is currently being processed by the marker.

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

AF21 assures that all packets with a DSCP of `18` receive a low-drop precedence but with only Class 2 priority. Thus, the possibility of network news traffic being dropped is low.

6. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

- **If you are done making changes, start the `ipqos` service.**

  See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

- **If you want to continue making changes in the IPQoS configuration file, choose another task.**

  See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

▼ 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 meter modules, `tokenmt` or `tswtclmt`, in the IPQoS configuration file.

This next procedure continues to build the IPQoS configuration file for the application server in Example 3-3. In the procedure, you configure the meter and two marker actions that are called within the meter `action` statement.

Before You Begin The procedure assumes that you have already defined a class and a filter for the application to be flow-controlled.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Open `/etc/inet/ipqosinit.conf`.**

   Begin making changes after the following marker action:

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

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

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

`module tokenmt`	Invokes the`tokenmt` meter.
`name meterftp`	Assigns the name `meterftp` to the `action` statement.

4. **Add parameters to configure the meter's rate.**

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

`committed_rate 50000000`	Assigns a transmission rate of 50,000,000 bps 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 87.

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

```
   red_action markAF31
    green_action_name markAF22
    global_stats TRUE
    }
}
```

`red_action_name markAF31`	Indicates that when the traffic flow of the `ftp` class exceeds the committed rate, packets are sent to the `markAF31` marker `action` statement.
`green_action_name 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 86.

6. **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
    }
}
```

`module dscpmk`	Invokes the marker module `dscpmk`.
`name markAF31`	Assigns the name `markAF31` to the `action` statement.
`global_stats TRUE`	Enables statistics for the `ftp` class.
`dscp_map{0–63:26}`	Assigns a DSCP 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`, and that these packets can return to the network stream.

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

AF31 assures that all packets with a DSCP of `26` receive a low-drop precedence but with only Class 3 priority. Therefore, the possibility of nonconformant FTP traffic being dropped is low. Table 6-2 lists possible AF codepoints.

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

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

`name markAF22`	Assigns the name `markAF22` to the `marker` action.
`dscp_map{0–63:20}`	Assigns a DSCP of `20` to the packet headers of the traffic class `ftp` whenever `ftp` traffic conforms to its configured rate.

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

AF22 assures that all packets with a DSCP 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. Table 6-2 lists possible AF codepoints.

8. **Add the DSCPs that you have created for the application server to the appropriate files on the Diffserv router.**

9. **Save changes to the `/etc/inet/ipqosinit.conf` file.**

   - **If you are done making changes, start the `ipqos` service.**

     See "How to Start the `ipqos` Service" on page 70 for specific instructions about starting or restarting the service.

   - **If you want to continue making changes in the IPQoS configuration file, choose another task.**

     See "General IPQoS Configuration Planning Task Map" on page 23 for a list of additional changes that might be needed.

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 24. The actual steps for configuring Diffserv on a router and updating that router's files are outside the scope of this guide.

This section provides general steps for coordinating the forwarding information among various IPQoS-enabled systems on the network and the Diffserv router.

First, review the configuration files for all IPQoS-enabled systems on your network for the codepoints that are used in the various QoS policies.

List the codepoints and the systems and classes to which the codepoints apply. Although using the same codepoint for different areas is acceptable, you should provide other criteria in the IPQoS configuration file, such as a `precedence` selector, to determine the precedence of identically marked classes.

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

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

After you have identified the codepoints from your network's IPQoS configuration files, add them to the appropriate files on the Diffserv router. The codepoints that you supply should help to configure the router's Diffserv scheduling mechanism. Refer to the router manufacturer's documentation and web sites for instructions.

♦ ♦ ♦ C H A P T E R 4

Starting and Maintaining IPQoS Tasks

This chapter contains tasks for activating an IPQoS configuration file and for logging IPQoS-related events. It covers the following topics:

- "Administering IPQoS" on page 69
- "Troubleshooting IPQoS Error Messages" on page 72

Note - The IPQoS facility might be removed in a future release. Instead, use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see "Managing Network Virtualization and Network Resources in Oracle Solaris 11.2 ".

Administering IPQoS

This section describes how to start and maintain IPQoS on an Oracle Solaris system. Before you start these tasks, you must have a completed IPQoS configuration file, as described in "Defining a QoS Policy Task Map" on page 39. This section covers the following tasks:

- "How to Add the `ipqos` Package" on page 69
- "How to Start the `ipqos` Service" on page 70
- "How to Enable Logging of IPQoS Messages During Booting" on page 71

▼ How to Add the `ipqos` Package

The `ipqos` package is not added by default in all configurations. This procedure describes how to add this package.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Verify that the IPQoS package is not already installed.**

```
# pkg list ipqos
pkg list: no packages matching 'ipqos' installed
```

3. **Verify that your IPS package repository contains the AI package.**

```
# pkg list -a ipqos
NAME (PUBLISHER)                          VERSION                   IFO
system/network/ipqos                      0.5.11-0.175.2.0.0.26.2   i--
```

4. **Install the AI package.**

```
# pkg install system/network/ipqos
           Packages to install:  1
       Create boot environment: No
Create backup boot environment: No
            Services to change:  1

DOWNLOAD                       PKGS       FILES    XFER (MB)   SPEED
Completed                       1/1       32/32      0.1/0.1  546k/s
```

▼ How to Start the `ipqos` Service

Once you have made changes to the IPQoS configuration file, you will need to start or restart the `ipqos` SMF service.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Make sure that the appropriate changes are in the `/etc/inet/ipqosinit.conf` file.**

3. **Determine if the `ipqos` service is running.**

```
# svcs svc:/network/ipqos
STATE          STIME    FMRI
disabled       Mar_11   svc:/network/ipqos:default
```

4. **Enable or restart the `ipqos` service.**

   - **If the `ipqos` service is disabled, start it.**

   ```
   # svcadm enable svc:/network/ipqos
   ```

   - **If the `ipqos` service is enabled, disable and re-enable it.**

     These steps will use the new configuration file when the service is started.

```
# svcadm disable svc:/network/ipqos
# svcadm enable svc:/network/ipqos
```

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

   Use UNIX utilities to track IPQoS behavior and to gather statistics on your IPQoS implementation. This information can help you determine whether the configuration operates as expected.

See Also

- To view statistics on how IPQoS modules are working, refer to "Gathering Statistical Information" on page 80.
- To log `ipqosconf` messages, refer to "How to Enable Logging of IPQoS Messages During Booting" on page 71.

▼ How to Enable Logging of IPQoS Messages During Booting

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

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Add the following text as the final entry in the `/etc/syslog.conf` file.**

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

   Use tabs rather than spaces between the columns.

   This entry logs all boot-time messages that are generated by IPQoS in the `/var/adm/messages` file.

3. **Reboot the system to apply the messages.**

Example 4-1 IPQoS Output From `/var/adm/messages`

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

```
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 that are similar to the following examples.

```
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 these error messages, see Table 4-1.

Troubleshooting IPQoS Error Messages

The following table lists error messages that are generated by IPQoS and their possible solutions.

TABLE 4-1 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. Then, 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. Alternatively, 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 IPQo S.	Add the format version, as explained in "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43.
`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 beginning with the Solaris 9 9/02 release of IPQoS.
`No ipgpc action defined.`	An action is not defined for the `ipgpc` classifier in the configuration file, which is an IPQo S requirement.	Define an action for `ipgpc`, as shown in "How to Create the IPQoS Configuration File and Define Traffic Classes" on page 43.
`Can't commit a null configuration`	`ipqosconf -c` was run to commit a configuration that was empty, which IPQoS does not allow.	Be sure to apply a configuration file before you attempt to commit a configuration. For instructions, see "How to Start the `ipqos` Service" on page 70.
`Invalid CIDR mask on line` *line-number*	In the configuration file, a CIDR mask is used as part of the IP	Change the mask value to be in the range of 1–32 for IPv4 and 1–128 for IPv6.

Error Message	Description	Solution
	address that is out of the valid range for IP addresses.	
`Address masks aren't allowed for host names line` *line-number*	In the configuration file, a CIDR mask is defined 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 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 of the `ipgpc` action in the configuration file is not the required name `ipgpc.classify`.	Rename the action `ipgpc.classify`.
`Second parameter clause not supported line` *line-number*	In the configuration file, two parameter clauses are specified 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, two actions have the same name.	Rename or remove one of the actions.
`Duplicate named filter/class in action` *action-name*	Two filters or two classes have the same name 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 information about the module that is called by the named action, refer to the module entry in "IPQoS Architecture and the Diffserv Model" on page 83. Alternatively, refer to the `ipqosconf`(1M) man page.
`Mandatory parameter missing for action` *action-name*	A required parameter is not defined for an action in the configuration file.	For information about the module that is called by the named action, refer to the module entry in "IPQoS Architecture and the Diffserv Model" on page 83. Alternatively, refer to the `ipqosconf`(1M) man page.
`Max number of classes reached in ipgpc`	More classes are specified 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`	More filters are specified than are allowed in the `ipgpc` action of	Review the configuration file, and remove unneeded filters. Alternatively, you can raise

Error Message	Description	Solution
	the IPQoS configuration file. The maximum number is 10007.	the maximum number of filters by adding the entry `ipgpc_max_filters` *filter-number* to the `/etc/system` file.
`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*	An action, filter, or class name begins with an exclamation mark (!), which is not allowed in the IPQo S file.	Remove the exclamation mark, or rename the action, class, or filter.
`Name exceeds the maximum name length line` *line-number*	An action, class, or filter name in the configuration file exceeds the maximum length of 23 characters.	Assign 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 83. Alternatively, refer to the `ipqosconf`(1M) man page.
`Quoted string exceeds line,` *line-number*	The string does not have 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 the module description in "IPQoS Architecture and the Diffserv Model" on page 83. Alternatively, refer to the `ipqosconf`(1M) man page.
`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 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 83. Alternatively, 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 the correct syntax of the module that is called by the `action` statement with the malformed value list, refer to the module description in "IPQoS Architecture and the Diffserv Model" on page 83. Alternatively, 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.

Error Message	Description	Solution
`Invalid action name line` *line-number*	The name of the action on *line-number* of the configuration file uses the predefined name "continue" or "drop."	Rename the action so that the action 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.	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`	An action 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.

♦ ♦ ♦ C H A P T E R 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. It covers the following topics:

- "Recording Information About Traffic Flows" on page 77
- "Gathering Statistical Information" on page 80

Note - The IPQoS facility might be removed in a future release. Instead, use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see "Managing Network Virtualization and Network Resources in Oracle Solaris 11.2 ".

Recording Information About Traffic Flows

You use the IPQoS `flowacct` module to collect information about traffic flows, for example, source and destination addresses, the number 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 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 Oracle Solaris extended accounting facility to obtain data on traffic flows.

For further information, see the following sources:

- For instructions on how to assign `flowacct` parameters in IPQoS configuration files, see "How to Enable Accounting for a Class in the IPQoS Configuration File" on page 51.

- 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 63.
- To learn how `flowacct` works, refer to "Classifier Module" on page 84.
- 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. `acctadm` can record either basic attributes or extended attributes in the file. All `flowacct` attributes are listed in Table 6-4. For detailed information, refer to the `acctadm`(1M) man page.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in "Securing Users and Processes in Oracle Solaris 11.2 ".

2. **Create a basic flow-accounting file.**

   The following example shows how to create a basic flow-accounting 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
   ```

`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 is 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`	Instructs `acctadm` to enable flow accounting.

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

   The `acctadm` output is similar to the following example:

   ```
   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 entries except the last four are for use with the Oracle Solaris Resource Manager feature. The entries that are specific to IPQoS are as follows:

`Flow accounting: active`	Indicates that flow accounting is turned on.
`Flow accounting file: /var/ipqos/goldweb/account.info`	Indicates 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`	Indicates that only the basic flow attributes are tracked.

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

   ```
   # 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.

6. **View the contents of a flow-accounting file.**

   For instructions for viewing the contents of a flow-accounting file, see "Perl Interface to libexacct" in "Administering Resource Management in Oracle Solaris 11.2 ".

See Also For detailed information about the extended accounting feature, refer to Chapter 4, "About Extended Accounting," in "Administering Resource Management in Oracle Solaris 11.2 ".

Next Steps

- To define `flowacct` parameters in the IPQoS configuration file, refer to "How to Enable Accounting for a Class in the IPQoS Configuration File" on page 51.
- To print the data in the file that was created with `acctadm`, refer to "Perl Interface to libexacct" in "Administering Resource Management in Oracle Solaris 11.2 ".

Gathering Statistical Information

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

`/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 would use the following command:

```
/bin/kstat -m dscpmk
```

For technical details, refer to the `kstat`(1M) man page.

EXAMPLE 5-1 `kstat` Statistics for IPQoS

The following example shows 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
```

`class: flacct`	Indicates the name of the class to which the traffic flows belong, in this example `flacct`.
`bytes_in_tbl`	Total number of bytes in the flow table. The total number of bytes 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 that this `kstat` output was created.
`epackets`	Number of packets that resulted in an error during processing, in this example 0.
`flows_in_tbl`	Number of flow records in the flow table, which in this example is 1. When no records are in the table, the value for `flows_in_tbl` is 0.
`nbytes`	Total number of bytes reported for this `flowacct` action instance, which is 84 in the example. The value includes bytes that are currently in the

	flow table. The value also includes bytes that have timed out and are no longer in the flow table.
`npackets`	Total number of packets reported for this `flowacct` action instance, which is 1 in the example. `npackets` includes packets that are currently in the flow table. `npackets` also includes packets 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 Detail Reference

This chapter provides in-depth details about the following IPQoS topics:

- "IPQoS Architecture and the Diffserv Model" on page 83
- "IPQoS Configuration File" on page 95

For further information, see the following resources:

- For an overview, refer to Chapter 1, "Introducing IPQoS".
- For planning information, refer to Chapter 2, "Planning for an IPQoS-Enabled Network".
- For procedures for configuring IPQoS, refer to Chapter 3, "Creating the IPQoS Configuration File Tasks".

Note - The IPQoS facility might be removed in a future release. Instead, use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see "Managing Network Virtualization and Network Resources in Oracle Solaris 11.2 ".

IPQoS Architecture and the Diffserv Model

This section describes the IPQoS architecture and how IPQoS implements the differentiated services (Diffserv) model that is defined inRFC 2475, An Architecture for Differentiated Services (http://www.ietf.org/rfc/rfc2475.txt?number=2475). 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 virtual local area network (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. In contrast, the IPQoS classifier `ipgpc` is designed to handle traffic flows on hosts that are 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, refer to the `ipgpc`(7ipp) man page.

The `ipgpc` classifier does the following:

1. Selects traffic flows that meet the criteria 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 codepoint (DSCP)

   The presence of 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 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 15. For information on invoking the classifier in the IPQoS configuration file, refer to "IPQoS Configuration File" on page 95.

IPQoS Selectors

The `ipgpc` classifier 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 number of filters you define can impact IPQoS performance.

The folowing table lists the selectors that are available for `ipgpc`.

TABLE 6-1 Filter Selectors for the IPQoS Classifier

Selector	Argument	Information Selected
`saddr`	IP address number.	Source address.
`daddr`	IP address number.	Destination address.
`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 (DSCP) with a value of 0–63.	DSCP, which defines any forwarding behavior to be applied to the packet. If this parameter is specified, the `dsfield_mask` parameter must also be specified.
`dsfield_mask`	Bit mask with a value of 0–255.	Used in tandem with the `dsfield` selector. `dsfield_mask` is applied to the `dsfield` selector to determine which of its bits to match against.
`if_name`	Interface name.	Interface to be used for either incoming or outgoing traffic of a particular class.
`user`	Number of the UNIX user ID or user name to be selected. If no user ID or user name is on the packet, the default –1 is used.	User ID 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`	Possible values are:	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.
`precedence`	Precedence value. Highest precedence is 0.	Used to order filters with the same priority.
`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. The meter then 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 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.

`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.

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` – Enables the awareness 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 a 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 outcome 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 be nonzero 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 `tokenmt` to be color aware.

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

```
action {
    module tokenmt
    name meter1
    params {
       committed_rate 4000000
       peak_rate 8000000
       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 enable 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 DSCP 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. However, you can change the default 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 *time window*.

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 about `tswtclmt`, refer to the `tswtclmt`(7ipp) man page.

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 available only for IPQoS systems with VLAN devices.

For technical information about these modules, refer to the `dscpmk`(7ipp) and `dlcosmk`(7ipp) man pages.

Using the `dscpmk` Marker for Forwarding Packets

The marker receives traffic flows after the flows are processed by the classifier or by the metering modules. The marker marks the traffic with a forwarding behavior. This forwarding behavior is the action to be taken on the flows after the flows leaving 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 in relation to other traffic classes. PHBs govern forwarding behaviors only on the IPQoS system's contiguous network. For more information, refer to "Per-Hop Behaviors" on page 19.

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 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 a value in the DS field of the packet header. This value is called the differentiated services codepoint (DSCP). The Diffserv architecture defines two types of forwarding behaviors, EF and AF, which use different DSCPs. For overview information about DSCPs, refer to "DS Codepoint" on page 19.

The IPQoS system reads the DSCP 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 DSCP enables the router to prioritize and schedule the concurrent traffic flows. The router forwards 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 packets with the recommended EF codepoint 46 (101110) receive the best treatment that is available on release to the network. Expedited 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.

Assured Forwarding (AF) PHB

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

TABLE 6-2 Assured Forwarding Codepoints

	Class 1	Class 2	Class 3	Class 4
Low-Drop Precedence	AF11 =	AF21 =	AF31 =	AF41 =
	10 (001010)	18 (010010)	26 (011010)	34 (100010)
Medium-Drop Precedence	AF12 =	AF22 =	AF32 =	AF42 =
	12 (001100)	20 (010100)	28 (011100)	36 (100100)
High-Drop Precedence	AF13 =	AF23 =	AF33 =	AF43 =
	14 (001110)	22 (010110)	30 (011110)	38 (100110)

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

When these packets reach a Diffserv router, the router evaluates the packets' codepoints along with DSCPs 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 packet's DSCPs. 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 mark packets with AF21 (010010), AF13 (001110), AF43 (100110), and EF (101110) codepoints. You then need to add the AF21, AF13, AF43, and EF DSCPs to the appropriate file on the Diffserv router.

See your router manufacturer's documentation for information about setting the AF PHB and instructions for setting DS codepoints on your equipment.

Supplying a DSCP to the Marker

The DSCP is 6 bits in length. The DS field is 1 byte long. When you define a DSCP, 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 DSCP, you use the following parameter within a marker action statement:

```
dscp_map{0-63:DS-name tcodepoint}
```

The `dscp_map` parameter is a 64-element array, which you populate with the (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 decimal notation. For example, you must translate the EF codepoint of 101110 into the decimal 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 decimal notation 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, compatible with commercial switches.

You can use the user-priority values in the `dlcosmk` marker action by defining the class of service marks that are listed in the following 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, 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
                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 user-priority value of 4 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*. Flow accounting produces 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, refer to the `flowacct`(7ipp) man page.

To enable flow accounting, you need to use the Oracle Solaris `exacct` accounting facility and the `acctadm` command, as well as `flowacct`. For more information about flow accounting, refer to Chapter 5, "Using Flow Accounting and Statistics Gathering Tasks".

`flowacct` Parameters

The `flowacct` module 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 table.

In the IPQoS configuration file, you define the following `flowacct` parameters to measure flow records and to write the records to the flow 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 an interval, in milliseconds, which specifies how long a packet flow must be inactive before the flow 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 flow 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 63.

Flow 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 include the `flowacct` 8-tuple:

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

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 timeout 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 attributes described in the following table.

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

Attribute Name	Attribute Contents	Type
`diffserv-field`	DSCP in the outgoing packet headers of the flow.	Extended only
`user`	Either a UNIX User ID 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, refer to the `acctadm`(1M) man page.

The `flowacct` module observes flows and fills the flow 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. These entries 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` *file-type* `-f` *filename* `flow`

`acctadm -e`	Invokes `acctadm` with the `-e` option. The -e indicates that a resource list follows.
file-type	Specifies the attributes to be gathered, 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 3, “Creating the IPQoS Configuration File Tasks”.

The syntax of the IPQoS configuration file is shown in Example 6-3.

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
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
```

`action` Statement

You use `action` statements to invoke the various IPQoS modules that are described in "IPQoS Architecture and the Diffserv Model" on page 83.

When you create the IPQoS configuration file, you must always begin with the version number. Then, you must add the following `action` statement 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
}
```

`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 modules listed in the following table.

TABLE 6-5 IPQoS Modules

Module Name	Definition
`ipgpc`	IP classifier
`dscpmk`	Marker to be used to create DSCPs 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.

The syntax to define the remaining classes in the IPQoS configuration is as follows:

```
class {

     name class-name
     next_action next-action-name
}
```

To enable statistics collection on a particular class, you must first enable global statistics in the `ipgpc.classify` `action` statement. For more information, refer to "`action` Statement" on page 96.

Use the `enable_stats TRUE` statement whenever you want to enable statistics collection 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 enables 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.

Index

Numbers and Symbols
, 40
`/etc/inet/ipqosinit.conf` file command
  overview, 40

A
`acctadm` command, for flow accounting, 17, 79, 95
`action` statement, 96
application server
  configuring for IPQoS, 56
assured forwarding (AF), 20, 89
  AF codepoints table, 89
  for a marker `action` statement, 49

B
bandwidth regulation, 13
  planning, in the QoS policy, 14

C
`class` clause
  in the IPQoS configuration file, 45, 97
class of service (CoS) mark, 17
classes
  defining
    in the IPQoS configuration file, 54, 58
  selectors, list of, 84
  syntax of `class` clause, 97
classes of service *See* classes
classifier module, 15
  `action` statement, 44
  functions of the classifier, 84
color awareness, 16, 87

D
differentiated services, 9
  differentiated services model, 15
  network topologies, 24
  providing different classes of service, 14
Diffserv model
  classifier module, 15
  flow example, 18
  IPQoS implementation, 15, 16, 17, 18
  marker modules, 17
  meter modules, 16
Diffserv-aware router
  evaluating DS codepoints, 90
  planning, 29
`dlcosmk` marker, 17
  planning datagram forwarding, 34
  user priority values, table of, 91
  VLAN tags, 91
DS codepoint (DSCP), 17, 19
  AF forwarding codepoint, 20, 89
  color-awareness configuration, 88
  configuring, on a diffserv router, 66, 89
  defining
    in the IPQoS configuration file, 49
  `dscp_map` parameter, 90
  EF forwarding codepoint, 20, 89
  PHBs and the DSCP, 19
  planning, in the QoS policy, 35
`dscpmk` marker, 17
  invoking
    in a marker `action` statement, 48, 55, 61, 64
  PHBs for packet forwarding, 89

E
error messages for IPQoS, 72
example IPQoS configuration files
  application server, 56
  best-effort web server, 42
  color-awareness segment, 87
  premium web server, 41
  VLAN device configuration, 92
expedited forwarding (EF), 20, 89
  defining
    in the IPQoS configuration file, 50

F
`filter` clause
  in the IPQoS configuration file, 46, 98
filters, 16
  creating
    in the IPQoS configuration file, 54, 59
  `filter` clause syntax, 98
  planning, in the QoS policy, 30
  selectors, list of, 84
flow accounting, 77, 93
  flow record table, 93
flow control
  through the metering modules, 16
`flowacct` module, 17, 93
  `acctadm` command, for creating a flow accounting file, 95
  `action` statement for `flowacct`, 52
  attributes of flow records, 94
  flow record table, 93
  flow records, 77
  parameters, 93
forwarding traffic
  datagram forwarding, 91
  effect of PHBs on packet forwarding, 89
  IP packet forwarding, with DSCP, 19
  planning, in the QoS policy, 14
  traffic flow through Diffserv networks, 20

H
hardware for IPQoS-enabled networks, 24

I
`ipgpc` classifier *See* classifier module
IPQoS, 9
  configuration example, 36, 37
  configuration file, 41, 95
    `action` statement syntax, 97
    `class` clause, 45
    `filter` clause, 46
    initial action statement, 44, 96
    list of IPQoS modules, 97
    marker `action` statement, 48
  configuration planning, 23
  Diffserv model implementation, 15
  error messages, 72
  features, 10
  message logging, 71
  network example, 41
  network topologies supported, 24, 25, 25, 26
  QoS policy planning, 27
  related RFCs, 10
  routers on an IPQoS network, 66
  statistics generation, 80
  traffic management capabilities, 12, 14
  VLAN device support, 91
`ipqosconf` command
  applying a configuration, 70

K
`kstat` command, use with IPQoS, 80

L
load balancing
  in an IPQoS-enabled network, 25

M
marker modules, 17, 17, 17, 88, 88
  See also dlcosmk marker
  See also dscpmk marker
  PHBs, for IP packet forwarding, 19
  specifying a DS codepoint, 90

support for VLAN devices, 91
metering modules, 16, 16, 86, 86
See also tokenmt meter
See also tswtclmt meter
introduction, 16
invoking
in the IPQoS configuration file, 63
outcomes of metering, 16, 86

N
network example for IPQoS, 41
network topologies for IPQoS, 24
configuration example, 37
LAN with IPQoS-enabled firewall, 26
LAN with IPQoS-enabled hosts, 25
LAN with IPQoS-enabled server farms, 24

P
`params` clause
defining global statistics, 44, 98
for a `flowacct` `action`, 52
for a marker `action`, 49
for a metering `action`, 64
syntax, 98
per-hop behavior (PHB), 19
AF forwarding, 20
defining
in the IPQoS configuration file, 65
EF forwarding, 20
using, with `dscpmk` marker, 89

Q
QoS policy, 12
creating filters, 30
implementing
in the IPQoS configuration file, 39
planning task map, 28
template for policy organization, 27
quality of service (QoS)
QoS policy, 12
tasks, 10

R
Requests for Comments (RFCs)
IPQoS, 10

S
selectors, 16
IPQoS 5-tuple, 15
planning, in the QoS policy, 30
selectors, list of, 84
service-level agreement (SLA), 11
billing clients, based on flow accounting, 77
providing different classes of service , 14
statistics for IPQoS
enabling class-based statistics, 98
enabling global statistics, 44, 98
generating, through the `kstat` command, 80
`svc:/network/ipqos` service
overview, 40
`syslog.conf` file logging for IPQoS, 71

T
task maps
IPQoS
configuration file creation, 39
configuration planning, 23
QoS policy planning, 28
`tokenmt` meter, 16
color-awareness configuration, 16, 87
metering rates, 86
rate parameters, 86
single-rate meter, 87
two rate-meter, 87
traffic conformance
defining, 63
outcomes, 16, 86
planning
outcomes in the QoS policy, 33
rates in the QoS policy, 33
rate parameters, 86, 86
traffic management
controlling flow, 16
forwarding traffic, 19, 20, 20, 21
planning network topologies, 24

prioritizing traffic flows, 13
regulating bandwidth, 12
`tswtclmt` meter, 16, 88
metering rates, 88

U

user priority value, 17

V

virtual LAN (VLAN) devices on an IPQoS network, 91

W

web servers
configuring for IPQoS, 41, 42, 53, 55