

Managing IP Quality of Service in Oracle® Solaris 11.1

Copyright © 1999, 2012, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf disposition de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, breveter, transmettre, distribuer, exposer, exécuter, publier ou afficher le logiciel, même partiellement, sous quelque forme et par quelque procédé que ce soit. Par ailleurs, il est interdit de procéder à toute ingénierie inverse du logiciel, de le désassembler ou de le décompiler, excepté à des fins d'interopérabilité avec des logiciels tiers ou tel que prescrit par la loi.

Les informations fournies dans ce document sont susceptibles de modification sans préavis. Par ailleurs, Oracle Corporation ne garantit pas qu'elles soient exemptes d'erreurs et vous invite, le cas échéant, à lui en faire part par écrit.

Si ce logiciel, ou la documentation qui l'accompagne, est concédé sous licence au Gouvernement des Etats-Unis, ou à toute entité qui délivre la licence de ce logiciel ou l'utilise pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Ce logiciel ou matériel a été développé pour un usage général dans le cadre d'applications de gestion des informations. Ce logiciel ou matériel n'est pas conçu ni n'est destiné à être utilisé dans des applications à risque, notamment dans des applications pouvant causer des dommages corporels. Si vous utilisez ce logiciel ou matériel dans le cadre d'applications dangereuses, il est de votre responsabilité de prendre toutes les mesures de secours, de sauvegarde, de redondance et autres mesures nécessaires à son utilisation dans des conditions optimales de sécurité. Oracle Corporation et ses affiliés déclinent toute responsabilité quant aux dommages causés par l'utilisation de ce logiciel ou matériel pour ce type d'applications.

Oracle et Java sont des marques déposées d'Oracle Corporation et/ou de ses affiliés. Tout autre nom mentionné peut correspondre à des marques appartenant à d'autres propriétaires qu'Oracle.

Intel et Intel Xeon sont des marques ou des marques déposées d'Intel Corporation. Toutes les marques SPARC sont utilisées sous licence et sont des marques ou des marques déposées de SPARC International, Inc. AMD, Opteron, le logo AMD et le logo AMD Opteron sont des marques ou des marques déposées d'Advanced Micro Devices. UNIX est une marque déposée d'The Open Group.

Ce logiciel ou matériel et la documentation qui l'accompagne peuvent fournir des informations ou des liens donnant accès à des contenus, des produits et des services émanant de tiers. Oracle Corporation et ses affiliés déclinent toute responsabilité ou garantie expresse quant aux contenus, produits ou services émanant de tiers. En aucun cas, Oracle Corporation et ses affiliés ne sauraient être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'accès à des contenus, produits ou services tiers, ou à leur utilisation.

Contents

Preface	7
1 Introducing IPQoS (Overview)	9
IPQoS Basics	9
What Are Differentiated Services?	9
IPQoS Features	10
Where to Get More Information About Quality-of-Service Theory and Practice	10
Providing Quality of Service With IPQoS	12
Implementing Service-Level Agreements	12
Assuring Quality of Service for an Individual Organization	12
Introducing the Quality-of-Service Policy	12
Improving Network Efficiency With IPQoS	13
How Bandwidth Affects Network Traffic	13
Using Classes of Service to Prioritize Traffic	14
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 (Tasks)	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
▼ How to Prepare a Network for IPQoS	29
▼ How to Define the Classes for Your QoS Policy	29
Defining Filters	31
▼ How to Define Filters in the QoS Policy	32
▼ How to Plan Flow Control	33
▼ How to Plan Forwarding Behavior	36
▼ How to Plan for Flow Accounting	38
Introducing the IPQoS Configuration Example	39
IPQoS Topology	39
3 Creating the IPQoS Configuration File (Tasks)	43
Defining a QoS Policy in the IPQoS Configuration File (Task Map)	43
Tools for Creating a QoS Policy	45
Basic IPQoS Configuration File	45
Creating IPQoS Configuration Files for Web Servers	46
▼ How to Create the IPQoS Configuration File and Define Traffic Classes	48
▼ How to Define Filters in the IPQoS Configuration File	50
▼ How to Define Traffic Forwarding in the IPQoS Configuration File	51
▼ How to Enable Accounting for a Class in the IPQoS Configuration File	54
▼ How to Create an IPQoS Configuration File for a Best-Effort Web Server	56
Creating an IPQoS Configuration File for an Application Server	58
▼ How to Configure the IPQoS Configuration File for an Application Server	60
▼ How to Configure Forwarding for Application Traffic in the IPQoS Configuration File ...	62
▼ How to Configure Flow Control in the IPQoS Configuration File	64
Providing Differentiated Services on a Router	67
▼ How to Configure a Router on an IPQoS-Enabled Network	67
4 Starting and Maintaining IPQoS (Tasks)	69
Administering IPQoS (Task Map)	69
Applying an IPQoS Configuration	70

▼ How to Apply a New Configuration to the IPQoS Kernel Modules	70
▼ How to Ensure That the IPQoS Configuration Is Applied After Each Reboot	71
Enabling sys log Logging for IPQoS Messages	71
▼ How to Enable Logging of IPQoS Messages During Booting	71
Troubleshooting with IPQoS Error Messages	72
5 Using Flow Accounting and Statistics Gathering (Tasks)	77
Setting Up Flow Accounting (Task Map)	77
Recording Information About Traffic Flows	78
▼ 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	92
IPQoS Configuration File	95
action Statement	96
Module Definitions	97
class Clause	97
filter Clause	98
params Clause	98
ipqosconf Configuration Utility	99
Index	101

Preface

Welcome to the Managing IP Quality of Service in Oracle Solaris 11.1. This book is part of a multi-volume set that covers a significant part of the Oracle Solaris system administration information. This book assumes that you have already installed Oracle Solaris. You should be ready to configure your network or ready to configure any networking software that is required on your network.

Note – This Oracle Solaris release supports systems that use the SPARC and x86 families of processor architectures. The supported systems appear in the *Oracle Solaris OS: Hardware Compatibility Lists*. This document cites any implementation differences between the platform types.

Who Should Use This Book

This book is intended for anyone responsible for administering systems that run Oracle Solaris, which are configured in a network. To use this book, you should have at least two years of UNIX system administration experience. Attending UNIX system administration training courses might be helpful.

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.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P-1 Typographic Conventions

Typeface	Description	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name%</code> su Password:
<i>aabbcc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . <i>A cache</i> is a copy that is stored locally. Do <i>not</i> save the file. Note: Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P-2 Shell Prompts

Shell	Prompt
Bash shell, Korn shell, and Bourne shell	\$
Bash shell, Korn shell, and Bourne shell for superuser	#
C shell	<code>machine_name%</code>
C shell for superuser	<code>machine_name#</code>

Introducing IPQoS (Overview)

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.

The following is a list of topics in this chapter:

- “IPQoS Basics” on page 9
- “Providing Quality of Service With IPQoS” on page 12
- “Improving Network Efficiency With IPQoS” on page 13
- “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. Users are encouraged to instead use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see *Using Virtual Networks in Oracle Solaris 11.1*.

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 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:

- `ipqosconf` Command-line tool for configuring the QoS policy
- Classifier that selects actions, which are based on filters that configure the QoS policy of your organization
- Metering module that measures network traffic, in compliance with the Diffserv model
- Service differentiation that is based on the ability to mark a packet's IP header with forwarding information
- Flow-accounting module that gathers statistics for traffic flows
- Statistics gathering for traffic classes, through the UNIX® `ksstat` 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 About Quality-of-Service Theory and Practice

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

Books About Quality of Service

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

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

Requests for Comments (RFCs) About Quality of Service

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

- 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
- RFC 2597, *Assured Forwarding PHB Group* (<http://www.ietf.org/rfc/rfc2597.txt?number=2597>) – Describes how the assured forwarding (AF) per-hop behavior works.
- RFC 2598, *An Expedited Forwarding PHB* (<http://www.ietf.org/rfc/rfc2598.txt?number=2598>) – Describes how the expedited forwarding (EF) per-hop behavior works
- Internet-Draft, *An Informal Management Model for Diffserv Routers* – Presents a model for implementing the Diffserv architecture on routers.

Web Sites With Quality-of-Service Information

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

Router manufacturers such as Cisco Systems and Juniper Networks provide information on their corporate web sites that describes how Differentiated Services are implemented in their products.

IPQoS Man Pages

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. The `acctadm` command 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. An example is 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. When computer networks expand, the need also increases for managing network traffic that is generated by increasing numbers of users and more powerful processors. Some symptoms of an overused network include lost data and traffic congestion. Both symptoms result in slow response times.

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

For example, for an enterprise or institution, you must maintain an efficient network to avoid traffic bottlenecks. You must also 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.

How Bandwidth Affects Network Traffic

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

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

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

Using Classes of Service to Prioritize Traffic

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

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

For an enterprise, you could create classes of service that are based on department requirements. Or, you could create classes that are based on the preponderance of a particular application in the network traffic.

Here are a few examples of traffic classes for an enterprise:

- Popular applications such as email and outgoing FTP to a particular server, 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 *Differentiated Services (Diffserv)* architecture that is defined in RFC 2475:

- Classifier
- Meter
- Marker

IPQoS adds the following enhancements to the Diffserv model:

- Flow-accounting module
- 802.1D datagram marker

This section introduces the Diffserv modules as they are used by IPQoS. You need to know about these modules, their names, and their uses to set up the QoS policy. For detailed information about each module, refer to “[IPQoS Architecture and the Diffserv Model](#)” on [page 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 is named `ipgpc`. The `ipgpc` classifier 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

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

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

For information on how to define classes, see [“How to Define the Classes for Your QoS Policy” on page 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 if the packet belongs to the filter's class. You can filter on a packet by using a variety of selectors, for example, the IPQoS 5-tuple and other common parameters:

- Source 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 on creating filters, see [“How to Define Filters in the QoS Policy” on page 32.](#)

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. Subsequent actions might include 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 a green, yellow, or red.

For information on defining parameters for the IPQoS meters, refer to [“How to Plan Flow Control” on page 33](#).

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 Diffserv model, as designed by the IETF.

For information on implementing a marker strategy for the QoS policy, see [“How to Plan Forwarding Behavior” on page 36](#).

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, as shown in the following list:

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

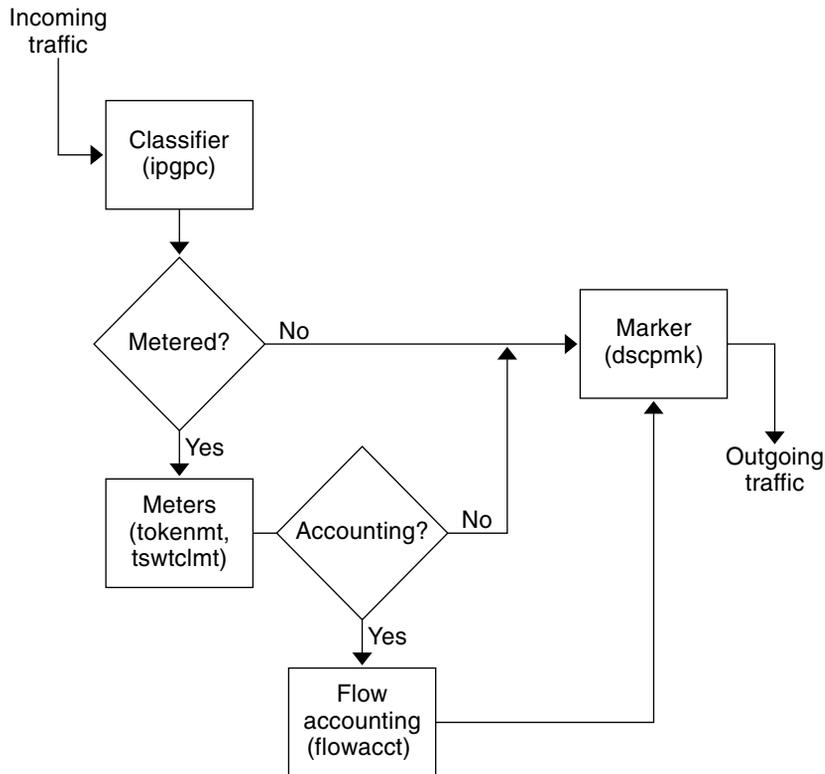
You can also gather statistics on other attributes, as described in [“Recording Information About Traffic Flows” on page 78](#), and in the `flowacct(7ipp)` and `acctadm(1M)` man pages.

For information on planning a flow-accounting strategy, see [“How to Plan for Flow Accounting” on page 38](#).

How Traffic Flows Through the IPQoS Modules

The next 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 a set of actions that are called *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 `d1cosmk` marker does not use the DSCP. Rather, `d1cosmk` 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. The exception is if another Diffserv system changes the DSCP. For more information on PHBs, refer to [“Using the `dscpmk` Marker for Forwarding Packets” on page 89](#).

The goal of a PHB is to provide a specified amount of network resources to a class of traffic on the contiguous network. You can achieve this goal in the QoS policy. 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 PHB. This traffic class then receives a low-drop precedence PHB from any Diffserv-aware router, which guarantees bandwidth to packets of this class. You can add to the QoS policy other DSCPs that assign varying levels of precedence to other traffic classes. The lower-precedence packets are given bandwidth by Diffserv systems in agreement with the priorities that are indicated in the packets' DSCPs.

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

Expedited Forwarding

The *expedited forwarding (EF)* 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

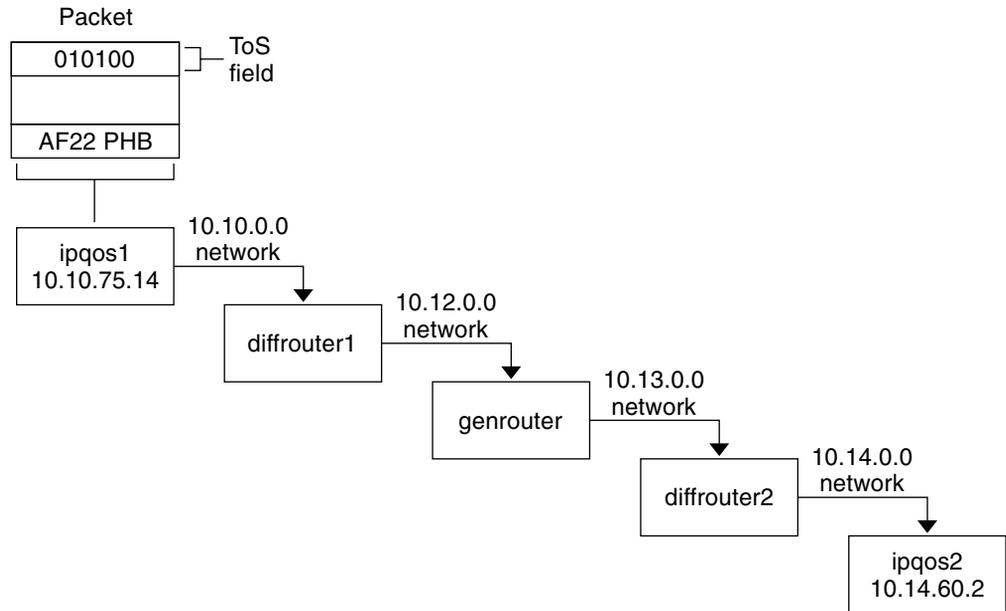
The *assured forwarding (AF)* per-hop behavior provides four different forwarding classes that you can assign to a packet. Every forwarding class provides three drop precedences, as shown in [Table 6-2](#).

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

Packet Forwarding in a Diffserv Environment

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

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



The next steps trace the flow of the packet that is shown in this figure. The steps begin 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 if 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.

Planning for an IPQoS-Enabled Network (Tasks)

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 tasks for adding IPQoS-enabled systems onto a Diffserv-aware network. The following topics are covered.

- [“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 39](#)

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 *Using Virtual Networks in Oracle Solaris 11.1*.

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.	Learn about 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 IPQoS system.	Decide on the classes, metering, and accounting features that are needed to implement each SLA.	“Planning the Quality-of-Service Policy” on page 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 your router does not currently offer Diffserv, consider the Diffserv solutions that are offered by Cisco Systems, Juniper Networks, and other router manufacturers. If the local router does not implement Diffserv, then the router passes marked packets on to the next hop without evaluating the marks.

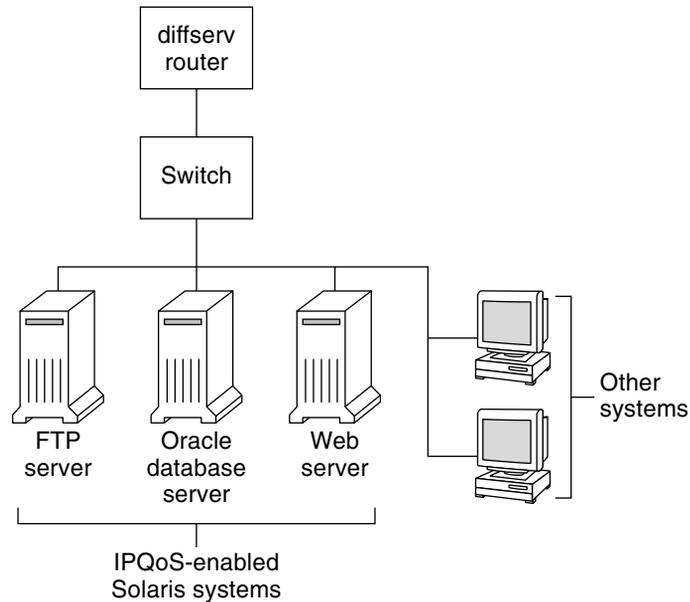
IPQoS Network Topologies

This section illustrates IPQoS strategies for various network needs.

IPQoS on Individual Hosts

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

FIGURE 2-1 IPQoS Systems on a Network Segment



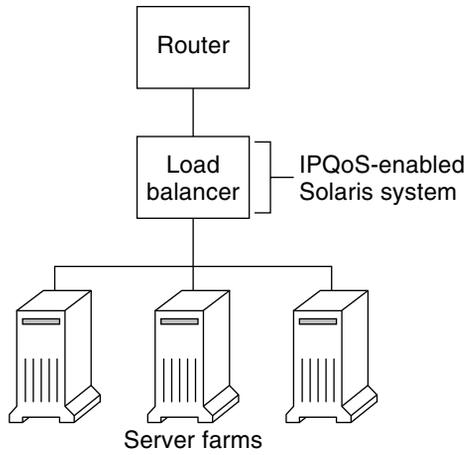
This network is but one segment of a corporate intranet. By enabling IPQoS on the application servers and web servers, you can control the rate at which each IPQoS system releases outgoing traffic. If you make the router Diffserv aware, you can further control incoming and outgoing traffic.

The examples in this guide use the “IPQoS on an individual host” scenario. For the example topology that is used throughout the guide, see [Figure 2-4](#).

IPQoS on a Network of Server Farms

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

FIGURE 2-2 Network of IPQoS-Enabled Server Farms



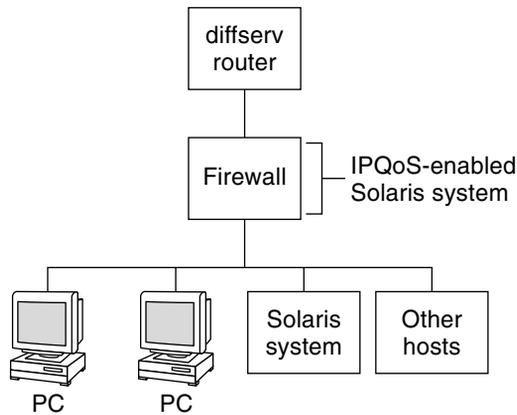
In such a topology, the router is Diffserv aware, and therefore able to queue and rate both incoming and outgoing traffic. The load balancer is also Diffserv-aware, and the server farms are IPQoS enabled. The load balancer can provide additional filtering beyond the router by using selectors such as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

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.

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

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

TABLE 2-1 QoS Planning Template *(Continued)*

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 that are shown in the template.

QoS Policy Planning (Task Map)

This 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.	“How to Prepare 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.	“How to Define 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 32
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 33
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.	“How to Plan Forwarding Behavior” on page 36
6. If applicable, set up a statistics-monitoring plan for traffic flows on the network.	Evaluate the traffic classes to determine which traffic flows must be monitored for accounting or statistical purposes.	“How to Plan for Flow Accounting” on page 38

Note – The rest of this section explains how to plan the QoS policy of an IPQoS-enabled system. To plan the QoS policy for the Diffserv router, refer to the router documentation and the router manufacturer's web site.

▼ How to Prepare a Network for IPQoS

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

- 1 Review your network topology. 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.

▼ How to Define the Classes for Your QoS Policy

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

Note – For an overview of classes, see [“IPQoS Classes”](#) on page 15.

The next procedure assumes that you have determined which systems on your network are to be IPQoS-enabled, as identified in [“How to Prepare a Network for IPQoS”](#) on page 29.

- 1 Create a QoS planning table for organizing the QoS policy information.**
For suggestions, refer to [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.

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

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

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

For example, your company might offer web site hosting to each customer, which indicates that you need to define a class for each customer web site. 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.

- 4 **List the classes that you have defined in the QoS planning table you created in Step 1.**

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

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

- 6 **When you finish defining classes, you next define filters for each class, as explained in [“How to Define Filters in the QoS Policy”](#) on page 32.**

More Information **Prioritizing the Classes**

As you create classes, you quickly realize which classes have highest priority, medium priority, and best-effort priority. A good scheme for prioritizing classes becomes particularly important when you assign per-hop behaviors to outgoing traffic, as explained in [“How to Plan Forwarding Behavior”](#) on page 36.

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.

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 then 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.

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 <code>/etc/services</code> , 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 <code>/etc/protocols</code> .
ip_version	Addressing style to use. Use either IPv4 or IPv6. IPv4 is the default.
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.
priority	Priority level that is assigned to the class. For more information, see “How to Define the Classes for Your QoS Policy” on page 29 .
user	Either the UNIX user ID or user name that is used when the upper-level application is executed.
projid	Project ID that is used when the upper-level application is executed.
direction	Direction of traffic flow. Value is either LOCAL_IN, LOCAL_OUT, FWD_IN, or FWD_OUT.

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

▼ How to Define Filters in the QoS Policy

Before You Begin Before you can perform the next steps, you should have completed the procedure [“How to Define the Classes for Your QoS Policy” on page 29](#).

- 1 Create at least one filter for each class in the QoS planning table that you created in [“How to Define the Classes for Your QoS Policy” on page 29](#).**

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 that was introduced in [Table 2–1](#) to fill in filters for the classes you defined.

Example 2–1 Defining Filters for FTP Traffic

The next table is an example that 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

- See Also**
- To define a flow-control scheme, refer to [“How to Plan Flow Control” on page 33](#).
 - To define forwarding behaviors for flows as the flows return to the network stream, refer to [“How to Plan Forwarding Behavior” on page 36](#).
 - To plan for flow accounting of certain types of traffic, refer to [“How to Plan for Flow Accounting” on page 38](#).
 - To add more classes to the QoS policy, refer to [“How to Define the Classes for Your QoS Policy” on page 29](#).
 - To add more filters to the QoS policy, refer to [“How to Define Filters in the QoS Policy” on page 32](#).

▼ How to Plan Flow Control

Flow control involves measuring traffic flow for a class and then releasing packets onto the network at a defined rate. When you plan flow control, you define parameters to be used by the IPQoS metering modules. The meters determine the rate at which traffic is released onto the network. For an introduction to the metering modules, see [“Meter \(tokenmt and tswtc_lmt\) Overview” on page 16](#).

The next procedure assumes that you have defined filters and selectors, as described in [“How to Define Filters in the QoS Policy” on page 32](#).

1 Determine the maximum bandwidth for your network.

2 Review any SLAs that are supported on your network. 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 the list of classes that you created in “How to Define the Classes for Your QoS Policy” on page 29.

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

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

Note – Not all classes need to be metered. Remember this guideline as you review your list of classes.

4 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. Suppose that 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.

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 organizational 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 “[tswtc_lmt Metering Module](#)” on page 88 and in the `tswtc_lmt(7ipp)` man page.

7 Add traffic conformance outcomes for the metered traffic.

The outcomes for both metering modules are green, red, and yellow. Add to your QoS organizational table the traffic conformance outcomes that apply to the rates you define. Outcomes for the meters are fully explained in “[Meter Module](#)” on page 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 next table is an example that 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

- See Also**
- To define forwarding behaviors for flows as the packets return to the network stream, refer to [“How to Plan Forwarding Behavior” on page 36](#).
 - To plan for flow accounting of certain types of traffic, refer to [“How to Plan for Flow Accounting” on page 38](#).
 - To add more classes to the QoS policy, refer to [“How to Define the Classes for Your QoS Policy” on page 29](#).
 - To add more filters to the QoS policy, refer to [“How to Define Filters in the QoS Policy” on page 32](#).
 - To define another flow-control scheme, refer to [“How to Plan Flow Control” on page 33](#).
 - To create an IPQoS configuration file, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes” on page 48](#).

▼ How to Plan 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: prioritize the flows of a class in relationship to other traffic classes or drop the flows entirely.

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

- `dscpmk` – Used to mark the DS field of an IP packet with a DSCP
- `dlcosmk` – Used to mark the VLAN tag of a datagram with a class-of-service (CoS) value

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

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

- 1 Review the classes that you have created thus far and the priorities that you have assigned to 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

Traffic is generally metered for the following reasons:

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

You use the marker with the meter to provide differentiated services and bandwidth management to these classes. For example, the following table shows a portion of a QoS policy. 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

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

The forwarding behaviors assign low-priority 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](#).

- See Also**
- To plan for flow accounting of certain types of traffic, refer to [“How to Plan for Flow Accounting”](#) on page 38.
 - To add more classes to the QoS policy, refer to [“How to Define the Classes for Your QoS Policy”](#) on page 29.
 - To add more filters to the QoS policy, refer to [“How to Define Filters in the QoS Policy”](#) on page 32.
 - To define a flow-control scheme, refer to [“How to Plan Flow Control”](#) on page 33.
 - To define additional forwarding behaviors for flows as the packets return to the network stream, refer to [“How to Plan Forwarding Behavior”](#) on page 36.
 - To create an IPQoS configuration file, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.

▼ How to Plan for Flow Accounting

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

1 Does your company offer SLAs to customers?

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

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

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

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

- See Also**
- To add more classes to the QoS policy, refer to [“How to Define the Classes for Your QoS Policy”](#) on page 29.
 - To add more filters to the QoS policy, refer to [“How to Define Filters in the QoS Policy”](#) on page 32.
 - To define a flow-control scheme, refer to [“How to Plan Flow Control”](#) on page 33.
 - To define forwarding behaviors for flows as the packets return to the network stream, refer to [“How to Plan Forwarding Behavior”](#) on page 36.
 - To plan for additional flow accounting of certain types of traffic, refer to [“How to Plan for Flow Accounting”](#) on page 38.
 - To create the IPQoS configuration file, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.

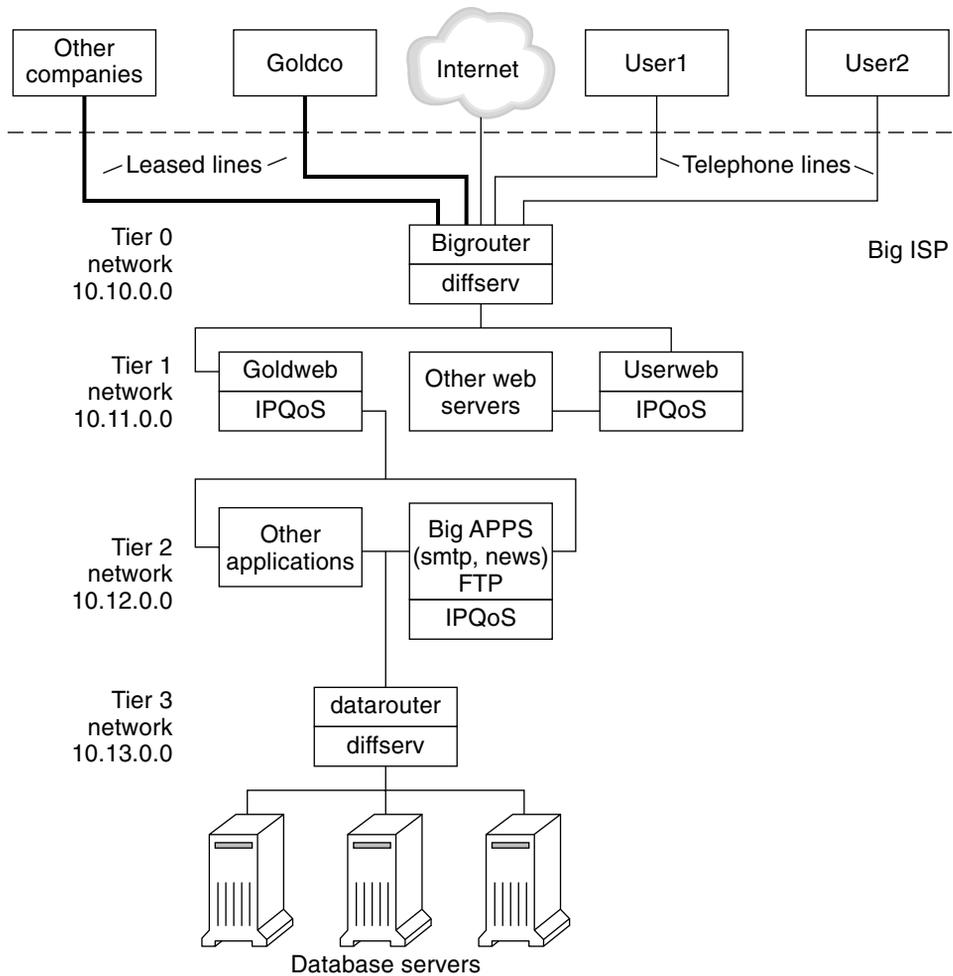
Introducing the IPQoS Configuration Example

Tasks in the remaining chapters of the guide use the example IPQoS configuration that is introduced in this section. The example shows the differentiated services solution 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 implemented 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.

Creating the IPQoS Configuration File (Tasks)

This chapter shows how to create IPQoS configuration files. Topics that are covered in the chapter include the following.

- “Defining a QoS Policy in the IPQoS Configuration File (Task Map)” on page 43
- “Tools for Creating a QoS Policy” on page 45
- “Creating IPQoS Configuration Files for Web Servers” on page 46
- “Creating an IPQoS Configuration File for an Application Server” on page 58
- “Providing Differentiated Services on a Router” on page 67

This chapter assumes that you have defined a complete QoS policy, and you are ready to use this policy as the basis for the IPQoS configuration file. For instructions on QoS policy planning, refer to “[Planning the Quality-of-Service Policy](#)” on page 27.

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 [Using Virtual Networks in Oracle Solaris 11.1](#).

Defining a QoS Policy in the IPQoS Configuration File (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.	“How to Prepare a Network for IPQoS” on page 29

Task	Description	For Instructions
2. Plan the QoS policy for IPQoS systems on your network.	Identify traffic flows as distinct classes of service. Then, determine which flows require traffic management.	“Planning the Quality-of-Service Policy” on page 27
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 48
4. Create filters for a class.	Add the filters that govern which traffic is selected and organized into a class.	“How to Define Filters in the IPQoS Configuration File” on page 50
5. Add more classes and filters to the IPQoS configuration file.	Create more classes and filters to be processed by the IP classifier.	“How to Create an IPQoS Configuration File for a Best-Effort Web Server” on page 56
6. Add an action statement with parameters that configure the metering modules.	If the QoS policy calls for flow control, assign flow-control rates and conformance levels to the meter.	“How to Configure Flow Control in the IPQoS Configuration File” on page 64
7. Add an action statement with parameters that configure the marker.	If the QoS policy calls for differentiated forwarding behaviors, define how traffic classes are to be forwarded.	“How to Define Traffic Forwarding in the IPQoS Configuration File” on page 51
8. Add an action statement with parameters that configure the flow-accounting module.	If the QoS policy calls for statistics gathering on traffic flows, define how accounting statistics are to be gathered.	“How to Enable Accounting for a Class in the IPQoS Configuration File” on page 54
9. Apply the IPQoS configuration file.	Add the content of a specified IPQoS configuration file into the appropriate kernel modules.	“How to Apply a New Configuration to the IPQoS Kernel Modules” on page 70
10. Configure forwarding behaviors in the router files.	If any IPQoS configuration files on the network define forwarding behaviors, add the resulting DSCPs to the appropriate scheduling files on the router.	“How to Configure a Router on an IPQoS-Enabled Network” on page 67

Tools for Creating a QoS Policy

The QoS policy for your network resides in the IPQoS configuration file. You create this configuration file with a text editor. Then, you provide the file as an argument to `ipqosconf`, the IPQoS configuration utility. When you instruct `ipqosconf` to apply the policy that is defined in your configuration file, the policy is written into the kernel IPQoS system. For detailed information about the `ipqosconf` command, refer to the `ipqosconf(1M)` man page. For instructions on the use of `ipqosconf`, refer to [“How to Apply a New Configuration to the IPQoS Kernel Modules” on page 70](#).

Basic IPQoS Configuration File

An IPQoS configuration file consists of a tree of `action` statements that implement the QoS policy that you defined in [“Planning the Quality-of-Service Policy” on page 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.

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

Configuring the IPQoS Example Topology

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

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

These three 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. The section then shows how to configure a completely different level of service in another configuration file for a server that hosts personal web sites. Both servers are part of the network example that is shown in [Figure 2-4](#).

The following configuration file defines IPQoS activities for the Goldweb server. 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 {
```

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

```

        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

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

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

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

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

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

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

```
fmt_version 1.0
```

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

`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.

- 3 **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`.

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

Global statistics are but one type of parameter you can define in a `params` clause. For syntactical and other details about `params` clauses, refer to “[params Clause](#)” on page 98 and the `ipqosconf(1M)` man page.

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

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

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

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

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

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

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

<code>name video</code>	Creates the class <code>video</code> to identify streaming video traffic that is outgoing from the <code>Goldweb</code> server.
<code>next_action markEF</code>	Instructs the <code>ipgpc</code> module to pass packets of the <code>video</code> class to the <code>markEF</code> statement after <code>ipgpc</code> completes processing. The <code>markEF</code> statement calls the <code>ds_cpmk</code> 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 not turned on.

- See Also**
- To define filters for the class you just created, refer to [“How to Define Filters in the IPQoS Configuration File”](#) on page 50.
 - To create another class clause for the configuration file, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.

▼ How to Define Filters in the IPQoS Configuration File

The next procedure shows how to define filters for a class in the IPQoS configuration file.

Before You Begin The procedure assumes that you have already started file creation and have defined classes. The steps continue building the `/var/ipqos/Goldweb.qos` file that is created in [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.

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

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

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

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

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

`name webout` Gives the name `webout` to the filter.

`sport 80` Selects traffic with a source port of 80, the well-known port for HTTP (web) traffic.

`direction LOCAL_OUT` Further selects traffic that is outgoing from the local system.

`class goldweb` Identifies the class to which the filter belongs, in this instance, class `goldweb`.

For syntactical and detailed information about the `filter` clause in the IPQoS configuration file, refer to “[filter Clause](#)” on page 98.

3 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` Gives the name `videoout` to the filter.

`sport videosrv` Selects traffic with a source port of `videosrv`, a previously defined port for the streaming video application on this system.

`direction LOCAL_OUT` Further selects traffic that is outgoing from the local system.

`class video` Identifies the class to which the filter belongs, in this instance, class `video`.

- See Also**
- To define forwarding behaviors for the marker modules, refer to “[How to Define Traffic Forwarding in the IPQoS Configuration File](#)” on page 51.
 - To define flow-control parameters for the metering modules, refer to “[How to Configure Flow Control in the IPQoS Configuration File](#)” on page 64.
 - To activate the IPQoS configuration file, refer to “[How to Apply a New Configuration to the IPQoS Kernel Modules](#)” on page 70.
 - To define additional filters, refer to “[How to Define Filters in the IPQoS Configuration File](#)” on page 50.
 - To create classes for traffic flows from applications, refer to “[How to Configure the IPQoS Configuration File for an Application Server](#)” on page 60.

▼ How to Define Traffic Forwarding in the IPQoS Configuration File

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

Before You Begin The procedure assumes that you have an existing IPQoS configuration file with already defined classes and already defined filters. The steps continue building the `/var/ipqos/Goldweb.qos` file from [Example 3-1](#).

Note – The procedure shows how to configure traffic forwarding by using the `dscpmk` marker module. For information about traffic forwarding on VLAN systems by using the `dlcosmk` marker, refer to [“Using the `dlcosmk` Marker With VLAN Devices” on page 91](#).

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

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

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

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

2 Invoke the marker with the following action statement.

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

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

`name markAF11` Gives the name `markAF11` to the action statement.

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

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

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

`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](#).

4 Start another marker action statement.

```
action {
    module dscpmk
    name marKEF
```

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

`name marKEF` Gives the name `marKEF` to the action statement.

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

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

`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.

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

For more information, refer to “[How to Configure a Router on an IPQoS-Enabled Network](#)” on page 67.

- See Also**
- 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 54.
 - To define forwarding behaviors for the marker modules, refer to “[How to Define Traffic Forwarding in the IPQoS Configuration File](#)” on page 51.
 - To define flow-control parameters for the metering modules, refer to “[How to Configure Flow Control in the IPQoS Configuration File](#)” on page 64.
 - To activate the IPQoS configuration file, refer to “[How to Apply a New Configuration to the IPQoS Kernel Modules](#)” on page 70.
 - To define additional filters, refer to “[How to Define Filters in the IPQoS Configuration File](#)” on page 50.
 - To create classes for traffic flows from applications, refer to “[How to Configure the IPQoS Configuration File for an Application Server](#)” on page 60.

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

The next procedure shows how to enable accounting on a traffic class in the IPQoS configuration file. The procedure shows how to define flow accounting for the video class, which is introduced in “[How to Create the IPQoS Configuration File and Define Traffic Classes](#)” on page 48. 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 already defined classes, filters, metering actions, if appropriate, and marking actions, if appropriate. The steps continue building the `/var/ipqos/Goldweb.qos` file from [Example 3-1](#).

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

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

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

2 Begin an action statement that calls flow accounting.

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

module flowacct Invokes the flow-accounting module flowacct.

name acct Gives the name acct to the action statement

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

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

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 time out value. A flow “times out” when packets for the flow are not seen during a time out interval. In this parameter, packets time out after 10000 milliseconds.

max_limit 2048 Sets the maximum number of active flow records in the flow table for this action instance.

next_action continue Indicates that no further processing is required on packets of the traffic class video, and that these packets can return to the network stream.

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

- See Also**
- To configure per-hop behaviors on a router, refer to [“How to Configure a Router on an IPQoS-Enabled Network” on page 67](#).
 - To activate the IPQoS configuration file, refer to [“How to Apply a New Configuration to the IPQoS Kernel Modules” on page 70](#).
 - To create classes for traffic flows from applications, refer to [“How to Configure the IPQoS Configuration File for an Application Server” on page 60](#).

▼ 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. As an example, the procedure uses the configuration file from [Example 3–2](#).

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

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

The `/var/ipqos/userweb.qos` file must begin with the partial `action` statement to invoke the `ipgpc` classifier. In addition, the `action` statement also has a `params` clause to turn on statistics collection. For an explanation of this `action` statement, see [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.

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

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

<code>name userweb</code>	Creates a class that is called <code>userweb</code> for forwarding web traffic from users.
<code>next_action markAF1</code>	Instructs the <code>ipgpc</code> module to pass packets of the <code>userweb</code> class to the <code>markAF12</code> action statement after <code>ipgpc</code> completes processing. The <code>markAF12</code> action statement invokes the <code>ds_cpmk</code> marker.
<code>enable_stats FALSE</code>	Enables statistics collection for the <code>userweb</code> class. However, because the value of <code>enable_stats</code> is <code>FALSE</code> , 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 48.

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

```
filter {
    name webout
    sport 80
}
```

```

        direction LOCAL_OUT
        class userweb
    }
}

```

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

For an explanation of the filter clause task, see [“How to Define Filters in the IPQoS Configuration File” on page 50](#).

5 Begin the action statement to invoke the dscpmk marker.

```

action {
    module dscpmk
    name markAF12
}

```

module dscpmk	Invokes the marker module dscpmk.
name markAF12	Gives the name markAF12 to the action statement.

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

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

```

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

```

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.

7 When you complete the IPQoS configuration file, apply the configuration.

- See Also**
- To add classes and other configuration for traffic flows from applications, refer to [“How to Configure the IPQoS Configuration File for an Application Server”](#) on page 60.
 - To configure per-hop behaviors on a router, refer to [“How to Configure a Router on an IPQoS-Enabled Network”](#) on page 67.
 - To activate your IPQoS configuration file, refer to [“How to Apply a New Configuration to the IPQoS Kernel Modules”](#) on page 70.

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
  }
}
```

EXAMPLE 3-3 Sample IPQoS Configuration File for an Application Server (Continued)

```

    }
    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
    }
}
}

```

EXAMPLE 3-3 Sample IPQoS Configuration File for an Application Server (Continued)

```

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

For example, you would create the `/var/ipqos/BigAPPS.qos` file for the application server. Begin with 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 48.

- 2 Create classes to select traffic from three applications on the BigAPPS server.

Add the class definitions after the opening `action` statement.

```

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

```

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

For more information about defining classes, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.

3 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` Gives the name `smtpout` to the filter.

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

- See Also**
- To define filters, refer to [“How to Define Filters in the IPQoS Configuration File”](#) on page 50.
 - To define forwarding behaviors for application traffic, refer to [“How to Configure Forwarding for Application Traffic in the IPQoS Configuration File”](#) on page 62.
 - To configure flow control by using the metering modules, refer to [“How to Configure Flow Control in the IPQoS Configuration File”](#) on page 64.
 - To configure flow accounting, refer to [“How to Enable Accounting for a Class in the IPQoS Configuration File”](#) on page 54.

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

The next procedure shows how to configure forwarding for application traffic. In the procedure, you define per-hop behaviors for application traffic classes that might have lower precedence than other traffic on a network. The steps continue building the `/var/ipqos/BigAPPS.qos` file in [Example 3-3](#).

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

- 1 **Open the IPQoS configuration file that you have created for the application server, and locate the end of the last filter clause.**

In the `/var/ipqos/BigAPPS.qos` file, the last filter is the following:

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

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

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

`module dscpmk` Invokes the marker module `dscpmk`.

`name markAF13` Gives the name `markAF13` to the action statement.

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

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

`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. For a table of possible AF codepoints, refer to [Table 6-2](#).

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

```
action {
    module dscpmk
    name markAF21
    params {
```

```

        global_stats FALSE
        dscp_map{0-63:18}
        next_action continue
    }
}

```

`name markAF21` Gives 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.

- See Also**
- To add configuration information for web servers, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.
 - To configure flow control by using the metering modules, refer to [“How to Configure Flow Control in the IPQoS Configuration File”](#) on page 64.
 - To configure flow accounting, refer to [“How to Enable Accounting for a Class in the IPQoS Configuration File”](#) on page 54.
 - To configure forwarding behaviors on a router, refer to [“How to Configure a Router on an IPQoS-Enabled Network”](#) on page 67.
 - To activate the IPQoS configuration file, refer to [“How to Apply a New Configuration to the IPQoS Kernel Modules”](#) on page 70.

▼ 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.

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

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

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

In the `/var/ipqos/BigAPPS.qos` file, you begin after the following marker action:

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

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

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

`module tokenmt` Invokes the `tokenmt` meter.

`name meterftp` Gives the name `meterftp` to the action statement.

3 Add parameters to configure the meter's rate.

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

`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](#).

4 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.

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

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

module dscpmk	Invokes the marker module dscpmk.
name markAF31	Gives the name markAF31 to the action statement.
global_stats TRUE	Enables statistics for the ftp class.
dscp_map{0-63:26}	Assigns a 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. Then 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. For a table of possible AF codepoints, refer to [Table 6-2](#).

6 Add a marker action statement to assign a per-hop behavior to 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	Gives 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. For a table of possible AF codepoints, refer to [Table 6–2](#).

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

- See Also**
- To activate the IPQoS configuration file, refer to [“How to Apply a New Configuration to the IPQoS Kernel Modules”](#) on page 70.
 - To add configuration information for web servers, refer to [“How to Create the IPQoS Configuration File and Define Traffic Classes”](#) on page 48.
 - To configure flow accounting, refer to [“How to Enable Accounting for a Class in the IPQoS Configuration File”](#) on page 54.
 - To configure forwarding behaviors on a router, refer to [“How to Configure a Router on an IPQoS-Enabled Network”](#) on page 67.

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

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

The next procedure uses as its example the topology in [Figure 2–4](#).

- Before You Begin** The next procedure assumes that you have already configured the IPQoS systems on your network by performing the previous tasks in this chapter.

1 Review the configuration files for all IPQoS-enabled systems on your network.

2 Identify each codepoint that is used in the QoS various policies.

List the codepoints, and the systems and classes, to which the codepoints apply. The next table can illustrate areas where you might have used the same codepoint. This practice is acceptable. However, 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)

3 Add the codepoints from your network's IPQoS configuration files 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.

Starting and Maintaining IPQoS (Tasks)

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

- “Administering IPQoS (Task Map)” on page 69
- “Applying an IPQoS Configuration” on page 70
- “Enabling `syslog` Logging for IPQoS Messages” on page 71
- “Troubleshooting with IPQoS Error Messages” on page 72

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 *Using Virtual Networks in Oracle Solaris 11.1*.

Administering IPQoS (Task Map)

This section lists the set of tasks for starting and maintaining IPQoS on an Oracle Solaris system. Before you use the tasks, you must have a completed IPQoS configuration file, as described in “Defining a QoS Policy in the IPQoS Configuration File (Task Map)” on page 43.

The following table itemizes and describes those tasks and contains links to the sections that detail how to complete these tasks.

Task	Description	For Instructions
1. Configure IPQoS on a system.	Use the <code>ipqosconf</code> command to activate the IPQoS configuration file on a system.	“How to Apply a New Configuration to the IPQoS Kernel Modules” on page 70
2. Make the Oracle Solaris startup scripts apply the debugged IPQoS configuration file after each system boot.	Ensure that the IPQoS configuration is applied each time the system reboots.	“How to Ensure That the IPQoS Configuration Is Applied After Each Reboot” on page 71.

Task	Description	For Instructions
3. Enable sys log logging for IPQoS.	Add an entry to enable sys log logging of IPQoS messages.	“How to Enable Logging of IPQoS Messages During Booting” on page 71.
4. Fix any IPQoS problems that arise.	Troubleshoot IPQoS problems by using error messages.	Refer to the error messages in Table 4–1 .

Applying an IPQoS Configuration

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

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

You use the `ipqosconf` command to read the IPQoS configuration file and to configure the IPQoS modules in the UNIX kernel. The next procedure uses as an example the file `/var/ipqos/Goldweb.qos`, which is created in [“Creating IPQoS Configuration Files for Web Servers” on page 46](#). For detailed information, refer to the `ipqosconf(1M)` man page.

1 Become an administrator.

For more information, see [“How to Use Your Assigned Administrative Rights” in Oracle Solaris 11.1 Administration: Security Services](#).

2 Apply the new configuration.

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

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

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

3 Test and debug the new IPQoS configuration.

Use UNIX utilities to track IPQoS behavior and to gather statistics on your IPQoS implementation. This information can help you determine if 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 [“Enabling syslog Logging for IPQoS Messages”](#) on page 71.
- To ensure that the current IPQoS configuration is applied after each boot, refer to [“How to Ensure That the IPQoS Configuration Is Applied After Each Reboot”](#) on page 71.

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

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

1 Become an administrator.

For more information, see [“How to Use Your Assigned Administrative Rights”](#) in *Oracle Solaris 11.1 Administration: Security Services*.

2 Test for the existence of an IPQoS configuration in the kernel modules.

```
# ipqosconf -l
```

If a configuration already exists, `ipqosconf` displays the configuration on the screen. If you do not receive output, apply the configuration, as explained in [“How to Apply a New Configuration to the IPQoS Kernel Modules”](#) on page 70.

3 Ensure that the existing IPQoS configuration is applied every time the IPQoS system reboots.

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

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

Enabling syslog Logging for IPQoS Messages

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

▼ How to Enable Logging of IPQoS Messages During Booting

1 Become an administrator.

For more information, see [“How to Use Your Assigned Administrative Rights”](#) in *Oracle Solaris 11.1 Administration: Security Services*.

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

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

Use tabs rather than spaces between the columns.

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

- 4 **Reboot the system to apply the messages.**

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.

```
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 in your IPQoS system's `/var/adm/messages` file.

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

For a description of these error messages, see [Table 4-1](#).

Troubleshooting with IPQoS Error Messages

This section contains a table of 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 <i>parameter-name</i> 's action <i>action-name</i>	In the IPQoS configuration file, the action name that you specified in <i>parameter-name</i> does not exist in the configuration file.	Create the action. Or, refer to a different, existing action in the parameter.

TABLE 4-1 IPQoS Error Messages (Continued)

Error Message	Description	Solution
action <i>action-name</i> involved in cycle	In the IPQoS configuration file, <i>action-name</i> 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 <i>action-name</i> isn't referenced by any other actions	A non- <i>ipgpc</i> 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 <i>fmt_version</i>	The format of the configuration file is not specified as the first entry of the file, which is required by IPQoS.	Add the format version, as explained in “ How to Create the IPQoS Configuration File and Define Traffic Classes ” on page 48.
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 <i>fmt_version 1.0</i> , which is required beginning with the Solaris 9 9/02 release of IPQoS.
No <i>ipgpc</i> action defined.	You did not define an action for the <i>ipgpc</i> classifier in the configuration file, which is an IPQoS requirement.	Define an action for <i>ipgpc</i> , as shown in “ How to Create the IPQoS Configuration File and Define Traffic Classes ” on page 48.
Can't commit a null configuration	When you ran <i>ipqosconf -c</i> to commit a configuration, that configuration was empty, which IPQoS does not allow.	Be sure to apply a configuration file before you attempt to commit a configuration. For instructions, see “ How to Apply a New Configuration to the IPQoS Kernel Modules ” on page 70.
Invalid CIDR mask on line <i>line-number</i>	In the configuration file, you used a CIDR mask as part of the IP address that is out of the valid range for IP addresses.	Change the mask value to be in the range of 1–32 for IPv4 and 1–128 for IPv6.
Address masks aren't allowed for host names line <i>line-number</i>	In the configuration file, you defined a CIDR mask for a host name, which is not allowed in IPQoS.	Remove the mask or change the host name to an IP address.
Invalid module name line <i>line-number</i>	In the configuration file, the module name that you specified in an action statement is invalid.	Check the spelling of the module name. For a list of IPQoS modules, refer to Table 6-5 .
<i>ipgpc</i> action has incorrect name line <i>line-number</i>	The name that you gave to the <i>ipgpc</i> action in the configuration file is not the required <i>ipgpc.classify</i> .	Rename the action <i>ipgpc.classify</i> .
Second parameter clause not supported line <i>line-number</i>	In the configuration file, you specified two parameter clauses for a single action, which IPQoS does not allow.	Combine all parameters for the action into a single parameters clause.
Duplicate named action	In the configuration file, you gave the same name to two actions.	Rename or remove one of the actions.

TABLE 4-1 IPQoS Error Messages (Continued)

Error Message	Description	Solution
Duplicate named filter/class in action <i>action-name</i>	You gave the same name to two filters or two classes in the same action, which is not allowed in the IPQoS configuration file.	Rename or remove one of the filters or classes.
Undefined class in filter <i>filter-name</i> in action <i>action-name</i>	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 <i>class-name</i> action <i>action-name</i>	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 <i>action-name</i>	In the configuration file, one of the parameters is invalid.	For the module that is called by the named action, refer to the module entry in “ IPQoS Architecture and the Diffserv Model ” on page 83. Alternatively, you can refer to the <code>ipqosconf(1M)</code> man page.
Mandatory parameter missing for action <i>action-name</i>	You have not defined a required parameter for an action in the configuration file.	For the module that is called by the named action, refer to the module entry in “ IPQoS Architecture and the Diffserv Model ” on page 83. Alternatively, you can refer to the <code>ipqosconf(1M)</code> man page.
Max number of classes reached in <code>ipgpc</code>	You specified more classes than are allowed in the <code>ipgpc</code> 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 <code>/etc/system</code> file the entry <code>ipgpc_max_classesclass-number</code> .
Max number of filters reached in action <code>ipgpc</code>	You specified more filters than are allowed in the <code>ipgpc</code> action of the IPQoS configuration file. The maximum number is 10007.	Review the configuration file, and remove unneeded filters. Alternatively, you can raise the maximum number of filters by adding to the <code>/etc/system</code> file the entry <code>ipgpc_max_filtersfilter-number</code> .
Invalid/missing parameters for filter <i>filter-name</i> in action <code>ipgpc</code>	In the configuration file, filter <i>filter-name</i> has an invalid or missing parameter.	Refer to the <code>ipqosconf(1M)</code> man page for the list of valid parameters.
Name not allowed to start with ‘!’, line <i>line-number</i>	You began an action, filter, or class name with an exclamation mark (!), which is not allowed in the IPQoS file.	Remove the exclamation mark, or rename the action, class, or filter.
Name exceeds the maximum name length line <i>line-number</i>	You defined a name for an action, class, or filter in the configuration file that exceeds the maximum length of 23 characters.	Give a shorter name to the action, class, or filter.
Array declaration line <i>line-number</i> is invalid	In the configuration file, the array declaration for the parameter on line <i>line-number</i> is invalid.	For the correct syntax of the array declaration that is called by the <code>action</code> statement with the invalid array, refer to “ IPQoS Architecture and the Diffserv Model ” on page 83. Alternatively, refer to the <code>ipqosconf(1M)</code> man page.

TABLE 4-1 IPQoS Error Messages (Continued)

Error Message	Description	Solution
Quoted string exceeds line, <i>line-number</i>	The string does not have the terminating quotation marks on the same line, which is required in the configuration file.	Make sure that the quoted string begins and ends on the same line in the configuration file.
Invalid value, line <i>line-number</i>	The value that is given on <i>line-number</i> of the configuration file is not supported for the parameter.	For the acceptable values for the module that is called by the <code>act ion</code> statement, refer to the module description in “ IPQoS Architecture and the Diffserv Model ” on page 83. Alternatively, you can refer to the <code>ipqos conf(1M)</code> man page.
Unrecognized value, line <i>line-number</i>	The value on <i>line-number</i> 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 <code>act ion</code> statement with the unrecognized line number, refer to “ IPQoS Architecture and the Diffserv Model ” on page 83. Alternatively, you can refer to the <code>ipqos conf(1M)</code> man page.
Malformed value list line <i>line-number</i>	The enumeration that is specified on <i>line-number</i> of the configuration file does not conform to the specification syntax.	For correct syntax for the module that is called by the <code>act ion</code> statement with the malformed value list, refer to the module description in “ IPQoS Architecture and the Diffserv Model ” on page 83. Alternatively, you can refer to the <code>ipqos conf(1M)</code> man page.
Duplicate parameter line <i>line-number</i>	A duplicate parameter was specified on <i>line-number</i> , which is not allowed in the configuration file.	Remove one of the duplicate parameters.
Invalid action name line <i>line-number</i>	You gave the action on <i>line-number</i> of the configuration file a name that 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 <i>line-number</i> , ignoring filter	<code>ipqos conf</code> 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 <i>line-number</i>	The IP version of the address on <i>line-number</i> is incompatible with the version of a previously specified IP address or <code>ip_ version</code> parameter.	Change the two conflicting entries to be compatible.
Action at line <i>line-number</i> has the same name as currently installed action, but is for a different module	You tried to change the module of an action that already exists in the system's IPQoS configuration, which is not allowed.	Flush the current configuration before you apply the new configuration.

Using Flow Accounting and Statistics Gathering (Tasks)

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

- “Setting Up Flow Accounting (Task Map)” on page 77
- “Recording Information About Traffic Flows” on page 78
- “Gathering Statistical Information” on page 80

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 *Using Virtual Networks in Oracle Solaris 11.1*.

Setting Up Flow Accounting (Task Map)

The following task map lists the generic tasks for obtaining information about traffic flows by using the `flowacct` module. The map also links to procedures to carry out these tasks.

Task	Description	For Instructions
1. Create a file to contain accounting information for traffic flows.	Use the <code>acctadm</code> command to create a file that holds the results of processing by <code>flowacct</code> .	“How to Create a File for Flow-Accounting Data” on page 78
2. Define <code>flowacct</code> parameters in the IPQoS configuration file.	Define values for the <code>timer</code> , <code>timeout</code> , and <code>max_limit</code> parameters.	“How to Enable Accounting for a Class in the IPQoS Configuration File” on page 54

Recording Information About Traffic Flows

You use the IPQoS `flowacct` module to collect information about traffic flows. For example, you can collect source and destination addresses, 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 is defined in their service-level agreements (SLAs). You can also use flow accounting to obtain flow statistics for critical applications. This section contains tasks for using `flowacct` with the Oracle Solaris extended accounting facility to obtain data on traffic flows.

The following information is contained in sources outside this chapter:

- For instructions on creating an action statement for `flowacct` in the IPQoS configuration file, refer to [“How to Configure Flow Control in the IPQoS Configuration File”](#) on page 64.
- 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. You use the `acctadm` command for this purpose. `acctadm` can record either basic attributes or extended attributes in the file. All `flowacct` attributes are listed in [Table 6–4](#). For detailed information about `acctadm`, refer to the `acctadm(1M)` man page.

1 Become an administrator.

For more information, see [“How to Use Your Assigned Administrative Rights”](#) in *Oracle Solaris 11.1 Administration: Security Services*.

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.

<code>/var/ipqos/goldweb/account.info</code>	Specifies the fully qualified path name of the file to hold the flow records from <code>flowacct</code> .
<code>flow</code>	Instructs <code>acctadm</code> to enable flow accounting.

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

`acctadm` generates the following output:

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

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

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

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.

Instructions for viewing the contents of a flow-accounting file are in [“Perl Interface to libexacct” in Oracle Solaris Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management](#).

- See Also**
- For detailed information on the extended accounting feature, refer to [Chapter 4, “Extended Accounting \(Overview\)”](#) in *Oracle Solaris Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management*.
 - 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 54.
 - To print the data in the file that was created with `acctadm`, refer to [“Perl Interface to libexacct”](#) in *Oracle Solaris Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management*.

Gathering Statistical Information

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

```
/bin/kstat -m ipqos-module-name
```

You can specify any valid IPQoS module name, as shown in [Table 6–5](#). For example, to view statistics that are generated by the `dscpmk` marker, you use the following form of `kstat`:

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

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

EXAMPLE 5–1 kstat Statistics for IPQoS

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

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

`class: flacct` Gives 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.

EXAMPLE 5-1 `kstat` Statistics for IPQoS (Continued)

<code>epackets</code>	Number of packets that resulted in an error during processing, in this example 0.
<code>flows_in_tbl</code>	Number of flow records in the flow table, which in this example is 1. When no records are in the table, the value for <code>flows_in_tbl</code> is 0.
<code>nbytes</code>	Total number of bytes that are seen by this <code>flowacct</code> 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.
<code>npackets</code>	Total number of packets that are seen by this <code>flowacct</code> action instance, which is 1 in the example. <code>npackets</code> includes packets that are currently in the flow table. <code>npackets</code> also includes packets that have timed out – are no longer in the flow table.
<code>usedmem</code>	Memory in bytes in use by the flow table that is maintained by this <code>flowacct</code> instance. The <code>usedmem</code> value is 256 in the example. The value for <code>usedmem</code> is 0 when the flow table does not have any flow records.

IPQoS in Detail (Reference)

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

- “IPQoS Architecture and the Diffserv Model” on page 83
- “IPQoS Configuration File” on page 95
- “ipqosconf Configuration Utility” on page 99

For an overview, refer to Chapter 1, “Introducing IPQoS (Overview).” For planning information, refer to Chapter 2, “Planning for an IPQoS-Enabled Network (Tasks).” 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. Users are encouraged to instead use the `dladm`, `flowadm`, and related commands, which support similar bandwidth resource control features. For more information, see *Using Virtual Networks in Oracle Solaris 11.1*.

IPQoS Architecture and the Diffserv Model

This section describes the IPQoS architecture and how IPQoS implements the differentiated services (Diffserv) model that is defined in RFC 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 `dicosmk` 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 of `ipgpc`, 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 next table lists the selectors that are available for `ipgpc`.

TABLE 6-1 Filter Selectors for the IPQoS Classifier

Selector	Argument	Information Selected
<code>saddr</code>	IP address number.	Source address.

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

Selector	Argument	Information Selected
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	Argument can be one of the following: LOCAL_IN LOCAL_OUT FWD_IN FWD_OUT	Direction of packet flow on the IPQoS machine. Input traffic local to the IPQoS system. Output traffic local to the IPQoS system. Input traffic to be forwarded. Output traffic to be forwarded.
precedence	Precedence value. Highest precedence is 0.	Precedence is 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. Committed rate and peak rate are explained in the next section.

tokenmt Metering Module

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

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

Configuration parameters for `tokenmt` are as follows:

- `committed_rate` – Specifies the committed rate of the flow in bits per second.
- `committed_burst` – Specifies the committed burst size in bits. The `committed_burst` parameter defines how many outgoing packets of a particular class can pass onto the network at the committed rate.
- `peak_rate` – Specifies the peak rate in bits per second.
- `peak_burst` – Specifies the peak or excess burst size in bits. The `peak_burst` parameter grants to a traffic class a peak-burst size that exceeds the committed rate.

- `color_aware` – Turns on 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 turn on color awareness by setting the `color_aware` parameter to `true`. As a color-aware meter, `tokenmt` assumes that the packet has already been marked as red, yellow, or green by a previous `tokenmt` action. Color-aware `tokenmt` evaluates a packet by using the 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`.

tswtc_lmt Metering Module

The `tswtc_lmt` metering module estimates average bandwidth for a traffic class by using a time-based *rate estimator*. `tswtc_lmt` 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*. The rate estimation algorithm is taken from RFC 2859, *A Time Sliding Window Three Colour Marker*.

You use the following parameters to configure `tswtc_lmt`:

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

For technical details on `tswtc_lmt`, refer to the `tswtc_lmt(7ipp)` man page. For general information on rate shapers that are similar to `tswtc_lmt`, see RFC 2963, *A Rate Adaptive Shaper for Differentiated Services* (<http://www.ietf.org/rfc/rfc2963.txt?number=2963>).

Marker Module

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

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

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 only govern forwarding behaviors on the IPQoS system's contiguous network. For more information on PHBs, 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 well-known forwarding behavior that is defined in the IPQoS configuration file. Thereafter, the IPQoS system and subsequent Diffserv-aware systems forward the traffic as indicated in the DS field until the mark changes. To assign a PHB, the IPQoS system marks 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. For technical information about EF, refer to RFC 2598, *An Expedited Forwarding PHB*.

Assured Forwarding (AF) PHB

Assured forwarding (AF) provides four different classes of forwarding behaviors that you can specify to the marker. The next table shows the classes, the three drop precedences that are

provided with each class, and the recommended 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 = 10 (001010)	AF21 = 18 (010010)	AF31 = 26 (011010)	AF41 = 34 (100010)
Medium-Drop Precedence	AF12 = 12 (001100)	AF22 = 20 (010100)	AF32 = 28 (011100)	AF42 = 36 (100100)
High-Drop Precedence	AF13 = 14 (001110)	AF23 = 22 (010110)	AF33 = 30 (011110)	AF43 = 38 (100110)

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

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 packets' 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.

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

Supplying a 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_codepoint}
```

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, which are well known and understood by 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 next table.

TABLE 6-3 801.D User-Priority Values

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

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

IPQoS Configuration for Systems With VLAN Devices

This section introduces a simple network scenario that shows how to implement IPQoS on systems with VLAN devices. The scenario includes two IPQoS systems, `machine1` and

machine2, that are connected by a switch. The VLAN device on machine1 has the IP address 10.10.8.1. The VLAN device on machine2 has the IP address 10.10.8.3.

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

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

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

    filter {
        name myfilter2
        daddr 10.10.8.3
        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 about flowacct, refer to the flowacct(7ipp) man page.

To enable flow accounting, you need to use the Oracle Solaris `acct` accounting facility and the `acctadm` command, as well as `flowacct`. For the overall steps in setting up flow accounting, refer to [“Setting Up Flow Accounting \(Task Map\)” on page 77](#).

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 64](#).

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 time out interval is 60 seconds. Entries that have timed out are then written to the accounting file that is created with the `acctadm` command.

flowacct Records

A `flowacct` record contains the attributes described in the following table.

TABLE 6-4 Attributes of a `flowacct` Record

Attribute Name	Attribute Contents	Type
<code>src-addr-address-type</code>	Source address of the originator. <i>address-type</i> is either v4 for IPv4 or v6 for IPv6, as specified in the IPQoS configuration file.	Basic
<code>dest-addr-address-type</code>	Destination address for the packets. <i>address-type</i> is either v4 for IPv4 or v6 for IPv6, as specified in the IPQoS configuration file.	Basic
<code>src-port</code>	Source port from which the flow originated.	Basic
<code>dest-port</code>	Destination port number to which this flow is bound.	Basic
<code>protocol</code>	Protocol number for the flow.	Basic
<code>total-packets</code>	Number of packets in the flow.	Basic
<code>total-bytes</code>	Number of bytes in the flow.	Basic
<code>action-name</code>	Name of the <code>flowacct</code> action that recorded this flow.	Basic
<code>creation-time</code>	First time that a packet is seen for the flow by <code>flowacct</code> .	Extended only
<code>last-seen</code>	Last time that a packet of the flow was seen.	Extended only
<code>diffserv-field</code>	DSCP in the outgoing packet headers of the flow.	Extended only
<code>user</code>	Either a UNIX User ID or user name, which is obtained from the application.	Extended only
<code>projid</code>	Project ID, which is obtained from the application.	Extended only

Using acctadm with the flowacct Module

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

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. *file-type* must be replaced by either `basic` or `extended`. For a list of attributes in each file type, refer to [Table 6–4](#).

`-f file-name` Creates the file *file-name* to hold the flow records.

`flow` Indicates that `acctadm` is to be run with IPQoS.

IPQoS Configuration File

This section contains full details about the parts of the IPQoS configuration file. The IPQoS boot-time activated policy is stored in the file `/etc/inet/ipqosinit.conf`. Although you can edit this file, the best practice for a new IPQoS system is to create a configuration file with a different name. Tasks for applying and debugging an IPQoS configuration are in [Chapter 3](#), “Creating the IPQoS Configuration File (Tasks).”

The syntax of the IPQoS configuration file is shown in [Example 6–3](#).

The example uses the following conventions:

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

EXAMPLE 6–3 Syntax of the IPQoS Configuration File

```
file_format_version ::= fmt_version version

action_clause ::= action {
    name action-name
    module module-name
    params-clause | ""
    cf-clauses
}
action_name ::= string
```

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

```

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

params_clause ::= params {
    parameters
    params-stats | ""
}
parameters ::= prm-name-value parameters | ""
prm_name_value ::= param-name param-value

params_stats ::= global-stats boolean

cf_clauses ::= class-clause cf-clauses |
              filter-clause cf-clauses | ""

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

filter_clause ::= filter {
    name filter-name
    class class-name
    parameters
}
filter_name ::= string

```

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

action Statement

You use action statements to invoke the various IPQoS modules that are described in [“IPQoS Architecture and the Diffserv Model”](#) on page 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
}
```

<i>name action_name</i>	Assigns a name to the action.
<i>module module_name</i>	Identifies the IPQoS module to be invoked, which must be one of the modules in Table 6-5 .
<i>params_clause</i>	Can be parameters for the classifier to process, such as global statistics or the next action to process.
<i>cf_clauses</i>	A set of zero or more class clauses or filter clauses

Module Definitions

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

TABLE 6-5 IPQoS Modules

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

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

```
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 turn on 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 turns on UNIX style statistics for the action statement where global statistics is invoked. You can view the statistics by using the `ksstat` command. You must enable action statement statistics before you can enable per-class statistics.

ipqosconf Configuration Utility

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

- Applies the configuration file to the IPQoS kernel modules (`ipqosconf -a filename`)
- Lists the IPQoS configuration file currently resident in the kernel (`ipqosconf -l`)
- Ensures that the current IPQoS configuration is read and applied each time the machine reboots (`ipqosconf -c`)
- Flushes the current IPQoS kernel modules (`ipqosconf -f`)

For technical information, refer to the [ipqosconf\(1M\)](#) man page.

Index

A

- acctadm command, for flow accounting, 17, 79, 94
- action statement, 96
- application server, configuring for IPQoS, 58
- assured forwarding (AF), 20, 89
 - AF codepoints table, 89
 - for a marker action statement, 53

B

- bandwidth regulation, 13–14
 - planning, in the QoS policy, 30

C

- class clause, in the IPQoS configuration file, 97
- class clause, in the IPQoS configuration file, 49
- class of service (CoS) mark, 17
- classes, 15
 - defining, in the IPQoS configuration file, 56, 60
 - selectors, list of, 84
 - syntax of class clause, 97
- classes of service, *See* classes
- classifier module, 15–16
 - action statement, 48
 - functions of the classifier, 84
- color awareness, 16, 87

D

- differentiated services, 9–10
 - differentiated services model, 15–19
 - network topologies, 24
 - providing different classes of service, 14
- Diffserv-aware router
 - evaluating DS codepoints, 90
 - planning, 29
- Diffserv model
 - classifier module, 15–16
 - flow example, 18
 - IPQoS implementation, 15–19, 17–18, 18
 - marker modules, 17
 - meter modules, 16–17
- dlsosmk marker, 17
 - planning datagram forwarding, 36
 - 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, 68, 89
 - defining, in the IPQoS configuration file, 52
 - dscp_map parameter, 90
 - EF forwarding codepoint, 20, 89
 - PHBs and the DSCP, 19
 - planning, in the QoS policy, 37
- dscpmk marker, 17
 - invoking, in a marker action statement, 52, 57, 63, 66
 - PHBs for packet forwarding, 89–91
 - planning packet forwarding, 36

E

- error messages for IPQoS, 72
- example IPQoS configuration files
 - application server, 58
 - best-effort web server, 47
 - color-awareness segment, 87
 - premium web server, 46
 - VLAN device configuration, 92
- expedited forwarding (EF), 20, 89
 - defining, in the IPQoS configuration file, 53

F

- filter clause, in the IPQoS configuration file, 50, 98
- filters, 16
 - creating, in the IPQoS configuration file, 56, 61
 - filter clause syntax, 98
 - planning, in the QoS policy, 31
 - selectors, list of, 84
- flow accounting, 78–80, 92
 - flow record table, 93–94
- flow control, through the metering modules, 16
- flowacct module, 17–18, 92
 - acctadm command, for creating a flow accounting file, 94
 - action statement for flowacct, 55
 - attributes of flow records, 94
 - flow record table, 93–94
 - flow records, 78
 - parameters, 93
- forwarding traffic
 - datagram forwarding, 91–92
 - effect of PHBs on packet forwarding, 89–91
 - IP packet forwarding, with DSCP, 19
 - planning, in the QoS policy, 30
 - traffic flow through Diffserv networks, 20

H

- hardware for IPQoS-enabled networks, 24

I

- ipgpc classifier, *See* classifier module
- IPQoS, 9
 - configuration example, 39–41
 - configuration file, 45, 95
 - action statement syntax, 97
 - class clause, 49
 - filter clause, 50
 - initial action statement, 48
 - initial action statement, 96
 - list of IPQoS modules, 97
 - marker action statement, 52
 - syntax, 95
 - configuration planning, 23
 - Diffserv model implementation, 15–19
 - error messages, 72
 - features, 10
 - man pages, 11–12
 - message logging, 71
 - network example, 45
 - network topologies supported, 24, 25, 26
 - QoS policy planning, 27
 - related RFCs, 11
 - routers on an IPQoS network, 67
 - statistics generation, 80
 - traffic management capabilities, 13, 14
 - VLAN device support, 91–92
- ipqosconf, 45
- ipqosconf command
 - applying a configuration, 70, 71
 - command options, 99
 - listing the current configuration, 71

K

- kstat command, use with IPQoS, 80

L

- load balancing, in an IPQoS-enabled network, 25

M

- marker modules, 17
 - See also* `dlcosmk` marker
 - See also* `dscpmk` marker
- PHBs, for IP packet forwarding, 19
- specifying a DS codepoint, 90–91
- support for VLAN devices, 91–92
- metering modules
 - See also* `tokenmt` meter
 - See also* `tswtclmt` meter
- introduction, 16–17
- invoking, in the IPQoS configuration file, 65
- outcomes of metering, 16, 86

N

- network example for IPQoS, 45
- network topologies for IPQoS, 24
 - configuration example, 39
 - 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, 48, 98
 - for a `flowacct` action, 55
 - for a marker action, 52
 - for a metering action, 65
 - syntax, 98
- per-hop behavior (PHB), 19
 - AF forwarding, 20
 - defining, in the IPQoS configuration file, 66
 - EF forwarding, 20
 - using, with `dscpmk` marker, 89–91

Q

- QoS policy, 13
 - creating filters, 31
 - implementing, in the IPQoS configuration file, 43

- QoS policy (*Continued*)
 - planning task map, 28
 - template for policy organization, 27
- quality of service (QoS)
 - QoS policy, 12–13
 - tasks, 10

R

- Requests for Comments (RFCs), IPQoS, 11

S

- selectors, 16
 - IPQoS 5-tuple, 15
 - planning, in the QoS policy, 31
 - selectors, list of, 84
- service-level agreement (SLA), 12
 - billing clients, based on flow accounting, 78
 - classes of services, 15
 - providing different classes of service, 14
- statistics for IPQoS
 - enabling class-based statistics, 98
 - enabling global statistics, 49, 98
 - generating, through the `ksstat` command, 80
- `syslog.conf` file logging for IPQoS, 71

T

- task map
 - IPQoS
 - configuration planning, 23
- task maps
 - IPQoS
 - configuration file creation, 43
 - flow-accounting setup, 77
 - QoS policy planning, 28
- `tokenmt` meter, 16
 - color-awareness configuration, 16, 87
 - metering rates, 86–88
 - rate parameters, 86
 - single-rate meter, 87

tokenmt meter (*Continued*)

two rate-meter, 87

traffic conformance

defining, 65

outcomes, 16, 86

planning

outcomes in the QoS policy, 35

rates in the QoS policy, 34

rate parameters, 86

traffic management

controlling flow, 16–17

forwarding traffic, 19, 20, 21

planning network topologies, 24

prioritizing traffic flows, 14

regulating bandwidth, 13

tswtclmt meter, 16, 88

metering rates, 88

U

user priority value, 17

V

virtual LAN (VLAN) devices on an IPQoS

network, 91–92

W

web servers

configuring for IPQoS, 46, 47, 55, 57