# QoS: Congestion Avoidance Configuration Guide, Cisco IOS Release 12.2SX

**Americas Headquarters**
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

# CONTENTS

## Last Updated: September 05, 2011

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Cisco and the Cisco Logo are trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and other countries. A listing of Cisco's trademarks can be found at www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1005R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

# Congestion Avoidance Overview

Congestion avoidance techniques monitor network traffic loads in an effort to anticipate and avoid congestion at common network bottlenecks. Congestion avoidance is achieved through packet dropping. Among the more commonly used congestion avoidance mechanisms is Random Early Detection (RED), which is optimum for high-speed transit networks. Cisco IOS QoS includes an implementation of RED that, when configured, controls when the router drops packets. If you do not configure Weighted Random Early Detection (WRED), the router uses the cruder default packet drop mechanism called tail drop.

This module gives a brief description of the kinds of congestion avoidance mechanisms provided by the Cisco IOS QoS features. It discusses the following features:

- Tail drop. This is the default congestion avoidance behavior when WRED is not configured.
- WRED. WRED and distributed WRED (DWRED)--both of which are the Cisco implementations of RED--combine the capabilities of the RED algorithm with the IP Precedence feature. Within the section on WRED, the following related features are discussed:
    - Flow-based WRED. Flow-based WRED extends WRED to provide greater fairness to all flows on an interface in regard to how packets are dropped.
    - DiffServ Compliant WRED. DiffServ Compliant WRED extends WRED to support Differentiated Services (DiffServ) and Assured Forwarding (AF) Per Hop Behavior (PHB). This feature enables customers to implement AF PHB by coloring packets according to differentiated services code point (DSCP) values and then assigning preferential drop probabilities to those packets.

# Finding Feature Information

Your software release may not support all the features documented in this module. For the latest feature information and caveats, see the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the Feature Information Table at the end of this document.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

# Tail Drop

Tail drop treats all traffic equally and does not differentiate between classes of service. Queues fill during periods of congestion. When the output queue is full and tail drop is in effect, packets are dropped until the congestion is eliminated and the queue is no longer full.

# Weighted Random Early Detection

This section gives a brief introduction to RED concepts and addresses WRED, the Cisco implementation of RED for standard Cisco IOS platforms.

WRED avoids the globalization problems that occur when tail drop is used as the congestion avoidance mechanism on the router. Global synchronization occurs as waves of congestion crest only to be followed by troughs during which the transmission link is not fully utilized. Global synchronization of TCP hosts, for example, can occur because packets are dropped all at once. Global synchronization manifests when multiple TCP hosts reduce their transmission rates in response to packet dropping, then increase their transmission rates once again when the congestion is reduced.

## About Random Early Detection

The RED mechanism was proposed by Sally Floyd and Van Jacobson in the early 1990s to address network congestion in a responsive rather than reactive manner. Underlying the RED mechanism is the premise that most traffic runs on data transport implementations that are sensitive to loss and will temporarily slow down when some of their traffic is dropped. TCP, which responds appropriately--even robustly--to traffic drop by slowing down its traffic transmission, effectively allows the traffic-drop behavior of RED to work as a congestion-avoidance signalling mechanism.

TCP constitutes the most heavily used network transport. Given the ubiquitous presence of TCP, RED offers a widespread, effective congestion-avoidance mechanism.

In considering the usefulness of RED when robust transports such as TCP are pervasive, it is important to consider also the seriously negative implications of employing RED when a significant percentage of the traffic is not robust in response to packet loss. Neither Novell NetWare nor AppleTalk is appropriately robust in response to packet loss, therefore you should not use RED for them.

### How It Works

RED aims to control the average queue size by indicating to the end hosts when they should temporarily slow down transmission of packets.

RED takes advantage of the congestion control mechanism of TCP. By randomly dropping packets prior to periods of high congestion, RED tells the packet source to decrease its transmission rate. Assuming the packet source is using TCP, it will decrease its transmission rate until all the packets reach their destination, indicating that the congestion is cleared. You can use RED as a way to cause TCP to slow down transmission of packets. TCP not only pauses, but it also restarts quickly and adapts its transmission rate to the rate that the network can support.

RED distributes losses in time and maintains normally low queue depth while absorbing spikes. When enabled on an interface, RED begins dropping packets when congestion occurs at a rate you select during configuration.

## Packet Drop Probability

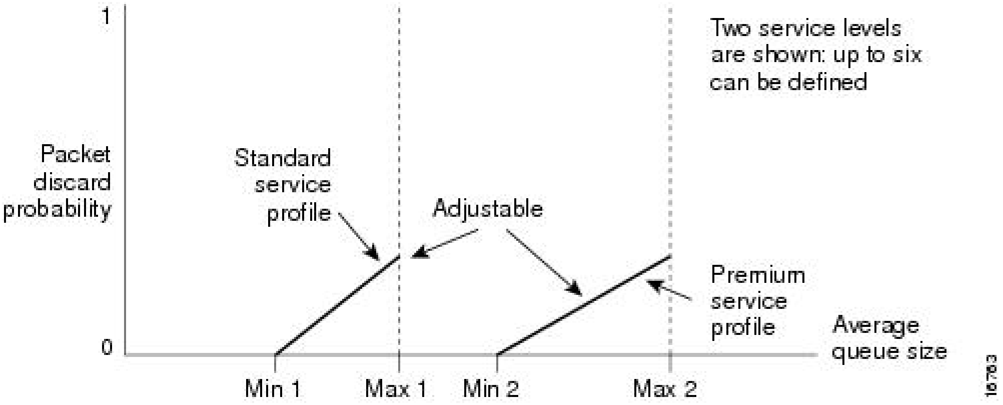
The packet drop probability is based on the minimum threshold, maximum threshold, and mark probability denominator.

When the average queue depth is above the minimum threshold, RED starts dropping packets. The rate of packet drop increases linearly as the average queue size increases until the average queue size reaches the maximum threshold.

The mark probability denominator is the fraction of packets dropped when the average queue depth is at the maximum threshold. For example, if the denominator is 512, one out of every 512 packets is dropped when the average queue is at the maximum threshold.

When the average queue size is above the maximum threshold, all packets are dropped. The figure below summarizes the packet drop probability.

*Figure 1*



The minimum threshold value should be set high enough to maximize the link utilization. If the minimum threshold is too low, packets may be dropped unnecessarily, and the transmission link will not be fully used.

The difference between the maximum threshold and the minimum threshold should be large enough to avoid global synchronization of TCP hosts (global synchronization of TCP hosts can occur as multiple TCP hosts reduce their transmission rates). If the difference between the maximum and minimum thresholds is too small, many packets may be dropped at once, resulting in global synchronization.

## How TCP Handles Traffic Loss

When the recipient of TCP traffic--called the receiver--receives a data segment, it checks the four octet (32-bit) sequence number of that segment against the number the receiver expected, which would indicate that the data segment was received in order. If the numbers match, the receiver delivers all of the data that it holds to the target application, then it updates the sequence number to reflect the next number in order, and finally it either immediately sends an acknowledgment (ACK) packet to the sender or it schedules an ACK to be sent to the sender after a short delay. The ACK notifies the sender that the receiver received all data segments up to but not including the one marked with the new sequence number.

Receivers usually try to send an ACK in response to alternating data segments they receive; they send the ACK because for many applications, if the receiver waits out a small delay, it can efficiently include its reply acknowledgment on a normal response to the sender. However, when the receiver receives a data segment out of order, it immediately responds with an ACK to direct the sender to resend the lost data segment.

When the sender receives an ACK, it makes this determination: It determines if any data is outstanding. If no data is outstanding, the sender determines that the ACK is a keepalive, meant to keep the line active, and it does nothing. If data is outstanding, the sender determines whether the ACK indicates that the receiver has received some or none of the data. If the ACK indicates receipt of some data sent, the sender determines if new credit has been granted to allow it to send more data. When the ACK indicates receipt of none of the data sent and there is outstanding data, the sender interprets the ACK to be a repeatedly sent ACK. This condition indicates that some data was received out of order, forcing the receiver to remit the first ACK, and that a second data segment was received out of order, forcing the receiver to remit the second ACK. In most cases, the receiver would receive two segments out of order because one of the data segments had been dropped.

When a TCP sender detects a dropped data segment, it resends the segment. Then it adjusts its transmission rate to half of what is was before the drop was detected. This is the TCP back-off or slow-down behavior. Although this behavior is appropriately responsive to congestion, problems can arise when multiple TCP sessions are carried on concurrently with the same router and all TCP senders slow down transmission of packets at the same time.

## How the Router Interacts with TCP

To see how the router interacts with TCP, we will look at an example. In this example, on average, the router receives traffic from one particular TCP stream every other, every 10th, and every 100th or 200th message in the interface in MAE-EAST or FIX-WEST. A router can handle multiple concurrent TCP sessions. Because network flows are additive, there is a high probability that when traffic exceeds the Transmit Queue Limit (TQL) at all, it will vastly exceed the limit. However, there is also a high probability that the excessive traffic depth is temporary and that traffic will not stay excessively deep except at points where traffic flows merge or at edge routers.

If the router drops all traffic that exceeds the TQL, as is done when tail drop is used by default, many TCP sessions will simultaneously go into slow start. Consequently, traffic temporarily slows down to the extreme and then all flows slow-start again; this activity creates a condition of global synchronization.

However, if the router drops no traffic, as is the case when queueing features such as fair queueing or custom queueing (CQ) are used, then the data is likely to be stored in main memory, drastically degrading router performance.

By directing one TCP session at a time to slow down, RED solves the problems described, allowing for full utilization of the bandwidth rather than utilization manifesting as crests and troughs of traffic.

# About WRED

WRED combines the capabilities of the RED algorithm with the IP Precedence feature to provide for preferential traffic handling of higher priority packets. WRED can selectively discard lower priority traffic when the interface begins to get congested and provide differentiated performance characteristics for different classes of service.

You can configure WRED to ignore IP precedence when making drop decisions so that nonweighted RED behavior is achieved.

For interfaces configured to use the Resource Reservation Protocol (RSVP) feature, WRED chooses packets from other flows to drop rather than the RSVP flows. Also, IP Precedence governs which packets are dropped--traffic that is at a lower precedence has a higher drop rate and therefore is more likely to be throttled back.

WRED differs from other congestion avoidance techniques such as queueing strategies because it attempts to anticipate and avoid congestion rather than control congestion once it occurs.

## Why Use WRED

WRED makes early detection of congestion possible and provides for multiple classes of traffic. It also protects against global synchronization. For these reasons, WRED is useful on any output interface where you expect congestion to occur.

However, WRED is usually used in the core routers of a network, rather than at the edge of the network. Edge routers assign IP precedences to packets as they enter the network. WRED uses these precedences to determine how to treat different types of traffic.

WRED provides separate thresholds and weights for different IP precedences, allowing you to provide different qualities of service in regard to packet dropping for different traffic types. Standard traffic may be dropped more frequently than premium traffic during periods of congestion.

WRED is also RSVP-aware, and it can provide the controlled-load QoS service of integrated service.

## How It Works

By randomly dropping packets prior to periods of high congestion, WRED tells the packet source to decrease its transmission rate. If the packet source is using TCP, it will decrease its transmission rate until all the packets reach their destination, which indicates that the congestion is cleared.

WRED generally drops packets selectively based on IP precedence. Packets with a higher IP precedence are less likely to be dropped than packets with a lower precedence. Thus, the higher the priority of a packet, the higher the probability that the packet will be delivered.

WRED reduces the chances of tail drop by selectively dropping packets when the output interface begins to show signs of congestion. By dropping some packets early rather than waiting until the queue is full, WRED avoids dropping large numbers of packets at once and minimizes the chances of global synchronization. Thus, WRED allows the transmission line to be used fully at all times.

In addition, WRED statistically drops more packets from large users than small. Therefore, traffic sources that generate the most traffic are more likely to be slowed down than traffic sources that generate little traffic.

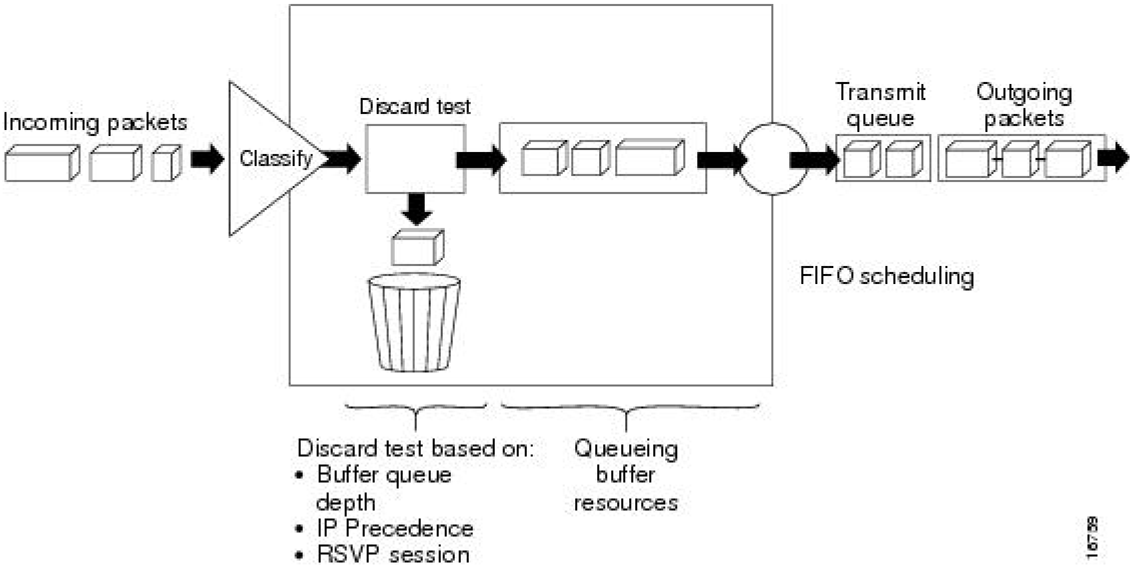
WRED avoids the globalization problems that occur when tail drop is used as the congestion avoidance mechanism. Global synchronization manifests when multiple TCP hosts reduce their transmission rates in response to packet dropping, then increase their transmission rates once again when the congestion is reduced.

WRED is only useful when the bulk of the traffic is TCP/IP traffic. With TCP, dropped packets indicate congestion, so the packet source will reduce its transmission rate. With other protocols, packet sources may not respond or may resend dropped packets at the same rate. Thus, dropping packets does not decrease congestion.

WRED treats non-IP traffic as precedence 0, the lowest precedence. Therefore, non-IP traffic, in general, is more likely to be dropped than IP traffic.

The figure below illustrates how WRED works.

***Figure 2***



## Average Queue Size

The router automatically determines parameters to use in the WRED calculations. The average queue size is based on the previous average and the current size of the queue. The formula is:

```
average = (old_average * (1-2 -n)) + (current_queue_size * 2
-n
)
```

where *n* is the exponential weight factor, a user-configurable value. The default value of the exponential weight factor is 9. It is recommended to use only the default value for the exponential weight factor. Change this value from the default value only if you have determined that your scenario would benefit from using a different value.

For high values of *n*, the previous average becomes more important. A large factor smooths out the peaks and lows in queue length. The average queue size is unlikely to change very quickly, avoiding drastic swings in size. The WRED process will be slow to start dropping packets, but it may continue dropping packets for a time after the actual queue size has fallen below the minimum threshold. The slow-moving average will accommodate temporary bursts in traffic.

**Note** If the value of *n* gets too high, WRED will not react to congestion. Packets will be sent or dropped as if WRED were not in effect.

For low values of *n*, the average queue size closely tracks the current queue size. The resulting average may fluctuate with changes in the traffic levels. In this case, the WRED process responds quickly to long queues. Once the queue falls below the minimum threshold, the process will stop dropping packets.

If the value of *n* gets too low, WRED will overreact to temporary traffic bursts and drop traffic unnecessarily.

## Restrictions

You cannot configure WRED on the same interface as Route Switch Processor (RSP)-based CQ, priority queueing (PQ), or weighted fair queueing (WFQ).

# Distributed Weighted Random Early Detection

Distributed WRED (DWRED) is an implementation of WRED for the Versatile Interface Processor (VIP). DWRED provides the complete set of functions for the VIP that WRED provides on standard Cisco IOS platforms.

The DWRED feature is only supported on Cisco 7000 series routers with an RSP-based RSP7000 interface processor and Cisco 7500 series routers with a VIP-based VIP2-40 or greater interface processor. A VIP2-50 interface processor is strongly recommended when the aggregate line rate of the port adapters on the VIP is greater than DS3. A VIP2-50 interface processor is required for OC-3 rates.

DWRED is configured the same way as WRED. If you enable WRED on a suitable VIP interface, such as a VIP2-40 or greater with at least 2 MB of SRAM, DWRED will be enabled instead.

In order to use DWRED, distributed Cisco Express Forwarding (dCEF) switching must be enabled on the interface.

You can configure both DWRED and distributed weighted fair queueing (DWFQ) on the same interface, but you cannot configure distributed WRED on an interface for which RSP-based CQ, PQ, or WFQ is configured.

## How It Works

When a packet arrives and DWRED is enabled, the following events occur:

- The average queue size is calculated. See the Average Queue Size, page 8 section for details.
- If the average is less than the minimum queue threshold, the arriving packet is queued.
- If the average is between the minimum queue threshold and the maximum queue threshold, the packet is either dropped or queued, depending on the packet drop probability. See the Packet-Drop Probability, page 8 section for details.
- If the average queue size is greater than the maximum queue threshold, the packet is automatically dropped.

## Average Queue Size

The average queue size is based on the previous average and the current size of the queue. The formula is:

```
average = (old_average * (1-1/2^n)) + (current_queue_size * 1/2^n)
```

where *n* is the exponential weight factor, a user-configurable value.

For high values of *n*, the previous average queue size becomes more important. A large factor smooths out the peaks and lows in queue length. The average queue size is unlikely to change very quickly, avoiding drastic swings in size. The WRED process will be slow to start dropping packets, but it may continue dropping packets for a time after the actual queue size has fallen below the minimum threshold. The slow-moving average will accommodate temporary bursts in traffic.

**Note** If the value of *n* gets too high, WRED will not react to congestion. Packets will be sent or dropped as if WRED were not in effect.

For low values of *n*, the average queue size closely tracks the current queue size. The resulting average may fluctuate with changes in the traffic levels. In this case, the WRED process responds quickly to long queues. Once the queue falls below the minimum threshold, the process stops dropping packets.

If the value of *n* gets too low, WRED will overreact to temporary traffic bursts and drop traffic unnecessarily.

## Packet-Drop Probability

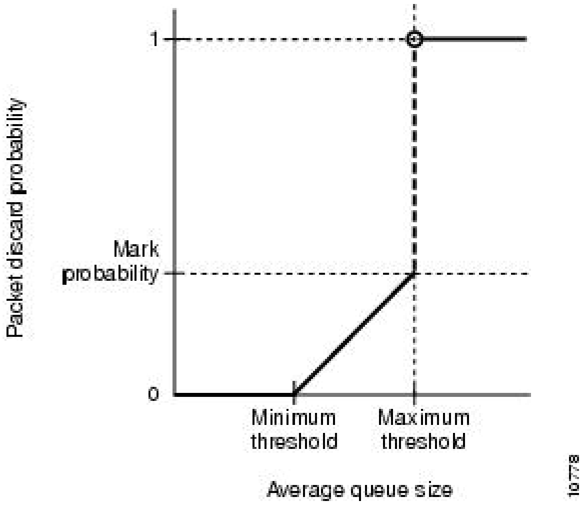
The probability that a packet will be dropped is based on the minimum threshold, maximum threshold, and mark probability denominator.

When the average queue size is above the minimum threshold, RED starts dropping packets. The rate of packet drop increases linearly as the average queue size increases, until the average queue size reaches the maximum threshold.

The mark probability denominator is the fraction of packets dropped when the average queue size is at the maximum threshold. For example, if the denominator is 512, one out of every 512 packets is dropped when the average queue is at the maximum threshold.

When the average queue size is above the maximum threshold, all packets are dropped. The figure below summarizes the packet drop probability.

*Figure 3*



The minimum threshold value should be set high enough to maximize the link utilization. If the minimum threshold is too low, packets may be dropped unnecessarily, and the transmission link will not be fully used.

The difference between the maximum threshold and the minimum threshold should be large enough to avoid global synchronization of TCP hosts (global synchronization of TCP hosts can occur as multiple TCP hosts reduce their transmission rates). If the difference between the maximum and minimum thresholds is too small, many packets may be dropped at once, resulting in global synchronization.

## Why Use DWRED

DWRED provides faster performance than does RSP-based WRED. You should run DWRED on the VIP if you want to achieve very high speed on the Cisco 7500 series platform--for example, you can achieve speed at the OC-3 rates by running WRED on a VIP2-50 interface processor.

Additionally, the same reasons you would use WRED on standard Cisco IOS platforms apply to using DWRED. For instance, when WRED or DWRED is not configured, tail drop is enacted during periods of congestion. Enabling DWRED obviates the global synchronization problems that result when tail drop is used to avoid congestion.

The DWRED feature provides the benefit of consistent traffic flows. When RED is not configured, output buffers fill during periods of congestion. When the buffers are full, tail drop occurs; all additional packets are dropped. Because the packets are dropped all at once, global synchronization of TCP hosts can occur as multiple TCP hosts reduce their transmission rates. The congestion clears, and the TCP hosts increase their transmission rates, resulting in waves of congestion followed by periods when the transmission link is not fully used.

RED reduces the chances of tail drop by selectively dropping packets when the output interface begins to show signs of congestion. By dropping some packets early rather than waiting until the buffer is full, RED avoids dropping large numbers of packets at once and minimizes the chances of global synchronization. Thus, RED allows the transmission line to be used fully at all times.

In addition, RED statistically drops more packets from large users than small. Therefore, traffic sources that generate the most traffic are more likely to be slowed down than traffic sources that generate little traffic.

DWRED provides separate thresholds and weights for different IP precedences, allowing you to provide different qualities of service for different traffic. Standard traffic may be dropped more frequently than premium traffic during periods of congestion.

## Restrictions

The following restrictions apply to the DWRED feature:

- Interface-based DWRED cannot be configured on a subinterface. (A subinterface is one of a number of virtual interfaces on a single physical interface.)
- DWRED is not supported on Fast EtherChannel and tunnel interfaces.
- RSVP is not supported on DWRED.
- DWRED is useful only when the bulk of the traffic is TCP/IP traffic. With TCP, dropped packets indicate congestion, so the packet source reduces its transmission rate. With other protocols, packet sources may not respond or may resend dropped packets at the same rate. Thus, dropping packets does not necessarily decrease congestion.
- DWRED treats non-IP traffic as precedence 0, the lowest precedence. Therefore, non-IP traffic is usually more likely to be dropped than IP traffic.

- DWRED cannot be configured on the same interface as RSP-based CQ, PQ, or WFQ. However, both DWRED and DWFQ can be configured on the same interface.

**Note** Do not use the **match protocol**command to create a traffic class with a non-IP protocol as a match criterion. The VIP does not support matching of non-IP protocols.

## Prerequisites

This section provides the prerequisites that must be met before you configure the DWRED feature.

- Weighted Fair Queueing, page 10
- WRED, page 10
- Access Control Lists, page 10
- Cisco Express Forwarding, page 10

### Weighted Fair Queueing

Attaching a service policy to an interface disables WFQ on that interface if WFQ is configured for the interface. For this reason, you should ensure that WFQ is not enabled on such an interface before configuring DWRED.

### WRED

Attaching a service policy configured to use WRED to an interface disables WRED on that interface. If any of the traffic classes that you configure in a policy map use WRED for packet drop instead of tail drop, you must ensure that WRED is not configured on the interface to which you intend to attach that service policy.

### Access Control Lists

You can specify a numbered access list as the match criterion for any traffic class that you create. For this reason, before configuring DWRED you should know how to configure access lists.

### Cisco Express Forwarding

In order to use DWRED, dCEF switching must be enabled on the interface.

# Flow-Based WRED

Flow-based WRED is a feature that forces WRED to afford greater fairness to all flows on an interface in regard to how packets are dropped.

- Why Use Flow-Based WRED, page 10
- How It Works, page 11

## Why Use Flow-Based WRED

Before you consider the advantages that use of flow-based WRED offers, it helps to think about how WRED (without flow-based WRED configured) affects different kinds of packet flows. Even before flow-based WRED classifies packet flows, flows can be thought of as belonging to one of the following categories:

- Nonadaptive flows, which are flows that do not respond to congestion.
- Robust flows, which on average have a uniform data rate and slow down in response to congestion.
- Fragile flows, which, though congestion-aware, have fewer packets buffered at a gateway than do robust flows.

WRED tends toward bias against fragile flows because all flows, even those with relatively fewer packets in the output queue, are susceptible to packet drop during periods of congestion. Though fragile flows have fewer buffered packets, they are dropped at the same rate as packets of other flows.

To provide fairness to all flows, flow-based WRED has the following features:

- It ensures that flows that respond to WRED packet drops (by backing off packet transmission) are protected from flows that do not respond to WRED packet drops.
- It prohibits a single flow from monopolizing the buffer resources at an interface.

## How It Works

Flow-based WRED relies on the following two main approaches to remedy the problem of unfair packet drop:

- It classifies incoming traffic into flows based on parameters such as destination and source addresses and ports.
- It maintains state about active flows, which are flows that have packets in the output queues.

Flow-based WRED uses this classification and state information to ensure that each flow does not consume more than its permitted share of the output buffer resources. Flow-based WRED determines which flows monopolize resources and it more heavily penalizes these flows.

To ensure fairness among flows, flow-based WRED maintains a count of the number of active flows that exist through an output interface. Given the number of active flows and the output queue size, flow-based WRED determines the number of buffers available per flow.

To allow for some burstiness, flow-based WRED scales the number of buffers available per flow by a configured factor and allows each active flow to have a certain number of packets in the output queue. This scaling factor is common to all flows. The outcome of the scaled number of buffers becomes the per-flow limit. When a flow exceeds the per-flow limit, the probability that a packet from that flow will be dropped increases.

# DiffServ Compliant WRED

DiffServ Compliant WRED extends the functionality of WRED to enable support for DiffServ and AF Per Hop Behavior PHB. This feature enables customers to implement AF PHB by coloring packets according to DSCP values and then assigning preferential drop probabilities to those packets.

**Note** This feature can be used with IP packets only. It is not intended for use with Multiprotocol Label Switching (MPLS)-encapsulated packets.

The Class-Based Quality of Service MIB supports this feature. This MIB is actually the following two MIBs:

- CISCO-CLASS-BASED-QOS-MIB

- CISCO-CLASS-BASED-QOS-CAPABILITY-MIB

The DiffServ Compliant WRED feature supports the following RFCs:

- RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*
- RFC 2475, *An Architecture for Differentiated Services Framework*
- RFC 2597, *Assured Forwarding PHB*
- RFC 2598, *An Expedited Forwarding PHB*

## How It Works

The DiffServ Compliant WRED feature enables WRED to use the DSCP value when it calculates the drop probability for a packet. The DSCP value is the first six bits of the IP type of service (ToS) byte.

This feature adds two new commands, **random-detect dscp** and **dscp**. It also adds two new arguments, *dscp-based* and *prec-based* **,** to two existing WRED-related commands--the **random-detect**(interface) command and the **random-detect-group** command.

The *dscp-based* argument enables WRED to use the DSCP value of a packet when it calculates the drop probability for the packet. The *prec-based* argument enables WRED to use the IP Precedence value of a packet when it calculates the drop probability for the packet.

These arguments are optional (you need not use any of them to use the commands) but they are also mutually exclusive. That is, if you use the *dscp-based* argument, you cannot use the *prec-based* argument with the same command.

After enabling WRED to use the DSCP value, you can then use the new **random-detect dscp** command to change the minimum and maximum packet thresholds for that DSCP value.

Three scenarios for using these arguments are provided.

## Usage Scenarios

The new *dscp-based* and *prec-based* arguments can be used whether you are using WRED at the interface level, at the per-virtual circuit (VC) level, or at the class level (as part of class-based WFQ (CBWFQ) with policy maps).

### WRED at the Interface Level

At the interface level, if you want to have WRED use the DSCP value when it calculates the drop probability, you can use the *dscp-based* argument with the **random-detect**(interface)command to specify the DSCP value. Then use the **random-detect dscp** command to specify the minimum and maximum thresholds for the DSCP value.

### WRED at the per-VC Level

At the per-VC level, if you want to have WRED use the DSCP value when it calculates the drop probability, you can use the *dscp-based* argument with the **random-detect-group** command. Then use the

**dscp** command to specify the minimum and maximum thresholds for the DSCP value or the mark-probability denominator.

This configuration can then be applied to each VC in the network.

### WRED at the Class Level

If you are using WRED at the class level (with CBWFQ), the *dscp-based* and *prec-based* arguments can be used within the policy map.

First, specify the policy map, the class, and the bandwidth. Then, if you want WRED to use the DSCP value when it calculates the drop probability, use the *dscp-based* argument with the **random-detect**(interface)command to specify the DSCP value. Then use the **random-detect dscp** command to modify the default minimum and maximum thresholds for the DSCP value.

This configuration can then be applied wherever policy maps are attached (for example, at the interface level, the per-VC level, or the shaper level).

## Usage Points to Note

Remember the following points when using the new commands and the new arguments included with this feature:

- If you use the *dscp-based* argument, WRED will use the DSCP value to calculate the drop probability.
- If you use the *prec-based* argument, WRED will use the IP Precedence value to calculate the drop probability.
- The *dscp-based* and *prec-based* arguments are mutually exclusive.
- If you do not specify either argument, WRED will use the IP Precedence value to calculate the drop probability (the default method).
- The **random-detect dscp** command must be used in conjunction with the **random-detect**(interface)command.
- The **random-detect dscp** command can only be used if you use the *dscp-based* argument with the **random-detect**(interface)command.
- The **dscp** command must be used in conjunction with the **random-detect-group** command.
- The **dscp** command can only be used if you use the *dscp-based* argument with the **random-detect-group** command.

# Configuring Weighted Random Early Detection

## Feature History

| Release | Modification |
|---|---|
| Cisco IOS | For information about feature support in Cisco IOS software, use Cisco Feature Navigator. |

This module describes the tasks for configuring Weighted Random Early Detection (WRED), distributed WRED (DWRED), flow-based WRED, and DiffServ Compliant WRED on a router.

**Note** WRED is useful with adaptive traffic such as TCP/IP. With TCP, dropped packets indicate congestion, so the packet source will reduce its transmission rate. With other protocols, packet sources may not respond or may resend dropped packets at the same rate. Thus, dropping packets does not decrease congestion. WRED treats non-IP traffic as precedence 0, the lowest precedence. Therefore, non-IP traffic is more likely to be dropped than IP traffic. You cannot configure WRED on the same interface as Route Switch Processor (RSP)-based custom queueing (CQ), priority queueing (PQ), or weighted fair queueing (WFQ). However, you can configure both DWRED and DWFQ on the same interface.

Random Early Detection (RED) is a congestion avoidance mechanism that takes advantage of the congestion control mechanism of TCP. By randomly dropping packets prior to periods of high congestion, RED tells the packet source to decrease its transmission rate. WRED drops packets selectively based on IP precedence. Edge routers assign IP precedences to packets as they enter the network. (WRED is useful on any output interface where you expect to have congestion. However, WRED is usually used in the core routers of a network, rather than at the edge.) WRED uses these precedences to determine how it treats different types of traffic.

When a packet arrives, the following events occur:

1. The average queue size is calculated.
2. If the average is less than the minimum queue threshold, the arriving packet is queued.
3. If the average is between the minimum queue threshold for that type of traffic and the maximum threshold for the interface, the packet is either dropped or queued, depending on the packet drop probability for that type of traffic.
4. If the average queue size is greater than the maximum threshold, the packet is dropped.

- WRED Configuration Examples, page 23
- DWRED Configuration Examples, page 25
- Flow-Based WRED Configuration Example, page 26
- DiffServ Compliant WRED Configuration Examples, page 27

# Finding Feature Information

Your software release may not support all the features documented in this module. For the latest feature information and caveats, see the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the Feature Information Table at the end of this document.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

# Weighted Random Early Detection Configuration Task List

- Enabling WRED, page 16
- Changing WRED Parameters, page 16
- Monitoring WRED, page 17

## Enabling WRED

**Note** To avoid counter issues do not configure WRED and queue-limit on the same interface at the same time.

| Command | Purpose |
|---|---|
| `Router(config-if)#` **random-detect** | Enables WRED. If you configure this command on a Versatile Interface Processor (VIP) interface, DWRED is enabled. |

## Changing WRED Parameters

**Note** The default WRED parameter values are based on the best available data. We recommend that you do not change the parameters from their default values unless you have determined that your applications will benefit from the changed values.

| Command | Purpose |
|---|---|
| `Router(config-if)#` **random-detect exponential-weighting-constant** *exponent* | Configures the weight factor used in calculating the average queue length. |

| Command | Purpose |
|---|---|
| `Router(config-if)#` **random-detect precedence** *precedence min-threshold max-threshold mark-prob-denominator* | Configures parameters for packets with a specific IP Precedence. The minimum threshold for IP Precedence 0 corresponds to half the maximum threshold for the interface. Repeat this command for each precedence. To configure RED, rather than WRED, use the same parameters for each precedence. |

## Monitoring WRED

| Command | Purpose |
|---|---|
| `Router#` **show queue** *interface-type interface-number* | Displays the header information of the packets inside a queue. This command does not support DWRED. |
| `Router#` **show queueing interface** *interface-number* [**vc** [[*vpi* /] *vci*]] | Displays the WRED statistics of a specific virtual circuit (VC) on an interface. |
| `Router#` **show queueing random-detect** | Displays the queueing configuration for WRED. |
| `Router#` **show interfaces** [*type slot* \| *port-adapter* \| *port*] | Displays WRED configuration on an interface. |

# DWRED Configuration Task List

- Configuring DWRED in a Traffic Policy, page 18
- Configuring DWRED to Use IP Precedence Values in a Traffic Policy, page 19
- Monitoring and Maintaining DWRED, page 19

# Configuring DWRED in a Traffic Policy

### SUMMARY STEPS

1. Router(config)# **policy-map** *policy-map*
2. Router(config-pmap)# **class** *class-name*
3. Steps 3, 4, and 5 are optional. If you do not want to configure the exponential weight factor, specify the amount of bandwidth, or specify the number of queues to be reserved, you can skip these three steps and continue with step 6.
4. Router(config-pmap-c)# **random-detect exponential-weighting-constant** *exponent*
5. Router(config-pmap-c)# **bandwidth** *bandwidth-kbps*
6. Router(config-pmap-c)# **fair-queue queue-limit** *queue-values*
7. Router(config-pmap-c)# **queue-limit** *number-of-packets*

### DETAILED STEPS

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | Router(config)# **policy-map** *policy-map* | Specifies the name of the traffic policy to be created or modified. |
| **Step 2** | Router(config-pmap)# **class** *class-name* | Specifies the name of a traffic class to be created and included in the traffic policy |
| **Step 3** | Steps 3, 4, and 5 are optional. If you do not want to configure the exponential weight factor, specify the amount of bandwidth, or specify the number of queues to be reserved, you can skip these three steps and continue with step 6. | |
| **Step 4** | Router(config-pmap-c)# **random-detect exponential-weighting-constant** *exponent* | Configures the exponential weight factor used in calculating the average queue length. |
| **Step 5** | Router(config-pmap-c)# **bandwidth** *bandwidth-kbps* | Specifies the amount of bandwidth, in kbps, to be assigned to the traffic class. |
| **Step 6** | Router(config-pmap-c)# **fair-queue queue-limit** *queue-values* | Specifies the number of queues to be reserved for the traffic class. |
| **Step 7** | Router(config-pmap-c)# **queue-limit** *number-of-packets* | Specifies the maximum number of packets that can be queued for the specified traffic class. |

## Configuring DWRED to Use IP Precedence Values in a Traffic Policy

### SUMMARY STEPS

1. Router(config)# **policy-map** *policy-map*
2. Router(config-pmap)# **class** *class-name*
3. Router(config-pmap-c)# **random-detect exponential-weighting-constant** *exponent*
4. Router(config-pmap-c)# **random-detect precedence** *precedence min-threshold max-threshold mark-prob-denominato r*

### DETAILED STEPS

| | Command or Action | Purpose |
|---|---|---|
| Step 1 | Router(config)# **policy-map** *policy-map* | Specifies the name of the traffic policy to be created or modified. |
| Step 2 | Router(config-pmap)# **class** *class-name* | Specifies the name of a traffic class to associate with the traffic policy |
| Step 3 | Router(config-pmap-c)# **random-detect exponential-weighting-constant** *exponent* | Configures the exponential weight factor used in calculating the average queue length. |
| Step 4 | Router(config-pmap-c)# **random-detect precedence** *precedence min-threshold max-threshold mark-prob-denominato r* | Configures the parameters for packets with a specific IP Precedence. The minimum threshold for IP Precedence 0 corresponds to half the maximum threshold for the interface. Repeat this command for each precedence. |

## Monitoring and Maintaining DWRED

| Command | Purpose |
|---|---|
| `Router#`<br>**show policy-map** | Displays all configured traffic policies. |
| `Router#` **show policy-map** *policy-map-name* | Displays the user-specified traffic policy. |
| `Router#`<br>**show policy-map interface** | Displays statistics and configurations of all input and output policies attached to an interface. |
| `Router#`<br>**show policy-map interface** *interface-spec* | Displays configuration and statistics of the input and output policies attached to a particular interface. |
| `Router#`<br>**show policy-map interface** *interface-spec input* | Displays configuration and statistics of the input policy attached to an interface. |

| Command | Purpose |
|---|---|
| `Router#` **show policy-map interface** *interface-spec* *output* | Displays configuration statistics of the output policy attached to an interface. |
| `Router#` **show policy-map interface** `[`*interface-spec* `[`*input* `\|` **output**`] [`*class class-name*`]]]]` | Displays the configuration and statistics for the class name configured in the policy. |

# Flow-Based WRED Configuration Task List

- Configuring Flow-Based WRED, page 20

## Configuring Flow-Based WRED

### SUMMARY STEPS

1. Router(config-if)# **random-detect flow**
2. Router(config-if)# **random-detect flow average-depth-factor** *scaling-factor*
3. Router(config-if)# **random-detect flow count** *number*

### DETAILED STEPS

| | Command or Action | Purpose |
|---|---|---|
| Step 1 | Router(config-if)# **random-detect flow** | Enables flow-based WRED. |
| Step 2 | Router(config-if)# **random-detect flow average-depth-factor** *scaling-factor* | Sets the flow threshold multiplier for flow-based WRED. |
| Step 3 | Router(config-if)# **random-detect flow count** *number* | Sets the maximum flow count for flow-based WRED. |

# DiffServ Compliant WRED Configuration Task List

- Configuring WRED to Use the Differentiated Services Code Point Value, page 20
- Verifying the DSCP Value Configuration, page 22

## Configuring WRED to Use the Differentiated Services Code Point Value

- WRED at the Interface Level, page 21
- WRED at the per-VC Level, page 21
- WRED at the Class Level, page 21

## WRED at the Interface Level

### SUMMARY STEPS

1. Router(config-if)# **random-detect** *dscp-based*
2. Router(config-if)# **random-detect dscp** *dscpvalue min-threshold max-threshold*[*mark-probability-denominator*]

### DETAILED STEPS

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | Router(config-if)# **random-detect** *dscp-based* | Indicates that WRED is to use the DSCP value when it calculates the drop probability for the packet. |
| **Step 2** | Router(config-if)# **random-detect dscp** *dscpvalue min-threshold max-threshold*[*mark-probability-denominator*] | Specifies the minimum and maximum thresholds, and, optionally, the mark-probability denominator for the specified DSCP value. |

## WRED at the per-VC Level

### SUMMARY STEPS

1. Router(config)# **random-detect-group** *group-name dscp-based*
2. Router(cfg-red-grp)# **dscp** *dscpvalue min-threshold max-threshold*[*mark-probability-denominator*]
3. Router(config-atm-vc)# **random-detect**[**attach** *group-name*]

### DETAILED STEPS

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | Router(config)# **random-detect-group** *group-name dscp-based* | Indicates that WRED is to use the DSCP value when it calculates the drop probability for the packet. |
| **Step 2** | Router(cfg-red-grp)# **dscp** *dscpvalue min-threshold max-threshold*[*mark-probability-denominator*] | Specifies the DSCP value, the minimum and maximum packet thresholds and, optionally, the mark-probability denominator for the DSCP value. |
| **Step 3** | Router(config-atm-vc)# **random-detect**[**attach** *group-name*] | Enables per-VC WRED or per-VC VIP-DWRED. |

## WRED at the Class Level

### SUMMARY STEPS

1. Router(config-if)# **class-map** *class-map-name*
2. Router(config-cmap)# **match** *match criterion*
3. Router(config-if)# **policy-map** *policy-map*
4. Router(config-pmap)# **class** *class-map-name*
5. Router(config-pmap-c)# **bandwidth** {*bandwidth-kbps* | **percent** *percent*}
6. Router(config-pmap-c)# **random-detect dscp-based**
7. Router(config-pmap-c)# **random-detect dscp** *dscpvalue min-threshold max-threshold*[*mark-probability-denominator*]
8. Router(config-if)# **service-policy output** *policy-map*

### DETAILED STEPS

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | Router(config-if)# **class-map** *class-map-name* | Creates a class map to be used for matching packets to a specified class. |
| **Step 2** | Router(config-cmap)# **match** *match criterion* | Configures the match criteria for a class map. |
| **Step 3** | Router(config-if)# **policy-map** *policy-map* | Creates or modifies a policy map that can be attached to one or more interfaces to specify a traffic policy. |
| **Step 4** | Router(config-pmap)# **class** *class-map-name* | Specifies the QoS actions for the default class. |
| **Step 5** | Router(config-pmap-c)# **bandwidth** {*bandwidth-kbps* \| **percent** *percent*} | Specifies or modifies the bandwidth allocated for a class belonging to a policy map. |
| **Step 6** | Router(config-pmap-c)# **random-detect dscp-based** | Indicates that WRED is to use the DSCP value when it calculates the drop probability for the packet. |
| **Step 7** | Router(config-pmap-c)# **random-detect dscp** *dscpvalue min-threshold max-threshold*[*mark-probability-denominator*] | Specifies the minimum and maximum packet thresholds and, optionally, the mark-probability denominator for the DSCP value. |
| **Step 8** | Router(config-if)# **service-policy output** *policy-map* | Attaches a policy map to an output interface or VC to be used as the traffic policy for that interface or VC. |

# Verifying the DSCP Value Configuration

| Command | Purpose |
|---|---|
| `Router#` **show queueing interface** | Displays the queueing statistics of an interface or VC. |
| `Router#` **show policy-map interface** | Displays the configuration of classes configured for traffic policies on the specified interface or permanent virtual circuit (PVC). |

# WRED Configuration Examples

## Example WRED Configuration

The following example enables WRED with default parameter values:

```
interface Serial5/0
 description to qos1-75a
 ip address 200.200.14.250 255.255.255.252
 random-detect
```

Use the **show interfaces** command output to verify the configuration. Notice that the "Queueing strategy" report lists "random early detection (RED)."

```
Router# show interfaces serial 5/0
Serial5/0 is up, line protocol is up
  Hardware is M4T
  Description: to qos1-75a
  Internet address is 200.200.14.250/30
  MTU 1500 bytes, BW 128 Kbit, DLY 20000 usec,
     reliability 255/255, txload 1/255, rxload 237/255
  Encapsulation HDLC, crc 16, loopback not set
  Keepalive not set
  Last input 00:00:15, output 00:00:00, output hang never
  Last clearing of "show interface" counters 00:05:08
  Input queue: 0/75/0 (size/max/drops); Total output drops: 1036
  Queueing strategy: random early detection(RED)
  5 minutes input rate 0 bits/sec, 2 packets/sec
  5 minutes output rate 119000 bits/sec, 126 packets/sec
     594 packets input, 37115 bytes, 0 no buffer
     Received 5 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     37525 packets output, 4428684 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 output buffer failures, 0 output buffers swapped out
     0 carrier transitions     DCD=up  DSR=up  DTR=up  RTS=up  CTS=up
```

Use the **show queue** command output to view the current contents of the interface queue. Notice that there is only a single queue into which packets from all IP precedences are placed after dropping has taken place. The output has been truncated to show only three of the five packets.

```
Router# show queue serial 5/0

Output queue for Serial5/0 is 5/0
Packet 1, linktype: ip, length: 118, flags: 0x288
  source: 190.1.3.4, destination: 190.1.2.2, id: 0x0001, ttl: 254,
  TOS: 128 prot: 17, source port 11111, destination port 22222
    data: 0x2B67 0x56CE 0x005E 0xE89A 0xCBA9 0x8765 0x4321
          0x0FED 0xCBA9 0x8765 0x4321 0x0FED 0xCBA9 0x8765
Packet 2, linktype: ip, length: 118, flags: 0x288
  source: 190.1.3.5, destination: 190.1.2.2, id: 0x0001, ttl: 254,
  TOS: 160 prot: 17, source port 11111, destination port 22222
    data: 0x2B67 0x56CE 0x005E 0xE89A 0xCBA9 0x8765 0x4321
          0x0FED 0xCBA9 0x8765 0x4321 0x0FED 0xCBA9 0x8765
Packet 3, linktype: ip, length: 118, flags: 0x280
  source: 190.1.3.6, destination: 190.1.2.2, id: 0x0001, ttl: 254,
  TOS: 192 prot: 17, source port 11111, destination port 22222
    data: 0x2B67 0x56CE 0x005E 0xE89A 0xCBA9 0x8765 0x4321
          0x0FED 0xCBA9 0x8765 0x4321 0x0FED 0xCBA9 0x8765
```

Use the **show queueing** command output to view the current settings for each of the precedences. Also notice that the default minimum thresholds are spaced evenly between half and the entire maximum threshold. Thresholds are specified in terms of packet count.

```
Router# show queueing
Current random-detect configuration:
     Serial5/0
       Queueing strategy:random early detection (WRED)
       Exp-weight-constant:9 (1/512)
       Mean queue depth:28

       Class   Random       Tail    Minimum    Maximum      Mark
                 drop       drop  threshold  threshold  probability
         0        330          0         20         40     1/10
         1        267          0         22         40     1/10
         2        217          0         24         40     1/10
         3        156          0         26         40     1/10
         4         61          0         28         40     1/10
         5          6          0         31         40     1/10
         6          0          0         33         40     1/10
         7          0          0         35         40     1/10
        rsvp        0          0         37         40     1/10
```

## Example Parameter-Setting DWRED

The following example specifies the same parameters for each IP precedence. Thus, all IP precedences receive the same treatment. Start by enabling DWRED.

```
interface FastEthernet1/0/0
 ip address 200.200.14.250 255.255.255.252
 random-detect
```

Next, enter the **show queueing random-detect** command to determine reasonable values to use for the precedence-specific parameters.

```
Router# show queueing random-detect
Current random-detect configuration:
  FastEthernet2/0/0
    Queueing strategy:fifo
    Packet drop strategy:VIP-based random early detection (DWRED)
    Exp-weight-constant:9 (1/512)
    Mean queue depth:0
    Queue size:0       Maximum available buffers:6308
    Output packets:5  WRED drops:0  No buffer:0
    Class   Random       Tail    Minimum    Maximum      Mark       Output
              drop       drop  threshold  threshold  probability  Packets
      0         0          0       109        218      1/10            5
      1         0          0       122        218      1/10            0
      2         0          0       135        218      1/10            0
      3         0          0       148        218      1/10            0
      4         0          0       161        218      1/10            0
      5         0          0       174        218      1/10            0
      6         0          0       187        218      1/10            0
      7         0          0       200        218      1/10            0
```

Complete the configuration by assigning the same parameter values to each precedence. Use the values obtained from the **show queueing random-detect** command output to choose reasonable parameter values.

```
interface FastEthernet1/0/0
 random-detect precedence 0 100 218 10
 random-detect precedence 1 100 218 10
 random-detect precedence 2 100 218 10
 random-detect precedence 3 100 218 10
 random-detect precedence 4 100 218 10
 random-detect precedence 5 100 218 10
 random-detect precedence 6 100 218 10
 random-detect precedence 7 100 218 10
```

## Example Parameter-Setting WRED

The following example enables WRED on the interface and specifies parameters for the different IP precedences:

```
interface Hssi0/0/0
 description 45Mbps to R1
 ip address 10.200.14.250 255.255.255.252
 random-detect
 random-detect precedence 0 32 256 100
 random-detect precedence 1 64 256 100
 random-detect precedence 2 96 256 100
 random-detect precedence 3 120 256 100
 random-detect precedence 4 140 256 100
 random-detect precedence 5 170 256 100
 random-detect precedence 6 290 256 100
 random-detect precedence 7 210 256 100
 random-detect precedence rsvp 230 256 100
```

# DWRED Configuration Examples

## Example DWRED on an Interface

The following example configures DWRED on an interface with a weight factor of 10:

```
Router(config)# interface hssi0/0/0
Router(config-if)# description 45mbps to R1
Router(config-if)# ip address 192.168.14.250 255.255.255.252
Router(config-if)# random-detect
Router(config-if)# random-detect exponential-weighting-constant 10
```

## Example Modular QoS CLI

The following example enables DWRED using the Legacy CLI (non-Modular QoS Command-Line Interface) feature on the interface and specifies parameters for the different IP precedences:

```
interface Hssi0/0/0
 description 45Mbps to R1
 ip address 200.200.14.250 255.255.255.252
 random-detect
 random-detect precedence 0 32 256 100
 random-detect precedence 1 64 256 100
 random-detect precedence 2 96 256 100
 random-detect precedence 3 120 256 100
 random-detect precedence 4 140 256 100
 random-detect precedence 5 170 256 100
 random-detect precedence 6 290 256 100
 random-detect precedence 7 210 256 100
 random-detect precedence rsvp 230 256 100
```

The following example uses the Modular QoS CLI to configure a traffic policy called policy10. For congestion avoidance, WRED packet drop is used, not tail drop. IP Precedence is reset for levels 0 through 5.

```
policy-map policy10
 class acl10
 bandwidth 2000
 random-detect exponential-weighting-constant 10
 random-detect precedence 0 32 256 100
 random-detect precedence 1 64 256 100
 random-detect precedence 2 96 256 100
 random-detect precedence 3 120 256 100
 random-detect precedence 4 140 256 100
 random-detect precedence 5 170 256 100
```

## Example Configuring DWRED in Traffic Policy

The following example configures policy for a traffic class named int10 to configure the exponential weight factor as 12. This is the weight factor used for the average queue size calculation for the queue for traffic class int10. WRED packet drop is used for congestion avoidance for traffic class int10, not tail drop.

```
policy-map policy12
 class int10
 bandwidth 2000
 random-detect exponential-weighting-constant 12
```

# Flow-Based WRED Configuration Example

The following example enables WRED on the serial interface 1 and configures flow-based WRED. The **random-detect** interface configuration command is used to enable WRED. Once WRED is enabled, the **random-detect flow** command is used to enable flow-based WRED.

After flow-based WRED is enabled, the **random-detect flow average-depth-factor** command is used to set the scaling factor to 8 and the **random-detect flow count** command is used to set the flow count to 16. The scaling factor is used to scale the number of buffers available per flow and to determine the number of packets allowed in the output queue for each active flow.

```
configure terminal
interface Serial1
 random-detect
 random-detect flow
 random-detect flow average-depth-factor 8
 random-detect flow count 16
end
```

The following part of the example shows a sample configuration file after the previous flow-based WRED commands are issued:

```
Router# more system:running-config
Building configuration...
Current configuration:
!
version 12.0
service timestamps debug datetime msec localtime
service timestamps log uptime
no service password-encryption
service tcp-small-servers
!
no logging console
enable password lab
!
```

```
clock timezone PST -8
clock summer-time PDT recurring
ip subnet-zero
no ip domain-lookup
!
interface Ethernet0
 no ip address
 no ip directed-broadcast
 no ip mroute-cache
 shutdown
!
interface Serial0
 no ip address
 no ip directed-broadcast
 no ip mroute-cache
 no keepalive
 shutdown
!
interface Serial1
 ip address 190.1.2.1 255.255.255.0
 no ip directed-broadcast
 load-interval 30
 no keepalive
 random-detect
 random-detect flow
 random-detect flow count 16
 random-detect flow average-depth-factor 8
!
router igrp 8
 network 190.1.0.0
!
ip classless
no ip http server
!
line con 0
 transport input none
line 1 16
 transport input all
line aux 0
 transport input all
line vty 0 4
 password lab
 login
!
end
```

# DiffServ Compliant WRED Configuration Examples

- Example WRED Configured to Use the DSCP Value, page 27
- Example DSCP Value Configuration Verification, page 28

## Example WRED Configured to Use the DSCP Value

The following example configures WRED to use the DSCP value 8. The minimum threshold for the DSCP value 8 is 24 and the maximum threshold is 40. This configuration was performed at the interface level.

```
Router(config-if)# interface seo/0
Router(config-if)# random-detect dscp-based
Router(config-if)# random-detect dscp 8 24 40
```

The following example enables WRED to use the DSCP value 9. The minimum threshold for the DSCP value 9 is 20 and the maximum threshold is 50. This configuration can be attached to other VCs, as required.

```
Router(config)# random-detect-group sanjose dscp-based
Router(cfg-red-grp)# dscp 9 20 50
Router(config-subif-vc)# random-detect attach sanjose
```

The following example enables WRED to use the DSCP value 8 for the class c1. The minimum threshold for the DSCP value 8 is 24 and the maximum threshold is 40. The last line attaches the traffic policy to the output interface or VC p1.

```
Router(config-if)# class-map c1
Router(config-cmap)# match access-group 101
Router(config-if)# policy-map p1
Router(config-pmap)# class c1
Router(config-pmap-c)# bandwidth 48
Router(config-pmap-c)# random-detect dscp-based
Router(config-pmap-c)# random-detect dscp 8 24 40
Router(config-if)# service-policy output p1
```

# Example DSCP Value Configuration Verification

When WRED has been configured to use the DSCP value when it calculates the drop probability of a packet, all entries of the DSCP table are initialized with the appropriate default values. The example in the following section are samples of the **show policy interface** command for WRED at the class level.

This example displays packet statistics along with the entries of the DSCP table, confirming that WRED has been enabled to use the DSCP value when it calculates the drop probability for a packet.

```
Router# show policy interface Serial6/3

 Serial6/3
  Service-policy output: test
    Class-map: c1 (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: protocol ip
        0 packets, 0 bytes
        5 minute rate 0 bps
      Weighted Fair Queueing
        Output Queue: Conversation 265
        Bandwidth 20 (%)
        Bandwidth 308 (kbps)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0
         exponential weight: 9
         mean queue depth: 0
dscp     Transmitted       Random drop      Tail drop    Minimum Maximum  Mark
          pkts/bytes        pkts/bytes       pkts/bytes    thresh  thresh  prob
af11          0/0               0/0              0/0           32      40  1/10
af12          0/0               0/0              0/0           28      40  1/10
af13          0/0               0/0              0/0           24      40  1/10
af21          0/0               0/0              0/0           32      40  1/10
af22          0/0               0/0              0/0           28      40  1/10
af23          0/0               0/0              0/0           24      40  1/10
af31          0/0               0/0              0/0           32      40  1/10
af32          0/0               0/0              0/0           28      40  1/10
af33          0/0               0/0              0/0           24      40  1/10
af41          0/0               0/0              0/0           32      40  1/10
af42          0/0               0/0              0/0           28      40  1/10
af43          0/0               0/0              0/0           24      40  1/10
cs1           0/0               0/0              0/0           22      40  1/10
cs2           0/0               0/0              0/0           24      40  1/10
cs3           0/0               0/0              0/0           26      40  1/10
cs4           0/0               0/0              0/0           28      40  1/10
cs5           0/0               0/0              0/0           30      40  1/10
cs6           0/0               0/0              0/0           32      40  1/10
cs7           0/0               0/0              0/0           34      40  1/10
ef            0/0               0/0              0/0           36      40  1/10
rsvp          0/0               0/0              0/0           36      40  1/10
default       0/0               0/0              0/0           20      40  1/10
```

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.