2

Introduction to QOS Mechanics and Architectures

2.1 What is Quality of Service?

In networking, the term quality of service (QOS) can mean many different things to different people, hence it is key that we start this chapter by defining what "QOS" means in the context of this book.

Firstly, how do we define a "service" in the context of IP networking? We consider that a service is a description of the overall treatment of a customer's traffic across a particular domain. A service is only practically useful if it meets the requirements of the end-user applications it is intended to support. Hence, the aim of the service is to maximize end-user satisfaction with the applications that the service is supporting. Maximizing user satisfaction requires that the end-user applications work effectively.

How then do we define "quality" in the context of a particular IP service? We can define service quality in terms of the underlying requirements for an application which can be defined in terms of the SLA metrics for IP service performance defined in Chapter 1: delay, jitter, packet loss, throughput, service availability, and per flow sequence preservation.

QOS, however, implies more than just ensuring that a network service is able to support the SLA requirements of the applications it is aiming to support. The problem of ensuring that a network can meet these requirements is fundamentally a problem of managing the available network capacity relative to the service load, i.e. a problem of managing congestion. If it is possible to ensure that there is always significantly more capacity available than there is traffic load then delay, jitter and loss will be minimized, throughput will be maximized, and the service requirements will be easy to meet. In practice, however, ensuring that the network is always overprovisioned relative to the actual traffic load is not always cost-effective. Hence, in engineering the QOS of a network service there is implicitly another important constraint, which is to minimize cost. If there is more traffic load than there is capacity to support it, i.e. if congestion occurs, then some traffic will either need to be delayed until there is capacity available or it will need to be dropped. Minimizing cost may demand that multiple services are supported or multiplexed on the same network, by classifying traffic into discrete classes, such that the problem of engineering traffic load relative to capacity can be performed on a per-class basis allowing per-class service differentiation.

In summary, at a high level we can describe QOS in terms of the goals that it is trying to achieve, which effectively define an optimization problem[1] of trying to maximize end-user satisfaction (utility or efficacy) while minimizing cost. Maximizing user satisfaction requires that the end-user applications work effectively, for example, that a voice over IP call quality is acceptable, which requires that the application's SLA requirements are met. Minimizing cost requires that we do not overengineer the network in order to deliver that call quality, which may require the need to differentiate the service levels offered to different applications.

2.1.1 Quality of Service vs Class of Service or Type of Service?

The terms "class of service" (COS) and "type of service" (TOS) are sometimes used interchangeably with quality of service; for the purposes of this book, we explicitly define them here to avoid confusion:

- *Class of service.* As well as being used interchangeably with quality of service, class of service is also sometimes used to refer to the layer 2 QOS capabilities provided by Ethernet or ATM. We prefer to use neither of those definitions but rather use "class of service" or

COS purely in the context of traffic classification. In Section 2.2.1 we define the concept of a traffic class as a set of traffic streams that will have common actions applied to them. Hence, to avoid confusion, we use the term "classes of service" to refer to the classification of an aggregate traffic stream into a number of constituent classes, where different actions will be applied to each individual "class of service."

- *Type of service*. We use the term "type of service" to refer specifically to the use of the Type of Service Octet in the IPv4 packet header as described in Section 2.3.2.

2.1.2 Best-effort Service

Networks engineered to deliver a particular quality of service are often contrasted to "best-effort" networks. "Best-effort" describes a network service which attempts to deliver traffic to its destination, but which does not provide any guarantees of delivery, and hence is without any commitments for delay, jitter, loss, and throughput.

The term best-effort, however, is often misused. Where a network supports multiple service classes simultaneously, best-effort is often used to refer to the service which offers the lowest SLA commitments. By definition, best-effort infers no SLA commitments and hence a service which provides any SLA commitments cannot be defined as best-effort, however lowly those commitments might be. Even if a network supports only a single service class, i.e. where packet forwarding is egalitarian and all packets receive the same quality of service, if that service provides defined SLA commitments, we contend that it cannot be considered a best-effort network service.

Confusion is also sometimes caused because IP can be referred to as a "best-effort" network layer protocol, in that it does not implicitly provide any capabilities to detect or retransmit lost packets. Despite this, however, with appropriate network engineering, it is possible to support IP services which have defined SLA commitments and hence an IP service does not imply a best-effort service. Conversely, TCP is sometimes considered to be a guaranteed transport layer protocol in that it provides the capability to detect and retransmit lost packets. This capability, however, may be of no practical use if the underlying IP service cannot deliver the TCP segments with the SLAs required by applications; the effective SLA of the TCP service is implicitly constrained by the SLA commitments provided by the underlying IP service.

In order to avoid any potential for confusion, we intentionally try not to use the term "best-effort."

2.1.3 The Timeframes that Matter for QOS

We find it useful to consider three timeframes relevant to engineering the quality of service of a network; different QOS techniques are applied in each timeframe:

- *O(milliseconds)*. The first timeframe we consider is in the order of milliseconds. Within this timeframe bursts of individual traffic streams or the aggregation of bursts for different streams at network aggregation points can cause congestion, where the traffic load exceeds the available capacity. QOS mechanisms relevant to this timescale include applying per-hop queuing, scheduling and dropping techniques to provide differentiation and isolation between different types of traffic, to prioritize some types of traffic over others thereby managing delay, and to ensure fair bandwidth allocations. These mechanisms are considered in Section 2.2.

- *O(100 milliseconds)*. The next timeframe we consider is in the order of 100s of milliseconds. This is the timeframe which defines network round trip times (RTT; see Chapter 1, Section 1.2.1). This is the timeframe which is important to applications that used closed-loop feedback between sender and receiver to apply flow control mechanisms, such as TCP-based applications. QOS mechanisms relevant to this timeframe therefore include active queue management (AQM) congestion control techniques such as random early detection (RED) (see Section 2.2.4.2.3).

- *O(10 seconds) or more*. Timeframes of seconds and minutes are relevant to the management of the long-term average network traffic rates and capacity, which is achieved through capacity planning and traffic engineering, which are discussed in Chapter 6.

2.1.4 Why IP QOS?

Layer 2 technologies such as ATM and Ethernet have their own defined QOS capabilities, hence it is a valid question to ask: "why use IP QOS rather than layer 2 QOS mechanisms?" The main reasons for using IP QOS stem from the fact that IP is the end-to-end network layer technology used by the vast majority of applications today. Added to this QOS is an end-to-end discipline where the service that a particular class of traffic receives is limited by the element on the end-to-end path which provides the worst service. Hence, in order to provide a low-delay, low-jitter and low-loss service (thus maximizing user satisfaction) the network must be engineered to remove all points of congestion on the end-to-end path for that service; in order to assure different SLAs for different classes of traffic (hence minimizing cost), we must apply differentiation at all points of congestion. Different underlying layer 2 technologies may be used for different legs of an end-to-end layer 3 path. Therefore, as IP is the lowest common end-to-end layer, it makes fundamental sense to use IP QOS techniques where possible, and to map them to underlying QOS capabilities in lower layer technologies, where required (see Section 2.5), rather than to attempt to map layer 2 QOS capabilities for one leg to the layer 2 QOS capabilities of the next leg. The SLAs provided at the IP layer, however, are implicitly limited by the SLAs of the underlying Layer 2 technology (see Section 2.5).

2.1.5 The QOS Toolset

In practical terms, QOS involves using a range of functions and features (e.g. classification, scheduling, policing, shaping), within the context of overriding architecture (e.g. Integrated Service, Differentiated

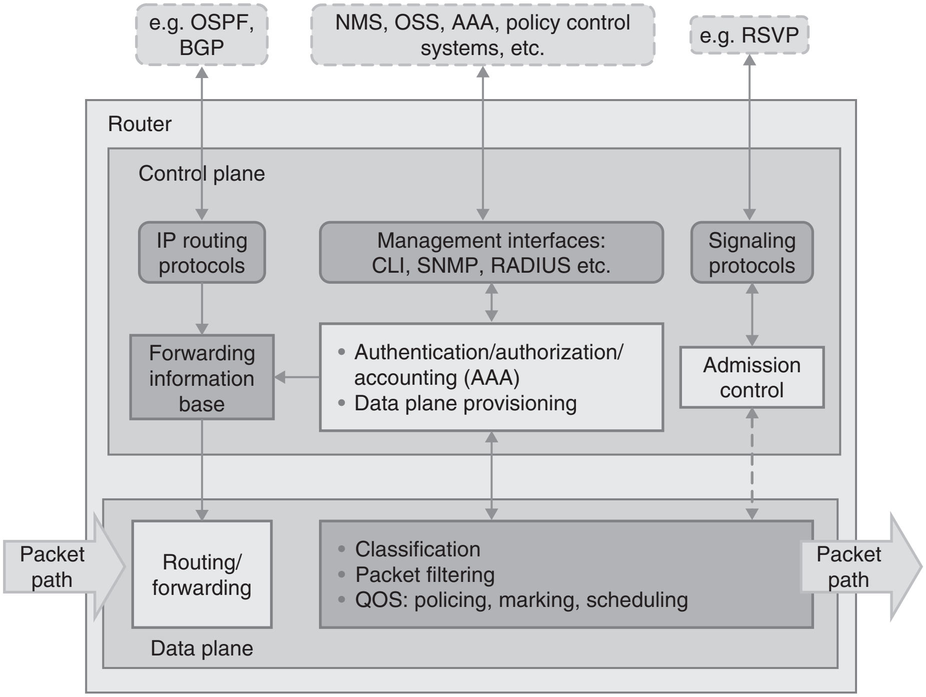

e.g. OSPF, BGP
NMS, OSS, AAA, policy control systems, etc.
e.g. RSVP
Router
Control plane
IP routing protocols
Management interfaces: CLI, SNMP, RADIUS etc.
Signaling protocols
Forwarding information base
• Authentication/authorization/accounting (AAA)
• Data plane provisioning
Admission control
Packet path
Routing/forwarding
• Classification
• Packet filtering
• QOS: policing, marking, scheduling
Packet path
Data plane

Figure 2.1 Control plane and data plane QOS functions

Services) in order to ensure that a network service delivers the SLA characteristics that the applications targeted by that service need to work effectively. The mechanisms used for engineering the QOS in a network can be broken down into data plane and control plane mechanisms applied on network devices such as routers, as shown in Figure 2.1 and which are introduced here and described in detail in the proceeding sections.

- ***Data plane***. Data plane QOS mechanisms are applied at network nodes and can directly impact the forwarding behavior of packets. They are processing intensive functions; in high-performance routers, they are typically implemented in hardware, along with other data plane functions such as packet forwarding lookups and packet filtering. Such data plane QOS mechanisms can be categorized in terms of the primitive behavioral characteristics that they impart to traffic streams to which they are applied:

  - *Classification*. Classification (see Section 2.2.1) is the process of categorizing an aggregate traffic stream into a number of constituent classes, such that any of the following actions can be applied to each individual "class of service."
  - *Marking*. Traffic marking (see Section 2.2.2) is the process of explicitly setting the value of the fields assigned for QOS classification in the IP or MPLS packet headers so that the traffic can subsequently be easily identified.
  - *Maximum rate enforcement*. Policing (see Section 2.2.3) and shaping (see Section 2.2.4.3) can be used to enforce a maximum rate for a traffic class.
  - *Prioritization*. Techniques such as priority scheduling (see Section 2.2.4.1.1) are used to prioritize some types of traffic over others thereby managing delay and jitter.
  - *Minimum rate assurance*. Scheduling techniques such as Weighted Fair Queuing (WFQ) and Deficit Round Robin (DRR) can be used to provide different traffic classes with different minimum bandwidth assurances (see Section 2.2.4.1.2).

- ***Control plane***. Control or signaling plane QOS mechanisms typically deal with admission control and resource reservation, and may in some cases be used to set up the data plane QOS functions. Control plane QOS functions are typically implemented as software processes, along with other control plane functions such as routing protocols. In practice, there is only one protocol widely used for control plane QOS signaling; that signaling protocol is RSVP. RSVP is used in several different contexts:
  - *Integrated services architecture*. RSVP is used in the context of the Integrated Services architecture to perform per flow resource reservation and admission control (see Section 2.3.3).
  - *MPLS traffic engineering*. RSVP is used in the context of MPLS traffic engineering, for admission control and to set up MPLS traffic engineering tunnels (see Chapter 6).

These QOS functions and mechanisms are, however, not generally used in isolation, but rather they are used together in concert, within the framework of an overriding QOS architecture, where mechanisms are combined for end-to-end effect. There are two defined IP QOS architectures: the Integrated Services architecture (see Section 2.3.3) and the Differentiated Services architecture (see Section 2.3.4).

2.2 Data Plane QOS Mechanisms

2.2.1 Classification

Classification in the process of identifying flows of packets and grouping individual traffic flows into aggregated streams such that actions can be applied to those streams; the actions that may be applied after classification could be other than QOS actions, for example packet filtering. To complete this definition we define the terms flow, stream and class:

- ***Flows***. An IPv4 flow is typically defined by the 5-tuple of source IP address, destination IP address, source TCP/UDP port, destination TCP/UDP port and the transport protocol (e.g. TCP or UDP). Fragmentation, encryption or tunneling can make some flows difficult to classify, as some of these fields may be unavailable. For example, assume that a flow is classified using this 5-tuple, but that the sizes of packets in the flow exceed the size of packets that can be supported (defined by the maximum transmission unit or MTU) on some of the underlying links which the flow transits. Those packets, which exceed the MTU of transit links, will need to be fragmented using IP level fragmentation [RFC 791]. When a packet is fragmented, the TCP/UDP port information is included in the first packet only; hence, it may not be possible to identify uniquely non-first fragments of a packet as belonging to the same flow. IPv6 introduced a Flow Label field which can be used to overcome this problem.

- ○ More complicated criteria than this 5-tuple may also be used to define a flow, such as using deep packet inspection/stateful inspection techniques (see Section 2.2.1.3).
- ○ ***Stream***. A traffic stream is an aggregation of flows based upon some common classification criteria. For example, all VoIP traffic from a particular VoIP gateway could be identified just by matching the source IP address of that gateway. A traffic stream may consist of a single flow, or a number of flows.
- ○ ***Traffic classes***. A traffic class is an aggregation of individual traffic flows or streams, for the purpose of applying a common action to the constituent flows or streams. For example, a class may represent all VoIP traffic from a particular site, which may consist of streams of traffic from a number of VoIP gateways. A traffic class may consist of a single stream, or a number of streams.

In the following sections we define the concepts “implicit,” “complex,” “deep packet/stateful” and “simple” when referring to classification techniques.

2.2.1.1 Implicit Classification

From the perspective of IP QOS, we define implicit classification as a broad brush classification approach, which requires no knowledge of the packet header or contents, and may for example use Layer 1/ Layer 2 context such as the received interface, or received virtual circuit (VC) in order to classify traffic.

2.2.1.2 Complex Classification

Complex classification (also known as multi-field classification) allows for more granularity than “implicit” classification and involves identifying traffic based upon values of specific fields or combinations of fields in the IP packet header, which were not explicitly intended for QOS classification. This includes those fields previously defined in the context of the 5-tuple for IP flow classification, i.e. classifying a flow based upon the 5-tuple is an inception of complex classification. Complex classification may also classify traffic using Layer 2 criteria, such as source or destination MAC addresses, e.g. identifying traffic from a specific host by that host's MAC address.

2.2.1.3 Deep Packet Inspection/Stateful Inspection

Some systems have the capability to classify traffic based upon more than the information contained in a single IP packet header. They may be able to look deeper into the packet, at the underlying data; this is referred to as deep packet inspection (DPI). They may also be able to classify flows by keeping state of the information contained in subsequent packets, rather than looking at each packet individually; this is referred to as stateful inspection (SI). When DPI is combined with SI, the combination can be useful to classify applications that cannot be identified using other means, such as some peer-to-peer applications for example.

Due to the traffic demands that can be placed upon the network by peer-to-peer applications, some peer-to-peer application developers intentionally try to make their applications hard to classify, or make them look like other applications, specifically so that they are difficult to classify and hence difficult to control. This situation is comparable to an arms race, where the application developers are constantly trying to stay one step ahead of the DPI/SI capabilities to classify them.

2.2.1.4 Simple Classification

Classification based upon fields in the packet headers which have been explicitly designed for QOS classification. We refer to it as "simple" classification, as it requires no understanding of other fields in the IP packet header or data, and need have no visibility of constituent flows within the traffic stream which may represent an aggregate of flows. The use of simple classification techniques makes QOS designs easier and requires a less complex underlying classification implementation on network equipment. The following schemes are defined for explicit QOS classification in IP and MPLS:

- *Type of service octet*. The original IPv4 specification [RFC 791] defined an 8-bit field to be used for IP QOS classification; this was called the Type of Service octet and is highlighted in Figure 2.2.

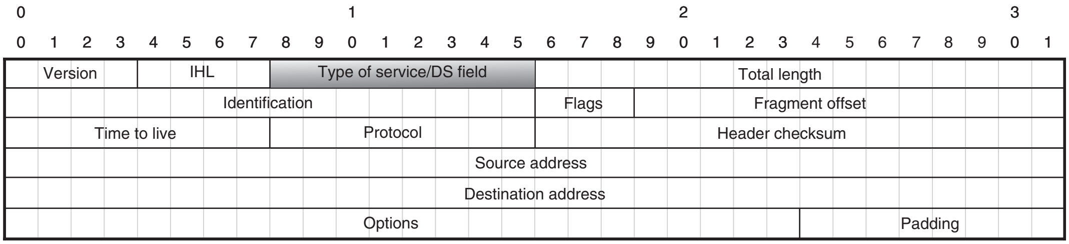


Figure 2.2 IPv4 packet header

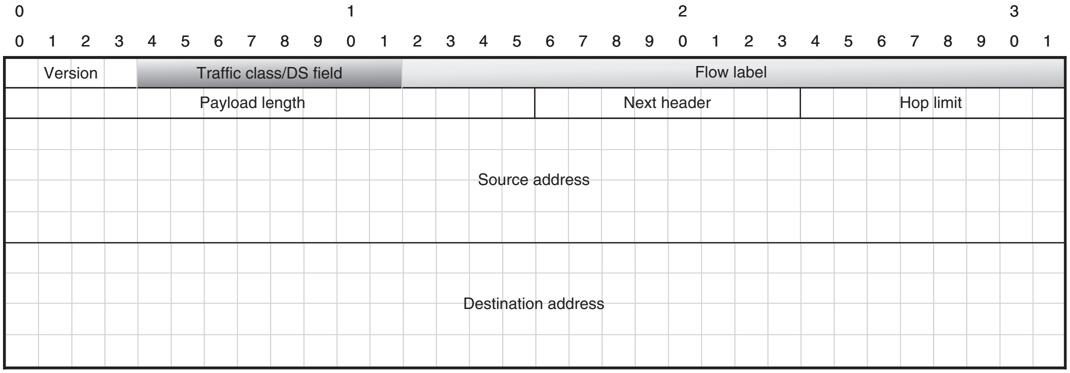


Figure 2.3 IPv6 packet header

The type of service octet has since been obsoleted by the Differentiated Services field. See Section 2.3.2 for more details on type of service.

- *IPv6 traffic class octet*. [RFC2460] originally defined the 8-bit traffic class field within the IPv6 header for QOS marking as shown in Figure 2.3. The IPv6 traffic class octet has been obsoleted by the Differentiated Services field.

  IPv6 packet headers also include a 20-bit flow label field. The flow label helps to unambiguously classify a flow, where some information used to identify the flow may be missing due to fragmentation, encryption or tunneling, for example. The 3-tuple of the flow label and the source and destination IPv6 address fields are used to classify an IPv6 flow [RFC 3697] uniquely.

- *Differentiated Service field*. [RFC2474] obsoleted both the IPv4 type of service octet and the IPv6 traffic class octet, by redefining them as the Differentiated Services (DS) field. Of the 8 bits of the DS field, 6 were defined as the Differentiated Services code point (DSCP), and the 2 low-order bits were initially undefined. See Section 2.3.4.1 for more details on the DS field.

  [RFC3168] later defined the use of the low-order 2 bits for Explicit Congestion Notification (ECN). ECN is described in Section 2.3.4.4.

- *MPLS EXP field*. [RFC3032] defined a 3-bit field in each MPLS header for "experimental use." [RFC3270] went on to redefine the use of this field for QOS marking. See Section 2.3.6.2 for more details on the MPLS EXP field.

- *Layer 2 marking*. In an IP QOS network enabled network, there may also be a need to make use of QOS marking at layer 2. The best example of this is Ethernet, where IEEE 802.1D [802.1D] defines the use of a 3-bit field for QOS marking. See Section 2.5 for more details of QOS at Layer 2 including IEEE 802.1D.

A particular classification policy could contain logical combinations of both complex and simple classification criteria, e.g. matching on all traffic with a particular DS field marking AND which is from a specific source IP address.

Why do we differentiate between complex and simple classification? Primarily because which classification technique you choose to use and how they are applied can have an impact on the complexity and scalability of the resulting QOS design. For example, using a particular server IP address may seem like a sensible way to identify a traffic stream. If this were used as the only classification criteria for that traffic stream, then classifiers would need to be configured throughout the network, which would classify traffic in that stream by matching the source or destination IP address to the address of the server. This would require that every router in the network were configured with this classifier. This may seem viable, but what if there were one hundred servers rather than just one? Further, what if the IP address of one or more of the servers changes? It quickly becomes apparent that manual configuration of complex classifiers throughout a network to classify streams becomes unmanageable. It is also noted that complex classification approaches may have an impact on router platform performance. On software-based platforms, complex classification may be more processor-intensive and have a consequent impact on the packets per second forward performance of the routers; on hardware-based platforms, there may be a limit to the number of complex classifiers that can be supported by the hardware.

Consequently, and as we will discuss later in this book, the QOS architectures used today do not use manually configured complex classification throughout networks. Instead, the Integrated Services architecture (see Section 2.3.4) uses a signaling protocol to set up per flow classifiers, and the Differentiated Services architecture generally uses simple classification, based upon matching aggregates of traffic identified by the marking of the DSCP field, where complex classification is required it is limited to the ingress edges of the network.

2.2.2 Marking

IP packet marking, which is also known as coloring, is the process of setting the value of the fields assigned for QOS classification in the IP or MPLS packet headers so that the traffic can easily be identified later, i.e. using simple classification techniques.

Marking may use any of the schemes described in Section 2.2.1; however, with the obsoletion of the IP precedence and TOS fields, the DSCP and the MPLS EXP field are becoming the main fields used for IP/MPLS packet marking and classification.

Traffic is generally marked at the source end-system or as close to the traffic source as possible in order to simplify the network design:

- *Source marking*. Packet marking may be applied at the source end-system itself; if the end-system is considered to be trusted then this marking may be relied upon throughout the rest of the network, subsequently requiring only simple classification to identify the traffic steam.

- *Ingress marking*. If the end-system is not capable of marking the packets it originates, or cannot be trusted to do so correctly, then on ingress to the network a trusted device close to the source may use implicit or complex classification to identify a traffic stream from the end-system and mark the traffic accordingly, such that it can be subsequently identified using simple classification techniques. If the source is not trusted to mark packets, then any markings that have previously been applied may be overwritten; this is sometimes termed "remarking."

  Such traffic marking can be applied unconditionally, e.g. mark the DSCP to 34 for all traffic received on a particular interface. Traffic marking can also be applied as a conditional result of a traffic policer (see Section 2.2.3 for more information on policers), e.g. for traffic received on an interface which conforms to a policer definition mark the DSCP to 34, for traffic which does not conform to (i.e. is in excess of) the policer definition, mark the DSCP to 36. This conditional marking behavior allows the enforcement of a traffic contract with an in- and out-of-contract concept, as described in Section 2.2.3.1.

  Even when a source end-system is trusted to mark the traffic it originates, a policer may still be applied to enforce that the traffic stream from the source conforms to the agreed traffic contract. For example, a source may originate all traffic with DSCP 34; simple classification matching DSCP 34 may then be used on ingress to classify the traffic stream, and a policer applied which leaves the DSCP marking at 34 if the traffic conforms to a policer definition (i.e. is "in-contract") and remarks (a.k.a. "demotes") traffic to DSCP 36 if it exceeds the policer definition (i.e. is "out-of-contract").

2.2.3 Policing and Metering

Policing is a mechanism which can be used to ensure that a traffic stream does not exceed a defined maximum rate. A policer is normally visualized as a token bucket mechanism – not to be confused with

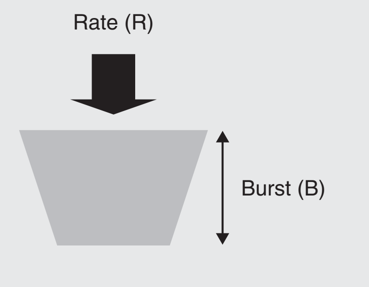


Figure 2.4 Token bucket

a leaky bucket algorithm, which has different properties, and is more commonly used for traffic shaping, as discussed in Section 2.2.4.3.

A simple one-rate token bucket policer has a defined maximum bucket depth (normally in bytes), known as the burst B, and a defined rate R (normally in bps) at which the bucket is filled with byte-sized tokens; see Figure 2.4. Depending upon the particular policer implementation, tokens are added to the bucket at rate R either every time a packet is processed by the policer, or at regular intervals, up to a maximum number of tokens that can be in the bucket, defined by B. The minimum number of tokens in the bucket is zero.

When a token bucket policer mechanism is applied to a traffic stream, and a packet from that stream arrives, the packet size b is compared against the number of tokens currently in the bucket. If there are at least as many byte tokens in the bucket as there are bytes in the packet, then we use the terminology that the packet has "conformed" to the token bucket definition; if there are less tokens in the bucket than bytes in the packet, then the packet has "exceeded" the token bucket definition. If the packet conforms then a number of tokens are decremented from the bucket equal to the packet size b. This simple policer behavior is described in the flowchart in Figure 2.5. Different actions can then be applied depending upon whether the packet conforms or exceeds the token bucket definition. The simplest actions are to transmit the packet if it conforms and to drop the packet if it exceeds; applied in this way the token bucket policer would enforce a maximum rate of R and burst of B on the traffic stream.

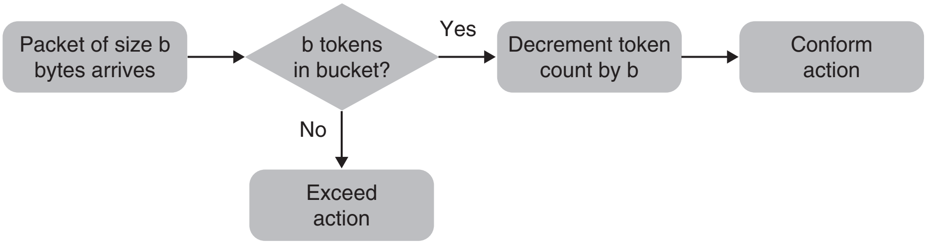


Figure 2.5 Simple one rate policer

The conform and exceed actions of a policer need not only be to transmit the traffic or to drop it; they could also include the marking of traffic or a combination of these actions, hence policers may also be know as markers. Marking is commonly used in conjunction with policing to enforce a defined committed "in-contract" rate on a traffic stream and to allow traffic in excess of this rate to be transmitted but to mark it differently from traffic within the contracted rate (this marking could potentially be to any of the fields described in Section 2.2.2) to indicate that it is "out-of-contract" such that it may potentially be given a less stringent SLA than the in-contract traffic; a more detailed description of applying the concept of in- and out-of-contract marking is given in the following section.

It is important to note that a token bucket policer never delays traffic, whereas a shaper does (see Section 2.2.4.3); there are no packets stored in the bucket; there are only tokens in the bucket! Hence, as a policer does not delay traffic, it cannot re-order or prioritize traffic as a scheduler can (see Section 2.2.4.1).

The simple token bucket policer described in this section is a subset of the functionality of the most commonly used policers, which are described in the following sections.

2.2.3.1 RFC 2697: Single Rate Three Color Marker

A commonly used policer is specified by the "single rate three color marker" (SR-TCM) defined in IETF RFC 2697 [RFC2697]; although this definition refers to a "marker" it can and is used to police traffic as well as to mark traffic. The "three colors" refer to the three possible

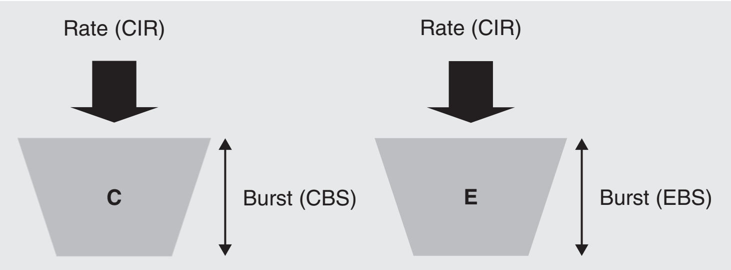


Figure 2.6 RFC 2697: A single rate three color marker

states that are outcomes of the SR-TCM, and which are described using a “traffic light” color scheme.

SR-TCM uses two token buckets, as shown in Figure 2.6, rather than the single token bucket described for the simple policer in the preceding section. The buckets are defined as C and E (for committed and excess, or conform and exceed) with maximum burst sizes CBS and EBS respectively.

Both buckets are filled with tokens at the same rate CIR. When the SR-TCM is applied to a traffic stream and a packet from that stream arrives:

- The packet size b is compared against the number of tokens currently in bucket C. IF there are at least as many tokens in bucket C as bytes in the packet, then the packet has conformed to the SR-TCM definition and the C bucket only is decremented by tokens equal to the number of bytes in the packet. Using the “traffic light” color scheme from RFC 2697 to indicate traffic conformance, in this case the packet is said to be green.
- ELSE IF there are not as many tokens in the C bucket as bytes in the packet, then the packet has exceeded the C bucket of the SR-TCM definition and is now compared against the E bucket. IF there are at least as many byte tokens in the E bucket as there are bytes in the packet, then the E bucket only is decremented by tokens equal to the number of bytes in the packet. Using the traffic light color scheme, in this case the packet is said to be yellow.

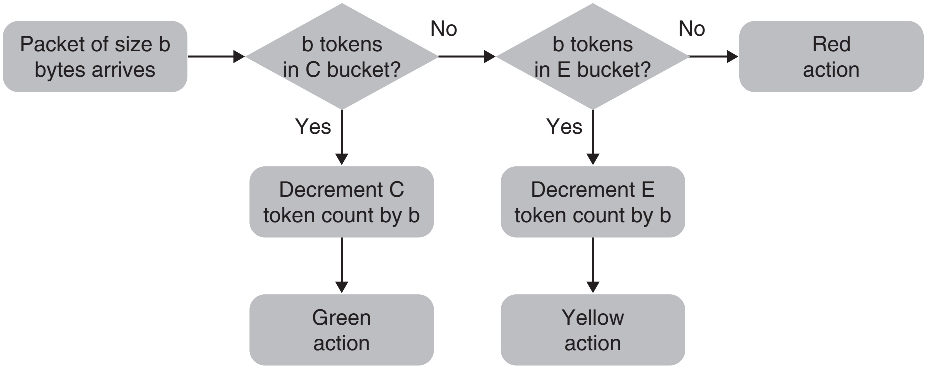


Figure 2.7 RFC 2697: A single rate three color marker (colorblind mode)

- ELSE IF there were neither as many tokens in the C bucket or E buckets as bytes in the packet, then we use the terminology that the packet has "violated" both buckets of the single rate three color token bucket definition. Using the traffic light color scheme, in this case, the packet is said to be red and neither bucket will be decremented. The flowchart in Figure 2.7 shows the operation of the SR-TCM.

CIR and CBS must be set >0, or else all packets will be red. If EBS = 0, then effectively the output of the marker has only two states, and packets will be marked either green or red, and the effective behavior of the SR-TCM is reduced to that of the simple one rate policer described in Section 2.2.3, i.e. "a single rate two color marker."

Different actions, such as transmitting, dropping, or marking the packet can then be applied – possibly in combination – depending upon whether the packet has been designated as green, yellow or red by the SR-TCM:

- Green packets will be transmitted and may be marked also; it makes no sense to apply a drop action to green packets.
- Yellow packets will be transmitted and may also be marked. It makes no sense to apply a drop action to yellow packets if EBS ≠ 0,

as the effect would be that all packets which are yellow or red would be dropped, i.e. there would be no differentiation between them.

- Red packets may either be transmitted, or marked and transmitted, or dropped. You might query why there would be a need to transmit red packets without remarking; this may be done if a policer is applied to meter (measure) whether the received traffic exceeds the defined rate (see Section 2.2.3.4).

An application of the SR-TCM could be to set EBS >0 and to apply a green action of {transmit + mark to indicate this traffic is "in-contract"}, a yellow action of {transmit + mark to indicate this traffic is "out-of-contract"} and a red action of drop. Applied in this way the SR-TCM would enforce a maximum rate of CIR and a burst of (CBS + EBS) on the traffic stream, and transmitted traffic would be marked as in- or out-of-contract depending upon whether it conformed or exceeded a burst of CBS. In practice, however, it is difficult to understand what meaningful service benefit is offered by differentiating between in- and out-of-contract traffic depending upon the level of burstiness of the traffic. Alternatively the same marker could be applied with green action of transmit, a yellow action of {transmit + mark to indicate this traffic is out-of-contract} and red action of {transmit + mark to indicate this traffic is "exceedingly-out-of-contract"} respectively; however, in practice it is similarly difficult to understand what meaningful service benefit is offered by differentiating between in-, out-, and exceedingly-out-of-contract traffic depending upon its level of burstiness. Hence, the RFC 2697 "Single Rate Three Color Marker" is not often used practically in this way; more commonly, it is used with EBS = 0; common applications include:

- Enforcing a maximum rate for a voice class of traffic, with EBS = 0 and applying a green action of transmit and red action of drop. Applied in this way the SR-TCM would enforce a maximum rate of CIR and a burst of CBS on the traffic stream, and any traffic in violation of this would be dropped, which is typical of the conditioning behaviors used for the Differentiated Services Expedited

Forwarding Per-Hop Behavior as described in Section 2.3.4.2.1. For a more detailed example like this, see Chapter 3, Section 3.2.2.3.1.

- Marking a certain amount of a traffic class as in-contract, and everything above that as out-of-contract, with EBS = 0 and applying a green action of transmit (if not pre-marked this could be combined with marking in-contract) and red action of {transmit + mark out-of-contract}. Applied in this way the SR-TCM would enforce a maximum rate of CIR and a burst of CBS on the traffic stream; any traffic in violation of this would not be dropped but would be marked out-of-contract, which is typical of the conditioning behaviors used for the Differentiated Services Assured Forwarding Per-Hop Behavior as described in Section 2.3.4.2.2. For a more detailed example like this, see Chapter 3, Section 3.2.2.4.5.2.

2.2.3.2 RFC 2698: Two Rate Three Color Marker

RFC 2698 [RFC2698] defines another commonly used policer/marker: the "two rate three color marker" (TR-TCM). Similarly to the SR-TCM, the TR-TCM uses two token buckets; however, a noticeable difference between the behaviors is that in the case of the two rate marker, the buckets – denoted in this case as C and P, with burst sizes of CBS and PBS respectively – are filled at different rates. C is filled at the committed information rate (CIR), while P is filled at the peak information rate (PIR), where PIR >= CIR and PBS >= CBS as represented in Figure 2.8.

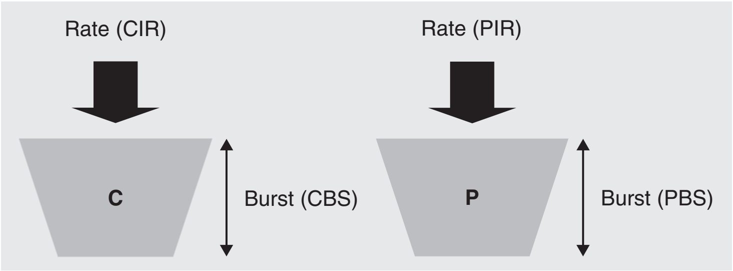


Figure 2.8 RFC 2698: A two rate three color marker

When the TR-TCM is applied to a traffic stream and a packet from that stream arrives:

- The packet size b is compared against the number of tokens currently in bucket P. IF there are fewer tokens in the bucket P than bytes in the packet, then the packet has violated the TR-TCM definition, and neither bucket will be decremented. Using the traffic light scheme, in this case the packet is said to be red.
- ELSE IF there are at least as many tokens in bucket P as bytes in the packet, then the packet is compared against the C bucket. IF there are fewer tokens in the bucket C than bytes in the packet, then the packet has exceeded the C bucket of the TR-TCM definition and the P bucket only is decremented by tokens equal to the number of bytes in the packet. Using the traffic light scheme, in this case the packet is said to be yellow.
- ELSE IF there are at least as many tokens in bucket C as bytes in the packet, then the packet has conformed to the TR-TCM definition and both the C and P buckets are decremented by tokens equal to the number of bytes in the packet. Using the traffic light scheme, in this case the packet is said to be green.

The flowchart in Figure 2.9 shows the operation of the TR-TCM.

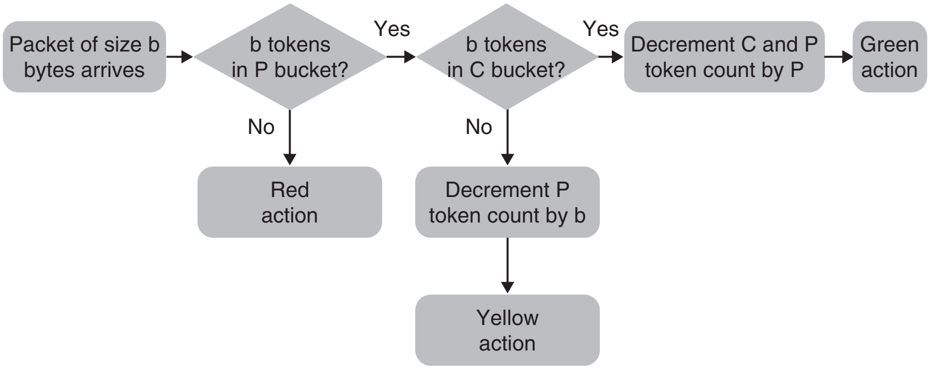

Packet of size b bytes arrives
b tokens in P bucket?
Yes
No
b tokens in C bucket?
Yes
No
Decrement C and P token count by P
Green action
Red action
Decrement P token count by b
Yellow action

Figure 2.9 RFC 2698: A two rate three color marker (colorblind mode)

As per the SR-TCM, different actions can then be applied, such as transmit, drop, or mark – potentially in combination – depending upon whether the packet has been designated as green, yellow, or red by the TR-TCM.

An example use of the TR-TCM is to mark a certain amount of a traffic class as in-contract, and everything above that as out-of-contract, up to a maximum rate, by applying a green action of transmit (if not pre-marked this could be combined with marking in-contract), yellow action of {transmit + mark out of contract}, and red action of drop. Applied in this way the TR-TCM would enforce a maximum rate of CIR and a burst of CBS on the traffic stream; any traffic in excess would then be marked out-of-contract up to a maximum rate of PIR and a burst of PBS. Although it is possible to have a red action of {transmit + mark}, for the same reasons as discussed for the SR-TCM, in practice it is difficult to understand what meaningful service benefit is offered by differentiating between traffic in terms of in-, out-, and exceedingly-out-of-contract. Hence, in practice the TR-TCM is most commonly used as a "two rate two color marker," i.e. with a red action of drop.

It is noted that if PIR = CIR and PBS = CBS then effectively the output of the marker has only two states, green and red; in this case the effective behavior would be the same as for the simple one rate policer described at the start of Section 2.2.3. Setting PIR < CIR or PBS < CBS may cause unpredictable results, where the same packet may violate the P bucket, i.e. be designated as red, where it would have conformed to the C bucket, i.e. be designated as green.

2.2.3.3 Color-aware Policers

The policing behaviors we have described so far have been color-unaware or "colorblind;" that is, once the packets have been classified into a stream that is being policed, the policer itself applies the policing actions indiscriminately of the packet markings. RFC 2697 and RFC 2698, however, also define "color-aware" modes of operation. When operating in color-aware mode, once the packets have been classified into a stream that is being policed, the policer takes into account any pre-existing markings that may have been set, by a policer at a previous network node for example, when determining the appropriate color-aware policing action for the packet, allowing different actions to be applied depending upon the pre-existing marking.

The RFC 2697 SR-TCM uses exactly the same token bucket definitions in color-aware mode as in colorblind mode. When applied to a traffic stream, the behavior in color-aware mode is as follows:

- IF the packet is pre-colored green and there are at least as many tokens in bucket C as bytes in the packet, then the packet conforms to the C bucket, is designated green and the C bucket only is decremented by tokens equal to the number of bytes in the packet.
- ELSE IF the packet is pre-colored green or yellow and there are at least as many tokens in bucket E as bytes in the packet, then the packet exceeds the C bucket, is designated yellow, and the E bucket only is decremented by tokens equal to the number of bytes in the packet.
- ELSE the packet violates the SR-TCM definition, is designated red and neither bucket will be decremented.

The flowchart in Figure 2.10 shows the operation of the SR-TCM in color-aware mode.

The SR-TCM, when operating in color-aware mode, ensures that packets pre-marked as yellow or red are not accounted against the C bucket and that packets pre-marked red are not accounted against the E bucket.

The RFC 2698 TR-TCM also uses exactly the same token bucket definitions in color-aware mode as in colorblind mode. When applied to a traffic stream, the behavior in color-aware mode is as follows:

- IF the packet is pre-colored red or if there are fewer tokens in the bucket P than bytes in the packet, then the packet violates the TR-TCM definition, is designated red, and neither bucket will be decremented.
- ELSE IF the packet is pre-colored yellow or if there are less tokens in the bucket C than bytes in the packet, then the packet exceeds

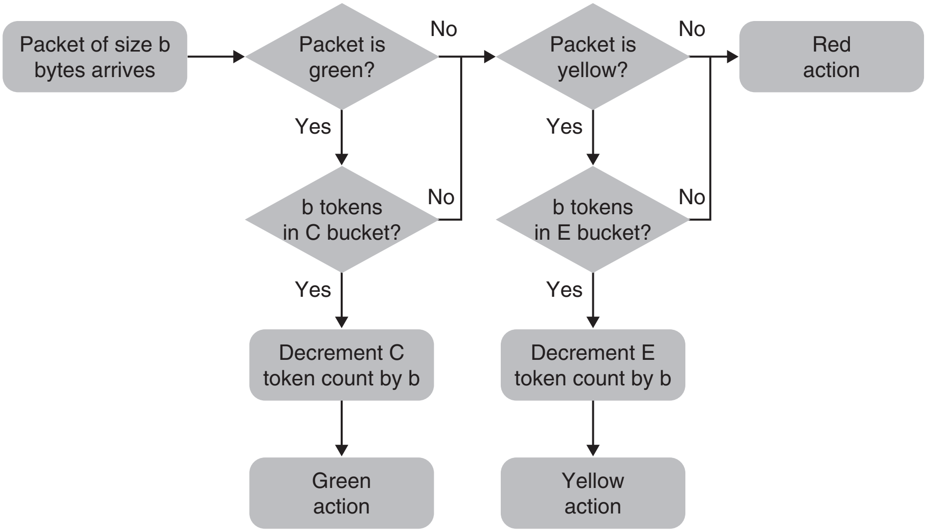


Figure 2.10 RFC 2697: Color-aware single rate three color marker

the C bucket definition, the packet is designated yellow, and the P bucket is decremented by tokens equal to the number of bytes in the packet.

- ELSE the packet conforms to the TR-TCM definition, is designated green, and both the C and P buckets are decremented by tokens equal to the number of bytes in the packet.

The flowchart in Figure 2.11 shows the operation of the TR-TCM in color-aware mode.

Color-aware policers are typically used at trust boundaries, where a downstream node (Node A) is expected to have applied a particular policer definition to condition a traffic stream to adhere to a traffic contract before sending the traffic to an upstream node (Node B). The color-aware policer is applied at Node B to ensure that traffic has been appropriately conditioned by Node A, while also trying to ensure that traffic is no indiscriminately re-marked. If a color-unaware policer was applied both at Node B, then as the policers operate independently, packets determined as green by the policer defined at

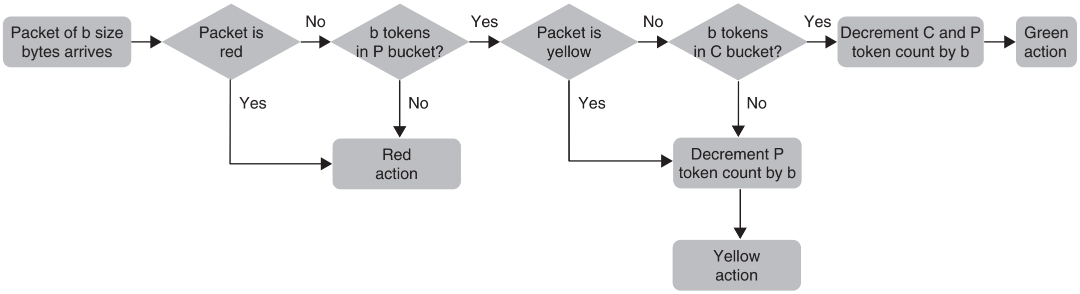


Figure 2.11 RFC 2698: Color-aware two rate three color marker

Node A may be determined to be yellow or red at Node B; similarly packets which A determined as yellow may be determined as green or red at B, and packets which A determined as red may be determined as green or yellow at B. The net effect of this may be that incorrect amounts of traffic are marked as green, yellow, and red, and hence the PIR and CIR commitments may not be met. Using a color-aware policer at Node B in conjunction with a color-unaware policer at Node A will overcome this problem and ensure that at Node B tokens of a particular color are only spent on packets of the same color.

2.2.3.4 Metering

Traffic metering is the process of measuring the rate and burst characteristic of a traffic stream for accounting or measurement purposes. A simple metering function could consist of applying either a single-rate or two-rate policer to a traffic stream, but with green, yellow and red actions all set to transmit; if statistics are maintained for the number of packets and bytes transmitted and dropped then these statistics could be used as a meter of a traffic stream. Alternatively, metering could be performed simply by taking the statistics of packet and bytes transmitted for traffic streams or classes, over a defined time interval; while this approach would provide a means to meter the average rate over the time interval, it would not provide any capability to measure the traffic burst.

2.2.4 Queuing, Scheduling, Shaping, and Dropping

In the most general terms, if the demands placed on any finite resource exceed the resource's ability to service them, contention happens. Scheduling mediates between demands when contention occurs, determining the time or order in which different demands are serviced. www.dictionary.com defines a schedule as "*A list of times of departures and arrivals;*" a scheduler is something that determines a schedule. Scheduling may re-order the demands relative to their arrival times; re-ordering is only possible if unserviced demands are delayed or queued.

In IP QOS terms, when, for example, the arriving traffic demands exceed a link's bandwidth, contention occurs and some of the traffic will need to be delayed or queued before it can be serviced. An IP packet scheduler acts upon the queued packets to determine their departure time (a real-time scheduler) or their departure order (a relative-time scheduler), shuffling packet departures according to rules derived from constraints of rates or priorities. Queues and schedulers are used in conjunction to control queuing delays and give bandwidth assurances to traffic streams.

2.2.4.1 Queuing and Scheduling

Queuing and scheduling in IP QOS has many close analogies in everyday life. Consider a simple example of an airline check-in, where there is single queue and a number of check-in counters, which service the queue in a first-come first-served (FCFS) basis, which is also referred to as first-in first-out (FIFO). The arrival rate at which the passengers turn up, relative to the rate at which the check-in counters are able to service the passengers, determines the length of the queue; arriving passengers start queuing at the "tail" of the queue and are serviced from the "head" of the queue. This is an example of the most basic queuing structure, as shown in Figure 2.12.

Several hours before the plane is due to depart, the rate of arrival of passengers at the check-in is relatively low; most passengers checking in at this time will be serviced straight away. An hour or two before departure, most people are checking in, the arrival rate of

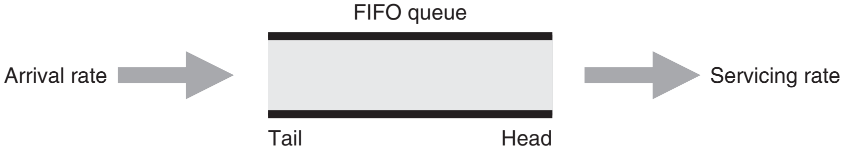


Figure 2.12 FIFO queue

passengers at the check-in is faster than the rate with which they can be serviced and a queue of passengers starts to form. The greater the difference between the arrival rate of the passengers and the servicing rate of the check-in desks, the longer the queue that will form and the greater the potential for delay for passengers in the queue. This case is analogous to a single service class IP network (we refrain from using the term "best-effort" for the reasons described in Section 2.1.2), where packet forwarding is egalitarian; all packets receive the same quality of service, and packets are typically forwarded using a strict FIFO queuing discipline.

If a check-in desk (or desks) is dedicated to business class passengers, which have a dedicated queue, then if the difference between the arrival rate and servicing rate for business class passengers is less than for economy passengers, then the business class queue length should be less than the economy queue and delay at check-in should be less also. Now consider an additional check-in desk and queue dedicated for first class passengers; if the difference between the arrival rate and servicing rate for first class passengers is less than for business passengers, then first class passengers should have even less time to wait. Hence, the queuing delays for the different classes can be managed by controlling the servicing rate (the number of check-in desks) relative to the arrival rate of passengers traveling at that class. The queuing delay for first class passengers could be further minimized by applying a prioritization scheme such that whenever a passenger arrives in the first class queue, they are serviced immediately by whichever check-in counter (economy, business or first class) next finishes servicing a customer. In addition, to make efficient use of resources and ensure that no check-in desks are unnecessarily left empty, if ever a

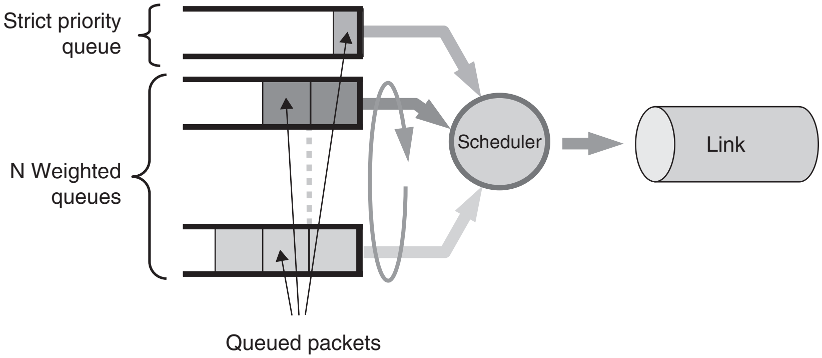


Figure 2.13 Basic IP scheduler

check-in desk (economy, business or first class) has an empty queue, it should next serve a passenger from a non-empty queue.

This airline check-in queuing and servicing scheme is very close to the basic IP scheduler implementations available today. In the same way that an airline check-in desk is a point of aggregation for passengers checking in for a flight, a router may be a point of aggregation for traffic; packets arriving on multiple links may be aggregated onto a single outbound link. Such aggregation can lead to congestion, hence the requirement for queuing and scheduling. Figure 2.13 illustrates the components of a typical basic IP packet scheduler, scheduling packets onto a physical link.

The components of such a scheduler are described in the following sections.

2.2.4.1.1 Priority Scheduling

Typically, most basic IP packet schedulers available today support a queue serviced with a priority scheduler for delay intolerant traffic such as voice and video.

Considered generally, priority scheduling can be either pre-emptive or non-pre-emptive. A pre-emptive priority algorithm would service a priority queue as soon as it becomes active, whereas non-pre-emptive priority algorithm would put the priority queue next on the list of queues to be serviced. In this context, pre-emption could be either at the packet level or at the quantum level. With packet level pre-emption, a non-priority packet currently being scheduled would be pre-empted before it is completely serviced by the scheduler; this approach provides the lowest delay characteristic for the priority queue, but results in bandwidth inefficiencies as the partially sent pre-empted packet needs to be resent in its entirety. With quantum level pre-emption, a non-priority queue currently being scheduled would be pre-empted before its full quanta (which may be a number of packets) has been serviced by the scheduler, but any packet currently being serviced from that queue would be allowed to finish; this approach is bandwidth-efficient, but may result in a higher delay characteristic for the priority queue.

Practical implementations of priority scheduling use quantum level pre-emption, i.e. if the priority queue is active, then the queue will be serviced next after any non-priority packet currently being serviced. This ensures that traffic in the priority queue receives bounded delay and jitter; if a packet arrives in the priority queue and the queue is empty, it should need to wait for at most one packet from another queue, before it is serviced by the scheduler. In practice, the delay impact on the priority queue may be more than just a single packet due to the presence of an interface FIFO queue (as described in Section 2.2.4.1.3).

As the priority queue is serviced with priority above other queues, if the priority queue is constantly active – that is, it always has packets to send – then the other queues may be starved of bandwidth. In order to prevent this from happening, it is common practice to police the traffic entering the priority queue, which enforces a maximum rate for the traffic using that queue. If this maximum rate is less than the available link rate, then there will always be bandwidth available for re-use by the other queues, irrespective of the load in the priority queue. Further, by controlling the offered load for the priority queue, the delay, jitter and loss characteristic for the queue can be bounded.

2.2.4.1.2 Weighted Bandwidth Scheduling

If the priority queue is inactive – that is, there are no packets in the queue – then there are a number of other queues which are each serviced in FIFO order. These queues will generally be serviced in a weighted fashion, where a weighting determines the service offered to one queue relative to another; the weightings effectively determine the share of the link bandwidth to each queue. A scheduling algorithm is used to ensure that the relative servicing between queues is achieved. By controlling the relative differences between the traffic arrival rates and servicing rates (determined by weightings) of different queues, the impact of queuing delays can be controlled and the relative service to those queues can be differentiated.

It is noted that queues can be assigned to discrete traffic flows (as in the case of Integrated Services architecture; see Section 2.3.3) or to traffic classes (as in the case of Differentiated Services architecture; see Section 2.3.4). Due to the scaling challenges of having per flow queues, it is more common to have per-class queues, where a class aggregates a number of flows which have common SLA requirements.

2.2.4.1.2.1 Weighted Round Robin Weighted Round Robin (WRR) is the simplest example of such a weighted bandwidth scheduling algorithm. It is easiest to explain how WRR works by way of an example.

EXAMPLE 1: Weighted Round Robin

Consider a scheduler which has three weighted queues (in addition to any priority queues), denoted as A, B and C with weights of 1, 2 and 4 respectively, as shown in Figure 2.14.

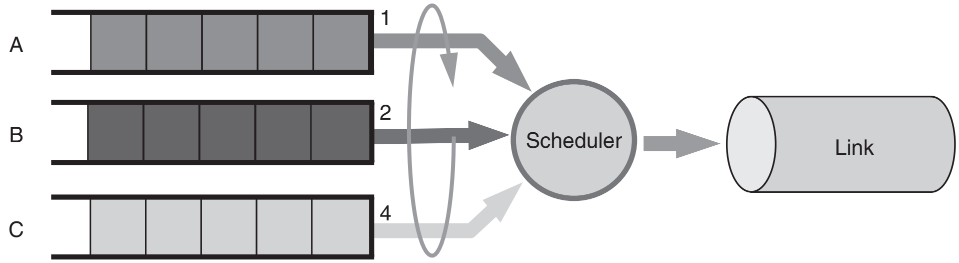


Figure 2.14 WRR scheduling example

In a round of the scheduler, the scheduler visits each queue and services an amount of traffic from that queue determined by the queue's weights. Hence, in this example, in each round, a WRR scheduler would service 1 packet, 2 packets and 4 packets from queues A, B and C respectively. If all queues were permanently full (i.e. their arrival rates constantly exceeded their servicing rates), the scheduling order would be A, B, B, C, C, C, C, A, B, B, C, C, C, C, A . . .

In the above example, the weighting of service between the queues is defined in terms of packets. If all packets in the different queues are the same size, then the allocation of the available link bandwidth is also weighted between the queues in the same proportion. In this case, if the link bandwidth was 512 kbps, queue A would be allocated 1/(1 + 2 + 4) * 512 = ~73 kbps, queue B would be allocated 2/(1 + 2 + 4) * 512 = ~146 kbps, and queue C would be allocated 4/(1 + 2 + 4) * 512 = ~293 kbps. In percentage terms, the link bandwidth has therefore been allocated with approximately 14% to queue A, 29% to queue B and 57% to queue C. Hence, the scheduler is providing these minimum bandwidth assurances for the queues; they are minimum bandwidth assurances as irrespective of the traffic load (but in this particular case not irrespective of packet sizes as we shall see shortly) in the other queues, each queue will be assured its respective bandwidth allocation at a minimum. Depending upon their particular equipment implementation, some vendors require that queues are configured by their relative weights, others allow configuration in terms of absolute (e.g. kbps) or relative percentage minimum bandwidth assurances, which are then converted to weights in order to program the scheduler.

In a WRR scheduler if any queues are inactive, then the scheduler moves on to the next queue and hence the unused bandwidth for the inactive queues is redistributed between the active queues in proportion to their relative weightings, i.e. in proportion to the queues' minimum bandwidth assurances. If queue B in the previous example was inactive, then the 146 kbps minimum bandwidth assurance for queue B would be redistributed between queue A and queue C in proportion to their weightings, i.e. in ratio 1:4. Hence queue A would now be allocated 73 + (146 * 1/5) = ~102 kbps and queue B would be allocated 292 + (146 * 4/5) = ~410 kbps. Schedulers that exhibit this characteristic – allowing unused bandwidth to be re-used by active queues – are called "work conserving" schedulers – they are only idle when there are no packets to send in any queues; most IP schedulers implemented today are work-conserving schedulers. In the preceding example, unused bandwidth is reallocated in proportion to the minimum bandwidth assurances of the active queues; some more advanced scheduler implementations support additional parameters, which allow the redistribution of unused bandwidth to be configured independently of the minimum bandwidth for the queue [CISCO].

A measure of the effectiveness of an IP scheduler is how closely the scheduler achieves the intended bandwidth allocation; this is referred to as the "fairness" of the scheduler. In the preceding example, WRR is fair as long as the packet sizes in the different queues are the same, and when considered over a complete round of the scheduler. If the average packets sizes in the different queues are the same, then WRR will be fair on average. If the average packet sizes of the different queues are not the same, then the scheduler could potentially normalize the weights of the queues according to the average packet size of each queue.

EXAMPLE 2: Weighted Round Robin

Continuing from the previous example, assume that queues A, B, and C have average packet sizes of 64 bytes, 1500 bytes and 300 bytes respectively and that the link is 512 kbps. With the weightings previously used the queue bandwidth allocations would be:

Queue A: 512 * (1 * 64)/((1 * 64) + (2 * 1500) + (4 * 300)) = ~8 kbps

Queue B: 512 * (2 * 1500)/((1 * 64) + (2 * 1500) + (4 * 300)) = ~360 kbps

Queue C: 512 * (4 * 300)/((1 * 64) + (2 * 1500) + (4 * 30)) = ~144 kbps

If the intended relative bandwidth allocation between the queues is 1:2:4, then this is clearly far from fair!

Normalizing each queue's weight by the average packet size for that queue, and expressing as an integer would give the following queue weightings:

$$\text{Queue A: weight} = 1/64 = 15 \times 10^{-3} \rightarrow \text{weight} = 150$$

$$\text{Queue B: weight} = 2/1500 = 1.3 \times 10^{-3} \rightarrow \text{weight} = 13$$

$$\text{Queue A: weight} = 4/300 = 13 \times 10^{-3} \rightarrow \text{weight} = 130$$

In practice, however, the average packet size in a queue may be difficult to estimate, and may vary over time. In these cases, the limitations of a simple WRR scheduler may be exposed as unfairness where some queues do not get their desired bandwidth allocation.

More advanced schedulers are able to overcome this issue, and some can be fair over time periods of less than a round of the scheduler. Fairness is measured by comparing the scheduler behavior against an idealized Generalized Process Sharing (GPS) scheme [KLEINROCK]. A GPS scheduler services an infinitesimally small amount from each queue on each round of the scheduler; hence, it visits all of the active queues in any finite time interval and therefore is fair in any time interval. Queues can have defined weights, and will receive service proportional to this weight whenever they have data in the queue.

A GPS scheduler is idealized in that it assumes that queues can be serviced in infinitesimally small amounts. This is clearly not possible in practice and for IP schedulers; the smallest unit that can be serviced from a queue is a single packet; in the timescale it takes to service a single packet, the scheduler must be unfair to other queues. Hence, no packet scheduler can be as fair as GPS, and in the following two sections we consider two practical scheduler implementations that are commonly used today and which aim to emulate a GPS scheme.

2.2.4.1.2.2 Weighted Fair Queuing Weighted fair queuing (WFQ) [DEMERS] computes the time that a packet would finish being serviced if it was being serviced using a GPS scheme; it then services packets in the order of their finish time, which in effect becomes a sequence number. WFQ is effectively the packet-based version of GPS.

Consider the following example.

EXAMPLE 3: Weighted Fair Queuing

Consider a scheduler that has three weighted queues (in addition to any priority queues): A, B, and C with desired relative bandwidth allocations of 1:2:4 (or 14%, 29%, and 57%) respectively. Assume that queues A, B, and C are permanently full and have packet sizes of 64 bytes, 1500 bytes, and 300 bytes respectively and that the link is 512 kbps. Consider packets arrive back-to-back in the order A1, A2, B1, C1, C2, C3, . . . faster than the scheduler can service the first packet.

In order to determine the servicing time of packets, a WFQ scheduler keeps track of a variable called the round number. If you considered a GPS scheduler servicing each queue byte-by-byte rather than in infinitesimally small amounts, the round number represents the number of complete rounds of byte-by-byte service that the WFQ scheduler has completed.

When a packet arrives at a previously inactive queue, its servicing time (i.e. sequence number) is calculated by taking the current round time and adding the size of the packet multiplied by the queue's weight; consequently with WFQ the bandwidth share of a queue is inversely proportional to that queue's weight. In this case, to achieve the desired bandwidth share of 1:2:4, weights of 4, 2, and 1 are allocated to queues A, B, and C respectively. With WFQ, whether a queue is active can be determined by whether there are any packets in the queue that have a sequence number greater than the current round number. When a packet arrives at an active queue, its sequence number is calculated by adding the size of the arriving packet multiplied by the queue's weight to the highest sequence number of packets already in the queue.

Consider the current round number is 0:

- Packet A1 arrives; the sequence number for the packet is calculated as 0 + 64 * 4 = 256
- Packet A2 arrives and, as the queue is active, the sequence number for the packet is calculated as 256 + 64 * 4 = 512
- Packet B1 arrives and, as the queue is inactive, the sequence number for the packet is calculated as 0 + 1500 * 2 = 3000
- Packet C1 arrives and because the queue is inactive, the sequence number for the packet is calculated as 0 + 300 * 1 = 300
- Packet C2 arrives and, as the queue is active, the sequence number for the packet is calculated as 300 + 300 * 1 = 600
- Packet C3 arrives and, as the queue is active, the sequence number for the packet is calculated as 600 + 300 * 1 = 900.

The scheduler services the packet with the lowest sequence number first, and updates the round to be the equal to the sequence number of that packet. If we compare the sequence numbers of the received packets in this case, we can see that the packets received in order A1, A2, B1, C1, C2, C3, are sent in order A1, C1, A2, C2, C3, B1.

2.2.4.1.2.3 Deficit Round Robin Deficit Round Robin (DRR) [SHREEDHAR] modifies WRR such that it can be fair without knowing the average packet sizes of packets in particular queues. This is achieved by keeping track of a deficit counter for each queue. A DRR scheduler visits each queue in a round and aims to service a weight or quantum's worth from each queue. Unlike WRR, the quantum is defined in bytes rather than in packets. When it is a queue's turn to be serviced, the scheduler will attempt to service a complete quantum from the queue. In practice, it is unlikely that the quantum will exactly equal the size of the next packet, or the next few packets at the front of the queue. In this case as many whole packets will be serviced from the front of the queue as can be accommodated by the quantum; if the first packet is greater than the available quantum, then no packets will be serviced from that queue in that round. If there are more packets in the queue than can be accommodated by the quantum, any unused quantum for the queue on that round of the scheduler will be carried forward to the next round, else the deficit counter is reset. In this way, queues which did not get their fair share in one round receive recompense on the next round. Consider the following example.

EXAMPLE 4: Deficit Round Robin

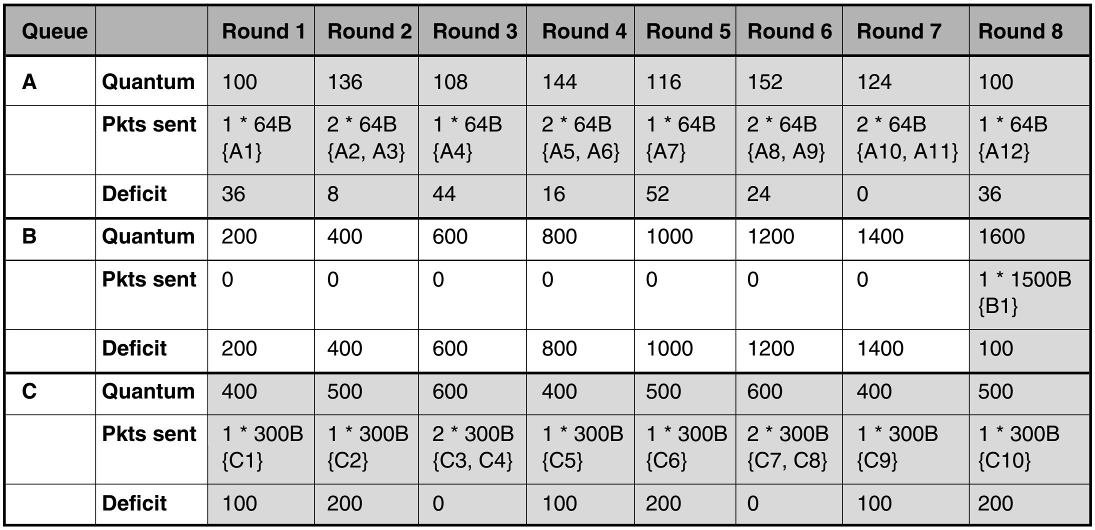
Consider a scheduler, which has three weighted queues (in addition to any priority queues): A, B, and C, which have desired relative bandwidth allocations of 1:2:4 (or 14%, 29%, and 57%) respectively and have quanta of 100, 200, and 400 accordingly. Assume that queues A, B, and C are permanently full and have packet sizes of 64 bytes, 1500 bytes, and 300 bytes respectively and that the link is 512 kbps.

All of the queue deficit counters are initially set to zero. On the first round of the scheduler, the quantum for queue A is 100, and the packets are 64 bytes, so there is sufficient quantum to service one complete packet. As there are more packets in queue A, the remaining quanta will be carried forward as a deficit to the next round; in this case, the deficit counter for queue A will be 100 − 64 = 36 bytes. This will be added to the queue quantum on the next round of the scheduler.

On the first round, after queue A the DRR scheduler next looks at queue B; the quantum for the queue is 200 and the packets are 1500 bytes, so there is insufficient quantum to service any packets and the remaining quantum will be carried forward as a deficit to the next round; in this case the deficit counter for queue B will be 200 bytes. The deficit counter for queue B will continue to increase until round 8, when the deficit counter + quantum will equal 1600 bytes and hence a single 1500-byte packet will be serviced. As there are more 1500-byte packets in the queue, the deficit counter will be set to 1600 − 1500 = 100 bytes and carried forward to round 9.

On the first round, after queue B the DRR scheduler next looks at queue C; the quantum for the queue is 400, and the packets are 300 bytes, so there is sufficient quantum to service one complete packet. As there are more packets in queue C, the remaining quantum will be carried forward as a deficit to the next round; in this case, the deficit counter for queue C will be 400 − 300 = 100 bytes. This will be added to the queue quantum on the next round of the scheduler.

The table in Figure 2.15 shows the status of the queues, in terms of quantum packets sent and deficit, at each round of the scheduler.



Queue		Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7	Round 8
A	Quantum	100	136	108	144	116	152	124	100
	Pkts sent	1 * 64B {A1}	2 * 64B {A2, A3}	1 * 64B {A4}	2 * 64B {A5, A6}	1 * 64B {A7}	2 * 64B {A8, A9}	2 * 64B {A10, A11}	1 * 64B {A12}
	Deficit	36	8	44	16	52	24	0	36
B	Quantum	200	400	600	800	1000	1200	1400	1600
	Pkts sent	0	0	0	0	0	0	0	1 * 1500B {B1}
	Deficit	200	400	600	800	1000	1200	1400	100
C	Quantum	400	500	600	400	500	600	400	500
	Pkts sent	1 * 300B {C1}	1 * 300B {C2}	2 * 300B {C3, C4}	1 * 300B {C5}	1 * 300B {C6}	2 * 300B {C7, C8}	1 * 300B {C9}	1 * 300B {C10}
	Deficit	100	200	0	100	200	0	100	200

Figure 2.15 Example: DRR queue status (shaded cell indicates packet dequeued)

Over the 8 rounds of the scheduler, the total number of bytes allocated to each queue is as follows:

Queue A: 12 * 64 = 768

Queue B: 1 * 1500 = 1500

Queue C: 10 * 300 = 3000

If we add on the value of the deficit counter, we can determine the effective relative bandwidth allocation to each queue over the 8 rounds:

Queue A: = (768 + 36)/((768 + 36) + (1500 + 100) + (3000 + 200)) = ~14%

Queue B: = (1500 + 100)/((768 + 36) + (1500 + 100) + (3000 + 200)) = ~29%

Queue C: = (3000 + 200)/((768 + 36) + (1500 + 100) + (3000 + 200)) = ~57%

Hence, we can see that the DRR demonstrates fairness irrespective of the packet sizes, albeit possibly over a number of rounds of the scheduler; the more rounds it is considered over, the more fair it becomes, as the outstanding deficit counter (which represents bytes not yet sent on that queue) has proportionally less impact.

2.2.4.1.2.4 Which Scheduling Algorithm? There are a number of characteristics which can be used to differentiate between scheduling algorithms, and which impact where they are used:

- *Fairness*. As previously described, the fairness of a scheduler is a measure of how closely the scheduler achieves the intended bandwidth allocation. Clearly, fairness is a desirable characteristic of any scheduler. Hence why DRR and WFQ are preferred IP packet scheduling algorithms to WRR, which will only provide a fair bandwidth allocation between queues if the packet sizes in the different queues are the same, which will generally not be the case.

- *Worst-case delay bounds*. Some scheduler implementations may attempt to support traffic which has low delay requirements from a weighted bandwidth queue, which is serviced using a scheduling algorithm such as WRR or WFQ. Different scheduling algorithms acting on the same set of queues would result in different packet dequeue orders, even when they may be configured to produce the same desired bandwidth allocation. Consequently, the worst-case delay bounds for a particular queue will depend upon the scheduling algorithm used and may also be dependent upon the number of queues used in the particular implementation. Further, for some scheduling algorithms, the weighting applied to the queue may need to be artificially inflated in order to reduce the worst-case delay bound by increasing the effective queue scheduling rate. By inflating the bandwidth of one class, the relative share of bandwidth available to the other classes may decrease, which can result in coarser relative granularity of bandwidth allocation to the other classes. Hence, it may be a desirable scheduler characteristic that the worst-case delay bound for a particular queue is as low as possible.

  In practical deployments, however, traffic which has low delay requirements is most commonly serviced using a strict priority queue rather than a weighted bandwidth queue. Hence, if a particular weighted bandwidth scheduler implementation is augmented with priority queues for low-delay traffic, the worst-case delay bound characteristic for the weighted bandwidth queues may not be critical when choosing the scheduling algorithm used for the weighted bandwidth queues.

- *Simplicity*. From a platform implementation perspective, there are benefits in an algorithm which is simple to implement; the fewer cycles and less state needed to implement a particular algorithm, the less processing power and memory required and hence the easier it is to scale and the lower the cost impact on the platform. DRR is less computationally intensive and simpler to implement than WFQ, and hence is generally preferred where a high degree of platform scalability is required, either for high-speed links or for a large number of lower-speed connections.

Although there are other scheduling algorithms used in IP networks, we have described those which are by far the most widely implemented together with the key characteristics which determine where the different algorithms are used. For a more detailed understanding of scheduling theory, see [KESHAV].

2.2.4.1.3 Interface FIFO

For most practical router implementations, the scheduler will not actually schedule queues directly onto the physical link, but rather will service its queues into the queue of the hardware line driver on the outgoing interface; this queue is designed to provide buffering before the hardware line driver allowing the line driver to maximize interface throughput. This queue is a FIFO queue, which is variously known as the interface FIFO or transmit ring buffer (tx-ring for short) and which is shown in Figure 2.16.

If the scheduler can dequeue packets into the interface FIFO faster than they can be serviced (i.e. faster that the link rate) then the transmit ring buffer may start to fill. It is common to implement a flow control mechanism to ensure that the interface FIFO does not continue to fill uncontrollably, but rather when the number of queued packets in the interface FIFO exceeds a defined threshold the flow control will stop the scheduler dequeuing any more packets (this is known as a

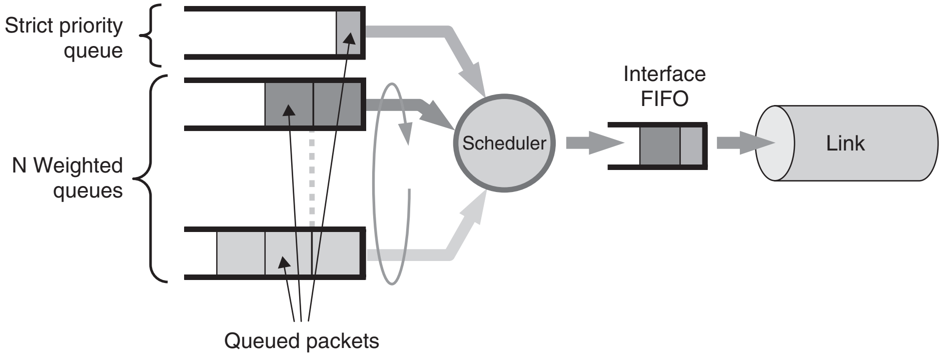


Figure 2.16 Interface FIFO

"flow off"). When the number of packets queued in the interface FIFO buffer falls below a threshold (which must be equal to or lower than the flow-off threshold) the flow control mechanism allows the scheduler to send more packets into the interface FIFO again (this is known as a "flow on"). This type of flow control is sometimes referred to as "back pressure" exerted from the interface FIFO to the scheduler. Therefore packets dequeued by the scheduler may be enqueued behind packets already in the interface FIFO. Even a priority packet can at best be enqueued at the tail of the interface FIFO and consequently the interface FIFO size can impact the queuing delays of all of the scheduler's queues; hence it is important that the size of the interface FIFO is not unnecessarily large.

The impact of even a reasonably sized interface FIFO on the delay of the priority queue can be significant on low-speed access links; optimally it would be tuned to 1-to-2 MTUs, where one packet is being clocked onto the link, while another packet is being enqueued in parallel into the interface FIFO. If the perturbing delay introduced by the interface FIFO exceeds the delay target for traffic in the priority queue, then link level fragmentation and interleaving mechanisms may be required, as described in Section 2.2.5. At higher speeds, i.e. for core network links, the delay impact of a reasonably sized interface FIFO will generally be negligible.

2.2.4.1.4 Advanced Concepts in Scheduling: Multi-level Strict Priority

Whilst the IP scheduler depicted in Figure 2.13 represents the de facto implementation found today, some more advanced implementations are adding support for more than one priority queue. The demand driving this requirement is the concurrent support of voice and video services. As described in Chapter 1, voice services generally have tighter network delay requirements than streaming video services, although the video services still require a delay-bounded service. In addition, video streaming applications often use large sized packets for bandwidth efficiency. Some vendors have implementations that allow multiple subsets of traffic within a class queue to be discretely policed, but to be serviced from the same queue; such implementations may limit the worst-case impact that the streams of traffic have on each other, but there will be an impact nonetheless. Hence, if voice and video are serviced from the same priority queue, large video packets can increase the worst-case delay experienced by the voice traffic; this effect can be significant on low-speed links.

In an alternative deployment approach, using the same scheduler, one of the weighted bandwidth queues could be used to support the video traffic. In this case, as described in Section 2.2.4.1.2.4, the delay bound that the video traffic will experience may vary depending upon the scheduler used, may also be dependent upon the number of other weighted bandwidth queues used in the particular implementation and the traffic in those queues, and at low-link speeds it may not be possible to achieve the required delay targets. Further, for some scheduling algorithms, the weighting applied to the queue may need to be artificially inflated in order to reduce the worst-case delay bound. By inflating the bandwidth of one class, the relative share of bandwidth available to the other classes decreases, which can result in coarser relative granularity of bandwidth allocation to the other classes.

In order to overcome these issues, some more advanced scheduler implementations provide support for more than one priority queue as shown in Figure 2.17.

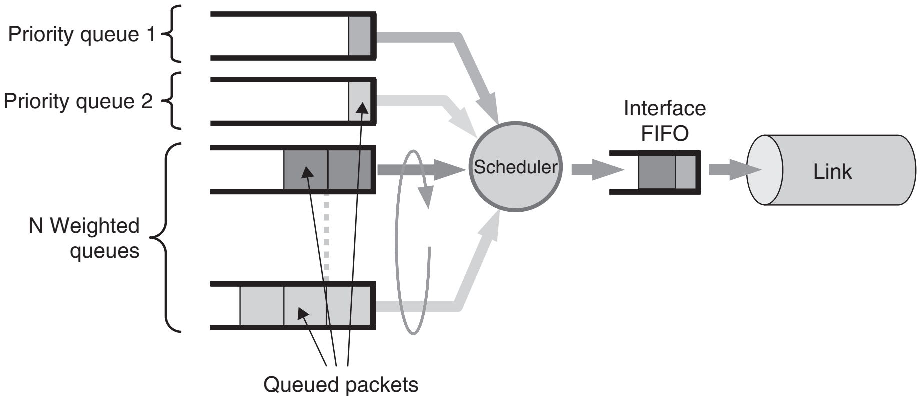


Figure 2.17 Multi-priority queue scheduler

The priority queue with the highest priority is serviced at line rate as soon as it becomes active; once this queue has been serviced, the second priority queue is serviced next. Finally, after the second priority queue, weighted bandwidth queues are serviced. When supporting voice and video, for example, using the highest priority queue for voice traffic, and the next priority queue for video traffic would enable the voice traffic to receive the lowest delay and jitter, while the video traffic would have a bounded delay and jitter, independent of the configuration and load of the weighted bandwidth queues. With multi-priority scheduler implementations such as this, the delay and jitter for the traffic at both levels of priority can be bounded and hence both levels of priority queue are compliant with the Differentiated Services (Diffserv) expedited forwarding (EF) per-hop behavior (PHB) definition, as described in Section 2.3.4.2.1.

While multi-priority queue scheduler implementations are currently the exception, they may become the norm as concurrent support for voice, video, and data services on IP networks becomes more widespread.

2.2.4.2 Dropping

At this point, before considering dropping, it is worth highlighting the difference between buffers and queues. Buffers are the physical memory locations where packets are temporarily stored while they are waiting to be transmitted. Queues on the other hand do not contain packets although it is common parlance to refer to "packets in a queue;" rather, a queue consists of an ordered set of pointers to the locations in buffer memory where packets in that particular queue are actually stored. Fast buffer memory is often an expensive component of a router implementation and hence buffer memory may be shared across all queues for more efficient buffer memory usage, rather than rigidly partitioned between queues. Buffer memory may also be organized in fixed sized chunks (which may be known as particles), which are typically 256–512 bytes, in order to facilitate fast memory lookups while making efficient use of buffer memory. For example, if a system used 256-byte particles, a 576-byte packet would consume three particles of buffer memory.

Considering the scheduler shown in Figure 2.13, if the traffic arrival rate continually exceeds the available link bandwidth, then the number of packets in at least one of the queues must continually increase. As the buffer memory used to queue packets must be finite, at some point, the queue's depth must exceed the available buffer memory, and inevitably packets must be dropped. It is important that routers have sufficient buffer memory to be able to buffer packets being queued in accordance with the configured Diffserv policy; buffer memory starvation can lead to "no buffer" packet drops that occur indiscriminately of class, resulting in violation of the class SLA commitments. Applying limits on the depths of queues has a consequent limiting impact on buffer memory usage.

While limiting buffer memory usage is one reason to drop a packet, in practice today's router platforms generally have sufficient memory that it is not the constraining factor on queues' depths. The main reasons to limit or manage the depth of a queue are either to bound the delay experienced by packets in the queue, or in an attempt to optimize the throughput achieved for traffic in the queue; different dropping techniques are applied depending upon the aim.

2.2.4.2.1 Tail Drop

A "tail drop" mechanism is used to place a hard limit on the number of packets that can be held in a queue. Before a packet is to be enqueued at the tail of a queue, the current depth of packets in the queue is checked and if the depth of the queue exceeds the maximum limit for the queue, which is normally specified in bytes, then the packet will be dropped rather than enqueued. We can consider that the probability of the packet being dropped on enqueue to that queue is zero while the queue depth is less than the tail drop queue limit, q_{limit}, but when the queue limit is reached the probability of being dropped is 100%, as shown in the drop probability graph in Figure 2.18.

Setting the maximum queue limit for a queue can be used to enforce a maximum delay bound on the traffic in the queue. Why might we want to set such a delay bound on a queue? Is it not better to send a packet if we possibly can, rather than to drop it? The answer to these questions depends upon the application. If we return to the

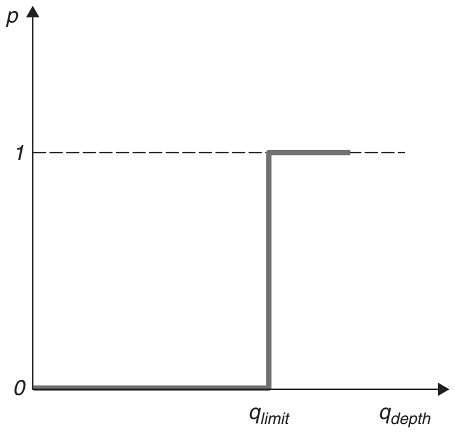


Figure 2.18 Tail drop

airline check-in analogy we introduced in Section 2.2.4.1, if the queue at check-in becomes too long, then the queuing delay may exceed the time to wait before the actual departure of the plane, in this case, there is no point in the passenger queuing as they will miss their plane. Similarly, with some applications such as VoIP, if a packet is delayed too much, it will be of no use and it is better to drop it rather than to consume bandwidth across the network and be dropped at the destination. Section 3.2.2.1.1 in Chapter 3 gives an example delay budget for VoIP, which determines a maximum acceptable delay at each hop in the network. Dropping can be considered the most extreme case of delay; that is, a packet that is infinitely delayed never arrives, and for all intents and purposes can be considered lost or dropped.

If the servicing rate of a queue is known, and the maximum size of the queue is also known, then the worst-case delay bound for a packet in the queue can be determined. If, for example, the servicing rate for a queue is 2 Mbps, and the maximum queue limit for the queue is 4 kilobytes, then the worst-case delay bound for a packet that is enqueued in the queue will be approximately 4096 * 8/2048000 = ~16 ms. Hence, if it is determined that the maximum per-hop delay that an application can sustain is 16 ms, then this may be an appropriate maximum queue depth setting. In practice, however, this is a simplified example, and there may be delays other than just the queuing delay which need to be taken into account and a more thorough analysis of these delays is provided in Chapter 3, Section 3.2.2.4.1.

Head drop (also known as "drop from the front" or DFF) is a possible alternative to tail drop; with head drop packets are dropped from the head of the queue rather than from the tail, when the depth of the queue exceeds the configured maximum limit for the queue. Lakshman et al. [LAKSHMAN] have shown that head drop improves the performance of TCP by allowing the congestion indication signal to reach the sender faster than waiting for the full queue to be transmitted first. Head drop, however, has mostly been a subject of academic research and is not generally supported by router vendor's implementations, hence we do not consider it further.

2.2.4.2.2 Weighted Tail Drop

Some queuing implementations support more that one queue limit within a queue; this is sometimes known as "weighted tail drop." The concept behind this is that if there is congestion in the queue – that is, the traffic arrival rate R_a for the queue exceeds the servicing rate R_s – and the queue depth starts to build, then some subset of the traffic in the queue will be preferentially discarded; the arrival rate of the traffic which will be preferentially discarded is R_{a1}, and the remainder is R_{a2}, such that $R_a = R_{a1} + R_{a2}$. The traffic that is to be preferentially discarded may be classified by a different marking from the remainder of the traffic; the traffic may have been differentially marked as in- and out-of-contract using a policer as described in Section 2.2.3. The preferential discard is achieved by applying a lower queue limit q_{limit1} to the subset of traffic which is to be discarded first than for the remainder, which has a queue limit q_{limit2}. If the arrival rate for the remainder of the traffic R_{a2} is less than the servicing rate of the queue R_s, and the burst of the remainder of the traffic is less than the difference between the two queue limits $(q_{limit2} - q_{limit1})$, then in congestion of the queue, only traffic from the subset to be preferentially discarded will be dropped. This scheme is illustrated by the drop probability graph in Figure 2.19.

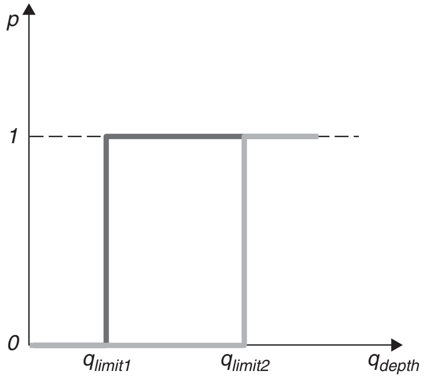


Figure 2.19 Weighted tail-drop

2.2.4.2.3 Random Early Detection

Random early detection (RED) [FLOYD1] is an active queue management (AQM) mechanism. AQM mechanisms detect congestion before queues overflow (i.e. before tail drop is invoked), and provide feedback of this congestion to the end-systems with the aim of avoiding excess packet loss due to congestion and maintaining high network throughput while minimizing queuing delays. Hence, AQM mechanisms are also known as "congestion avoidance" techniques. RED was originally designed as an algorithm aimed at improving throughput for TCP-based sessions, by aiming to prevent the observed behavior of "global synchronization" [DORAN] between TCP sessions. Global synchronization is a behavior which can occur where TCP sessions are aggregated on a single connection (or queue); when congestion occurs, the queue limit is exceeded, causing packet drops across multiple TCP sessions. Due to the adaptive nature of TCP (see Chapter 1, Section 1.3.3.1), on realizing that packets have been dropped the impacted sessions react by each slowing their rate of sending, hence the congestion abates and the effective aggregate throughput drops below line rate. As there is no congestion there are no packets dropped and the sessions then all increase their rate of sending until congestion occurs again and the cycle repeats, potentially creating a sawtooth aggregate throughput characteristic, as illustrated in Figure 2.20.

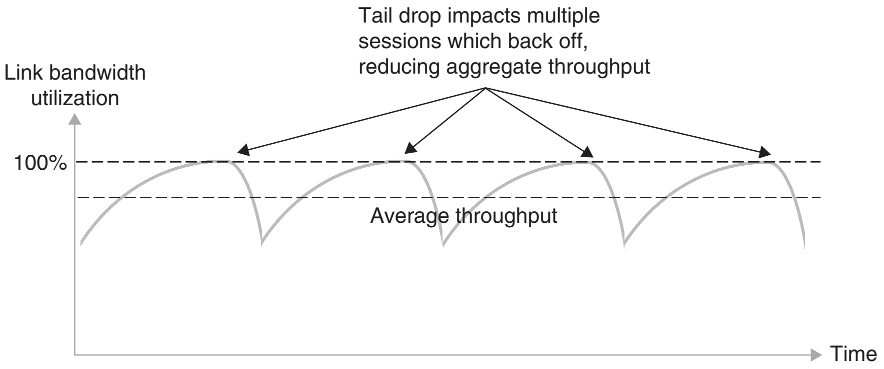


Figure 2.20 TCP "global synchronization"

RED aims to try and break TCP global synchronization by keeping track of the average depth of the queue and using it as an indication of when congestion is approaching; the average queue depth is tracked rather than the actual queue depth, which is used in tail drop, in order to accommodate the bursty nature of TCP. This indication of congestion is fed back to the end-systems by randomly discarding packets from individual sessions as the average queue depth increases, rather than dropping packets across all sessions. The aim of this approach is to cause individual sessions to back off in order to reduce the aggregate throughput in a controlled manner such that a higher aggregate throughput is maintained on average.

RED makes a drop decision prior to enqueuing a packet into a queue based upon the current average queue depth of that queue and a set of four parameters, which are configurable in most implementations:

- The average queue (q_{avg}) depth is calculated using the following formula:

$$q_{avg} = q_{avg\_old} \times \left(1 - \frac{1}{2^w}\right) + \left(q_{current} \times \frac{1}{2^w}\right)$$

where:

q_{avg_old} = the previously calculated average queue depth

$q_{current}$ = the current (not averaged) queue depth

w = the exponential weighting constant

RED uses an exponential weighted moving average; the exponential weighting constant (w, which is normally configurable) determines how closely the average queue depth tracks the actual queue depth; the lower w the more closely the average queue depth tracks the actual queue depth, i.e. the more sensitive the RED drop behavior is to traffic bursts.

- If the current average queue depth (q_{avg}) is below a configurable minimum threshold (q_{minth}), the packet is enqueued.
- If the current average queue depth (q_{avg}) is above a defined maximum threshold (q_{maxth}) then the packet is always dropped; this is referred to as a "forced drop."
- If the current average queue depth (q_{avg}) is above q_{minth} and below q_{maxth}, the packet may be dropped with an increasing, but randomized, probability; this is referred to as a "random drop" and the probability of a random drop (p) is determined by the following formula:

$$p = \left(\frac{q_{avg} - q_{minth}}{q_{maxth} - q_{minth}} \right) \times P_{max} \times RAND(1)$$

where p_{max} is the probability of packet loss at q_{maxth}, which impacts how quickly the probability of the packet being dropped increases between q_{minth} and q_{maxth}.

The RED dropping behavior is illustrated by the drop probability graph in Figure 2.21, where the specific parameters chosen define a particular RED drop profile.

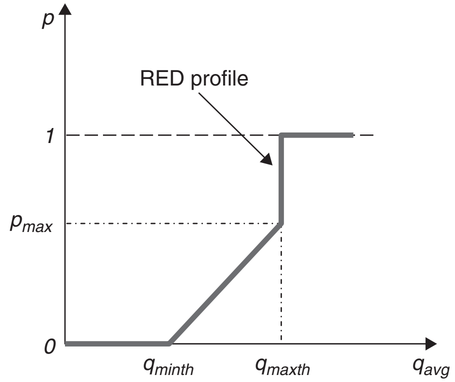


Figure 2.21 Random early detection drop profile

An enhancement to RED was proposed in "Red-Light" [JACOBSON] and is used in some implementations; RED-LIGHT does not have the concept of a configurable exponential weighting constant. Further enhancements to RED have been proposed in [FLOYD2].

The widespread use of RED was advocated in [RFC2309]; however, the benefit of RED is difficult to quantify in practice. AQM has been a favored subject in academic research and some research has recommended against the deployment of RED [MAY]. There have also been a number of new algorithms proposed for AQM; in [BITORIKA] they note that more than 50 new algorithms were proposed between 1999 and 2003 alone. In practice, none of these schemes have been widely implemented today and RED remains the most widely supported AQM algorithm implemented by router vendors, and most widely implemented in networks. In the authors' experience, with appropriate tuning (see Chapter 3, Section 3.4), RED is not generally worse than tail drop; however, the claims of both its benefits and its disadvantages appear to have been overstated.

RED was designed for TCP and as such is used for queues that carry TCP applications. RED is not intended for use with inelastic applications such as VoIP or video, which commonly use UDP as these applications cannot adapt to RED drops. Further, with applications such as voice or video, it is generally preferable to have a bounded worst-case delay for the queue enforced with a firm queue limit. With

RED, however, the maximum queue depth is dependent upon the maximum threshold and current average-queue depth, hence the actual maximum queue depth may be significantly greater than the maximum threshold and the worst-case queuing delay bound may not be easily determined. There are some elastic applications that use UDP rather than TCP, such as the Trivial File Transfer Protocol (TFTP) for example. UDP does not have any implicit reliability or flow control mechanisms built in, hence if they are required they need to be built into the application implementation. For applications such as these detailed knowledge of the specific application implementation is required in order to understand what impact RED would have on them; however, in general the performance with RED should not be significantly worse than with tail drop.

2.2.4.2.4 Weighted Random Early Detection

Weighted RED (WRED) extends the basic concept of RED, by allowing a number of different RED profiles to be used for the same queue, where each profile may be applied to a different subset of the traffic destined for the queue. The concept is very similar to weighted tail drop, in that if there is congestion in the queue, then some subset of the traffic in the queue will be preferentially discarded; this is achieved by having a more aggressive WRED profile (lower q_{minth} and q_{maxth} settings) for the traffic that will be discarded first. The traffic that is to be preferentially discarded may for example be identified using a different marking from the remainder of the traffic; the traffic may have been differentially marked as in- and out-of-contract using a policer as described in Section 2.2.3. The WRED dropping behavior is illustrated by the drop probability graph in Figure 2.22.

2.2.4.2.5 Advanced Concepts in Dropping

A new breed of dropping algorithms has been defined [PAN1, PAN2], which combine FIFO packet scheduling with differential dropping on packet enqueue and are claimed to be capable of approximating a variety of bandwidth allocation and control behaviors including those traditionally supported by scheduling algorithms. Although such mechanisms have yet to be generally implemented or deployed,

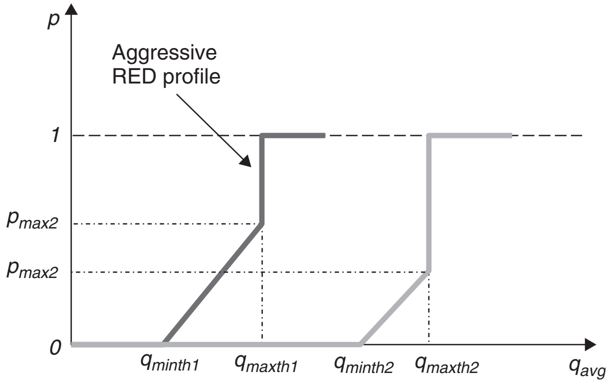


Figure 2.22 Weighted random early detection drop profile

they potentially offer the benefit of scheduling-like bandwidth allocation behaviors with lower complexity implementations.

2.2.4.3 Shaping

Shaping, like policing, is a mechanism which can be used to ensure that a traffic stream does not exceed a defined maximum rate. Also like policing, a shaper can be visualized as a token bucket mechanism like that shown in Figure 2.4, with a defined maximum depth (normally in bytes), known as the burst B, and a defined rate R (normally in bps) at which the bucket is filled with byte-sized tokens. Depending upon the particular shaper implementation, tokens are added to the bucket either every time a packet is processed by the shaper, or at regular intervals, up to a maximum number of tokens that can be in the bucket, defined by B. The minimum number of tokens in the bucket is zero.

The difference between a shaper and a policer becomes apparent when considering what happens when a shaper is applied to a traffic stream. When a packet from that stream arrives, the packet size b is compared against the number of tokens currently in the bucket. If there are at least as many byte tokens in the bucket as there are bytes in the packet, then the packet is transmitted without delay. If there are fewer tokens in the bucket than bytes in the packet, then the packet is delayed (i.e. queued, hence shapers are implicitly used in conjunction with queues) until there are sufficient tokens in the bucket; when there are sufficient tokens in the bucket, the packet is sent and the bucket is decremented by a number of tokens equal to the number of bytes in the packet. In this respect a shaper is significantly different from a policer, which acts to drop or mark non-conformant traffic rather than to delay; a policer can be thought of as a special case of a shaper with a queue with a maximum queue length of zero packets. Hence, while policing acts to cut the peaks off bursty traffic, shaping acts to smooth the traffic profile by delaying the peaks.

It is noted that not all shapers need be implemented with a token bucket mechanism. Another mechanism that is used for shaping is the leaky bucket; leaky buckets and token buckets are often confused but have significant and fundamental differences. With a leaky bucket algorithm, it can be visualized that packets – rather than tokens – are stored in the bucket; arriving packets are placed in a bucket which effectively has a hole in the bottom. The depth of the leaky bucket, B, determines the maximum number of packets that can be queued in the bucket (in effect the same as a queue limit applied to the queue that the bucket represents); if a packet arrives when the bucket is already full, the packet is dropped. Packets drain from the hole in the bucket (i.e. they are transmitted) at a constant rate R, thus smoothing traffic bursts. The best known example of a leaky bucket algorithm is the Generic Cell Rate Algorithm (GCRA) used in traffic shaping of ATM networks [GCRA].

Real-time schedulers, which determine packet dequeue times rather than relative dequeue orders, can also be used to shape traffic streams; such schedulers are non–work-conserving, in that they can be idle (i.e. not sending traffic) even when there is traffic to send, in order to shape the traffic stream. Most practical IP shaping implementations today, however, are based upon token bucket mechanisms. It is noted that while there are standardized definitions of shapers for ATM, and for FR, there are no such standardized definitions for IP.

A shaper can be applied to enforce a maximum rate across all traffic on a physical or logical interface as shown in Figure 2.23.

This could be used to offer a subrate service, for example, where a customer buys a service to provide connectivity to a site from a service provider which defines an aggregate committed rate (i.e. across all classes) for the access connection. The service provider could enforce this service at the edge of their network, either by provisioning the

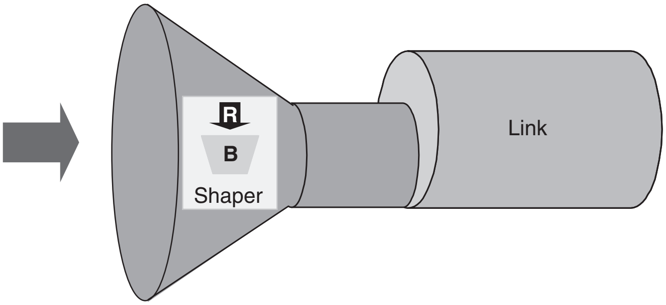


Figure 2.23 Aggregate shaper

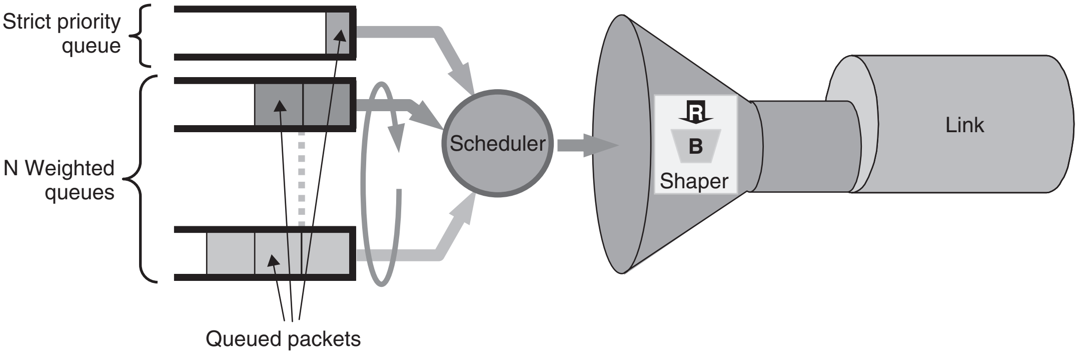


Figure 2.24 Aggregate shaper with class scheduler

access link to the contracted rate, or alternatively by provisioning a higher rate access link and shaping the traffic on the link to the contracted per-customer aggregate rate, which would be sub the line rate; in this way the shaper can acts as an artificial bottleneck, limiting the customer's traffic.

Alternatively, a shaper can be applied to enforce a maximum rate for an individual class of traffic, in which case the delayed traffic will be queuing in the queue for that class. A shaper could also be applied to an aggregate traffic stream which comprises a number of classes; in this case, it is important that a shaper is combined with a scheduler, as shown in Figure 2.24. Such that if the aggregate rate of the traffic exceeds the shaped rate, traffic is delayed in queues per-class, and the scheduler defines the order in which those queues are serviced at the shaped rate, assuring differentiation between different traffic classes.

The example shown in Figure 2.24 is the simplest form of a scheduling/shaping hierarchy; effectively, the shaper and scheduler have a parent/child relationship, where the scheduling policy is a child to the parent shaping policy. Some deployments may require additional levels of shaping and scheduling.

2.2.5 Link Fragmentation and Interleaving

Even with a strict priority scheduler for delay sensitive traffic, such as VoIP, a newly arrived priority packet can at best be enqueued after the last packet to be scheduled. On relatively low-speed links, a single 1500 byte (the maximum transmission unit for Ethernet) non-priority packet scheduled just before a priority packet arrives can have significant impact on the priority packet delay. For a 512 kbps connection this would be ~23 ms which would exceed the 15 ms maximum acceptable access link delay target for a VoIP class derived in the example given in Chapter 3, Section 3.2.2.1.1. In practical implementations, the problem may be worse, with several non-priority packets potentially being queued ahead of a priority packet due to the presence of an interface FIFO (as described in Section 2.2.4.1.3).

Consider, for example, that a particular queuing implementation has been designed such that if a priority queue packet arrives when the priority queue is empty, at most 2 non-priority packets can be serviced before that priority queue packet, i.e. a maximum interface FIFO size of 2 packets, which is representative of practical implementations. If this implementation was used on a 512 kbps link, and assuming non-priority packets of 1500 bytes, then even if a priority packet arrives at the priority queue when it is empty, the packet may be delayed by up to (2 * 1500 * 8/512000) = ~47 ms before the priority packet can even start to be sent out of the interface. This would significantly exceed a typical access link delay budget and consume a significant component of an end-to-end delay budget.

In such cases, link layer fragmentation and interleaving (LFI) mechanisms – such as FRF.12 [FRF12], which is specific to frame-relay or mechanisms which rely on the Multilink Point-to-Point Protocol (MLPPP) [RFC 1990] – are needed to reduce the impact of non-priority packets on priority traffic delay. Link layer fragmentation breaks large non-priority packets into smaller fragments with which priority packets can be interleaved rather than having to wait for whole non-priority packets to be transmitted. Link layer fragmentation therefore reduces the impact of the delay induced by the non-priority packets. The fragments are each transported as unique layer 2 frames, which contain an identifier enabling them to be differentiated from priority packets, and a sequence number enabling the fragments to be reassembled into whole packets at the far end of the link. Typical LFI implementations have a configurable fragment size such that non-priority packets which are greater than the fragment size will be broken up into fragments of at most that size. Consider the previous example: if an LFI technique were used with a fragment size of 300 bytes and a priority queue packet arrives in an empty priority queue it would now be delayed only by 2 * 300 byte fragments in the interface FIFO, i.e. 2 * 300 * 8/512000 = ~9 ms before it could start to be sent out of the interface, and the delay budget would be met. Although IP layer fragmentation could be used to similar effect, it suffers many disadvantages [SHANNON] and is therefore generally not recommended.

Link layer fragmentation and interleaving mechanisms are processor intensive functions and hence they may have a performance impact on software-based router implementations.

2.3 IP QOS Architectures

2.3.1 A Short History of IP Quality of Service

In order to understand the history of IP QOS architectures, we first need to define what an architecture means in this context. QOS architectures define the structures within which we deploy QOS mechanisms to deliver end-to-end QOS assurances or SLAs. To be completely defined, they need to provide the context in which mechanisms such as classification, marking, policing, queuing, and scheduling,

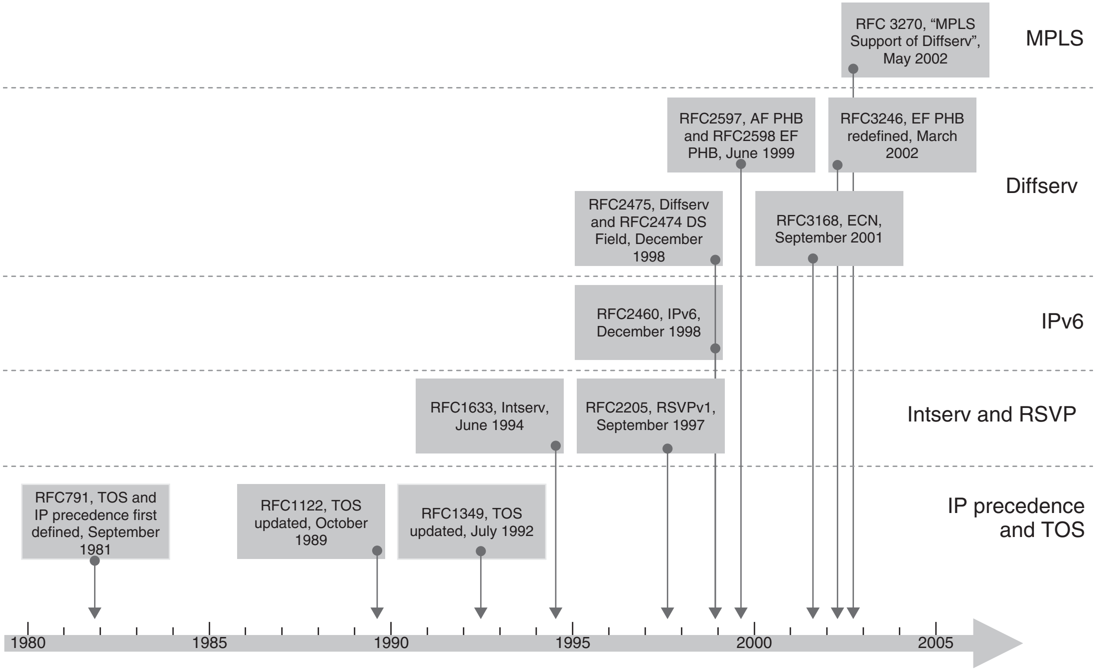


Figure 2.25 IP QOS standards timeline

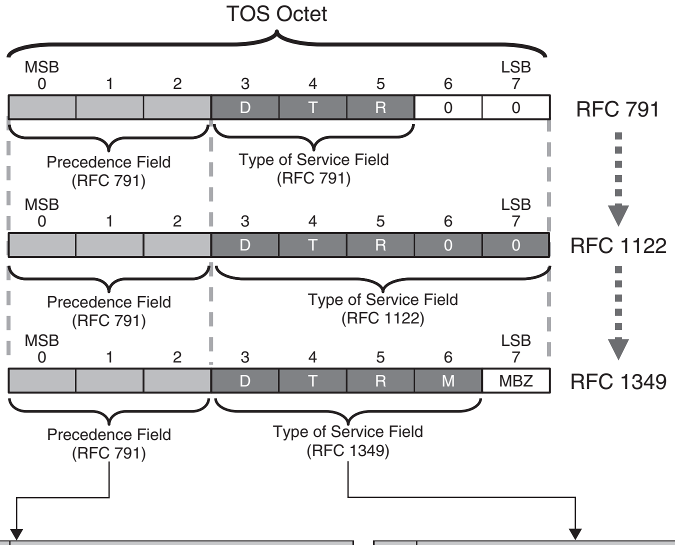
dropping, and shaping are used together to assure a specified SLA for a service.

The standards which define the different architectures for IP QOS have been defined by the Internet Engineering Task Force (IETF, www.ietf.org). Figure 2.25 shows a timeline for the publication of the major IETF milestones in defining the QOS architectures for IP and MPLS.

The following sections describe the evolution of IP QOS architectures from IP Precedence and Type of Service, through Integrated Services and Differentiated Services, and also describe how IP QOS architectures apply to MPLS.

2.3.2 Type of Service/IP Precedence

In 1981, the original IPv4 specification [RFC 791], defined an 8-bit field to be used for IP QOS classification; this was called the Type of Service



Binary	Decimal	Name
111	7	Network Control
110	6	Internet work Control
101	5	Critical and Emergency Call Processing (ECP)
100	4	Flash Override
011	3	Flash
010	2	Immediate
001	1	Priority
000	0	Routine

Bit	Meaning
3	Minimize Delay (D)
4	Maximize Throughput (T)
5	Maximize Reliability (R)
6	Minimize Monetary Cost (M)

Figure 2.26 Type of service octet evolution

octet. The type of service octet subsequently went through several stages of redefinition in [RFC 1122] and [RFC 1349], as shown in Figure 2.26.

Originally, in RFC 791 the type of service octet was subdivided with 3 bits used for the IP precedence field (bits 0–2), and 3 bits (bits 3–5) used for the type of service (TOS) field; bits 6 and 7 of the type of service octet were listed as “Reserved for Future Use,” and are shown set to zero. We note that it is somewhat confusing that the **TOS field** is a subset of the type of service **octet**; we explicitly differentiate between them.

[RFC1122] went on to extend the TOS field to include bits 3–7, although at that time the meaning of the low-order 2 bits (bits 6 and 7) was not defined.

[RFC1349] explicitly defined the meaning of bit 6 of the type of service octet as belonging to the TOS field and being used to indicate a desire to "minimize monetary cost" for packets marked in this way. The use of the low-order bit (bit 7) was redefined as "currently unused" and labeled "MBZ" for "must be zero." RFC 1349 also stated that *"The originator of a datagram sets [the MBZ] field to zero (unless participating in an Internet protocol experiment which makes use of that bit)."*

The specific meanings of the precedence and TOS fields are described in Sections 2.3.2.1 and 2.3.2.2. The definitions of the type of service octet have since been obsoleted by "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers" [RFC2474], which is described in Section 2.3.4.1.

2.3.2.1 IP Precedence

The use of the IP precedence is only defined in RFC 791, which defines the notion of precedence as "*An independent measure of the importance of this datagram*." Retrospectively, it is apparent that IP precedence defined only a relative priority marking scheme, rather than an overarching QOS architecture.

RFC 791 defined a number of traffic denominations – indicating network control traffic, routing traffic, and various levels of privilege – and an associated marking scheme using the Precedence Field in order to be able to identify to which denomination a particular packet belongs. The bits of the Precedence Field have no individual meaning but rather the field value is taken as a whole to determine the "IP precedence" of a particular packet; hence as there are three precedence bits, there are eight different IP precedence values. The IP precedence values and their corresponding denominations are shown in the table in Figure 2.26; the notation normally used when referring to particular IP precedence values is either to use the decimal values or to use the denomination names shown. Appendix 2.A provides a guide to conversion between precedence, TOS, and DSCP values.

IP precedence provided a capability for marking packets, such that simple classification could be used at later node to determine what scheduling treatment, i.e. by which queue in the scheduler, the packet should be serviced. As such, it allowed packets with different markings to receive different treatments; however, there was no definition either in relative or absolute terms of how traffic with different IP precedence values should be treated with respect to delay, jitter, loss, throughput, or availability. There was no definition, for example, of how a packet marked precedence 3 ("Flash") should be treated compared to a packet marked precedence 2 ("Immediate"). RFC 791 recognizes this: "*Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic.*"

Hence, IP precedence did not define the architectural framework of capabilities needed to support services with defined SLA requirements and consequently it did not achieve widespread deployment. Nonetheless, the use of some IP precedence markings have become de facto – e.g. most router vendors today mark routing protocol traffic IP precedence 6 by default. Even though the use of IP precedence has been superseded by the DS field, this de facto marking does not provide any issues with respect to backward compatibility, as Diffserv provides a backward compatibility mode through the use of the class selector code points (see Section 2.3.4.1).

2.3.2.2 Type of Service

The definition of the type of service field evolved through RFC 791, RFC 1122, and RFC 1349. RFC 1349 provides the most recent and comprehensive definition, hence we refer to that definition in this section.

RFC 1349 defined a scheme using the 4-bit TOS field (bits 3–6 of the type of service octet) to indicate in each packet the service that it required from the network. Unlike the precedence field, where individual bits do not have a specific meaning, each bit of the TOS field was set in a packet if that packet required the service represented by that bit as shown in Figure 2.26; all bits set to zero indicates that a packet requires normal service.

As defined in RFC 1349, TOS field marking was not intended to determine which queue a particular packet would be queued in at a network node – that was the function of the IP precedence field – but rather the TOS marking of a packet was to be used to determine which path that packet would take through the network. Specifications for some routing protocols provided support for TOS routing, where a separate (and possibly distinct) set of routes could be calculated for each IP TOS value, such that an IP packet could be routed based upon both the packet's destination IP address and its TOS field value.

While we have already differentiated between the type of service octet and the TOS field, the terms "type of service" or "TOS," when used on their own, are generally referring to the use of the TOS field for TOS routing. The notation generally used when referring to TOS is to take the entire value of the type of service octet (including the precedence field and bit 7 of the type of service octet) expressed in decimal, where bit 0 is taken as the most significant bit; this is generally called the "TOS value." For example, assuming a precedence field value of 101 binary, and a TOS field value of 1000 binary, the TOS value would generally be referred to as "176" decimal (i.e. 10110000 binary). This notation can sometimes cause confusion, because it consumes the IP precedence value; hence even though a packet may have a TOS field value of 0000 binary (i.e. normal service), if the packet has a precedence field value of 101 binary, the TOS value would generally be referred to as "160" decimal (10100000). Appendix 2.A provides a guide to conversion between precedence, TOS, and DSCP values.

With type of service, the markings defined were subjective; it was not defined, for example, how a packet marked with "minimize delay" should be treated relative to one marked without; RFC 1349 goes as far as to say "*setting the TOS field to 1000 (minimize delay) does not guarantee that the path taken by the datagram will have a delay that the user considers 'low'*." Similarly, with IP precedence, type of service did not provide the facilities needed to support services with defined SLA requirements and hence type of service was never widely implemented or deployed. At one time, the Open Shortest Path First (OSPF) [RFC 1583] interior gateway routing protocol specification supported the capability to calculate separate routing topologies for each type of service; however, this was never widely implemented and was subsequently removed from the OSPF specification [RFC 2178].

The limitations in IP precedence and type of service were realized and this led to the definition of the Integrated Services and Differentiated Services QOS architectures, which resulted in the obsoletion of the TOS field by the Differentiated Services field.

2.3.2.3 IPv6 Traffic Class Octet

The IPv6 Traffic Class Octet existed only in theory and for a short period of time, hence it is not worthy of lengthy discussion. The IPv6 Traffic Class was defined in [RFC2460] in December 1998, and then was redefined in the same month by the definition of Differentiated Services (DS) field, which is specified in [RFC2474] (see Section 2.3.4.1).

2.3.3 Integrated Services Architecture

[RFC1633] laid out the philosophy of the Integrated Services or "Intserv" IP QOS architecture. It was designed to address the issues identified with IP precedence and type of service, providing the capabilities needed to support applications with bounded SLA requirements, such as VoIP and video. Intserv tackles the problem of providing services level assurances to applications by explicitly managing bandwidth resources and schedulers on a per flow basis; resources are reserved and admission control is performed for each flow. The resource ReSerVation Protocol (RSVP) is the end-to-end signaling protocol used to setup Intserv reservations. Intserv and RSVP are described in more detail in Chapter 4, Section 4.4.

2.3.4 Differentiated Services Architecture

Scalability concerns with Intserv lead to the definition of the Differentiated Services (DS) or "Diffserv" IP QOS Architecture [RFC 2475]. Diffserv comprises the following key components, which are used together to enable end-to-end differentiated delay, jitter, and loss commitments to be supported on the same Diffserv-enabled IP network – referred to as a *Diffserv domain* – for different types or classes of service. These components are shown in Figure 2.27.

- *Traffic classification and conditioning*. The edge of the Diffserv domain is the provider/customer boundary for the Diffserv-enabled services being offered. This does not, however, infer that Diffserv is only

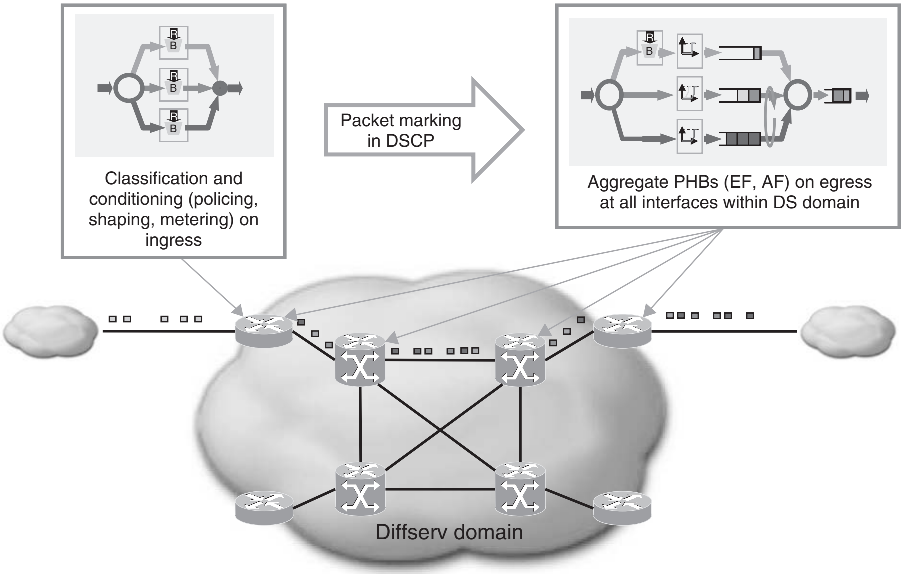

B
B
B
Classification and conditioning (policing, shaping, metering) on ingress
Packet marking in DSCP
B
Aggregate PHBs (EF, AF) on egress at all interfaces within DS domain
Diffserv domain

Figure 2.27 Diffserv architecture – RFC 2475

applicable to network service providers; the networking department of an enterprise organization is a service provider to their enterprise. On ingress to the Diffserv domain, the customer's traffic is classified using implicit, simple, or complex classification into a limited number of traffic classes, which are also known as "behavior aggregates" in Diffserv-speak. These aggregates are checked for conformance against agreed profiles – referred to in Diffserv as Traffic Conditioning Agreements (TCAs). QOS mechanisms, such as shaping or policing, are used to "condition" the traffic to ensure that traffic ingressing the Diffserv domain is conformant with the TCA; non-conformant traffic may be delayed, dropped, or re-marked. The TCA is derived from the SLA between the provider and the customer and defines the characteristics of the offered traffic for which the SLA is assured. For example, if a site is provided with a 128 kbps assured VoIP service, the TCA might be to police the received VoIP traffic from that site to 128 kbps (with an appropriate burst), dropping any excess traffic.

- *DSCP marking*. Packets either are pre-marked using the Diffserv Code Point (DSCP) within the DS field in the IP packet header, or are marked on ingress to the Diffserv domain, in order to identify to which particular class or behavior aggregate the packets belong. The marking could be performed by a policer enforcing the TCA, for example. Subsequent Diffserv nodes therefore only need to perform simple classification using the DSCP in order to determine the class of a packet. The DS field and DSCP are described in Section 2.3.4.1.

- *Per-hop behaviors*. The conditioning applied at the edges of the network ensures that all traffic ingressing the Diffserv domain is within the committed TCAs and is appropriately marked. Within the Diffserv domain, the objective is then simply to ensure that the per-class SLAs are met, for the limited number of classes supported. Per-class scheduling and queuing control mechanisms are applied to the traffic classes based upon the DSCP marking in order to ensure per-class SLA differentiation. Diffserv is not prescriptive in defining the scheduling and queuing control algorithms that should be implemented at each hop, but rather, uses a level of abstraction in defining the externally observable forwarding behaviors – termed Per-Hop Behaviors (PHBs) – that can be applied to traffic at each hop. Diffserv PHBs are described in Section 2.3.4.2.

Unlike Intserv, Diffserv configurations are provisioned, either by manual configuration or by an NMS system, rather than being set up by a network signaling protocol.

End-to-end SLAs are assured with Diffserv by ensuring that per-class resources are appropriately provisioned at each hop relative to the traffic load within the class. Per-class traffic loads within the Diffserv domain will change over time and hence performing per-class capacity planning is an essential component of Diffserv to ensure that the per-class resources are appropriately provisioned. Capacity planning is discussed in Chapter 5.

Diffserv achieves scalability by performing complex per-customer QOS functions and maintaining per-customer state (e.g. complex classification criteria), only at the edges of the network. Distributing these functions at the edge of the network facilitates scaling, as the edge of the network naturally expands as the network grows. Unlike Intserv, within the Diffserv domain, there is no per flow or per-customer state, but rather scalability is achieved through using only a limited number of classes, which are simply classified using the DSCP marking within each packet, hence only per-class state is required.

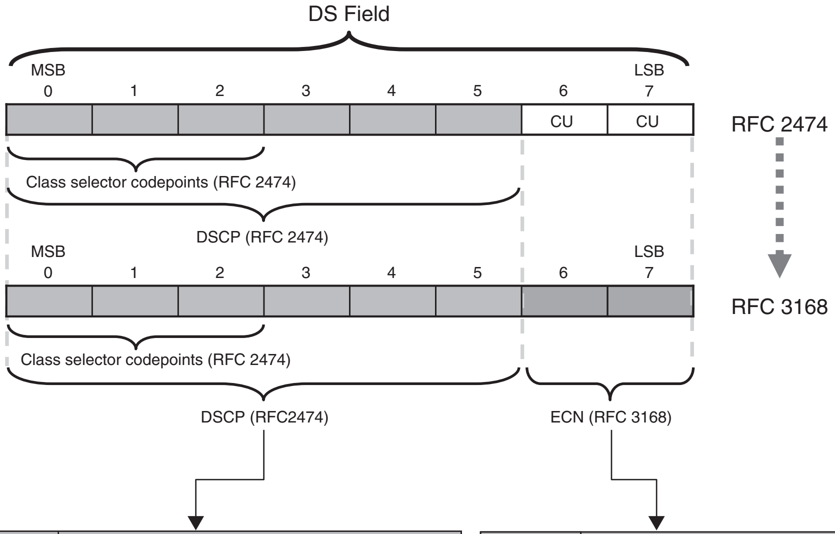
Diffserv can be applied equally to IPv6 as to IPv4. Diffserv can also be applied to MPLS, although the limited size of the field available for QOS marking in MPLS introduces some complexities, which are discussed in Section 3.6.2.

Diffserv is by far the most widely deployed IP QOS architecture; it is widely deployed in enterprise networks and in SP networks providing virtual private network (VPN) services to enterprises. Diffserv is also being deployed to support the move toward so-called "Next Generation Networks" (NGN), which support the migration of PSTN telephony services to IP/MPLS networks. In NGN networks, IP-based PSTN-replacement services coexist on the same network with "best-effort" Internet access services and business oriented VPN services; Diffserv is used to ensure that the SLA requirements for each service are met and that there is isolation between the behaviors of the different services.

SLAs based upon Diffserv are not generally committed for Internet access services, i.e. services to the wider Internet, because services accessing the Internet may transit a number of different service provider networks, and unless they all provide aligned Diffserv capabilities, there would be no benefit in a single provider supporting Diffserv for such services. Further, as Internet access services are at the commodity end of IP service offerings, there is no incentive for service providers to incur the cost and complexity of providing Diffserv-assured Internet access services to their customers.

2.3.4.1 DS Field

The Differentiated Services (DS) field, which is specified in [RFC 2474], redefined the use of the 8-bit field, which had been the type of service octet in IPv4 [RFC1349] and the traffic class octet in IPv6 [RFC 2460]. The DS field definition specified that the first six bits of the



Codepoint	DSCP		
Default/CS0	000000		
EF PHB	101110		
CS1	001000		
CS2	010000		
CS3	011000		
CS4	100000		
CS5	101000		
CS6	110000		
CS7	111000		
AF PHB Group	**Drop Precedence**		
AF Class	**Low (AFx1)**	**Medium (AFx2)**	**High (AFx3)**
AF1x	AF11 = 001001	AF12 = 001010	AF13 = 001011
AF2x	AF21 = 010001	AF22 = 010010	AF23 = 010011
AF3x	AF31 = 011001	AF32 = 011010	AF33 = 011011
AF4x	AF41 = 100001	AF42 = 100010	AF43 = 100011

ECN Field		Meaning
0	0	Not ECT
0	1	ECT(0): not defined in [RFC 2481]
1	0	ECT(1)
1	1	CE

Figure 2.28 DS field format

field (bits 0–5) were designated as the Differentiated Services code point (DSCP). The DSCP field is unstructured and the value of the field is taken as a whole, i.e. there is no distinct meaning for each specific bit. A particular combination of DSCP bits is referred to as a "codepoint," which is set such that it can be used to select the PHB a packet will experience at each node. The DS Field is shown in Figure 2.28.

Codepoints can be expressed in binary or decimal notation. In binary format the notation “xxxxxx” is commonly used, where “x” may be equal to “0” or “1” and the left-most bit signifies bit 0 of the DSCP. In decimal notation all 6 bits of DSCP are expressed in decimal, where bit 0 is taken as the most significant bit; for example, a DSCP of 010000 binary is represented as DSCP 16. Appendix 2.A provides a guide to conversion between precedence, TOS, and DSCP values.

The DSCP is a 6-bit field and therefore it can be used to indicate 64 distinct codepoints. RFC 2474 partitioned this codepoint space into three pools for the purpose of codepoint assignment and management:

- *Pool 1 – standards action*. The first pool consists of the 32 codepoints in the range xxxxx0 were defined to be assigned to standardized PHBs defined in the IETF.
- *Pool 2 – experimental or local use*. The 16 codepoints in the range xxxx11 were reserved for experimental or local use (EXP/LU).
- *Pool 3 – experimental or local use*. The 16 codepoints in the range xxxx01 were also reserved for experimental or local use (EXP/LU). The difference between Pool 3 and Pool 2, however, is that the use of this field may be subsequently redefined for standardized assignments if Pool 1 is ever exhausted.

The DSCP codepoint space assignment is summarized in the table in Figure 2.29.

Some, although not all, of the DSCP values in the Pool 1 codepoint space have been assigned to particular PHBs standardized within the IETF:

- DSCP 0 (000000 binary) has been assigned to the default PHB, which is discussed in Section 2.3.4.2.3.
- The expedited forwarding (EF) PHB, which is discussed in Section 2.3.4.2.1, has been assigned DSCP 46 (101110 binary).
- A set of 12 codepoints, which are shown in Figure 2.28, have been allocated to the AF PHB group, which is discussed in Section 2.3.4.2.2.

Pool	Codepoint Space	Assignment Policy
1	xxxxx0	Standards Action
2	xxxx11	EXP/LU
3	xxxx01	EXP/LU

Figure 2.29 DSCP codepoint space assignment

- A set of codepoints defined as the "Class Selector" (CS) codepoints are any of the eight codepoints in the range "xxx000," where "x" may equal "0" or "1." The notation commonly used for the CS codepoints is "CS" followed by the value of the first three bits of the DSCP expressed in decimal, where bit 0 is taken as the most significant bit, e.g. codepoint 101000 is expressed as CS5.

  The CS codepoints only use the first three bits of the DSCP, which are the bits that were previously defined for the IP precedence field (as described in Section 2.3.2.1), hence use of the CS codepoints provides backward compatibility with IP precedence, i.e. marking or classifying a packet as CS5 is, to all intents and purposes the same as marking or classifying a packet as IP precedence 5 (assuming the TOS field is set to 0). The CS codepoints have an associated PHB definition, which is described in Section 2.3.4.2.4.

As well as being specified in the documents which define a particular PHB, there is a central DS Field Codepoints Registry [DSCR] maintained by the Internet Assigned Numbers Authority (IANA).

Take note that the allocation of Pool 1 codepoints specified above, for standardized PHBs defined in the IETF, are recommended rather than being mandated. This means that while it may make sense to use these values, if there are valid reasons to use DSCP values other than those recommended, then it is up to the network designer's discretion to do so. Hence, contrary to popular opinion, using values other than the recommended values does not mean that a particular IP QOS design is not "Diffserv compliant." Further, while RFC 2474 says, "*Recommended codepoints SHOULD map to specific, standardized PHBs,*" it also says, "*the mapping of codepoints to PHBs MUST* *be configurable*" in order to allow for alternative configurations, where codepoints other than the recommended codepoints are used.

It is further noted that all packets with the same DSCP should be treated with the same PHB. If packets with the same DSCP were treated with different PHBs, they may for example be placed in different queues, with the result that packets from the same flow may be resequenced. Hence, it is essential to treat all packets with the same DSCP with the same PHB in order to prevent resequencing within a flow due to the adverse impact that packet re-ordering can have on the performance of some applications (this is discussed in more detail in Chapter 1, Section 1.2.5).

Multiple codepoints, however, may be mapped to the same PHB. As a consequence, even though the CS codepoints are defined as those codepoints within the range xxx000, in order to be backward compatible with IP precedence, an alternative approach to be backward compatible is to ignore the markings in bits 3–5 of the DSCP, and classify packets based only on bits 0–2. If this approach were taken, DSCP values of 101000 and 101001 would both be mapped to the same PHB.

RFC 2474 originally defined bits 6 and 7 of the DS field as being reserved and annotated them "Currently Unused" (CU) as shown in Figure 2.28. The use of the field was subsequently redefined in [RFC 3168] for use with Explicit Congestion Notification (ECN), as described in Section 2.3.4.4.

2.3.4.2 Per-Hop Behaviors

Diffserv is not prescriptive in defining the scheduling and queuing control algorithms that should be implemented at each hop, but rather, uses a level of abstraction in defining the externally observable "black box" forwarding behaviors, termed Per-Hop Behaviors (PHBs), that are applied to traffic at each hop.

Four types of PHBs are defined which are described in the following sections. The PHBs are formally defined, such that an implementation which complies with a particular PHB is assured to support the behavioral characteristics intended by that PHB. Each PHB definition consists of two components: a formal definition of the required forwarding behavior and a recommended marking scheme to be used for classifying packets that will be subjected to that PHB.

2.3.4.2.1 The Expedited Forwarding (EF) PHB

The EF PHB [RFC 3246][2] is used to support applications with low-delay, low-jitter, low-loss, assured bandwidth requirements, such as VoIP. The characteristics of an implementation which supports the requirements of the EF PHB are that it is able to service the EF traffic at a specified rate or higher, measurable over a defined time interval and independent of the offered load of any non-EF traffic at the point where the EF PHB is applied. If these characteristics are supported and the EF class traffic rate and burst characteristics are controlled, using a token bucket policer as described in Section 2.2.3 for example, then the delay and jitter for the EF traffic can be bounded. Further, if the available buffer space for the EF class traffic is greater than the burst characteristic, then the loss can also be controlled (i.e. will be zero).

Some scheduler implementations may attempt to support EF traffic using a scheduling algorithm such as WRR or WFQ; however, with such implementations the worst-case delay bounds for the EF traffic will depend upon the particular scheduling algorithm used and may also be dependent upon the number of queues used in the particular scheduler implementation as described in Section 2.2.4.1.2.4. Consequently, the EF PHB is typically implemented using a strict priority queuing mechanism, such as that described in Section 2.2.4.1.1. With implementations that support multiple priority queues, typically they all support the EF PHB as described in Section 2.2.4.1.4.

RFC 3246 also specifies that if the EF PHB is implemented using a scheduler that allows the EF traffic to pre-empt other traffic (e.g. a strict priority queue), then there must also be some mechanism (e.g. a token bucket policer) supported to limit the EF traffic in order to constrain the impact it can have on the other traffic, i.e. to prevent the other traffic from being starved.

Hence, the application of a policer to an EF traffic stream serves two purposes: firstly, to limit the EF traffic load such that when servicing the traffic with an EF PHB, the delay, jitter and loss can be assured; secondly, to limit the impact that the EF traffic can have on non-EF traffic.

The EF PBH is assigned a recommended DSCP of 101110 binary, 46 decimal.

2.3.4.2.2 The Assured Forwarding (AF) PHB

The Assured Forwarding (AF) PHB group [RFC 2597] defines a set of AF classes, which are designed to support data applications with assured bandwidth requirements, such as absolute or relative minimum bandwidth guarantees, with a work-conserving property.

The key concept behind the AF PHB group definition is that a particular class could be used by a DS domain to offer a service, to a particular site for example, with an assurance that IP packets within that class will be forwarded with a high probability as long as the class rate from the site does not exceed a defined contracted rate. If the rate is exceeded, then the excess traffic may be forwarded, but with a probability that may be lower than for traffic which was below the contracted rate.

There are four defined AF classes denoted as AF1x, AF2x, AF3x, and AF4x. Within each class a packet can be assigned to one of three levels of drop precedence, for example AF11, AF12, and AF13 within class AF1x. Within a particular class, the probability of forwarding AFx1 must not be less than for AFx2, which in turn must not be less than for AFx3, i.e. AFx1 has a low drop precedence, AFx2 has a medium drop precedence, and AFx3 has a high drop precedence. Within a class, the drop precedence therefore indicates the relative importance of the packet. A set of twelve recommended DSCP values have been allocated to indicate the four classes and three drop precedence levels within each class, as shown in Figure 2.28. Although only four AF classes are defined, in theory there is nothing, apart from the size of the DSCP field, to limit the number of classes serviced with an AF forwarding behavior. If more than four AF classes are required then as the recommended DSCP markings are only defined for four classes, non-recommended DSCP values need to be used for the additional AF classes.

A particular AF class is realized by combining conditioning behaviors on ingress to the Diffserv domain – where a particular AF class is offered to a customer – which control the amount of traffic accepted at each level of drop precedence within that class and marks the traffic accordingly. At that and subsequent nodes, the AF class bandwidth is allocated to ensure that traffic within the contracted rate is delivered with a high probability. If congestion within the class is experienced, the congested node aims to ensure that packets of a higher drop precedence are dropped with a higher probability than packets with a lower drop precedence. Hence, at a DS node the forwarding assurance of a particular packet depends upon the forwarding resources allocated to the class, the current offered load for that class, and if congestion occurs within the class, the packet's drop precedence within the class.

The edge conditioning behaviors for an AF class will typically be implemented using a one rate or two rate policer as described in Section 2.2.3 respectively. Although both of these policers are capable of marking "3 colors" which can correspond to 3 levels of drop precedence, in practice it is difficult to understand what meaningful service benefit is offered by differentiating between traffic in terms of a 3 color scheme, i.e. "in-contract," "out-of-contract," and "exceedingly-out-of-contract." Hence, it is more common for the policers to be used to mark 2 colors only ("in-contract" and "out-of-contract"), and hence typically use only 2 drop precedence levels, e.g. AFx1 and AFx2.

An AF PHB class is typically allocated to a queue which is serviced from a weighted scheduling mechanism such as WFQ (Section 2.2.4.1.2.2) or DRR (Section 2.2.4.1.2.3), where the weighting for the queue and queue depths are configured to ensure that low drop precedence packets within the class are forwarded with a high probability. If congestion occurs within the class then WRED is commonly used in order to drop, for example, AFx2 traffic with a higher probability than AFx1 traffic; this is done by having a more aggressive RED drop profile for the AFx2 traffic as described in Section 2.2.4.2.4.

As for all of the other defined Diffserv PHBs, it is a requirement that in applying an AF PHB there is no possibility that packets from a single flow will be resequenced, due to the impact that this can have on application performance. Hence packets from the same flow should always be assigned to the same AF PHB, although they may be assigned different drop precedences, such that they will always be serviced from the same queue and hence will always be in sequence, even though they may have different drop probabilities.

2.3.4.2.3 The Default PHB

The default PHB [RFC 2474] is defined as being the PHB used for packets not explicitly mapped to other PHBs. The default PHB is somewhat ambiguously defined as a PHB which has no committed resources and yet cannot be starved by other PHBs, but potentially can re-use unused bandwidth from other classes when available, i.e. has an implied work conserving property. RFC 2474 also says that the default PHB could be supported *"by a mechanism in each node that reserves some minimal resources (e.g., buffers, bandwidth) for default behavior aggregates*." Hence, in practice, the difference between the service provided to an AF PHB, which has a minimal but quantifiable bandwidth assurance and the default PHB is semantic. Hence, to avoid confusion we choose not to use the default PHB; if a class requires only a minimal bandwidth assurance, we consider it as serviced with an AF PHB, which has a minimal but quantifiable bandwidth assurance.

There can be confusion between the default PHB and the concept of a default class for classification purposes. Most router implementations have the concept of a default class to which all packets that are not explicitly classified into other classes are assigned; this default class serves to simplify QOS configuration. The default class will be assigned to a queue and the bandwidth assurances to this queue will generally be configurable; RED will also generally be able to be configured on this queue in order to optimize throughput for TCP. For example, consider a case where traffic consisting of 5 distinct DSCP markings is being classified into 3 separate classes, each being serviced with AF PHBs. If discrete DSCP markings are individually mapped to each of two classes, then the remaining DSCP could be explicitly mapped to the third class, or alternatively if the concept of a default class is supported, they could implicitly be mapped to the default class without requiring explicit configuration.

2.3.4.2.4 Class Selector PHB

[RFC 2474] defines a set of PHB requirements associated with the Class Selector codepoints (Section 2.3.4.1). The intent of the CS PHB requirements is to define a PHB group that could replace (and hence provide backward compatibility with) the behaviors applied to packets based upon their IP precedence marking. In doing so, the CS PHB requirements assume that packets with a numerically higher IP precedence value were treated with a higher probability of forwarding (i.e. lower probability of drop) than packets with numerically lower precedence values. Therefore, RFC 2474 specifies that packets with a higher numerical CS codepoint value must not have a lower probability of timely forwarding than packets with a lower CS codepoint value. The definition for the CS PHB requires a minimum of two classes servicing the eight CS codepoint values; hence, in the minimal case multiple CS codepoint values may need to be mapped to a single CS PHB. Where multiple PHBs are used in this way, they are referred to as a CS compliant PHB group.

In practice, as IP precedence was not really used in a consistent way (as discussed in Section 2.3.2.1), there has been little need to deploy the forwarding behaviors specified by the CS PHB requirements. If this were required, this could effectively be achieved by classifying them into classes based upon their CS codepoint values and configuring the AF classes with the appropriate bandwidth resources relative to class load, to achieve the relative differentiation required of a CS PHB group.

A more common practical reason for using the CS codepoint makings is not to facilitate backward compatibility with IP precedence, but instead to ease the process of mapping IP packet markings to the MPLS experimental field (as described in Section 2.3.6.2.1). In cases such as this, the EF or AF PHBs may be applied to classes where traffic is classified into those classes based upon CS codepoint classification.

2.3.4.3 Per-Domain Behaviors

[RFC 3086] defines Diffserv “Per-Domain Behaviors” (PDBs); PDBs are intended to define particular end-to-end behaviors delivered by a Diffserv domain. PHBs can be considered to be the externally observable “black box” forwarding behaviors experienced at a particular hop in the Diffserv domain, and similarly a PDB can be considered to be a definition of the “black box” forwarding behaviors experienced by a class of packets across the Diffserv domain as a whole. As such, a PDB can be considered to be the Diffserv definition of the end-to-end engineering SLAs described in Chapter 1, Section 1.4.1.

Only a single PDB has been defined and that is the lower-effort PDB.

2.4.4.3.1 Lower effort PDB

[RFC3662] is an informational RFC which defines *"A Lower Effort (LE) Per-Domain Behavior (PDB)."* The service provided by the LE PDB can be characterized as one where all other traffic takes precedence over LE traffic in consumption of network link bandwidth, but the traffic supported by the LE PHB is able to use unused bandwidth from other classes when available, i.e. it has a work conserving behavior. Hence, if any congestion is experienced in the Diffserv domain, the service provided by the LE PDB may be completely starved; that is, the other classes can consume all of the available bandwidth such that the LE PDB will get nothing. This is different to the end-to-end service that would be provided by using the Default PHB, as the definition of the Default PHB (Section 2.3.4.2.3) explicitly specifies that it should not be starved. The LE PDB and the terms "scavenger class" and "lower than best-effort" are synonymous.

RFC 3662 says of the LE PDB: *"This behavior could be obtained, for example, by using a [class-based queuing] scheduler with a small share and with borrowing permitted."* Hence, in practice, there is little difference between the LE PDB, and the service provided by the Default PHB. In turn, as discussed in Section 2.3.4.2.3, there is no significant difference between the service provided to an AF PHB, which has a minimal but quantifiable bandwidth assurance, and the default PHB. Hence, as per the default PHB, we choose not to use the LE PDB; rather, if the required bandwidth assurance for a class is negligible, we consider it as serviced with an AF PHB, which has a minimal bandwidth assurance.

2.3.4.4 Explicit Congestion Notification

As described in Chapter 1, Section 1.3.3.1, TCP effectively treats the network as a "black box," in that it does not rely on any explicit network behaviors when performing flow control, in order to determine the status of available network bandwidth and whether congestion has occurred. Instead, TCP relies on TCP timeouts or the reception of duplicate ACKs to determine implicitly when packets are dropped. AQM mechanisms such as RED are used to detect congestion before the queues overflow (i.e. before tail drop), and selectively drop packets to feedback indication of this congestion to the end-systems so that they will reduce their rate of sending with the aim of avoiding excess packet loss due to congestion and maintaining high network throughput while minimizing queuing delays.

Explicit congestion notification (ECN) aims to further improve throughput for TCP (and potentially for other transport protocols) and reduce queuing delays by adding the capability for the network to explicitly indicate to end-systems when congestion has occurred; support for explicit congestion notification (ECN) was added to Diffserv in [RFC 3168]. The main concept underlying ECN is that, rather than using AQM mechanisms like RED to drop packets when congestion is experienced, they are instead used to explicitly mark packets; the end-system TCP stacks would then use the packet marking to determine when congestion has occurred and hence to slow their rate of sending. ECN relies on the proactive indication of congestion before packets are actually dropped, rather than reacting to packet loss as with non-ECN TCP stacks, hence ECN reduces packet loss and improves overall throughput.

This explicit indication is provided by marking the ECN field, which RFC 3168 defined as bits 6 and 7 of the DS field, which were previously undefined. The ECN field is set both by end-systems, to indicate that they are using an ECN capable transport (ECT) layer protocol, and by routers, to indicate explicitly when congestion is experienced (CE). The possible markings (referred to as codepoints) of the ECN field are given in Figure 2.28.

From Figure 2.28 it can be seen that there are two values of the ECN field which indicate ECT; either can be used by end-systems and routers should treat both values as equivalent. Thus although the ECN field has four possible values, effectively it defines only three states. The reason for having two values to indicate ECT is largely a legacy from the first experimental definition of ECN in [RFC 2481]. RFC 2481 defined that the first bit of the ECN field be used for ECT and the second bit for CE, therefore the 01 codepoint was not undefined; this was changed as per Figure 2.28 by RFC 3168, which obsoleted RFC 2481.

ECN requires the following behaviors in ECN capable end-systems and routers:

- Before using ECN, the transport protocol might employ negotiation between the end-systems to determine that they are both ECN capable. In the case of TCP, this is done during session establishment using two new flags in the TCP header, the ECN-Echo (ECE) and Congestion Window Reduced (CWR) flags, which are defined in RFC 3168 and are shown in Figure 2.30.

  Negotiation of the use of ECN between two TCP end-systems, A and B, where A is the initiator, requires that when A sends the TCP SYN to B it sets both the ECE and CWR flags to indicate that it is ECN capable. If B is also ECN capable, it responds with a SYN-ACK with the ECE flag set and the CWR flag unset.

  With ECN negotiation complete, both A and B can originate packets on this TCP session with ECT set, indicating that they both support ECN.

- A router which receives an ECT packet uses a mechanism such as RED (see Section 2.2.4.2.3) to determine whether or not to set CE; the modified RED behavior to support ECN is as follows:
  - When there is no congestion, i.e. if the current average queue depth (q_{avg}) is below the configurable minimum threshold (q_{minth}), the packet is enqueued. ECT packets received with CE already set are left unchanged and the packet is enqueued as normal.

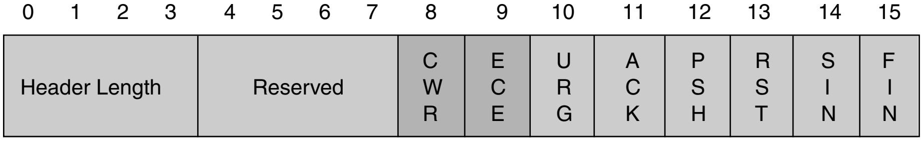


Figure 2.30 TCP header updated with flags ECN

  - ○ If there is moderate congestion, i.e. the current average queue depth (q_{avg}) is above q_{minth} and below q_{maxth}, the packet will be marked CE (instead of being dropped as would be the case for a non-ECT packet) with an increasing, but randomized, probability, using the formula defined in Section 2.2.4.2.3.
  - ○ If there is extreme congestion, i.e. the current average queue (q_{avg}) depth is above the defined maximum threshold (q_{maxth}) then the ECT packet will always be dropped, as for non-ECT packets.

- Upon receiving a packet with CE set the receiver sets the ECN-Echo flag in its next TCP ACK sent to the sender.

- Upon receiving a TCP ACK with the ECE flag set, the sender applies the same congestion control algorithms as would be applied by a non-ECT end-system in the presence of a single dropped packet (see Chapter 1, Section 1.3.3.1). The sender also sets the CWR flag in the TCP header of the next packet sent to the receiver to acknowledge its receipt of and reaction to the ECN-Echo flag.

Even though ECN was designed to be incrementally deployable, it has not been widely deployed. End-users will not get any benefit from ECN until it is supported both in the TCP stacks of their end-systems and by the routers in the networks they use, and one reason often cited for the lack of deployment of ECN is a chicken and egg problem with respect to the availability of ECN capable implementations:

- Support for ECN by router vendors will inevitably require development efforts and they are unlikely to undertake this development unless they have requests for support from their enterprise or SP customers.

- Enterprise or SP customers are unlikely to ask their router vendors to add support for ECN until it is supported in end-systems' TCP stacks.

- There is no incentive for the vendors of TCP stacks to develop support for ECN, if there is no support by router vendors.

Another oft-cited reason for the lack of deployment of ECN has been concerns over the potential for subversion of the use of ECN capabilities, where packets are falsely indicating ECN capability, for example. If the falsely ECT marked packets encounter moderate congestion at an ECN capable router, the router may set the CE codepoint instead of dropping the packet. If the transport protocol in fact is not ECN-capable or is not adhering to the defined ECN behaviors, then the transport protocol may not reduce its rate of sending, as intended by ECN. The consequences of this action are two-fold:

- The end-systems that are falsely claiming to be ECN-capable receive a lower probability of packet loss when moderate congestion is experienced than others which are correctly indicating that they are not ECN-capable.
- If the end-systems that are falsely claiming to be ECN-capable do not reduce their rate of sending when they should due to CE marking, the level of congestion may increase, thereby increasing the rate of packet marking or dropping impacting all flows.

Hence, the benefits of ECN are only really gained when all end-systems cooperate in adhering to the ECN behaviors; if this cannot be assured the benefits of ECN also cannot be assured. Similar reasons are often cited for the lack of deployment of equivalent layer 2 mechanisms used for adaptive shaping such as forward explicit congestion notification (FECN) in Frame Relay and Explicit Forward Congestion Indication (EFCI) in ATM.

In comparing ECN in IP to FECN in frame-relay or EFCI in ATM, other than the obvious difference that ECN is applied at layer 3, the main difference is that frame-relay and ATM implementations generally use a notion of the actual queue depths at a transit node to determine when to set FECN/EFCI, whereas ECN relies on AQM mechanisms such as RED, which set ECN based upon a measure of the average queue depth at a transit node in order to try and achieve a more stable network-wide behavior. Notionally, assuming that frame-relay and ATM networks supported a REDlike capability, a router which received frames or cells with FECN/EFCI set, could also use this as an indication to set CE in the corresponding IP packet. Where ATM and frame-relay networks are used as links in an end-to-end IP network, this would facilitate the mapping between IP QOS and the layer 2 QOS provided by the underlying ATM and frame-relay networks. This is, however, somewhat of a moot discussion because like ECN, neither FECN nor EFCI have been widely deployed in frame-relay or ATM.

There are some proposals to re-use the capability to mark the ECN bits for admission control; this is discussed in more detail in Chapter 4, Section 4.6. [RFC4774] defines considerations on the re-use of the ECN field.

2.3.4.5 Diffserv Tunneling Models

There are a number of ways to "tunnel" IP traffic within IP traffic, where tunneling involves encapsulating a received IP packet within another IP packet header at the tunnel source, such that packets within the tunnel have two headers (actually at least two headers – as it is possible to have tunnels within tunnels). Such tunneling is commonly used to create a virtual or simulated physical connection between two networks across an intermediate network. Packets which are within the tunnel, i.e. between the tunnel source and tunnel destination, are routed using only the outer packet header. At the tunnel destination the outer IP header is stripped off, revealing the underlying IP packet (and possibly layer 2 headers), which is then forwarded as normal. There are a number of IP tunneling techniques used including:

- simple IP-in-IP tunnels such as [RFC 2003] and GRE [RFC 2784]
- multi-protocol tunnels, such as IP in PPP [RFC 1661] in L2TP [RFC 2661]
- secure tunneling techniques such as IPSec [RFC 2401].

Whichever particular tunneling technique is used, when used with Diffserv, if a tunnel is not used end-to-end (i.e. from traffic source to traffic destination) then as the DSCP of the underlying "tunneled"

IP packet is not visible to nodes on the tunnel path, consideration needs to be given to how the DSCP of the tunnel (outer) packet header is set at the tunnel source, relative to the inner (tunneled) packet. In addition, if the DSCP of the tunnel packet is changed (i.e. re-marked) by an intermediate node somewhere between the tunnel source and tunnel destination, consideration needs to be given as to if and how the DSCP of the underlying IP packet is changed relative to the outer (i.e. “tunnel”) DSCP at the tunnel destination, where the “tunnel” IP header, which contains the re-marked DSCP value, is stripped off.

[RFC 2983] defines two conceptual models to describe how to deal with the treatment of Diffserv in the context of IP tunnels, which are described in the following sections.

It is noted that individual IP tunnels are unidirectional entities. If bidirectional behavior is required then a tunnel will be required in each direction and the respective tunneling models will need to be applied to each tunnel.

2.3.4.5.1 IP Uniform Model

With the uniform model, any classification, marking, and re-marking are performed using the DSCP field of the outermost IP packet header only. At the tunnel source, the DSCP value of the underlying IP data packet is copied into the DSCP value of the tunnel IP header, and then at the destination of the tunnel the DSCP of the tunnel IP header is copied back into the DSCP value of the underlying data packet IP header. In this way, the DSCP value of the underlying IP packet propagates up through any added layers of tunnel header and should the outermost DSCP be re-marked, this re-marked value is similarly propagated down to the underlying IP packet when the tunnel header is stripped off at the tunnel destination.

Consider the example shown in Figure 2.31 and the following description:

1. Before entering the IP tunnel, in this example, IP data packets are marked with DSCP 34 (i.e. AF41).
2. At the tunnel source (Router B), the IP tunnel header is added and the DSCP value of the data packet IP header (DSCP 34) is copied

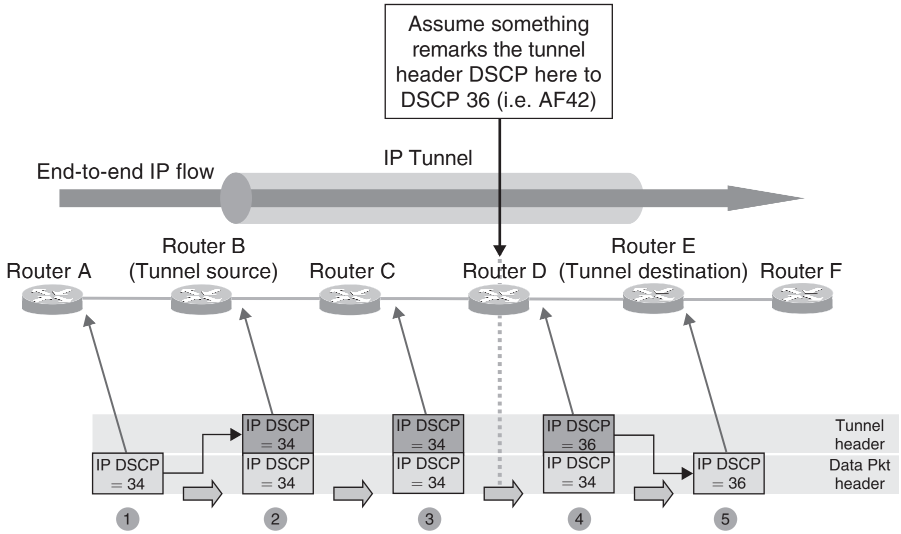


Figure 2.31 IP tunnel: uniform model

into the DSCP of the tunnel header, i.e. the DSCP of both the tunnel and data packet headers are 34.

3. On egress to Router B and intermediate routers between the tunnel source and tunnel destination, e.g. at Router C, when classifying tunnel packets by DSCP, they will look only at the DSCP value of the tunnel header (the outermost tunnel header where multiple layers of tunnel are used), which in this case is DSCP 34 and happens to be the same as the DSCP of the underlying data packet.

4. Assume that some function at Router D re-marks some or all of the tunneled packets to DSCP 36 (i.e. AF42); the re-marking only affects the DSCP of the tunnel header, hence the DSCP of the (outermost) tunnel header is now DSCP 36 (i.e. AF42), while the DSCP of the underlying IP data packet header is still DSCP 34 (i.e. AF41). Router D and any subsequent routers between the tunnel source and tunnel destination, when classifying tunnel packets by DSCP, will look only at the DSCP value of the tunnel header, which is now DSCP 36.

5. At the tunnel destination, the tunnel header is stripped off and DSCP value of the tunnel IP header is copied back into the DSCP value of the underlying data packet IP header, which is now DSCP 36 and which will be used by all subsequent routers when classifying the packets by DSCP.

Effectively, with the uniform model, for each packet, there is one piece of Diffserv information which is carried end-to-end, which may change along the path and which is always represented by the DSCP of the outermost IP header. Hence, when using the uniform model, with an IP tunnel there is really no difference in the resultant Diffserv behavior compared to where no tunnel is present; similarly, there is no difference in the case that multiple layers of IP tunnels are used. Effectively with the uniform model, tunnels are transparent to Diffserv operations.

2.3.4.5.2 IP Pipe Model

Unlike the uniform model, the pipe model treats the DSCP markings on the inner (data) and outer (tunnel) packet headers as independent (although possibly associated) entities. As for the uniform model, outside of the tunnel, as there is no tunnel header the IP data packet DSCP is used; between the tunnel source and tunnel destination, the DSCP marking in the tunnel header is used. The pipe model differs from the uniform model in that DSCP value of the underlying IP packet is not copied into the tunnel header DSCP at the tunnel source (although the tunnel DSCP setting may be derived from the underlying IP packet DSCP value), nor is the tunnel header DSCP copied back into the underlying IP packet DSCP at the tunnel destination. The pipe model behavior is the same where multiple layers of tunnel are used, with each subsequent tunnel layer being treated independently of the last.

Consider the example shown in Figure 2.32 and the following description:

1. Before entering the IP tunnel, in this example, IP data packets are marked with DSCP 34 (i.e. AF41).

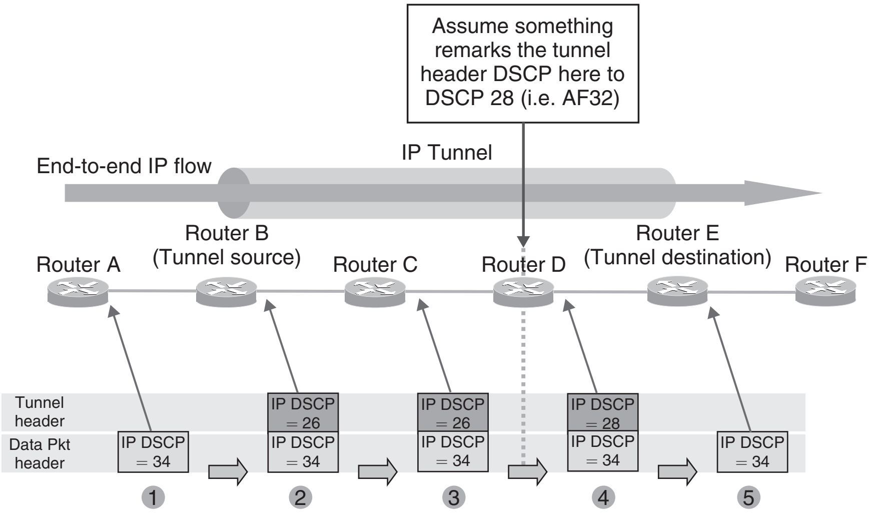


Figure 2.32 IP tunnel: pipe model

2. At the tunnel source (Router B), the tunnel header is added and the DSCP value of the data packet IP header is set. The tunnel DSCP value may be derived from (could be copied from) the DSCP value of the underlying IP packet, or may be set independently of that value. In this example, we assume that the tunnel DSCP is set to DSCP 26 (i.e. AF31) independently of the underlying IP packet DSCP.

3. On egress to Router B and to intermediate routers between the tunnel source and tunnel destination, e.g. at Router C, when classifying tunnel packets by DSCP, they will look only at the DSCP value of the tunnel header (outermost tunnel header where multiple layers of tunnel are used), which in this case is DSCP 26.

4. Assume that some function at Router D re-marks the tunneled packets to DSCP 28 (i.e. AF32); the re-marking only affects the DSCP of the (outermost) tunnel header, hence the DSCP of the underlying IP data packet header is still DSCP 34 (i.e. AF41). On egress to Router D and any subsequent routers between the tunnel source and tunnel destination, when classifying tunnel packets by DSCP,

they will look only at the DSCP value of the tunnel header, which is now DSCP 28.

5. At the tunnel destination, the tunnel header is stripped off. In the case of the pipe model, however, the DSCP value of the tunnel IP header is **not** copied back into the DSCP value of the underlying data packet IP header. Hence, the original DSCP value of the underlying data packet IP header is preserved through the tunnel.

Effectively with the pipe model, there are two separate pieces of Diffserv information which are used; one is used within the bounds of the tunnel and another is used outside of the tunnel. Therefore, the pipe model enables different marking schemes to be used within the tunnel than outside the tunnel. This capability can be useful where the tunnel represents a different Diffserv domain than the networks on either side of the tunnel. This could be in the context of a VPN service provided by an SP, where for example the SP uses a different marking scheme within their portion of the network than their customers do at the edge, while allowing their customers' marking scheme to be preserved end-to-end across the SP VPN service. This capability is sometimes referred to as "QOS transparency."

It is noted that a typical default router implementation would copy the DSCP value of the underlying IP packet into tunnel DSCP value at the tunnel source, but would not copy the DSCP value of the tunnel IP header back into the DSCP value of the underlying data packet IP header at the tunnel destination.

Although [RFC 2983] only considers IP tunneling technologies, the concepts can also be applied to "tunnels" formed by encapsulation in layer 2 (link) or MPLS headers. These approaches are not a form of "IP tunneling" as they do not add an additional IP header, but nonetheless, they can be considered a form of "tunnel." MPLS Diffserv Tunneling modes are described in Section 2.3.6.2.3.

2.3.5 IPv6 QOS Architectures

There is a common misperception that IPv6 provides fundamentally better QOS capabilities than IPv4, which is incorrect. The Intserv

(Section 2.3.3) and Diffserv (Section 2.3.4) IP QOS architectures can be applied equally to IPv6 as to IPv4.

The only practical difference between IPv6 and IPv4 from a QOS perspective is that IPv6 packet headers also include a 20-bit flow label field [RFC 3697]. The flow label helps to classify a flow unambiguously, where some information used to identify the flow may be missing due to packet fragmentation or encryption (Section 2.2.1).

2.3.6 MPLS QOS Architectures

MPLS [RFC 3031] enables new forwarding paradigms from conventional IPv4 or IPv6, allowing forwarding based upon criteria other than the destination IP address. When a traffic stream traverses an MPLS network (also known as an MPLS "domain"), the IP packets (or protocol data units [PDUs] from another protocol, as the "M" in MPLS stands for "Multiprotocol") are "labeled" at the ingress edge router of the MPLS domain (referred to as an Edge Label Switched Router or ELSR). The MPLS "label" is 32 bits long, is of local significance and is most commonly "pushed" on top of (or imposed onto) the original IP header as what is known as a "shim" header; in some cases (where MPLS VPNs or MPLS traffic engineering are used, for example) more than one label (a "label stack") may be imposed. The label that is pushed at the ELSR determines the path that the packet will take across the MPLS domain; this path is termed a label switch path (LSP). Each router within the MPLS domain – termed label switched routers or LSRs – will not look at the IP destination address or within the underlying IP header of labeled packets to determine how to forward the packet but rather the label (or topmost label if there is a label stack) is used to determine which interface and outbound label to use when forwarding the packet onwards to the next hop on the LSP. At the egress of the MPLS domain the labels are popped or stripped off the packet by the egress ELSR and are then forwarded using the normal conventions of IP. Where ATM switches are used as LSRs, they do not add a label "shim" header to the IP packet, but rather encode the label stack into the ATM VPI/VCI field; such ATM LSR deployments are no longer common and hence we do not consider this case in detail.

The new forwarding paradigms made possible with MPLS enable IP networks to support new functionality. Different techniques and signaling protocols are used to determine and establish LSP paths, depending upon the particular paradigm being used. MPLS is most commonly deployed by service providers to provide one or more of the following functions:

- To allow many virtual private networks (VPNs) to be built on top of a single IP/MPLS network. These can be layer 3 VPNs, such as using BGP MPLS VPNs as described in [RFC 4364], or layer 2 VPNs such as those defined in the IETF L2VPN working group [L2VPN].
- To provide traffic engineering (TE) capabilities using MPLS RSVP-TE as defined in [RFC 3209], and as described in Chapter 6, Section 6.2.3.
- To provide fast recovery around network element failures using MPLS TE Fast Reroute (FRR) as defined in [RFC 4090], as discussed in Section 2.6.

With multi-protocol label switching (MPLS), there is a common perception that MPLS provides fundamentally better QOS capabilities than IPv4; as for IPv6, this is not correct. The Intserv (Section 2.3.3) and Diffserv (Section 2.3.4) IP QOS architectures can be applied to MPLS, but with some practical differences from "vanilla" IPv4 and IPv6.

2.3.6.1 MPLS and Intserv/RSVP

Intserv requires admission control and resource reservation on a per flow basis – where a flow is identified by the 5-tuple of source and destination IP addresses, source and destination UDP/TCP port numbers and IP protocol number. These fields are not visible to LSRs within an MPLS domain, which forward labeled packets based upon the outermost label only, hence support for Intserv would require that LSPs are provisioned on a per flow basis, which is not a scalable approach. In practice, Intserv/RSVP is supported in the context of providing reservations to aggregations of flows through the use of MPLS TE tunnels, as discussed in more detail in Chapter 4, Section 4.4.5.

2.3.6.2 MPLS and Diffserv

Diffserv can be applied in an MPLS network essentially in the same way as a plain IP networks, as defined in [RFC3270]. Traffic conditioning is performed at the edge of the Diffserv domain in exactly the same way although it is noted that the router at the edge of the MPLS domain (termed the provider edge or PE router) is not necessarily also the router at the edge of the Diffserv domain, which may be at the customer edge, or CE router. Packets are marked to indicate the particular class of traffic to which they belong and then within the core of the MPLS Diffserv network different PHBs are applied depending upon the marking.

There are, however, some differences between how Diffserv is applied to plain IP packets compared to MPLS labeled packets and these stem from the fact that within an MPLS network, all forwarding is done based upon the outermost label rather that the IP packet header. As the DSCP value is in the IP packet header, which is not used by an LSR, this cannot be used for PHB selection within an MPLS domain. Instead, there is a 3-bit field within the MPLS label shim header as shown in Figure 2.33 – termed the EXP field – which is used for classification when Diffserv is used with MPLS.

[RFC3032] initially defined the EXP field for experimental use; this was subsequently updated by [RFC3270] which redefined it for use with Diffserv, although the field is still commonly referred to as the "EXP" bits. There are two mechanisms by which the EXP field is used for PHB selection within an MPLS Diffserv network; these are described in the following sections.

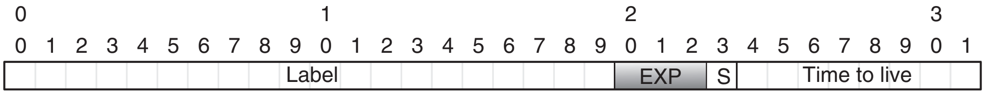


Figure 2.33 MPLS label stack encoding: Label = label value (20 bits), EXP = EXP field (3 bits), S = bottom of stack indicator (1 bit), TTL = time to live field (8 bits)

2.3.6.2.1 EXP Inferred PHB Selection

The most commonly used approach for PHB selection within an MPLS network is to use the EXP field to determine with which PHB a labeled packet should be serviced; this is referred to as EXP Inferred PHB selection.

The EXP field is only three bits long, and therefore it can only represent 8 distinct values, whereas there are 64 possible DSCP values, hence it is not possible to treat the EXP field in labeled packets as directly equivalent to the DS Field in plain IP packets. Therefore, there may need to be a many-to-one mapping of DSCP values to EXP values at the ingress ELSR such that a particular EXP value may need to represent a group of DSCP values, in which case it is referred to as a PHB scheduling class (PSC). LSPs where the EXP field marking is used to determine a PSC are called as EXP-Inferred-PSC LSPs (E-LSPs).

The typical default behavior at an ingress LSR is to copy bits 0 to 2 of the DS field – which are the class selector (CS) codepoints (which are functionally equivalent to the precedence bits) – into the 3 bits of the EXP field in the MPLS label, as shown in Figure 2.34.

The typical default behavior applied at LSRs within the MPLS domain is to copy the EXP field from label-to-label. If additional labels

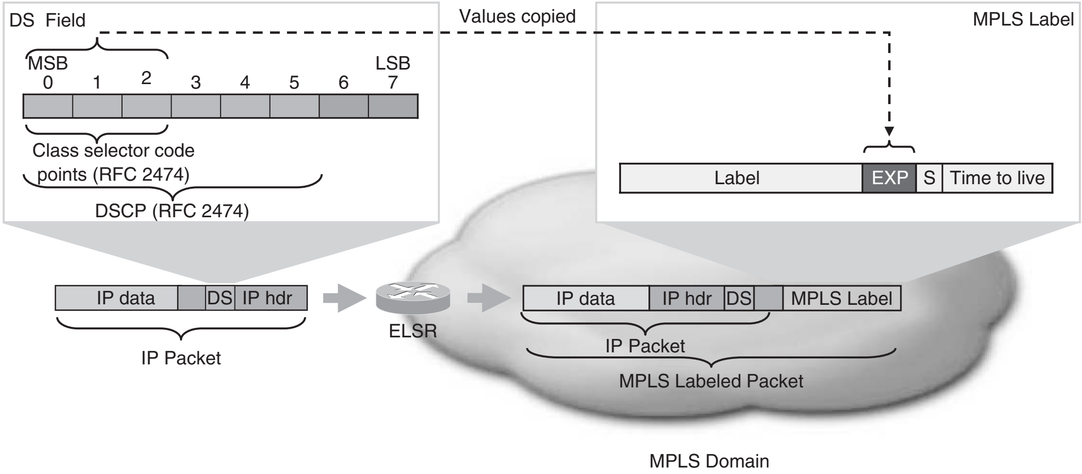


Figure 2.34 Copying CS codepoint to EXP field

are imposed (i.e. a label stack is used), the typical default behavior is to copy the EXP field up the label stack. Hence, where these behaviors are supported, by default the CS codepoint marking of the underlying IP packet is propagated up the label stack. If labels are popped off, the typical default behavior is not to copy the EXP field down the label stack or to the CS codepoints. Therefore if the EXP field marking is changed within the MPLS network this change is not normally propagated down the label stack, but rather the underlying CS codepoint and DSCP values are preserved across the MPLS domain. Behaviors other than this are possible; these are discussed in Section 2.3.6.2.3 on Diffserv MPLS tunnel modes.

2.3.6.2.2 Label Inferred PHB Selection

The use of E-LSPs is the norm in MPLS Diffserv deployments; however, should 8 distinct PSC markings be insufficient to support the number of PHBs required in an MPLS Diffserv network design – or where ATM LSRs are deployed, where a shim header is not used – an alternative approach is defined.

With E-LSPs, a single LSP can be used to carry labeled packets marked with a number of different PSCs; however, [RFC 3270] also defines a scheme where an LSP carries a single PSC only; LSPs that use this scheme are referred to as label-only-inferred-PSC LSPs (L-LSPs). Where a “shim” header is used, the EXP field can then be used to represent the drop precedence to be applied by the LSR to the labeled packet (see the AF PHB, Section 2.3.4.2.2).

ATM LSRs are not widely deployed and in practice there have not been significant requirements for more than 8 distinct PSCs in MPLS Diffserv networks (see the backbone Diffserv deployment case study in Chapter 3, Section 3.3.2). Hence, L-LSPs are not widely used.

2.3.6.2.3 MPLS Diffserv Tunneling Models

[RFC 3270] considers the application of the Diffserv tunneling models (as described in Section 2.3.4.5) and concepts specifically to MPLS. There are a number of conceptual similarities between the Diffserv tunneling models used for IP tunneling and those used for MPLS, where MPLS LSPs are used instead of IP tunnels. As for IP tunnels, MPLS LSPs are unidirectional. In addition, comparably with IP tunnels, intermediate nodes on the path of an LSP look at the marking of the outermost label only. There are also, however, a number of differences due to the implicit differences between IP and MPLS. [RFC 3270] defines three MPLS Diffserv tunneling models, which are described in the following sections.

2.3.6.2.3.1 MPLS Uniform Model The uniform model for MPLS is conceptually similar to the IP tunneling case. Consider the example shown in Figure 2.35 and the following description. In this example, which describes an IP-to-MPLS forwarding case, only a single level of label is added, which could be assigned by LDP [RFC 3036] for example.

1. Outside of the MPLS domain, e.g. at router 1, assume IP data packets are marked with DSCP 34 (i.e. AF41).

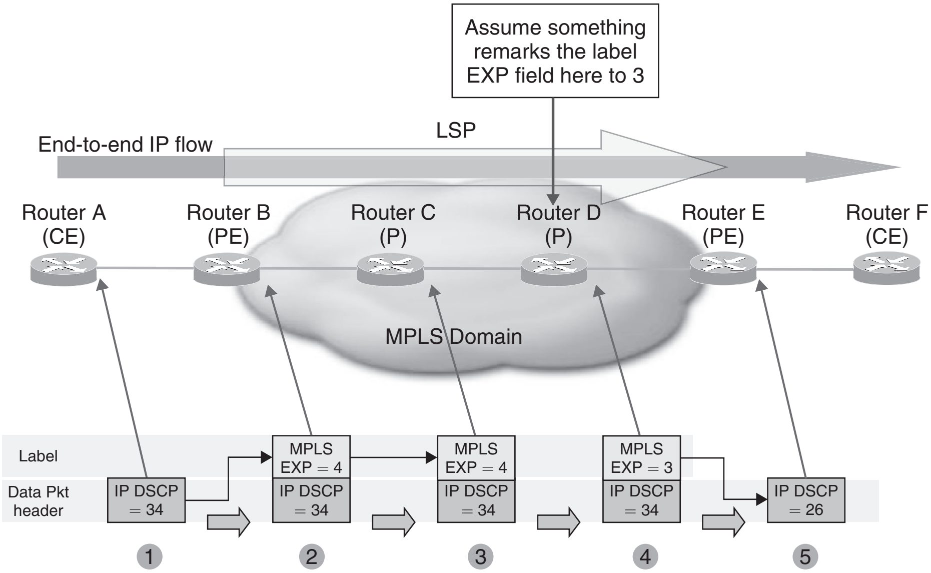


Figure 2.35 MPLS Diffserv tunnel modes: uniform model

2. At the ingress ELSR (Router B), the initial LSP label is imposed and the EXP value for the label is set. The EXP may be derived from the underlying IP packet DSCP value; however, as the EXP field is only 3 bits and the DSCP is 8 bits, it is not possible to copy the whole DSCP value to the EXP field (as would be the case with an IP tunnel). In this example, we assume that bits 0–2 of the DSCP are copied into the label EXP field as described in Section 2.3.6.2.1 with the result that the EXP field would be set to 4. Alternative mappings between the underlying DSCP and the EXP field are possible.

   In an MPLS-to-MPLS forwarding case, where the uniform model is used, the EXP value would be copied up the label stack, as additional layers of label are imposed.

3. Intermediate routers on the LSP, e.g. Router C, will label switch the packet based upon the label value (outermost label value if there is a label stack); as packets are labeled switched, the EXP is by default copied from ingress label to egress label. Hence, in this example, the packet would have an EXP value of 4 on egress to Router C. If there is a label stack, when classifying packets by the EXP field, LSRs will look only at the EXP field of the outermost label.

4. Assuming that some function at Router D re-marks the packets on the LSP from EXP 4 to EXP 3, if there is a label stack, the re-marking only affects the EXP value of the outermost label. Hence, the EXP value of the label is now EXP 3, while the DSCP of the underlying IP data packet header is still DSCP 34 (i.e. AF41). Router D and any subsequent routers on the LSP, when classifying labeled packets by the EXP field, will look only at the EXP field of the outermost label only, which is now EXP 3.

5. In this case, we assume that penultimate hop popping (PHP) is not used, and hence the label is stripped off at the egress ELSR which is the final hop of the LSP, being Router E in this example. The DSCP field value of the underlying IP packet may be re-marked to a value derived from the EXP field value of the deposed label. If the EXP field value were copied back to bits 0–2 of the DSCP field, the resultant DSCP value of the exposed packet on egress to

Router E would be DSCP 26 (i.e. AF31), although alternative mappings between the EXP field and the underlying packet DSCP value are possible. As the label is popped off, on egress to Router E and at subsequent routers the underlying DSCP may be used to classify the packets.

Where PHP is used, the label is popped off at the penultimate hop on the LSP, Router D, in which case as no label is present at Router E, router D would need to perform any required mapping/copying from EXP to DSCP.

The intent with the uniform model is that when used with MPLS, there is really no difference in the resultant Diffserv behavior compared to where MPLS is not used. However, the one factor which prevents total transparency of MPLS to Diffserv operations (unlike the IP tunneling case), is the fact that the MPLS EXP field is only 3 bits long while the DSCP is 8 bits long; this may therefore demand a mapping from DSCP values to EXP values and back to DSCP values, rather than the simple copying which is used in the IP tunneling case.

In practice, it is not normal to apply re-marking within an MPLS domain; rather, such conditioning functions are normally performed at the edge of the Diffserv domain. Hence, the uniform model is rarely used in the context of MPLS and it is defined as optional by [RFC 3270].

2.3.6.2.3.2 MPLS Pipe Model The pipe model for MPLS is conceptually similar to the IP tunneling case, where MPLS LSPs are used instead of IP tunnels, although implicitly there are some differences due to the differences between IP and MPLS. Considering the case of a single level of label (i.e. no label stack), the MPLS pipe model treats the DSCP markings on an underlying IP packet and the MPLS EXP markings on the LSP used by the packet as independent entities. At the start of the LSP, the MPLS EXP field is set; this setting may be derived from the marking in the DCSP of the underlying packet in the case of IP-to-MPLS forwarding, or from the MPLS EXP field of the received label in the case of MPLS-to-MPLS forwarding where a hierarchy of LSPs result in a label stack. Along the path of the LSP, any classification,

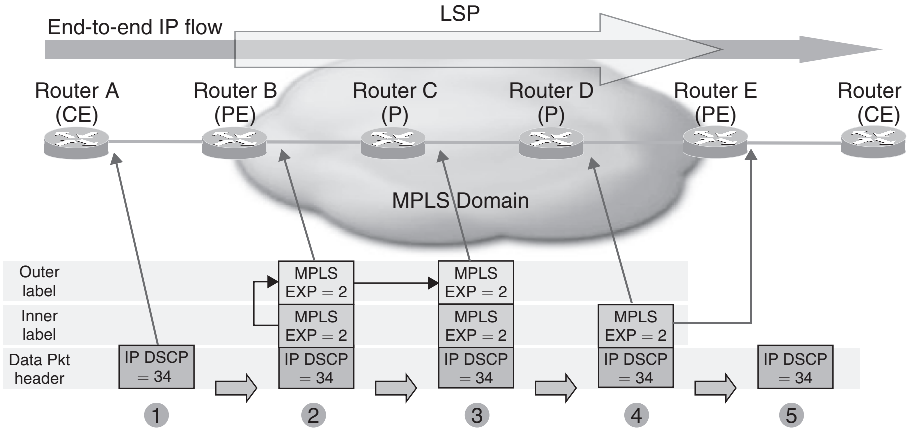


Figure 2.36 MPLS Diffserv tunnel modes: pipe model

marking and re-marking are performed using the EXP field of the outermost label only. Where the outermost label is stripped off, at the end of an LSP (or at the penultimate hop, where PHP is used), the MPLS EXP value of the deposed label is not copied down into the underlying IP packet DSCP or underlying label EXP field.

Consider the example shown in Figure 2.36 and the following description. In this example, which describes an IP-to-MPLS transition, a label stack is used, which could for example represent an MPLS VPN deployment [as per RFC 4364], where the inner label is assigned by Multiprotocol BGP (MBGP) [RFC 2858] and the outer label is assigned by LDP [RFC 3036]. The example also assumes that PHP is used:

1. Outside of the MPLS domain, e.g. at router 1, assume IP data packets are marked with DSCP 34 (i.e. AF41).
2. At the ingress ELSR (Router B), the label stack is imposed and the EXP values for the labels are set. The EXP value for the inner label may be derived from the DSCP value of the underlying IP packet DSCP, or may be set independently of that value. In this example, we assume that the EXP value is set to 2 independently of the underlying IP packet DSCP. As in this example, a label stack is used and

the EXP value of the inner (e.g. MBGP, in the context of MPLS VPN) label is copied up the label stack into the outer (e.g. LDP) label, hence the inner and outer labels both have an EXP value of 2.

3. Intermediate routers on the LSP path, e.g. Router C, will label switch the packet based upon the outermost label value; as packets are labeled switched, the EXP is by default copied from ingress label to egress label, hence in this example, the packet would have an EXP value of 4 on egress to Router C. If there is a label stack, in classifying labeled packets by the EXP field, LSRs will look only at the EXP field of the outermost label.

4. As PHP is used in this example, the outer (e.g. LDP) label is popped off at the penultimate LSR on the LSP, which is Router D in this example. Although the outer label is popped off, the inner (e.g. MBGP) label still remains, hence when classifying labeled packets by the EXP field, on egress Router D will look at the EXP field of this remaining label.

5. Router E, the egress ELSR, will receive the packet with this single (MBGP) label, which it will pop off. Even though it pops off this label, the pipe model defines that Router E retains the value of the EXP field in the received label, such that it can be used to classify the (now unlabeled) packets on egress from Router E to Router F.

   Where a label stack is not used, and penultimate hop popping is used, Router E does not receive a labeled packet, and has no EXP information to retain and use for classification on egress to Router F; therefore, the pipe model cannot be used where penultimate hop popping is used without a label stack.

The pipe model is widely used in MPLS VPN deployments and hence is defined as mandatory by [RFC 3270].

It is noted that the typical default behavior implemented on LSRs is an inception of the pipe model. An ingress LSR will typically copy bits 0 to 2 of the DS field into the 3 bits of the EXP field in the MPLS label by default. The typical default behavior at the egress LSR is not

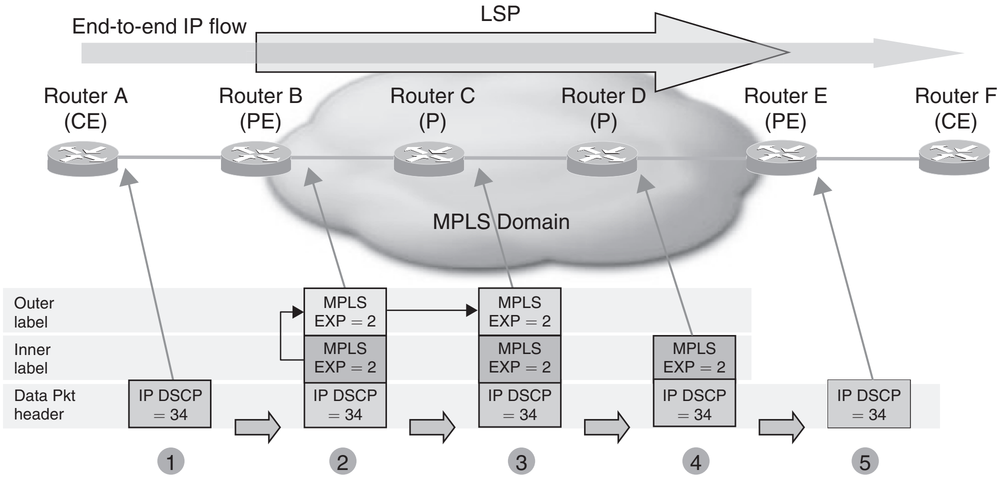


Figure 2.37 MPLS Diffserv tunnel modes: short pipe model

to copy the EXP field down the label stack or to the CS codepoints, as labels are popped.

2.3.6.2.3.3 MPLS Short Pipe Model The short pipe model is a variation on the pipe model; consider the example in Figure 2.37 and the following description.

Steps 1–4 as per the MPLS pipe model described in Section 2.3.6.2.3.2.

Step 5: The short pipe model differs from the pipe model in terms of the behavior applied at the egress ELSR; rather than retaining the EXP value of the received label, the value of the underlying DSCP is used to classify the (now unlabeled) packets on egress from Router E to Router F.

2.3.7 IP Multicast and QOS

The Intserv (Section 2.3.3) and Diffserv (Section 2.3.4) IP QOS architectures were designed to support both IP unicast and IP multicast [RFC1112] traffic from the outset.

- ***IP Multicast and Diffserv***. Support for the basic mechanisms of Diffserv is no different for IP multicast than for IP unicast. Multicast traffic is designated by the destination address of the packets; other than that, the IP headers for multicast traffic are the same as for unicast traffic. Multicast replicated packets have exactly the same DSCP as the original packet, and therefore will be treated with the same PHB as the incoming packets of their respective multicast group. Hence, the DSCP field can be used to mark and classify IP multicast traffic exactly as for IP unicast and Diffserv PHBs can be applied accordingly.

  Implicitly, however, the impact of multicast traffic flows on a network is different from the impact of unicast traffic flows; by definition, multicast flows are point-to-multipoint or multipoint-to-multipoint – a single multicast stream from a source may be replicated to multiple destinations – whereas unicast flows are point-to-point. Hence, the key differences between a unicast and a multicast Diffserv deployment are two-fold:

  - *Multicast capacity planning*. Capacity planning in a Diffserv network manages the provisioning of available capacity relative to network load, potentially on a per-class basis. In a multicast deployment, this needs to take into account the traffic matrix (the matrix of ingress to egress flows) resulting from multicast replication. Capacity planning is discussed in detail in Chapter 6.
  - *Multicast SLAs*. Whereas a received unicast flow can be limited on ingress to the network and the impact on the network thereby constrained, a received multicast flow may be replicated to many destinations within the network after it is received, and hence the impact on the network may be significantly greater. Any SLA definitions for Diffserv-enabled multicast services must take multicast replication within the network into account.

- ***IP Multicast and Intserv/RSVP***. Intserv [RFC1633] and RSVP [RFC2205] were fundamentally designed to cater for multicast as well as unicast reservations. As a consequence, RSVP gives flexible control over the manner in which reservations can be shared along branches of the multicast delivery trees, using “Wildcard Filter” and "Shared-Explicit Filter" reservation styles to allow for reservation state merging. Further, RSVP allows the elementary actions of adding or deleting one sender and/or receiver to or from an existing reservation. Intserv and RSVP are discussed in more detail in Chapter 4, Section 4.4.

2.4 Typical Router QOS Implementations in Practice

Diffserv is the most widely deployed QOS IP QOS architecture and hence most router QOS implementations are optimized for Diffserv deployments. In Section 2.2 we described the main data plane tools that are used in IP QOS: classification, marking, policing, queuing and scheduling, dropping, and shaping. In practice, which of these components are used and how they are combined to create a QOS policy depend upon where they are being applied in the network – at the edge or in the core – and whether they are being applied on egress to an interface or on ingress. Many networks are built with a hierarchy consisting of core routers (CRs), which provide connectivity between distribution routers (DRs), which in turn aggregate connections to routers at remote sites, each of which have local access routers (ARs). If, for example, we consider a Diffserv deployment in this type of network, as shown in Figure 2.38, the access link will normally be the edge of the Diffserv domain, with edge policies being applied outbound on the access router core facing interface and on the access facing interface of the distribution router (which may have

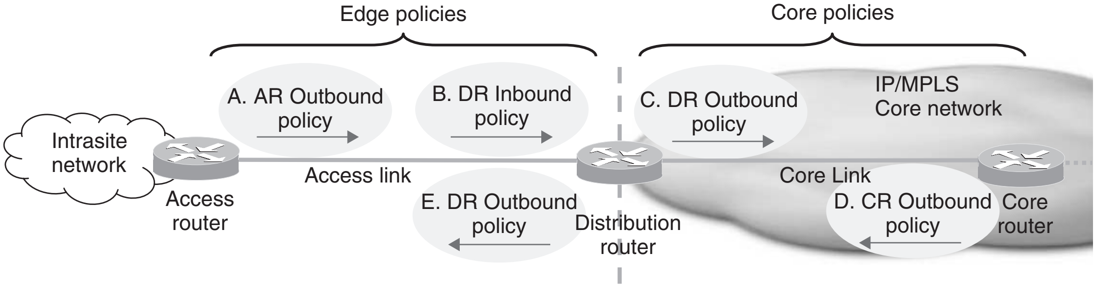


Figure 2.38 Where in the network QOS policies are applied

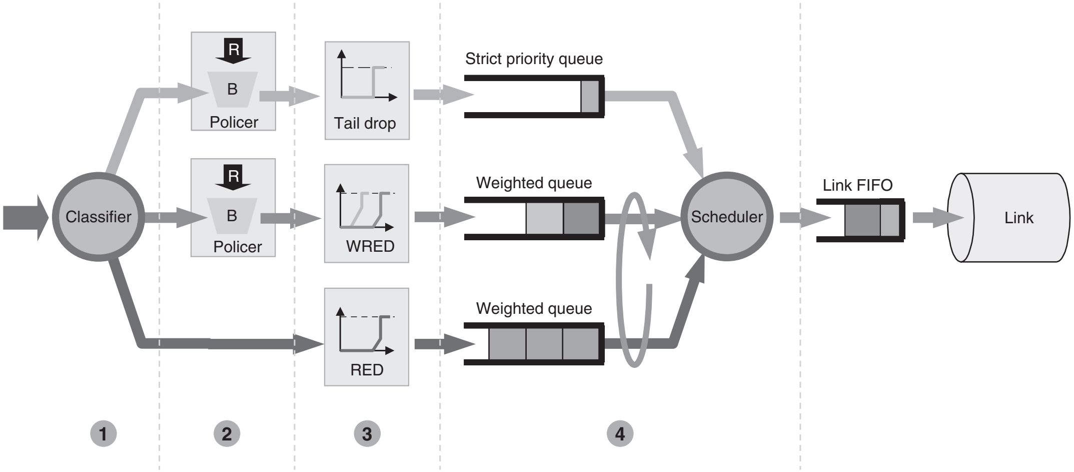


Figure 2.39 Typical access router outbound QOS implementation

inbound policies in some cases also); core policies are then typically applied outbound on the core facing interfaces of the distribution router and outbound on all core router interfaces.

As complex classification and conditioning is performed at the edge of a Diffserv network, router QOS policies applied on routers at the edge of the network (i.e. access routers and distribution routers) are typically more complicated than those used on core routers. Figure 2.39 shows how the different Diffserv QOS components are used together in a typical AR QOS policy applied outbound on the DR facing interface (i.e. policy A in Figure 2.38). Considering Figure 2.39:

- The traffic that is destined for the interface is first classified using simple or complex classification. In this example, we show three classes, although there could be more or less.
- The top traffic class (diagramatically) in Figure 2.39 is serviced from a strict priority queue, for the lowest delay and jitter; this is likely to include applications such as voice and video. An SR-TCM policer with an exceed action of drop is applied to the class before packets are enqueued into the strict priority queue, to enforce a maximum rate for the class and in order to prevent this class from

starving the other classes of bandwidth; packets which are dropped by the policer are not enqueued in the class queue. Tail drop is used to impose a maximum queue limit for the queue, hence enforcing a maximum delay bound for traffic in the queue; the tail drop queue limit may be somewhat redundant in the presence of the applied policer as the policer burst will also implicitly limit the maximum queue depth, and hence delay bound.

- The middle traffic class in Figure 2.39 is serviced from a weighted bandwidth queue. An SR-TCM policer may be applied to the class before packets are enqueued in order to enforce a maximum rate for the in-contract traffic within the class. This could for example be achieved with a conform action of transmit (if the traffic is not pre-marked this could be combined with marking in-contract), and violate action of transmit + mark out-of-contract. WRED may be used to maximize throughput for TCP-based applications within the class and to drop in- and out-of-contract traffic differentially, by using a more aggressive WRED profile for the out-of-contract traffic.
- The bottom traffic class in Figure 2.39 is serviced from a weighted bandwidth queue. RED is used to maximize throughput for TCP-based applications within the class.
- The scheduler ensures that the top class is treated with appropriate priority, and that the middle and red classes receive minimum bandwidth assurances due to their respective configured weightings.
- In most practical router implementations a hardware "line driver" will deal with sending the packets onto the actual line and the scheduler will service its queues into the queue of the hardware line driver on the outgoing interface, which is known as the interface FIFO.

From the above description, it can be seen that there is an implied ordering of OQS actions:

1. Classification is performed first to determine which class packets will be assigned to.

2. Then policing and marking functions are applied to the respective classes; packets dropped by the policer will not be enqueued and will not be subject to tail drop or (W)RED drop decisions. Packets should not be re-classified into other classes at this step, or else there would be the possibility of a loop, where a packet is remarked in one class, then reclassified into another class, where it is also remarked, then reclassified into another class and so on.

3. Tail drop or (W)RED decisions are performed before packets are enqueued into their respective queues. If policing/marking policies have been applied to the class in step #2, it is important that an (W)RED profile selection and drop decisions are based upon the resulting markings from that step, such that any in-/out-of-contract marking will be effective for example. Note that, although (W)RED profile selection should be based upon the result of step #2, packets should not be re-classified into other classes at this step, or else there would be the possibility of a classification loop.

4. Then scheduling decisions are performed and the packets are enqueued in the interface FIFO.

The AR outbound QOS implementation described above could further be augmented with the addition of an aggregate shaper, as described in Section 2.2.4.3, where required to offer subline rate services, as shown in Figure 2.40.

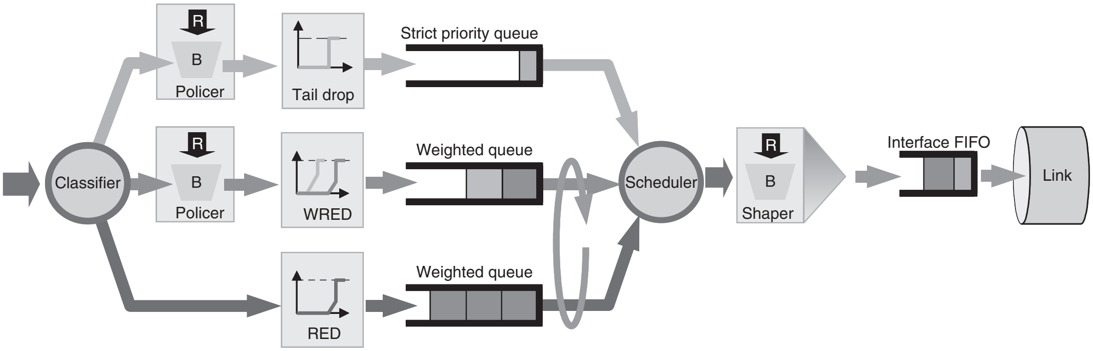


Figure 2.40 Access router outbound QOS implementation with subline rate shaping

Outbound DR QOS policies on AR facing interfaces (i.e. policy E in Figure 2.38) will typically be similar, only with no policers applied to the weighted bandwidth queues, as outbound traffic will have been subjected to traffic conditioning on ingress to the Diffserv domain.

Depending on a particular deployment, QOS policies may be applied inbound on the DR on the AR facing interface (i.e. policy B in Figure 2.38) to perform conditioning on ingress to the Diffserv domain. This is likely to be the case where the access link represents a trust boundary between a network service provider and a customer. While, conceptually, the egress access QOS implementation shown in Figure 2.39 could also be implemented on ingress to a router, this is not commonly done in practice. The ingress to an interface is less commonly an aggregation point for traffic than the egress; if traffic is not aggregated then congestion will not occur, and hence there may be no need to implement scheduling or queuing on ingress to a router. Instead, on ingress to a router interface it is more common to support only per-class policing to perform conditioning, where an SR-TCM or TR-TCM is applied to each class. Figure 2.41 shows a

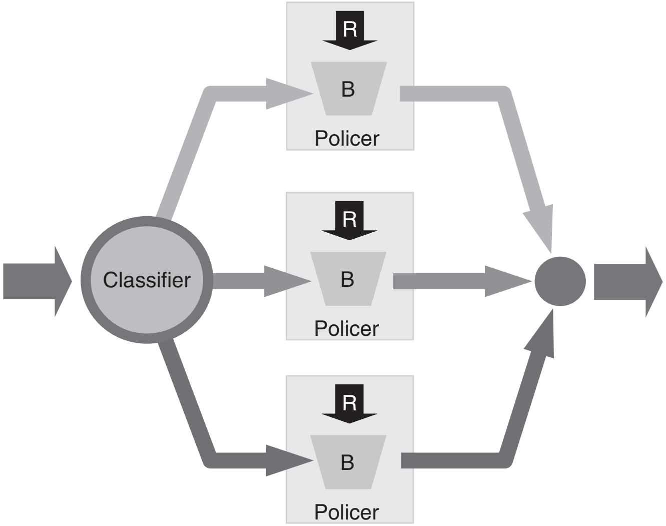


Figure 2.41 Typical distribution router ingress QOS implementation

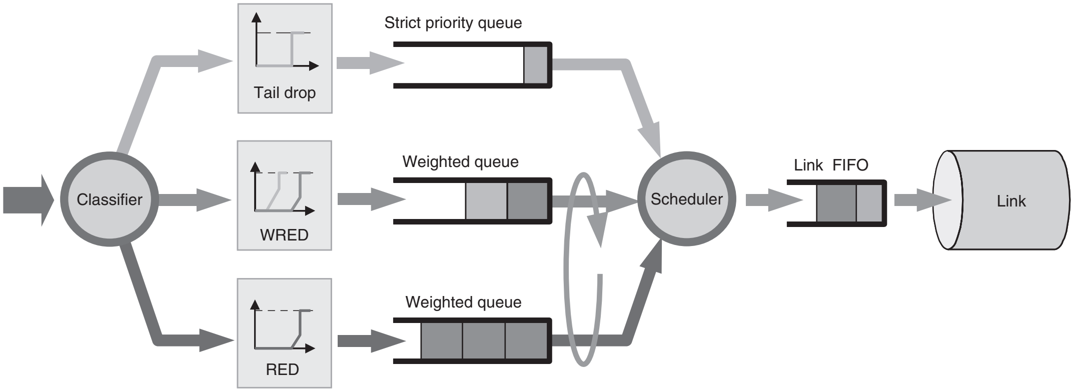


Figure 2.42 Typical core router egress QOS implementation

typical QOS implementation applied to traffic on ingress to a distribution router AR facing interface (i.e. policy B in Figure 2.38).

A typical core router egress QOS implementation (i.e. policy D in Figure 2.38), as shown in Figure 2.42, would generally be a simpler subset of the access router implementation; without policers applied to the weighted bandwidth queues, nor with support for aggregate shapers; the reasons for the differences between core and access router capabilities with respect to QOS are discussed in Section 2.3.4. The same implementation would also typically be used on egress to the DR core facing interfaces (i.e. policy C in Figure 2.38). There are not typically any QOS policies applied on ingress to core interfaces.

When considered as a whole, a typical distribution router QOS implementation supporting a number of access interfaces is shown in Figure 2.43.

Depending upon the architecture of the router, the QOS mechanisms may be implemented centrally in the router, or on distributed platforms, they may be implemented on the interface linecards. On platforms which have a centralized switching fabric, the switching fabric may be a point of aggregation of traffic and hence queuing and scheduling mechanisms may be implemented toward the switching fabric itself. If Diffserv EF/AF forwarding behaviors have an impact on router forwarding performance, the router will support less aggregate

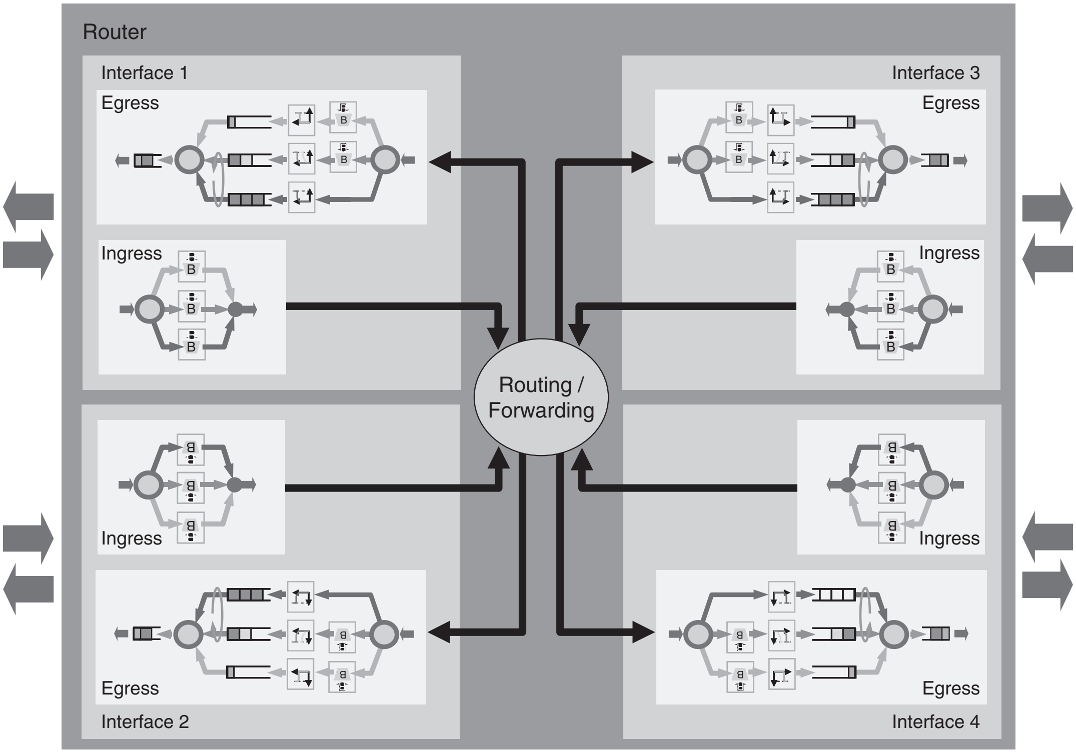


Figure 2.43 Typical distribution router QOS implementation

throughput with Diffserv enabled, and consequently, the per-port cost of the network deployment will be higher. High-performance routers typically implement the EF/AF forwarding behaviors in ASICs, ensuring that there is no forwarding penalty associated with the support of the Diffserv functionality.

The case studies presented in Chapter 3 consider the definition and application of specific QOS policies in more detail.

2.5 Layer 2 QOS

Although the focus of this book is on IP (i.e. layer 3) QOS, IP networks use underlying layer 1 and 2 technologies in order to provide connectivity between layer 3 nodes (i.e. routers). Therefore, in building end-to-end IP services with contracted SLA commitments, it is essential that the underlying layer 1 and layer 2 technologies are able to support the network requirements needed to deliver the contracted SLAs. The SLAs provided at the IP layer are, however, implicitly limited by the SLAs of the underlying layer 2 technology; for example, it would not be possible to deliver an IP service supporting VoIP with a bounded delay, jitter and loss, using an underlying layer 2 network, which did not provide SLAs for delay, jitter and loss at least as good (generally better) as those required for the IP service, e.g. an ATM ABR service.

Some point-to-point layer 2 technologies, such as leased lines delivered on underlying TDM or Synchronous Optical Network (SONET)/ Synchronous Digital Hierarchy (SDH) networks, generally have well-defined SLAs at layer 2, which are capable of supporting layer 3 applications with tight SLA requirements. Such connections generally have a defined delay, a defined bit error rate (BER) and committed layer 2 minimum bandwidth, which is normally equal to the maximum bandwidth for the service. In these cases, Intserv or Diffserv queuing policies can be attached directly to the interface that represents the connection; in the case of Diffserv the policy applied would be similar to that illustrated in Figure 2.40.

Other considerations can apply to multiaccess layer 2 technologies, such as ATM, Frame relay and Ethernet, which have some of their own explicit QOS capabilities at layer 2; in such cases end-to-end IP SLAs can be achieved by interworking between layer 3 QOS functions and the underlying layer 2 QOS capabilities as described in the following sections.

2.5.1 ATM

Asynchronous transfer mode (ATM) is a cell relay technology which can be used to provide layer 2 connectivity in IP networks. ATM is a connection-oriented technology in which virtual circuits (VCs) are established between the end points of the connection before data can be exchanged. The ATM Forum Traffic Management Specification Version 4.0 [af-tm-0056] defines the following five service categories that determine the QOS that a particular VC receives:

- *Constant bit rate (CBR)*. The CBR service category was designed to support low delay (cell transfer delay or CTD), low jitter (cell delay variation or CDV), low loss, constant bit rate applications such as circuit emulation. CBR VCs have a specified peak cell rate (PCR); as long as the PCR is not exceeded, the CTD and CDV will be assured.

- *Variable bit rate – real-time (VBR-rt)*. The VBR-rt service category was designed to support low delay, low jitter, low loss, variable bit rate applications such as VoIP and video. VBR VCs (both VBR-rt and VBR-nrt) have a specified sustained cell rate (SCR) but can burst at rates above this up to their defined PCR as defined by the maximum burst size (MBS).

- *Variable bit rate – non-real-time (VBR-nrt)*. The VBR-nrt service category was designed to support bursty, non-real-time applications that require a committed minimum amount of bandwidth but do not have tightly bounded requirements for delay and jitter. With VBR-nrt, the CTD and CDV are not assured and hence VBR-nrt is unsuitable for real-time applications.

- *Available bit rate (ABR)*. Similar to VBR-nrt the ABR service category was designed to support bursty, non-real-time applications that require a committed minimum amount of bandwidth while the VC is active. In addition, however, ABR uses closed-loop feedback conveyed via resource management (RM) cells from the network to the end-systems in order to adapt their rate of sending based upon the congested state of the network. This allows end-systems to increase their transmission rates to take advantage of the available bandwidth when the network is not congested. If the end-systems correctly adapt their rate of sending based upon the network feedback, the network commits to a ratio of dropped cells to transmitted cells defined by the cell loss ratio (CLR).

With ABR, the CTD and CDV can be large and hence ABR is unsuitable for real-time applications. ABR VCs have a specified minimum cell rate (MCR) and PCR; the bandwidth available to a VC at a particular point in time is defined as the available cell rate (ACR), which is between MCR and PCR.

- *Unspecified bit rate (UBR)*. UBR VCs have a specified PCR which defines the maximum rate for the VC; VCs can use up to their configured PCR when bandwidth is available, although no bandwidth reservations are made for the VC within the ATM network. There are no bounds with respect to the CTD or CLR, i.e. cell delivery is not guaranteed; retransmission at higher layers is assumed and hence UBR is unsuitable for real-time applications or applications with committed loss requirements.

  The UBR + variant of UBR allows end-systems to signal a requested MCR to an ATM switch in a connection request, and the ATM network attempts to maintain this as an end-to-end guarantee.

ATM traffic shaping is performed by ATM end-systems at the ingress point to the ATM network (the user-to-network interface or UNI) in order to ensure that the traffic on the VC adheres to the respective VC traffic contract. If the traffic contract is exceeded then policing within the ATM network may drop excess traffic or may set the cell loss priority (CLP) bit within the cell header to 1. If congestion is subsequently experienced within the ATM network cells marked CLP = 1 are discarded in preference to cells marked CLP = 0. VCs may be grouped together in a virtual path (VP), which may also be shaped.

Within the ATM network, schedulers are used to differentiate between traffic from the different service categories. Typical ATM schedulers service the different ATM service categories in strict priority order: CBR is serviced first with highest priority, then VBR-rt, next VBR-nrt, and ABR with UBR last.

The different ATM service classes can be used when supporting Intserv or Diffserv over an ATM connection. This “mapping” of Diffserv to ATM QOS is described in the following section, while Intserv is described in detail in Chapter 4.

2.5.1.1 Mapping Diffserv to ATM QOS

There are two ways that Diffserv can be supported over an intermediate layer 2 ATM connection between two layer 3 nodes:

- *Single VC approach*. With this approach a single VC is used between connected layer 3 nodes and a Diffserv IP QOS policy, i.e. a queuing policy implementing Diffserv PHBs, is applied to that VC. The service category and SLA commitments for the VC need to be sufficient to support the sum of the tightest SLA requirements for constituent Diffserv classes being transported.

  For example, if 256 kbps of VoIP traffic and a minimum of 512 kbps of data traffic needs to be concurrently supported between two sites using ATM, the VC would need to be of a service category that can meet the delay, jitter, and loss requirements of the VoIP service (i.e. typically CBR or VBR-rt), and the VC would need to be provisioned for (512 kbps + 256 kbps) = 768 kbps. The ATM service provider would only assure the delay, jitter, and loss commitments for the VC for up to 768 kbps of traffic, and therefore ATM traffic shaping would be applied to the VC at each of the L3 nodes at either end of the VC. A Diffserv policy would also be applied to each shaped VC, which would use an EF PHB (i.e. priority queue) to prioritize the 256 kbps of VoIP traffic, and an AF PHB (i.e. weighted bandwidth queue) to assure a minimum of 512 kbps of data traffic. The policy applied would be similar to that illustrated in Figure 2.40, where the aggregate shaper was performing ATM traffic shaping. If the VoIP traffic were inactive then the data traffic could potentially re-use the unused bandwidth up to the 768 kbps VC maximum rate.

- *Multi-VC approach*. With this approach, a separate ATM VC is used for each Diffserv PHB type between connected layer 3 nodes. Each VC has a different ATM service category, where the service category and SLA commitments for each VC need to be sufficient to support the SLA requirements of the respective Diffserv classes being transported.

  Considering the previous example, a 256 kbps VC could be used to support the voice (EF) traffic; this VC would need to be of a service category that can meet the delay and jitter requirements of the VoIP service, i.e. typically CBR or VBR-rt. A separate 512 kbps VC would be used to support the data (AF) traffic. The data VC would need to be of a service category that can meet the delay and loss requirements of the data service, i.e. typically VBR-nrt, but potentially ABR or UBR+ depending upon the specific service requirements.

SLAs for ATM services available from ATM network service providers are generally specified on a per-VC basis (unless ATM VPs are used), rather than across a group of VCs. Therefore, this approach can have the disadvantage that the SLAs for the service do not allow for unused capacity on the voice VC to be directly available to be re-used by the data traffic, for example. A potential benefit of this approach, however, is that rental cost of 1 × 256 kbps CBR VC and 1 × 512 kbps VBR-nrt VC may be less than for 1 × 768 kbps CBR VC (VCs with higher priority service categories tend to be more expensive than lower priority categories). This approach may require that the multiple VCs are effectively treated as a single routed connection, with the DSCP of each packet determining which particular VC will be used for that packet.

Section 2.3.4.4 discusses the possible interworking of ATM explicit forward congestion indication (EFCI) with ECN.

2.5.2 Frame-relay

Compared to ATM, the QOS capabilities provided with frame-relay networks are rudimentary. Frame-relay, like ATM, is a connection-oriented technology in which VCs are established between the end points of the connection before data can be exchanged. The service provided by a frame-relay VC is defined in terms of a token bucket traffic shaper, as described in Section 2.2.4.3. The bucket has a specified maximum depth, which is the sum of the committed burst (*Bc*) and the excess burst (*Be*) and is filled at a rate defined by the committed information rate (*CIR*). *CIR* defines the average rate at which the frame-relay network service guarantees to transport data on the VC during a time interval *Tc*. *Bc* defines the maximum amount of data that can be transmitted during the interval *Tc*, i.e. *Bc* = *CIR* * *Tc*. *Be* defines the amount of excess data in addition to *Bc* that can be sent during the first interval after the token bucket is full, i.e. it contains *Bc* + *Be* tokens. Therefore, if the bucket is full *Bc* + *Be* bytes can be sent in the first interval; if these tokens are all used then *Bc* bytes can be sent in the subsequent time interval.

Frame-relay traffic shaping is performed by frame-relay end-systems at the ingress point to the frame-relay network (i.e. at the UNI) in order to ensure that the traffic on the VC adheres to the respective VC traffic contract. If the traffic contract is exceeded then policing within the frame-relay network may drop excess traffic or may set the discard eligibility (DE) bit within the cell header. If congestion is subsequently experienced within the network, the frames with DE set are discarded in preference to frames where DE is not set. Frame-relay provides explicit notification of congestion via forward explicit congestion notification (FECN); if congestion is experienced along the path of a VC, this bit is set in transiting frames; the destination frame-relay end-system will set the backward error congestion notification (BECN) bit in frames sent back to the source end-system. The explicit congestion notification provided by BECN can be used to allow end-systems to increase their transmission rates (through adapting their traffic shaping rates) to take advantage of the available bandwidth when the network is not congested, resorting to sending at CIR if BECN is received.

When Diffserv is applied to frame-relay, a Diffserv IP QOS policy is applied to a shaped frame-relay VC; the policy applied would be similar to that illustrated in Figure 2.40, where the aggregate shaper would be performing frame-relay traffic shaping. Unlike ATM, frame-relay does not have the concept of different service categories and therefore the SLA commitments for the VC assured by the frame-relay service provider would need to be sufficient to support the sum of the tightest SLA requirements for constituent Diffserv classes being transported.

Section 2.3.4.4 discusses the possible interworking of frame-relay forward explicit congestion notification (FECN) with ECN.

2.5.3 Ethernet

The original IEEE 802.3 specifications for Ethernet included no provision for differentiated QOS, i.e. only supported a single service class. The subsequent 802.1Q [802.1Q] project in the IEEE 802 standards process added support for VLAN trunking. The 802.1Q frame format added an extra 4-byte "VLAN tag" to the original Ethernet header, 3-bits of which are defined as the *user_priority* field as shown in Figure 2.44.

The use of the *user_priority* field, to assign a priority indication to each frame, was defined by the 802.1P project (and hence is also commonly referred to as the "802.1P" field), the results of which were merged into 802.1D [802.1D] Annex G. The use of the user_priority field is analogous to the use of the DSCP field in IP and the EXP field in an MPLS network; the field is used to indicate the "priority" of the frame, which is used to determine the forwarding behavior of that frame at each bridging hop. As the field is 3 bits long it can be used to present 8 distinct markings.

Annex G of 802.1D is considered as "informative" only, and provides only a high-level description of behaviors that should be applied based upon *user_priority* markings. 802.1D Annex G acknowledges that not all 8 markings may be used in any particular deployment and specifies strict priority behavior as the minimal implementation where only a few classes are deployed. It acknowledges the need to support schedulers which can provide minimum bandwidth assurances where

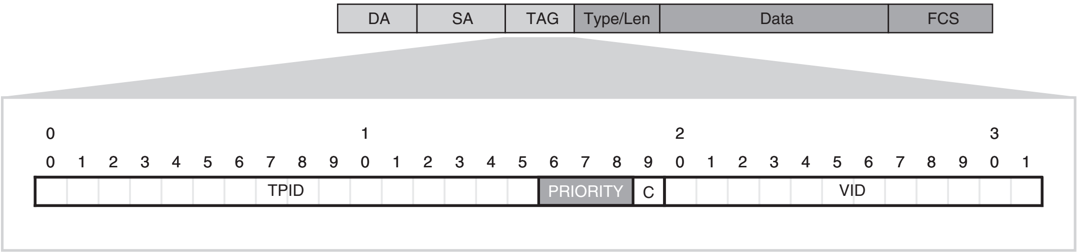


Figure 2.44 802.1Q Frame Format. DA = Destination Address, SA = Source Address, FCS = Frame Check Sequence, TPID = Tag Protocol Identifier (16 bits), PRIORITY = User priority field (3 bits), C = Canonical Format Indicator (1 bit), VID = VLAN Identifier (12 bits)

greater numbers of classes are supported. In practice, in interpreting the specification, most Ethernet switch vendors today support Diffserv like scheduling, with a strict priority queue (i.e. EF-like) and a number of weighted bandwidth queues (i.e. AF-like), to which different *user_priority* values can be assigned.

Hence, in practice, it is often possible to use these capabilities and to treat Ethernet networks, which are components of an end-to-end IP network, almost like any other part of the Diffserv domain, but with EF-like and AF-like behaviors being applied based upon classification of the 802.1Q *user_priority* field. As with MPLS EXP field, it is noted that as the *user_ priority* field is 3 bits long and therefore it can only represent 8 distinct values, whereas there are 64 possible DSCP values, hence when used as a component in an end-to-end Diffserv network a particular *user_ priority* value may need to represent a group of DSCP values.

2.6 Complementary Technologies

Complementary to IP QOS technologies, there are a set of additional techniques and technologies that have been developed within the IP technical community and which further enable IP networks to be engineered to support tightly bounded SLA commitments. While these technologies are not covered in detail in this book, they are described here in overview.

- ***Fast IGP convergence***. Advancements and developments in the implementation and deployment of IGPs have resulted in significant improvements to the IGP convergence times that can be achieved without any compromise in routing protocol stability. This has resulted in convergence times of a few hundred milliseconds being realistically achievable in well-designed IP networks today [FRANCOIS], which significantly reduces the loss of connectivity experienced following network element (e.g. link or node) failures. This reduction in convergence times allows higher availability targets and lower packet loss rates to be offered for SLAs across all service classes. Consequently, fast IGP convergence is also recommended as a foundation for multiservice IP network designs.

- ***Fast reroute technologies***. Developments in local protection schemes for both IP and MPLS – generically termed fast reroute (FRR) technologies – enable further reductions in the loss of connectivity following network element failures.
  - *MPLS traffic engineering fast reroute*. The use of MPLS traffic engineering (TE) for admission control is discussed in Chapter 4, and for bandwidth management is discussed in Chapter 6. However, there is another application of MPLS TE which is in the context if MPLS TE Fast Re-route (FRR) [RFC 4090].
  - *IP fast re-route*. IP fast re-route [SHAND] is a recent development which provides similar capabilities to MPLS TE FRR, but for IP environments.

  These FRR technologies are local protection schemes, unlike IGP convergence which is a distributed computation process. With FRR, on occurrence of a failure there are no delays associated with the distribution of updated routing information or routing table recalculation prior to IP connectivity being restored. Consequently, the restoration times achieved with FRR are always likely to be faster – typically within fifty milliseconds – and more deterministic than those achieved with IGP fast convergence are. This allows the highest availability of service to be offered in support of VoIP services; for example, ensuring that link failures have minimal impact on IP telephony users. Where such levels of protection are required, a subsecond convergence IGP design should be complemented with the deployment of FRR.

2.7 Where QOS cannot make a difference

In concluding this chapter, we highlight the fact that QOS is not a panacea to all networking ills. There will undoubtedly be cases where, even using QOS mechanisms the SLA requirements cannot be met on a particular network, and hence techniques other than QOS may need to be considered.

- *Network engineering*. In some cases, it may be necessary to re-engineer a network in order to ensure that the SLA requirements of an application can be met. For example, a satellite connection could be replaced by a terrestrial link, in order to reduce the end-to-end delay experienced by an application.

- *Application engineering*. There may be cases where it is more appropriate (cost-effective) to re-engineer an application, or rather re-engineer how an application uses the network such that the applications SLA demands on the network are reduced, than to re-engineer the network to support the original requirements of the application. For example, if the de-jitter buffer on a video end-system is set unnecessarily large it may add unnecessarily to the end-to-end delay, which may increase the channel change time or VoD responsiveness above acceptable thresholds. In this case the right approach to solving the problem is to reduce the de-jitter buffer in the video end-system rather than trying to reduce the network delay. Another example of application engineering is the deployment of distributed application caches throughout the network, which can both reduce the network traffic load due to the application, and also reduce the end-user response times.

References

[802.1D] IEEE Std. 802.1D-2004, Media Access Control (MAC) Bridges

[802.1Q] IEEE Std. 802.1Q-2003, Virtual Bridged Local Area Networks

[af-tm-0056] ATM Forum Traffic Management Specification Version 4.0, April 1996

[BITORIKA] A. Bitorika, M. Robin, and M. Huggard, A survey of active queue management schemes, Trinity College Dublin, Department of Computer Science, Tech. Rep., Sept. 2003

[CISCO]http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s28/12sl3ncd.htm

[DEMERS] A. Demers, S. Keshav, and S. Shenkar, Analysis and simulation of a fair queueing algorithm, *Journal of Internetworking Research and Experience*, vol. 1, 1990

[DORAN] S. Doran, RED Experience and Differentiated Queueing, *NANOG 13*, June 1998

[DSCR] http://www.iana.org/assignments/dscp-registry

[FLOYD1] S. Floyd, and V. Jacobson, Random Early Detection gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, Volume 1, Number 4, August 1993, pp. 397–413

[FLOYD2] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker, Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management, August 1, 2001

[FRANCOIS] Pierre Francois, Clarence Filsfils, John Evans and Olivier Bonaventure, Achieving subsecond IGP convergence in large IP networks, *ACM SIGCOMM Computer Communication Review*, Vol. 35, Issue 3 (July 2005), pp. 35–44

[FRF.12] Frame-Relay Fragmentation Implementation Agreement, FRF.12, Frame-Relay Forum, December 1997

[GCRA] ATM Forum, *ATM User-Network Interface Specification*, Prentice Hall, 1993

[JACOBSON] V. Jacobson, K. Nichols, K. Poduri, RED in a different Light, Technical Report, Cisco Systems, Sept. 1999

[KESHAV] S. Keshav, *An Engineering Approach to Computer Networking*, Addison-Wesley, 1997

[KLEINROCK] L. Kleinrock, *Queueing Systems Vol. 2: Computer Applications*, Wiley, 1976

[L2VPN] IETF L2 VPN working Group: http://www.ietf.org/html.charters/l2vpn-charter.html

[LAKSHMAN] T. V. Lakshman, A. Neidhardt, and T. J. Ott, The Drop from Front Strategy in TCP and in TCP over ATM, *Proc. IEEE INFOCOM*, San Francisco, CA, March 1996

[MAY] M. May, J. Bolot, C. Diot, and B. Lyles, Reasons not to deploy RED, in *Proc. IWQoS'99*, June 1999, pp. 260–262

[PAN1] R. Pan, L. Breslau, B. Prabhakar, S. Shenker, Approximate Fairness through Differential Dropping, *ACM Computer Communication Review*, July 2003

[PAN2] R. Pan, L. Breslau, B. Prabhakar, S. Shenker, A Flow Table-based Design to Approximate Fairness, BEST PAPER AWARD, *Proceedings of Hot Interconnects*, 2002 and by invitation IEEE Micro January/February 2003

[RFC791] Internet Protocol Protocol Specification, RFC 791, September 1981

[RFC1112] S. E. Deering, Host extensions for IP multicasting, *RFC* 1112, 1989

[RFC1122] R. Braden, Editor, Requirements for Internet Hosts – Communication Layers, *RFC* 1122, October 1989

[RFC1349] P. Almquist, Type of Service in the Internet Protocol Suite, *RFC* 1349, July 1992

[RFC1583] J. Moy, OSPF Version 2, *RFC* 1583, March 1994

[RFC1633] R. Braden, D. Clark, S. Shenker, Integrated Services in the Internet Architecture: an Overview, *RFC* 1633, June 1994

[RFC1661] W. Simpson, Editor, The Point-to-Point Protocol (PPP), *RFC* 1661, July 1994

[RFC1990] Skwloer et al., The PPP Multilink Protocol (MP), *RFC* 1990, August 1996

[RFC2003] C. Perkins, IP Encapsulation within IP, *RFC* 2003, October 1996

[RFC2309] D. Clark et al., Recommendations on Queue Management and Congestion Avoidance in the Internet, *RFC* 2309, April 1998

[RFC2328] J. Moy, OSPF Version 2, *RFC* 2328, April 1998

[RFC2401] S. Kent, R. Atkinson, Security Architecture for the Internet Protocol, *RFC* 2401, November 1998

[RFC2460] S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, *RFC* 2460, December 1998

[RFC2474] K. Nichols et al., Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, *RFC* 2474, December 1998

[RFC2475] S. Blake et al., An Architecture for Differentiated Service, *RFC* 2475, December 1998

[RFC2481] K. Ramakrishnan, S. Floyd, A Proposal to add Explicit Congestion Notification (ECN) to IP, *RFC* 2481, January 1999

[RFC2597] J. Heinanen et al., Assured Forwarding PHB Group, *RFC* 2597, June 1999

[RFC2661] W. Townsley et al., Layer Two Tunneling Protocol 'L2TP', *RFC* 2661, August 1999

[RFC2697] J. Heinanen, R. Guerin, A Single Rate Three Color Marker, *RFC* 2697, September 1999

[RFC2698] J. Heinanen, R. Guerin, A Two Rate Three Color Marker, *RFC* 2698, September 1999

[RFC2784] D. Farinacci et al., Generic Routing Encapsulation (GRE), *RFC* 2784, March 2000

[RFC2858] T. Bates, Y. Rekhter, R. Chandra and D. Katz, Multiprotocol Extensions for BGP-4, *RFC* 2858, June 2000

[RFC2983] D. Black, Differentiated Services and Tunnels, *RFC* 2983, October 2000

[RFC3031] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol Label Switching Architecture, *RFC* 3031, January 2001

[RFC3032] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, MPLS Label Stack Encoding, *RFC* 3032, January 2001

[RFC3036] L. Andersson et al., LDP Specification, *RFC* 3036, January 2001

[RFC3086] K. Nichols, and B. Carpenter, Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification, *RFC* 3086, April 2001

[RFC3168] K. Ramakrishnan et al., The Addition of Explicit Congestion Notification (ECN) to IP, *RFC* 3168, September 2001

[RFC3209] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, RSVP-TE: Extensions to RSVP for LSP Tunnels, *RFC* 3209, December 2001

[RFC3246] B. Davie, ed. et al., An Expedited Forwarding PHB, *RFC* 3246, March 2002

[RFC3270] F. Le Faucheur et al., Multi-Protocol Label Switching (MPLS) Support of Differentiated Services, *RFC* 3270, May 2002

[RFC3270] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, Multi-Protocol Label Switching (MPLS) Support of Differentiated Services, *RFC* 3270, May 2002

[RFC3662] Nichols, K. Wehrle, A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services, R. Bless, K. December 2003

[RFC3697] J. Rajahalme et al., IPv6 Flow Label Specification, *RFC* 3697, March 2004

[RFC4090] P. Pan, Ed., G. Swallow, Ed., A. Atlas, Ed., Fast Reroute Extensions to RSVP-TE for LSP Tunnels, *RFC* 4090, May 2005

[RFC4364] E. Rosen and Y. Rekhter, BGP/MPLS IP Virtual Private Networks (VPNs), *RFC* 4364, February 2006

[REC4774] S. Floyd, Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field, November 2006

[SHAND] M. Shand, S. Bryant, IP Fast Reroute Framework, Internet Draft, draft – ietf-rtgwg-ipfrr-Framework (work in progress)

[SHANNON] C. Shannon, D. Moore, and K. Claffy, Beyond Folklore: Observations on Fragmented Traffic, Cooperative Association for Internet Data Analysis (CAIDA), 2002. Available at: www.caida.org/outreach/papers/2002/Frag/

[SHREEDHAR] M. Shreedhar, George Varghese, Efficient Fair Queuing using Deficit Round Robin, SIGCOMM 1995

Appendix 2.A: Precedence, TOS, and DSCP Conversion

It can be confusing trying to understand how to convert between corresponding IP precedence values, TOS values, and DSCP values, hence this appendix aims to help solve that problem.

2.A.1 Notation

The numeric notations most commonly used for IP precedence, TOS and DSCP values are as follows.

- *IP precedence.* The notation normally used when referring to particular IP precedence values is to take the entire value of the precedence field (bits 0 to 2 of the TOS octet) and express it in decimal, where bit 0 is the most significant bit; e.g. an IP precedence value of 010 binary is represented as Precedence 2.

  IP precedence is discussed in detail in Section 2.3.2.1.

- *Type of service.* The notation generally used when referring to the "TOS value" is to take the entire vale of the type of service octet (including the precedence field and bit 7 of the type of service octet) expressed in decimal, where bit 0 is taken as the most significant bit; e.g. assuming a precedence field value of 110 binary, and a

TOS field value of 0100 binary, the TOS value would generally be referred to as 212 decimal (i.e. 11010100 binary).

Type of service is discussed in detail in Section 2.3.2.2.

- *DSCP*. When referring to DSCP values, all 6 bits of the DSCP are expressed in decimal, where bit 0 is taken as the most significant bit; e.g. a DSCP of 010000 binary is represented as DSCP 16.

  The DS field is discussed in detail in Section 2.3.4.1.

- *Class selector codepoints*. The numeric notation used for the class selector (CS) codepoints is exactly the same as for IP precedence. When converting to/from the CS codepoints, use the same conversion as for to/from IP precedence.

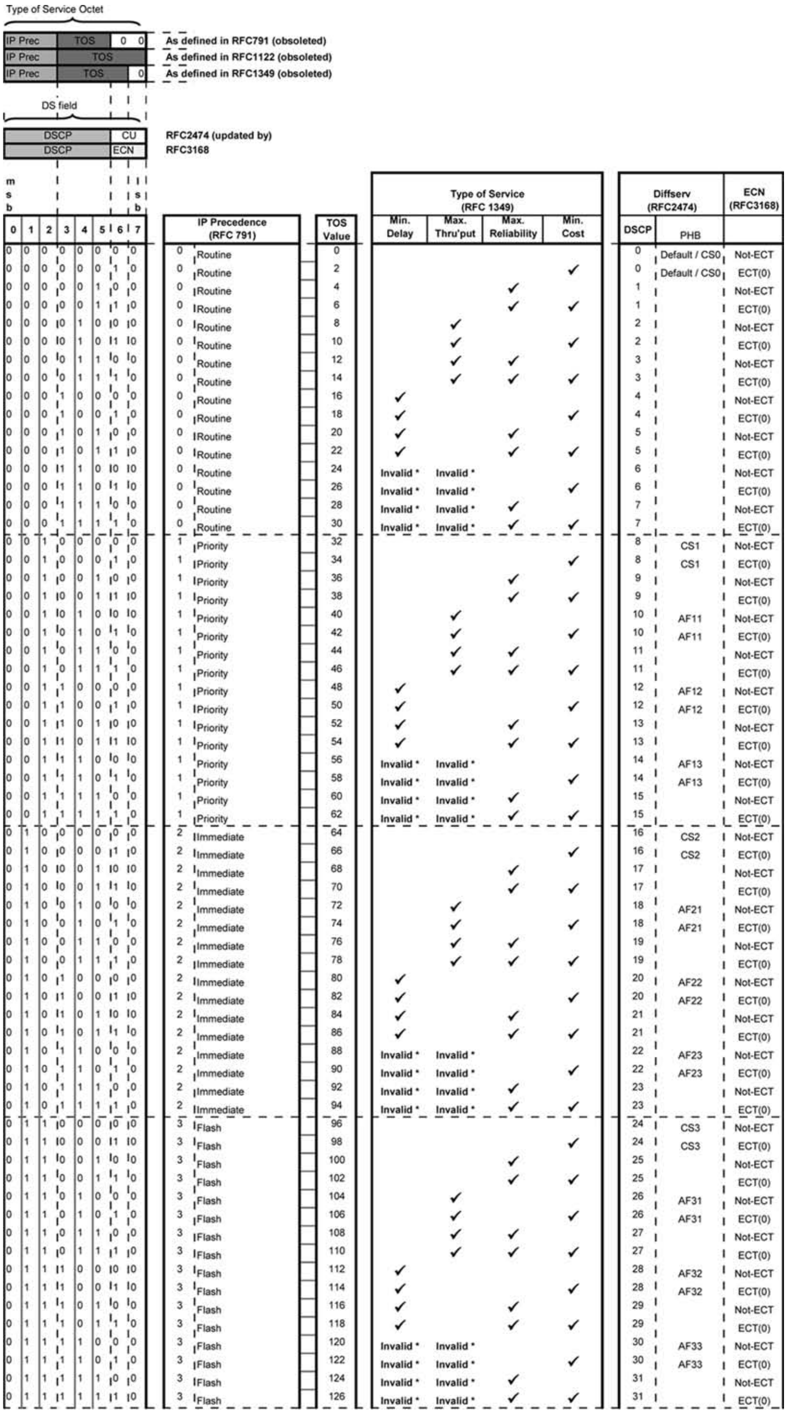
  The use of the class selector codepoints is discussed in Section 2.3.4.1.

2.A.2 Conversion

Use the following formula when converting between the numeric notations most commonly used for IP precedence, TOS and DSCP values:

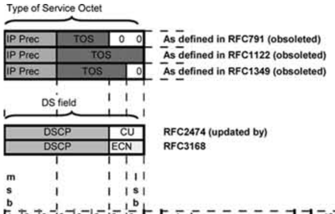
- *DSCP_value = INT (TOS_value/4)*
- *DSCP_value = PREC * 8*
- *TOS_value = DSCP_value * 4*
- *TOS_value = PREC_value * 32*
- *PREC_value = INT (DSCP_value/8)*
- *PREC_value = INT (TOS_value/32)*

The ready-reckoner in Figure 2.45 provides a quick reference for conversion.


Type of Service Octet
IP Prec
TOS
0 0
As defined in RFC791 (obsoleted)
IP Prec
TOS
As defined in RFC1122 (obsoleted)
IP Prec
TOS
0
As defined in RFC1349 (obsoleted)
DS field
DSCP
CU
RFC2474 (updated by)
DSCP
ECN
RFC3168
m
s
b
l
s
b

											Type of Service (RFC 1349)				Diffserv (RFC2474)		ECN (RFC3168)
0	1	2	3	4	5	6	7	IP Precedence (RFC 791)		TOS Value	Min. Delay	Max. Thru'put	Max. Reliability	Min. Cost	DSCP	PHB	
0	0	0	0	0	0	0	0	0	Routine	0					0	Default / CS0	Not-ECT
0	0	0	0	0	0	1	0	0	Routine	2				✓	0	Default / CS0	ECT(0)
0	0	0	0	0	1	0	0	0	Routine	4			✓		1		Not-ECT
0	0	0	0	0	1	1	0	0	Routine	6			✓	✓	1		ECT(0)
0	0	0	0	1	0	0	0	0	Routine	8		✓			2		Not-ECT
0	0	0	0	1	0	1	0	0	Routine	10		✓		✓	2		ECT(0)
0	0	0	0	1	1	0	0	0	Routine	12		✓	✓		3		Not-ECT
0	0	0	0	1	1	1	0	0	Routine	14		✓	✓	✓	3		ECT(0)
0	0	0	1	0	0	0	0	0	Routine	16	✓				4		Not-ECT
0	0	0	1	0	0	1	0	0	Routine	18	✓			✓	4		ECT(0)
0	0	0	1	0	1	0	0	0	Routine	20	✓		✓		5		Not-ECT
0	0	0	1	0	1	1	0	0	Routine	22	✓		✓	✓	5		ECT(0)
0	0	0	1	1	0	0	0	0	Routine	24	Invalid *	Invalid *			6		Not-ECT
0	0	0	1	1	0	1	0	0	Routine	26	Invalid *	Invalid *		✓	6		ECT(0)
0	0	0	1	1	1	0	0	0	Routine	28	Invalid *	Invalid *	✓		7		Not-ECT
0	0	0	1	1	1	1	0	0	Routine	30	Invalid *	Invalid *	✓	✓	7		ECT(0)
0	0	1	0	0	0	0	0	1	Priority	32					8	CS1	Not-ECT
0	0	1	0	0	0	1	0	1	Priority	34				✓	8	CS1	ECT(0)
0	0	1	0	0	1	0	0	1	Priority	36			✓		9		Not-ECT
0	0	1	0	0	1	1	0	1	Priority	38			✓	✓	9		ECT(0)
0	0	1	0	1	0	0	0	1	Priority	40		✓			10	AF11	Not-ECT
0	0	1	0	1	0	1	0	1	Priority	42		✓		✓	10	AF11	ECT(0)
0	0	1	0	1	1	0	0	1	Priority	44		✓	✓		11		Not-ECT
0	0	1	0	1	1	1	0	1	Priority	46		✓	✓	✓	11		ECT(0)
0	0	1	1	0	0	0	0	1	Priority	48	✓				12	AF12	Not-ECT
0	0	1	1	0	0	1	0	1	Priority	50	✓			✓	12	AF12	ECT(0)
0	0	1	1	0	1	0	0	1	Priority	52	✓		✓		13		Not-ECT
0	0	1	1	0	1	1	0	1	Priority	54	✓		✓	✓	13		ECT(0)
0	0	1	1	1	0	0	0	1	Priority	56	Invalid *	Invalid *			14	AF13	Not-ECT
0	0	1	1	1	0	1	0	1	Priority	58	Invalid *	Invalid *		✓	14	AF13	ECT(0)
0	0	1	1	1	1	0	0	1	Priority	60	Invalid *	Invalid *	✓		15		Not-ECT
0	0	1	1	1	1	1	0	1	Priority	62	Invalid *	Invalid *	✓	✓	15		ECT(0)
0	1	0	0	0	0	0	0	2	Immediate	64					16	CS2	Not-ECT
0	1	0	0	0	0	1	0	2	Immediate	66				✓	16	CS2	ECT(0)
0	1	0	0	0	1	0	0	2	Immediate	68			✓		17		Not-ECT
0	1	0	0	0	1	1	0	2	Immediate	70			✓	✓	17		ECT(0)
0	1	0	0	1	0	0	0	2	Immediate	72		✓			18	AF21	Not-ECT
0	1	0	0	1	0	1	0	2	Immediate	74		✓		✓	18	AF21	ECT(0)
0	1	0	0	1	1	0	0	2	Immediate	76		✓	✓		19		Not-ECT
0	1	0	0	1	1	1	0	2	Immediate	78		✓	✓	✓	19		ECT(0)
0	1	0	1	0	0	0	0	2	Immediate	80	✓				20	AF22	Not-ECT
0	1	0	1	0	0	1	0	2	Immediate	82	✓			✓	20	AF22	ECT(0)
0	1	0	1	0	1	0	0	2	Immediate	84	✓		✓		21		Not-ECT
0	1	0	1	0	1	1	0	2	Immediate	86	✓		✓	✓	21		ECT(0)
0	1	0	1	1	0	0	0	2	Immediate	88	Invalid *	Invalid *			22	AF23	Not-ECT
0	1	0	1	1	0	1	0	2	Immediate	90	Invalid *	Invalid *		✓	22	AF23	ECT(0)
0	1	0	1	1	1	0	0	2	Immediate	92	Invalid *	Invalid *	✓		23		Not-ECT
0	1	0	1	1	1	1	0	2	Immediate	94	Invalid *	Invalid *	✓	✓	23		ECT(0)
0	1	1	0	0	0	0	0	3	Flash	96					24	CS3	Not-ECT
0	1	1	0	0	0	1	0	3	Flash	98				✓	24	CS3	ECT(0)
0	1	1	0	0	1	0	0	3	Flash	100			✓		25		Not-ECT
0	1	1	0	0	1	1	0	3	Flash	102			✓	✓	25		ECT(0)
0	1	1	0	1	0	0	0	3	Flash	104		✓			26	AF31	Not-ECT
0	1	1	0	1	0	1	0	3	Flash	106		✓		✓	26	AF31	ECT(0)
0	1	1	0	1	1	0	0	3	Flash	108		✓	✓		27		Not-ECT
0	1	1	0	1	1	1	0	3	Flash	110		✓	✓	✓	27		ECT(0)
0	1	1	1	0	0	0	0	3	Flash	112	✓				28	AF32	Not-ECT
0	1	1	1	0	0	1	0	3	Flash	114	✓			✓	28	AF32	ECT(0)
0	1	1	1	0	1	0	0	3	Flash	116	✓		✓		29		Not-ECT
0	1	1	1	0	1	1	0	3	Flash	118	✓		✓	✓	29		ECT(0)
0	1	1	1	1	0	0	0	3	Flash	120	Invalid *	Invalid *			30	AF33	Not-ECT
0	1	1	1	1	0	1	0	3	Flash	122	Invalid *	Invalid *		✓	30	AF33	ECT(0)
0	1	1	1	1	1	0	0	3	Flash	124	Invalid *	Invalid *	✓		31		Not-ECT
0	1	1	1	1	1	1	0	3	Flash	126	Invalid *	Invalid *	✓	✓	31		ECT(0)

Figure 2.45 Precedence, TOS, DSCP, and ECN ready reckoner



1	0	0	0	0	0	0	0	4	Flash Overide	128					32	CS4	Not-ECT
1	0	0	0	0	0	1	0	4	Flash Overide	130				✓	32	CS4	ECT(0)
1	0	0	0	0	1	0	0	4	Flash Overide	132			✓		33		Not-ECT
1	0	0	0	0	1	1	0	4	Flash Overide	134			✓	✓	33		ECT(0)
1	0	0	0	1	0	0	0	4	Flash Overide	136		✓			34	AF41	Not-ECT
1	0	0	0	1	0	1	0	4	Flash Overide	138		✓		✓	34	AF41	ECT(0)
1	0	0	0	1	1	0	0	4	Flash Overide	140		✓	✓		35		Not-ECT
1	0	0	0	1	1	1	0	4	Flash Overide	142		✓	✓	✓	35		ECT(0)
1	0	0	1	0	0	0	0	4	Flash Overide	144	✓				36	AF42	Not-ECT
1	0	0	1	0	0	1	0	4	Flash Overide	146	✓			✓	36	AF42	ECT(0)
1	0	0	1	0	1	0	0	4	Flash Overide	148	✓		✓		37		Not-ECT
1	0	0	1	0	1	1	0	4	Flash Overide	150	✓		✓	✓	37		ECT(0)
1	0	0	1	1	0	0	0	4	Flash Overide	152	Invalid *	Invalid *			38	AF43	Not-ECT
1	0	0	1	1	0	1	0	4	Flash Overide	154	Invalid *	Invalid *		✓	38	AF43	ECT(0)
1	0	0	1	1	1	0	0	4	Flash Overide	156	Invalid *	Invalid *	✓		39		Not-ECT
1	0	0	1	1	1	1	0	4	Flash Overide	158	Invalid *	Invalid *	✓	✓	39		ECT(0)
1	0	1	0	0	0	0	0	5	Critical	160					40	CS5	Not-ECT
1	0	1	0	0	0	1	0	5	Critical	162				✓	40	CS5	ECT(0)
1	0	1	0	0	1	0	0	5	Critical	164			✓		41		Not-ECT
1	0	1	0	0	1	1	0	5	Critical	166			✓	✓	41		ECT(0)
1	0	1	0	1	0	0	0	5	Critical	168		✓			42		Not-ECT
1	0	1	0	1	0	1	0	5	Critical	170		✓		✓	42		ECT(0)
1	0	1	0	1	1	0	0	5	Critical	172		✓	✓		43		Not-ECT
1	0	1	0	1	1	1	0	5	Critical	174		✓	✓	✓	43		ECT(0)
1	0	1	1	0	0	0	0	5	Critical	176	✓				44		Not-ECT
1	0	1	1	0	0	1	0	5	Critical	178	✓			✓	44		ECT(0)
1	0	1	1	0	1	0	0	5	Critical	180	✓		✓		45		Not-ECT
1	0	1	1	0	1	1	0	5	Critical	182	✓		✓	✓	45		ECT(0)
1	0	1	1	1	0	0	0	5	Critical	184	Invalid *	Invalid *			46	EF	Not-ECT
1	0	1	1	1	0	1	0	5	Critical	186	Invalid *	Invalid *		✓	46	EF	ECT(0)
1	0	1	1	1	1	0	0	5	Critical	188	Invalid *	Invalid *	✓		47		Not-ECT
1	0	1	1	1	1	1	0	5	Critical	190	Invalid *	Invalid *	✓	✓	47		ECT(0)
1	1	0	0	0	0	0	0	6	Internetwork Control	192					48	CS6	Not-ECT
1	1	0	0	0	0	1	0	6	Internetwork Control	194				✓	48	CS6	ECT(0)
1	1	0	0	0	1	0	0	6	Internetwork Control	196			✓		49		Not-ECT
1	1	0	0	0	1	1	0	6	Internetwork Control	198			✓	✓	49		ECT(0)
1	1	0	0	1	0	0	0	6	Internetwork Control	200		✓			50		Not-ECT
1	1	0	0	1	0	1	0	6	Internetwork Control	202		✓		✓	50		ECT(0)
1	1	0	0	1	1	0	0	6	Internetwork Control	204		✓	✓		51		Not-ECT
1	1	0	0	1	1	1	0	6	Internetwork Control	206		✓	✓	✓	51		ECT(0)
1	1	0	1	0	0	0	0	6	Internetwork Control	208	✓				52		Not-ECT
1	1	0	1	0	0	1	0	6	Internetwork Control	210	✓			✓	52		ECT(0)
1	1	0	1	0	1	0	0	6	Internetwork Control	212	✓		✓		53		Not-ECT
1	1	0	1	0	1	1	0	6	Internetwork Control	214	✓		✓	✓	53		ECT(0)
1	1	0	1	1	0	0	0	6	Internetwork Control	216	Invalid *	Invalid *			54		Not-ECT
1	1	0	1	1	0	1	0	6	Internetwork Control	218	Invalid *	Invalid *		✓	54		ECT(0)
1	1	0	1	1	1	0	0	6	Internetwork Control	220	Invalid *	Invalid *	✓		55		Not-ECT
1	1	0	1	1	1	1	0	6	Internetwork Control	222	Invalid *	Invalid *	✓	✓	55		ECT(0)
1	1	1	0	0	0	0	0	7	Network Control	224					56	CS7	Not-ECT
1	1	1	0	0	0	1	0	7	Network Control	226				✓	56	CS7	ECT(0)
1	1	1	0	0	1	0	0	7	Network Control	228			✓		57		Not-ECT
1	1	1	0	0	1	1	0	7	Network Control	230			✓	✓	57		ECT(0)
1	1	1	0	1	0	0	0	7	Network Control	232		✓			58		Not-ECT
1	1	1	0	1	0	1	0	7	Network Control	234		✓		✓	58		ECT(0)
1	1	1	0	1	1	0	0	7	Network Control	236		✓	✓		59		Not-ECT
1	1	1	0	1	1	1	0	7	Network Control	238		✓	✓	✓	59		ECT(0)
1	1	1	1	0	0	0	0	7	Network Control	240	✓				60		Not-ECT
1	1	1	1	0	0	1	0	7	Network Control	242	✓			✓	60		ECT(0)
1	1	1	1	0	1	0	0	7	Network Control	244	✓		✓		61		Not-ECT
1	1	1	1	0	1	1	0	7	Network Control	246	✓		✓	✓	61		ECT(0)
1	1	1	1	1	0	0	0	7	Network Control	248	Invalid *	Invalid *			62		Not-ECT
1	1	1	1	1	0	1	0	7	Network Control	250	Invalid *	Invalid *		✓	62		ECT(0)
1	1	1	1	1	1	0	0	7	Network Control	252	Invalid *	Invalid *	✓		63		Not-ECT
1	1	1	1	1	1	1	0	7	Network Control	254	Invalid *	Invalid *	✓	✓	63		ECT(0)
x	x	x	x	x	x	0	0	-	-	-	-	-	-	-	-	-	Not-ECT
x	x	x	x	x	x	0	1	-	-	-	-	-	-	-	-	-	ECT(1)
x	x	x	x	x	x	1	0	-	-	-	-	-	-	-	-	-	ECT(0)
x	x	x	x	x	x	1	1	-	-	-	-	-	-	-	-	-	CE

Figure 2.45 (Continued)

Notes

1. Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit.

2. The EF PHB was first defined in [RFC2598], however, the formal definition was subsequently determined to be incorrect and hence was superseded by RFC3246.