CHAPTER 6

Quality of Service with MPLS TE

Quality of service (QoS) and MPLS are, at a political level, similar. They're both technologies that have been gaining popularity in recent years. They both seem to be technologies that you either love or hate—some people are huge QoS fans, and others can't stand it. The same is true of MPLS—some people like it, and others don't.

At a technical level, though, QoS and MPLS are very different.

QoS is an umbrella term that covers network performance characteristics. As discussed in Chapter 1, "Understanding Traffic Engineering with MPLS," QoS has two parts:

- Finding a path through your network that can provide the service you offer
- Enforcing that service

The acronym QoS in respect to IP first showed up in RFC 1006, "ISO Transport Service on Top of the TCP: Version 3," published in 1987. The term QoS has been around for even longer, because it is a general term used to describe performance characteristics in networks. In the IP and MPLS worlds, the term QoS is most often used to describe a set of techniques to manage packet loss, latency, and jitter. QoS has been rather appropriately described as "managed unfairness": If you have contention for system resources, who are you unfair to, and why?

Two QoS architectures are in use today:

- Integrated Services (IntServ)
- Differentiated Services (DiffServ)

For various reasons, IntServ never scaled to the level it needed to get to for Internet-size networks. IntServ is fine for small- to medium-sized networks, but its need to make end-to-end, host-to-host, per-application microflows across a network means it can't grow to the level that large service provider networks need.

DiffServ, on the other hand, has proven quite scalable. Its use of classification on the edge and per-hop queuing and discard behaviors in the core means that most of the work is done at the edge, and you don't need to keep any microflow state in the core.

This chapter assumes that you understand QoS on an IP network. It concentrates on the integration of MPLS into the IP QoS spectrum of services. This means that you should be comfortable with acronyms such as CAR, LLQ, MDRR, MQC, SLA, and WRED in order to get the most out of this chapter. This chapter briefly reviews both the DiffServ architecture and the Modular QoS CLI (MQC), but see Appendix B, "CCO and Other References," if you want to learn more about the portfolio of Cisco QoS tools.

QoS, as used in casual conversation and in the context of IP and MPLS networks, is a method of packet treatment: How do you decide which packets get what service?

MPLS, on the other hand, is a switching method used to get packets from one place to another by going through a series of hops. Which hops a packet goes through can be determined by your IGP routing or by MPLS TE.

So there you have it—MPLS is about getting packets from one hop to another, and QoS (as the term is commonly used) is what happens to packets at each hop. As you can imagine, between two complex devices such as QoS and MPLS, a lot can be done.

This chapter covers five topics:

- The DiffServ architecture
- DiffServ's interaction with IP Precedence and MPLS EXP bits
- The treatment of EXP values in a label stack as packets are forwarded throughout a network
- A quick review of the Modular QoS CLI (MQC), which is how most QoS features on most platforms are configured
- Where DiffServ and MPLS TE intersect—the emerging DiffServ-Aware Traffic Engineering (DS-TE) devices and how they can be used to further optimize your network performance

DiffServ and MPLS TE

It is important to understand that the DiffServ architecture and the sections of this chapter that cover DiffServ and MPLS have nothing to do with MPLS TE. DiffServ is purely a method of treating packets differently at each hop. The DiffServ architecture doesn't care what control plane protocol a given label assignment comes from. Whether it's RSVP or LDP, or BGP, or something else entirely, the forwarding plane doesn't care. Why does this chapter exist then, if it's not about TE? Partly because MPLS TE and DiffServ treatment of MPLS packets go hand in hand in many network designs, and partly because of the existence of something called DS-TE, discussed later in this chapter.

The DiffServ Architecture

RFC 2475 defines an architecture for Differentiated Services—how to use DiffServ Code Point (DSCP) bits and various QoS mechanisms to provide different qualities of service in your network.

DiffServ has two major components:

- **Traffic conditioning**—Includes things such as policing, coloring, and shaping. Is done only at the edge of the network.
- **Per-hop behaviors**—Essentially consist of queuing, scheduling, and dropping mechanisms. As the name implies, they are done at every hop in the network.

Cisco IOS Software provides all sorts of different tools to apply these architecture pieces. You can configure most services in two ways—a host of older, disconnected, per-platform methods, and a newer, unified configuration set called the MQC. Only MQC is covered in this chapter. For information on the older configuration mechanisms, see Appendix B or the documentation on CCO. Not all platforms support MQC, so there might be times when you need to configure a service using a non-MQC configuration method; however, MQC is where all QoS configuration services are heading, so it's definitely worth understanding.

Traffic conditioning generally involves classification, policing, and marking, and per-hop behaviors deal with queuing, scheduling, and dropping. Each of these topics are discussed briefly.

Classification

The first step in applying the DiffServ architecture is to have the capability to classify packets. Classification is the act of examining a packet to decide what sort of rules it should be run through, and subsequently what DSCP or EXP value should be set on the packet.

Classifying IP Packets

Classifying IP packets is straightforward. You can match on just about anything in the IP header. Specific match capabilities vary by platform, but generally, destination IP address, source IP address, and DSCP values can be matched against. The idea behind DSCP is discussed in the section "DiffServ and IP Packets."

Classifying MPLS Packets

The big thing to keep in mind when classifying MPLS packets is that you can't match on anything other than the outermost EXP value in the label stack. There's no way to look past the MPLS header at the underlying IP packet and do any matching on or modification of that packet. You can't match on the label value in the top of the stack, and you can't match on TTL (just as you can't match on IP TTL). Finally, you can't do any matching of EXP values on any label other than the topmost label on the stack.

Policing

Policing involves metering traffic against a specified service contract and dealing with in-rate and out-of-rate traffic differently. One of the fundamental pieces of the DiffServ architecture is that you don't allow more traffic on your network than you have designed for, to make sure that you don't overtax the queues you've provisioned. This is generally done with policing, although it can also be done with shaping.

Policing is done on the edge of the network. As such, the packets coming into the network are very often IP packets. However, under some scenarios it is possible to receive MPLS-labeled packets on the edge of the network. For example, the Carrier Supporting Carrier architecture (see Appendix B) means that a provider receives MPLS-labeled packets from a customer.

Marking

The marking configuration is usually very tightly tied to the policing configuration. You can mark traffic as in-rate and out-of-rate as a result of policing traffic.

You don't need to police in order to mark. For example, you can simply define a mapping between the IP packet's DSCP value and the MPLS EXP bits to be used when a label is imposed on these packets. Another possibility is to simply mark all traffic coming in on an interface, regardless of traffic rate. This is handy if you have some customers who are paying extra for better QoS and some who are not. For those who are not, simply set the EXP to 0 on all packets from that customer.

Being able to set the EXP on a packet, rather than having to set the IP Precedence, is one of the advantages of MPLS. This is discussed in more detail in the sections "Label Stack Treatment" and "Tunnel Modes."

Queuing

Queuing is accomplished in different ways on different platforms. However, the good news is that you can treat MPLS EXP just like IP Precedence.

Multiple queuing techniques can be applied to MPLS, depending on your platform and code version:

- First In First Out (FIFO)
- Modified Deficit Round Robin (MDRR) (GSR platforms only)
- Class-Based Weighted Fair Queuing (CBWFQ) (most non-GSR platforms)
- Low-Latency Queuing (LLQ)

FIFO exists on every platform and every interface. It is the default on almost all of those interfaces.

MDRR, CBWFQ, and LLQ are configured using the MQC, just like most other QoS mechanisms on most platforms. Just match the desired MPLS EXP values in a class map and then configure a bandwidth or latency guarantee via the **bandwidth** or **priority** commands. The underlying scheduling algorithm (MDRR, CBWFQ/LLQ) brings the guarantee to life.

Queuing is one of two parts of what the DiffServ architecture calls *per-hop behaviors* (PHBs). A per-hop behavior is, surprisingly, a behavior that is implemented at each hop. PHBs have two fundamental pieces—queuing and dropping.

Dropping

Dropping is the other half of DiffServ's PHB. Dropping is important not only to manage queue depth per traffic class, but also to signal transport-level backoff to TCP-based applications. TCP responds to occasional packet drops by slowing down the rate at which it sends. TCP responds better to occasional drops than to tail drop after a queue is completely filled up. See Appendix B for more information.

Weighted Random Early Detection (WRED) is the DiffServ drop mechanism implemented on most Cisco platforms. WRED works on MPLS EXP just like it does on IP Precedence. See the next section for WRED configuration details.

As you can see, implementing DiffServ behavior with MPLS packets is no more and no less than implementing the same behavior with IP.

A Quick MQC Review

Over time, various Cisco IOS Software versions and hardware platforms have invented different ways to control QoS behaviors (policing, marking, queuing, and dropping). There have been many different CLIs that accomplish very similar things. A GSR QoS configuration was different from a 7200 configuration was different from a 7500 configuration, and so forth.

Along came MQC. MQC's goal is to unify all the QoS configuration steps into a single, flexible CLI that can be used across all IOS platforms.

This chapter covers only the MQC, because that is the preferred way to configure QoS. You can configure QoS services in other ways, but they are not covered here. If you have questions about a specific router model or software version, see the relevant documentation.

MQC is very powerful, and it can be a bit complex to the beginner. This section is a quick review of MQC. This is not a book on QoS, so this chapter doesn't go into great detail on the different types of services—merely how you can configure them with MQC.

MQC has three pieces:

- **Class map**—How you define what traffic you're interested in
- **Policy map**—What you do to the traffic defined in a class map
- **Service policy**—How you enable a policy map on an interface

The options you have within each of these pieces vary between code versions and platforms, so some of the options shown here might not be available to you. The basic idea remains constant, though.

Configuring a Class Map

The first step in using MQC is to build a class map. Not surprisingly, you do this with the **class-map** command:

```
vxr12(config)#class-map ?
  WORD       class-map name
  match-all  Logical-AND all matching statements under this classmap
  match-any  Logical-OR all matching statements under this classmapm
```

The **match-all** and **match-any** keywords let you specify whether traffic matches this class, if it matches *all* the rules in this class or *any* of them, respectively. The default is **match-all**.

You create a class by giving it a name. This puts you in a submode called config-cmap:

```
vxr12(config)#class-map voice
vxr12(config-cmap)#?
QoS class-map configuration commands:
  description  Class-Map description
  exit         Exit from QoS class-map configuration mode
  match        classification criteria
  no           Negate or set default values of a command
  rename       Rename this class-mapm
```

The most useful option under config-cmap is **match**, which lets you define the traffic you want to match with this class map. Its options are as follows:

```
vxr12(config-cmap)#match ?
  access-group         Access group
  any                  Any packets
  class-map            Class map
  cos                    IEEE 802.1Q/ISL class of service/user priority values
  destination-address    Destination address
  fr-de                  Match on Frame-relay DE bit
  input-interface        Select an input interface to match
  ip                     IP specific values
  mpls                   Multi Protocol Label Switching specific values
  not                    Negate this match result
  protocol               Protocol
  qos-group              Qos-group
  source-address         Source addressm
```

There are a few interesting options here. Note that you can match a class map; this lets you define hierarchical classes. A class called "Business," for example, can match an "Email" class and a "Payroll" class you define so that all business-class e-mail and all traffic to or from the payroll department gets the same treatment. You can also match **not**, which lets you match everything *except* a specific thing. You might now start to understand the awesome power of MQC.

Table 6-1 lists the matches of interest here.

Table 6-1 *Class Map Matches*

Match Type	Function	Configuration Syntax
access-group	Matches a named or numbered access list.	**access-group** {**name** *acl-name* | *acl-number 1-2699*}
ip	Matches IP packets of a specific DSCP or IP Precedence value (or a range of RTP ports, but that's not addressed here).	**ip** {**dscp** *name-or-number* | **precedence** *name or number* | **rtp** *lower-bound range*}
mpls	Matches MPLS packets that have a particular EXP value.	**mpls experimental** *0-7*

For the example in this chapter, create a simple LLQ policy matching MPLS EXP 5 traffic, and assume it is Voice over IP (VoIP) traffic. The necessary configuration is as follows:

```
class-map match-all voice
  match mpls experimental  5
policy-map llq
  class voice
    priority percent 30
```

This defines a class that matches any MPLS traffic that has the EXP bits set to 5 and then defines a policy for that traffic that gives the traffic 30 percent of a link's bandwidth. The policy map hasn't been enabled on an interface yet; you'll see that in a minute.

You can match multiple values with the same line using the following command sequence:

```
class-map match-any bronze-service
  match mpls experimental  0  1
```

This matches any packets that have an MPLS EXP of 0 or 1. As with route maps, multiple values specified within the same match clause (such as **match mpls experimental 0 1**) are implicitly ORed together; a packet can't have both EXP 0 *and* EXP 1, so this implicit ORing makes sense.

You can also match more than one criteria within a class, and you can use the **match-any** and **match-all** statements to decide how you want to match traffic. For example, the following policy matches traffic that has MPLS EXP 5 *or* traffic that entered the router on interface POS3/0:

```
class-map match-any gold
  match mpls experimental  5
  match input-interface POS3/0
```

The following policy matches any traffic that has MPLS EXP 5 *and* that came in on interface POS3/0:

```
class-map match-all gold
  match mpls experimental  5
  match input-interface POS3/0
```

See the difference? The first class map is **match-any** (the default), and the second is **match-all**.

The output for **show class-map** {*class-map-name*} shows you the configured class maps, as demonstrated in Example 6-1.

Example 6-1 *Displaying the Configured Class Maps*

```
vxr12#show class-map
 Class Map match-all gold (id 2)
   Match mpls experimental  5
   Match input-interface POS3/0

 Class Map match-any class-default (id 0)
   Match any

 Class Map match-all voice (id 3)
   Match mpls experimental  5
```

Note the **class-default** class in this list. **class-default** is a predefined class; it can be used to match any traffic that doesn't match any other class. You'll read more about this in the next section.

Configuring a Policy Map

After you define the class maps you want to match, you need to associate the class of traffic with a behavior. You create the behavior with the **policy-map** command, which, like **class-map**, puts you in a special submode:

```
vxr12(config)#policy-map ?
  WORD  policy-map name

vxr12(config)#policy-map llq
vxr12(config-pmap)#?
```

```
QoS policy-map configuration commands:
  class        policy criteria
  description  Policy-Map description
  exit         Exit from QoS policy-map configuration mode
  no           Negate or set default values of a command
  rename       Rename this policy-map
```

Under the config-pmap submode, you specify the class you want to match. This puts you in the config-pmap-c submode:

```
vxr12(config)#policy-map llq
vxr12(config-pmap)#class voice
vxr12(config-pmap-c)#?
QoS policy-map class configuration commands:
  bandwidth       Bandwidth
  exit            Exit from QoS class action configuration mode
  no              Negate or set default values of a command
  police          Police
  priority        Strict Scheduling Priority for this Class
  queue-limit     Queue Max Threshold for Tail Drop
  random-detect   Enable Random Early Detection as drop policy
  service-policy  Configure QoS Service Policy
  set             Set QoS values
  shape           Traffic Shaping
```

You can do all sorts of things here. Table 6-2 shows the options of interest for the purposes of this chapter.

Table 6-2 **policy-map** *Command Options*

Policy Type	Function	Configuration Syntax
bandwidth	Allocates the configured amount of bandwidth to the matched class. This is CBWFQ.	**bandwidth** {*bandwidth-kbps* | **remaining percent** *percentage* | **percent** *percentage*}
police	A token bucket policer that conforms to RFCs 2697 and 2698.	**police** {**cir** *cir*} [**bc** *conform-burst*] {**pir** *pir*} [**be** *peak-burst*] [**conform-action** *action* [**exceed-action** *action* [**violate-action** *action*]]]
priority	Allocates the configured amount of bandwidth to the matched class. This differs from the bandwidth option in that the priority keyword is LLQ.	**priority** {*bandwidth-kbps* | **percent** *percentage*} [*burst*]
random-detect	Sets the WRED parameters for this policy.	**random-detect** {**prec** *precedence min-threshold max-threshold* [*mark-probability-denominator*]
set	Sets IP Precedence, DSCP, or the EXP value on a packet.	**set** {**ip** {**dscp** *value* | **precedence** *value*} | {**mpls experimental** *value*}}
shape	Shapes the matched traffic to a certain profile.	**shape** {**average** *value* | **max-buffers** *value* | **peak** *value*}

The syntax for all those commands makes them look intimidating, but it's easy. Example 6-2 shows the LLQ policy you read about earlier, using the **voice** class map that's already been defined.

Example 6-2 *LLQ Policy with a Defined* **voice** *Class Map*

```
class-map match-any voice
  match mpls experimental  5
policy-map llq
  class voice
    priority percent 30
```

A predefined class called class-default implicitly matches anything not matched by a specific class; this gives you an easy way to give all unmatched packets the same treatment if you want to. For example, you can expand this class map to cover two types of service: voice and business. Business class is any data that has an MPLS EXP of 3 or 4, that is allocated 60 percent of the link bandwidth, and in which MPLS-based VoIP packets (EXP 5) get 30 percent of the link bandwidth. All other traffic is matched with **class-default**, which gets the remaining 10 percent of link bandwidth. This requires both a new class (to match the EXP 3 and 4 traffic) and a new policy map to define the treatment these classes get (see Example 6-3).

Example 6-3 *Defining a Policy Map with a Business Class*

```
class-map match-all business
  match mpls experimental  3  4
class-map match-all voice
  match mpls experimental  5
policy-map business-and-voice
  class voice
    priority percent 30
  class business
   bandwidth percent 60
  class class-default
   bandwidth percent 10
```

The **show policy-map** command gives you a fair amount of detail about the policy maps defined on a router and the class maps inside them. Example 6-4 shows two policy maps—llq and business-and-voice; the llq policy matches the voice class, and the business-and-voice policy matches the voice and business classes.

Example 6-4 *Displaying Policy Map Information*

```
vxr12#show policy-map
  Policy Map business-and-voice
    Class voice
      Weighted Fair Queuing
            Strict Priority
            Bandwidth 30 (%)
    Class business
```

Example 6-4 *Displaying Policy Map Information (Continued)*

```
      Weighted Fair Queuing
            Bandwidth 60 (%) Max Threshold 64 (packets)
    Class class-default
      Weighted Fair Queuing
            Bandwidth 10 (%) Max Threshold 64 (packets)

  Policy Map llq
    Class voice
      Weighted Fair Queuing
            Strict Priority
            Bandwidth 30 (%)
```

Configuring a Service Policy

This is the easiest part of the MQC. So far, you've seen a few class definitions and a policy definition. Now all that's left is to apply the policy to an interface, as shown in Example 6-5.

Example 6-5 *Applying a Service Policy to a Router Interface*

```
vxr12(config-if)#service-policy ?
  history  Keep history of QoS metrics
  input    Assign policy-map to the input of an interface
  output   Assign policy-map to the output of an interface

vxr12(config-if)#service-policy out
vxr12(config-if)#service-policy output ?
  WORD  policy-map name

vxr12(config-if)#service-policy output llq
```

That's it. Note that you can have both an inbound and an outbound service policy.

show policy-map interface *interface* gives you details about which policy maps are applied to an interface, as demonstrated in Example 6-6.

Example 6-6 *Determining Which Policy Maps Are Applied to an Interface*

```
vxr12#show policy-map interface pos3/0
 POS3/0

  Service-policy output: llq

    Class-map: voice (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: mpls experimental  5
      Weighted Fair Queuing
        Strict Priority
        Output Queue: Conversation 264
        Bandwidth 30 (%)
        Bandwidth 46500 (kbps) Burst 1162500 (Bytes)
```

continues

Example 6-6 *Determining Which Policy Maps Are Applied to an Interface (Continued)*

```
        (pkts matched/bytes matched) 0/0
        (total drops/bytes drops) 0/0

    Class-map: class-default (match-any)
      21 packets, 15744 bytes
      5 minute offered rate 3000 bps, drop rate 0 bps
      Match: any
```

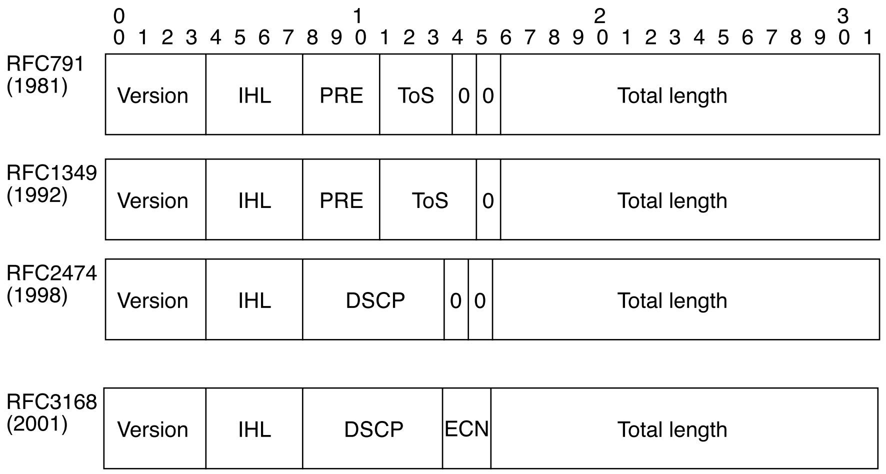
If you're still not comfortable with either MQC or the underlying QoS concepts such as CBWFQ and LLQ, it is highly recommended that you check out the references in Appendix B.

DiffServ and IP Packets

QoS markings in the IP packet have evolved over time. The IP header has always contained a byte known as the *type of service* (ToS) byte. The 8 bits within this byte have evolved and have been redefined over time. It's a little confusing to start with, because the ToS *byte* has contained multiple things, some of which were called ToS *bits*.

Figure 6-1 shows the first four bytes of the IP header as defined in the original IP header (RFC 791, circa 1981) and as redefined in RC 1349 (circa 1992), RFC 2474 (circa 1998), and RFC 3168 (circa 2001).

Figure 6-1 *Evolution of the First 4 Bytes of the IP Header*



Originally, the IP header had 3 precedence bits and 3 ToS bits, as well as 2 unused bits. The precedence bits were (and still are) used to make various decisions about packet treatment. Precedence values 0 through 5 are designated for user data; precedence values 6 and 7 are reserved to make network control traffic. RFC 1349 reassigned one of the unused bits to the ToS bits, giving the IP header a total of 3 precedence bits, 4 ToS bits, and one unused bit.

Use of ToS bits was never well-defined or well-deployed. The original intent was to be able to mark packets that preferred low delay, high throughput, or high-reliability paths, but service architectures were never designed or built around these bits.

The DiffServ set of RFCs (RFCs 2474 and 2475) redefined the entire ToS byte. The ToS byte now contains 6 bits of information that declare desired packet treatment—DSCP bits. The remaining two bits in the ToS byte are used for a TCP mechanism called Explicit Congestion Notification (ECN), which isn't addressed here but is defined in RFC 3168.

When discussing QoS and the ToS byte, some people use IP Precedence terminology, and others use DiffServ terminology. This chapter uses DiffServ terminology, but we recognize that not everyone is familiar with DiffServ and its code points. If you are familiar with IP Precedence but are new to DiffServ, two things you should know are

- How to map DSCP bits to IP Precedence bits, and vice versa
- What services DSCP offers above and beyond mapping to IP Precedence

The first part is easy—how to map DSCP bits to IP Precedence bits. See Table 6-3.

Table 6-3 *Mapping DSCP Bits to IP Precedence Bits*

IP Precedence (Decimal)	IP Precedence (Bits)	DSCP (Decimal)	DSCP (Bits)
0	000	0	000000
1	001	8	001000
2	010	16	010000
3	011	24	011000
4	100	32	100000
5	101	40	101000
6	110	48	110000
7	111	56	111000

If you're accustomed to dealing with IP Precedence values 0, 1, 2, and 5 (for example), you just need to refer to them as DSCP values 0, 8, 16, and 40. It's easy: To convert IP Precedence to DSCP, just multiply by 8.

The terminology is simple, too. The eight IP Precedence values are called *classes,* and the DSCP bits that map to them (in Table 6-3) are called *Class Selector Code Points* (CSCP). Sometimes you see these class selectors abbreviated as CS (CS1, CS2, CS5, and so on). These are referred to simply as *class selectors;* the term Class Selector Code Point isn't all that widely used.

In addition to the eight class selectors that are defined, RFCs 2597 and 2598 define 13 additional DSCP values—12 Assured Forwarding (AF) values and an Expedited Forwarding (EF) value (see Table 6-4). The decimal values are shown for reference only; almost all discussions of DSCP use the names given in the Name column.

Table 6-4 *Additional DSCP Values in RFCs 2597 and 2598*

Name	DSCP (Decimal)	DSCP (Bits)
Default	0	000000
AF11	10	001010
AF12	12	001100
AF13	14	001110
AF21	18	010010
AF22	20	010100
AF23	22	010110
AF31	26	011010
AF32	28	011100
AF33	30	011110
AF41	34	100010
AF42	36	100100
AF43	38	100110
EF	46	101110

There are 12 AF values, all in the format Af*xy,* where *x* is the class number and *y* is the drop precedence. There are four classes (AF1*y* through AF4*y*), each of which has three drop precedences (AF*x*1 through AF*x*3). AF is a method of providing low packet loss within a given traffic rate, but it makes minimal guarantees about latency.

EF is a defined behavior that asks for low-delay, low-jitter, low-loss service. EF is typically implemented using some form of LLQ. Only one EF class is defined, because it doesn't make sense to have more than one class whose goals are minimal delay and jitter. These two classes would compete with each other for the same resources.

DiffServ would be extremely simple if it weren't for all the confusing terminology, not all of which is covered here. If you want to learn more about the full set of DiffServ terminology and the DiffServ architecture, see the references in Appendix B.

DiffServ and MPLS Packets

One thing you might have noticed is that there are 6 DSCP bits and only 3 EXP bits. Because only eight possible EXP values and 64 possible DSCP values (21 of which are currently defined) exist, how do you offer DSCP services over MPLS?

On a frame-mode network (as opposed to a cell-mode network), you are stuck with the 3 EXP bits; you need to map multiple DSCP classes to these EXP bits. However, this has operationally not yet proven to be an issue in production networks, because hardly any QoS deployments offer services that can't be provisioned with the 3 MPLS EXP bits.

Work is being done to define something called L-LSPs (Label-Only Inferred PSC LSPs) that will help alleviate this problem. The basic idea behind L-LSPs is that you use both the EXP bits and the label to define different service classes. This is actually how label-controlled ATM MPLS mode with multi-VC works. However, this book doesn't cover cell mode, because, as of this writing, there is no MPLS TE cell mode implementation available.

L-LSPs for frame-mode MPLS aren't discussed in this chapter because they're not yet implemented or even fully defined. See RFC 3270, "Multiprotocol Label Switching (MPLS) Support of Differentiated Services," for more details on how L-LSPs work.

Label Stack Treatment

As mentioned earlier in this book, MPLS has 3 EXP bits in the label header that are used in much the same way as IP Precedence bits or the DSCP CS bits.

You should consider three cases. The first case is when IP packets enter an MPLS network. This is known as the *ip-to-mpls* case, often written as *ip2mpls*.

The second case is when packets that already have one or more labels have their label stack manipulated. This is known as the *mpls-to-mpls* case, or *mpls2mpls*.

The third case is when packets exit an MPLS network and have all their labels removed. This is known as the *mpls-to-ip* path, or *mpls2ip*.

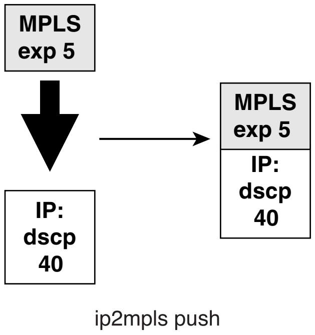
ip2mpls

Packets entering an MPLS network have one or more labels applied to an underlying IP packet. This is an *ip2mpls push,* because labels are pushed onto an unlabeled IP packet.

By default, when Cisco IOS Software pushes labels onto an IP packet, the most significant bits in the DiffServ field (the IP Precedence bits) are copied to the EXP field of all imposed labels.

Figure 6-2 shows an ip2mpls push.

Figure 6-2 *ip2mpls Push*



mpls2mpls

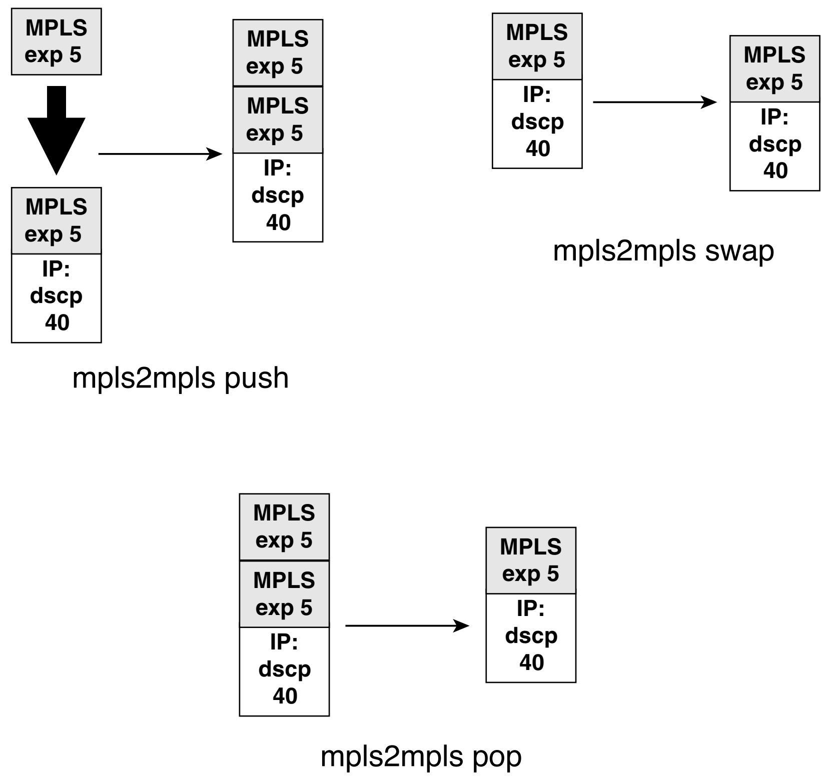
Similarly, when a label is pushed onto a packet that already has a label, the EXP value from the underlying label is copied into the EXP field of the newly imposed label. This is known as the *mpls2mpls* path.

Three actions are possible in the mpls2mpls path:

- **Push**—An mpls2mpls push is when one or more labels are added to an already-labeled packet.
- **Swap**—An mpls2mpls swap is when the topmost label on a packet is swapped for another label.
- **Pop**—An mpls2mpls pop is when one or more labels are removed from a packet, but at least one label is left.

Figure 6-3 illustrates these three cases.

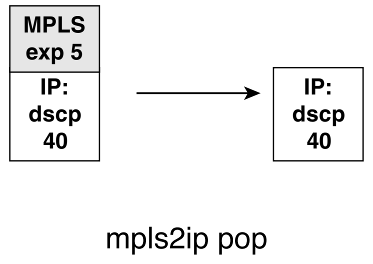
Figure 6-3 *mpls2mpls Operations*



mpls2ip

Packets leaving an MPLS network have their label stack removed; this is known as the *mpls-to-ip* operation, or *mpls2ip*. The only operation in the mpls2ip case is a pop, as illustrated in Figure 6-4.

Figure 6-4 *mpls2ip Pop*



EXP and DSCP Are Independent

As you can see, the default Cisco IOS Software behavior is to propagate the IP Precedence bits (the three most significant DSCP bits) through your network.

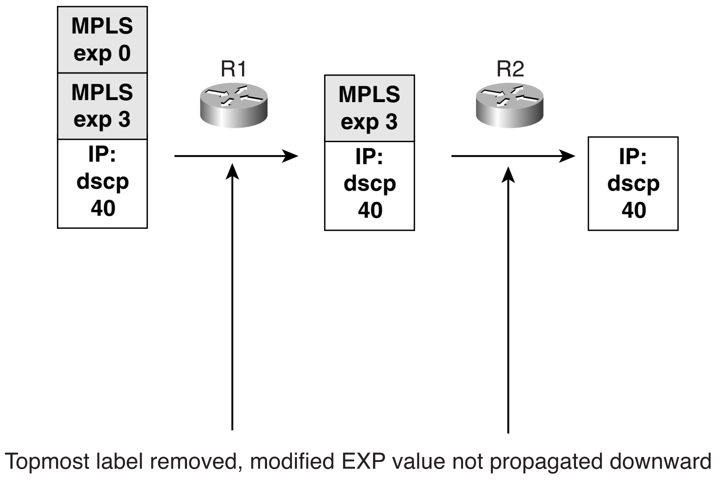
NOTE To avoid confusion and to make things simpler, this chapter calls the bits that are copied from the IP header to the EXP field the "IP Precedence bits." This is deliberate; there is no DiffServ term that means "only the three most significant bits of the DSCP field." There are CSs, but to talk about copying the DSCP CS to EXP isn't accurate, because CS definitions cover all six of the DSCP bits. So, when you see text such as "IP Precedence is copied to EXP," this reminds you that only the three most significant DSCP bits are copied to the MPLS EXP field.

Two things interact to define the basic EXP and IP Precedence interaction:

- Cisco IOS Software allows you to set the EXP value on a label independently of any IP or EXP values that might already be set.
- In both the ip2mpls and mpls2mpls pop cases, nothing is done to the lower-level packet when labels are removed.

These two facts, when considered together, mean that if you set a packet's EXP values differently from the underlying IP packet, or if you change EXP values on the top of a label stack midway through your network, these changes are not propagated downward. Figure 6-5 shows this in action, by default, in both the mpls2mpls pop and mpls2ip.

Figure 6-5 *Default EXP Treatment When Popping*



Generally, this is acceptable. However, you might want to do things differently in some cases.

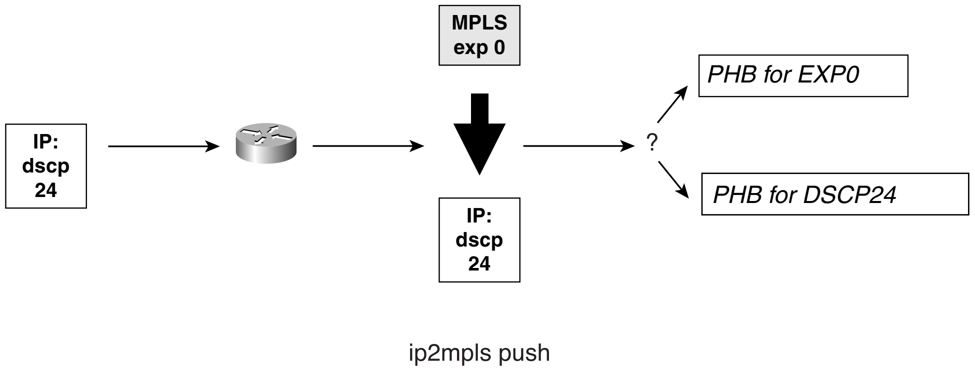
In Figure 6-5, you might want to preserve the fact that the outermost label has an EXP value of 0, even after that outermost label is removed and the underlying label (with an EXP of 3) is exposed.

This means copying the value of EXP 0 *down* and overwriting the value of EXP 3 in the underlying label. The motivation and mechanics of this scenario are explained in the upcoming "Tunnel Modes" section.

Per-Hop Behaviors in the ip2mpls and mpls2ip Cases

Another thing to consider is how your packets are treated if the outermost label on a packet has its EXP value set to something other than the EXP or IP Precedence underneath it. If you push a label of EXP 0 onto a packet with EXP 3 (or IP Precedence 3), how do you treat the packet as it leaves the box? Is the packet given PHB treatment as if it has EXP 0, or as if it has IP Precedence 3/DSCP 24? Figure 6-6 illustrates this question.

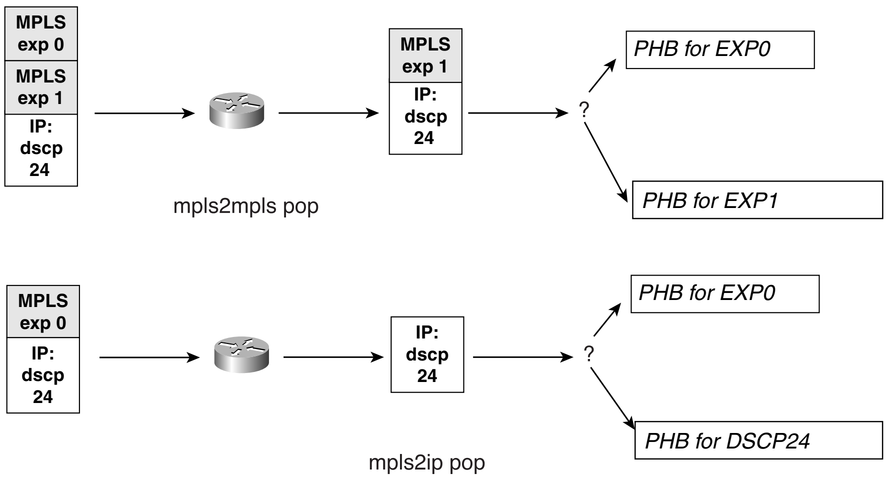
Figure 6-6 *PHB Decision on Label Push*



As it turns out, the case of label imposition is an easy one. The router doing the imposition always treats the packet according to the new outgoing PHB indicator. In Figure 6-6, this means that the packet is given PHB treatment for EXP 0.

The mpls2mpls and mpls2ip pop cases are not as straightforward. Figure 6-7 illustrates these cases.

Figure 6-7 *PHB Decision on Label Pop*



In some cases, you will want the resulting packet to receive treatment according to the label that was removed, and in other cases, you will want the resulting packet to receive treatment according to the outermost indicator in whatever remains after the POP (MPLS EXP in the case of mpls2mpls and IP DSCP in the case of mpls2ip).

All these label operations can be confusing. The next section aims to unify them and make things easier to understand.

Tunnel Modes

So far, you've seen two independent questions:

- How is the queuing indicator (IP Precedence or MPLS EXP) propagated up and down the label stack and upon exit from the MPLS network?
- In the mpls2mpls and mpls2ip pop cases, which PHB does the resulting packet get?

This section covers reasons why you might want to enforce different behaviors, as well as the mechanisms being defined to standardize those behaviors.

Currently, a set of behaviors are being defined that give you a set of mechanisms to control EXP values in various scenarios. These mechanisms are called *tunnel modes*. Three tunnel modes are defined in RFC 3270:

- Uniform
- Short-Pipe
- Pipe

These modes are defined in the same IETF draft that defines L-LSP—"MPLS Support of Differentiated Services" (*draft-ietf-mpls-diff-ext*), which might well be an RFC by the time you read this. Because this is a developing technology, this chapter does not cover CLI commands for configuring tunnel modes. However, it's worth understanding the concepts behind tunnel modes so that you can design your network accordingly. If you're interested in the current state of affairs regarding tunnel mode in Cisco IOS Software, check CCO for availability.

Uniform Mode

In Uniform mode, any changes made to the EXP value of the topmost label on a label stack are propagated both *upward* as new labels are added and *downward* as labels are removed. The idea here is that the network is a single DiffServ domain, so any changes made to the EXP values on the MPLS packet in transit are supposed to be applied to all labels underneath the packet, as well as to the underlying IP packet.

The rules for Uniform mode are as follows:

- On imposition, copy the DSCP/EXP upward.
- On disposition, copy the removed EXP downward to both IP packet and MPLS (if stacked).

The question of deciding which PHB is applied on label disposition is irrelevant; no matter when you apply the PHB according to the received EXP value or the outgoing EXP/DSCP value, they're both the same.

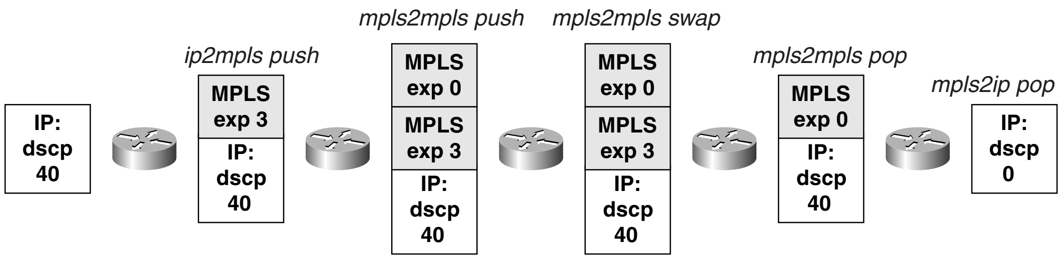
Table 6-5 shows the EXP treatment in Uniform mode.

Table 6-5 *EXP Manipulation in Uniform Mode*

	Push	**Swap**	**Pop**
ip2mpls	Copies the IP Precedence into the newly imposed label.	N/A	N/A
mpls2mpls	Copies the received EXP into the newly imposed EXP.	Copies the received EXP into the newly imposed EXP.	Copies the removed EXP into the newly revealed label.
mpls2ip	N/A	N/A	Copies the removed EXP into the DSCP.

Figure 6-8 illustrates a case where a new label is pushed onto the stack with EXP 0 and, as this label is popped off, the underlying label (previously EXP 3) is changed to EXP 0.

Figure 6-8 *Uniform Mode Packet Handling*



You would apply Uniform mode when both the IP and the MPLS network are in the same DiffServ domain. You use Uniform mode if you want a change in EXP somewhere in your network to affect how the IP packet is treated after it exits the MPLS portion of the network.

Short-Pipe Mode

Short-Pipe mode is useful for ISPs implementing their own QoS policy independent of their customer's QoS policy. The IP Precedence bits on an IP packet are propagated upward into the label stack as labels are added. When labels are swapped, the existing EXP value is kept. If the topmost EXP value is changed, this change is propagated downward only within the label stack, not to the IP packet.

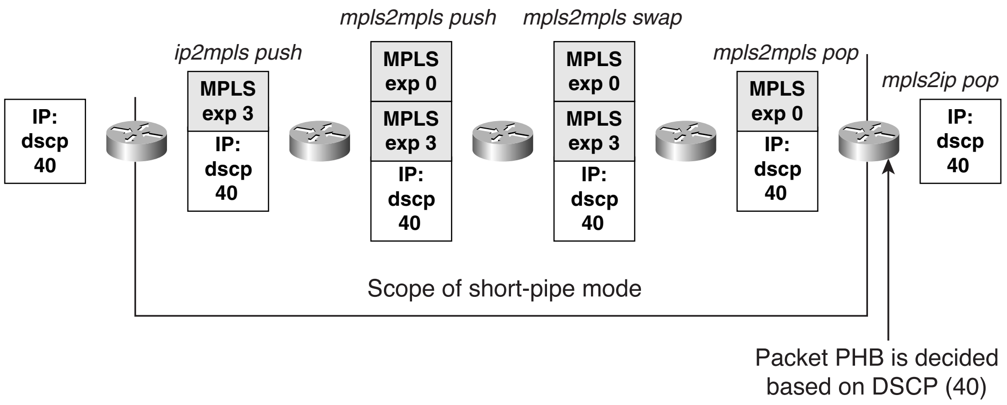
Table 6-6 shows EXP treatment in Short-Pipe mode.

Table 6-6 *EXP Manipulation in Short-Pipe Mode*

	Push	**Swap**	**Pop**
ip2mpls	Copies the IP Precedence into the EXP.	N/A	N/A
mpls2mpls	Copies the received EXP into the newly imposed EXP.	Copies the received EXP into the newly imposed EXP.	Copies the removed EXP into the newly revealed EXP.
mpls2ip	N/A	N/A	Doesn't modify the DSCP; selects the PHB based on the DSCP.

Figure 6-9 illustrates this.

Figure 6-9 *Short-Pipe Mode Packet Handling*



The only difference between Uniform and Short-Pipe mode is that any changes to the label stack EXP are propagated throughout the MPLS network, but—in Short-Pipe mode—the underlying IP packet DSCP is not touched.

What about PHBs? There are two rules in Short-Pipe mode:

- In the mpls2mpls pop case, the received EXP is propagated downward, so the question of which EXP value decides the PHB is moot.
- In the mpls2ip pop case, the PHB is decided based on the DSCP on the IP packet that's revealed after the label stack is removed.

The assumption in Short-Pipe mode is that the link between the provider and the customer is where the mpls2ip processing happens. Because the customer is paying for the link between the provider and the customer, the customer is the one in charge of the queuing on that link, so the outgoing packet in the mpls2ip case is queued according to the DSCP the customer sent the packet into the network with.

Pipe Mode

Pipe mode is just like Short-Pipe mode, except the PHB on the mpls2ip link is selected based on the removed EXP value rather than the recently-exposed DSCP value. The underlying DSCP in the packet is not touched, but the mpls2ip path does not consider the DSCP for queuing on the egress link.

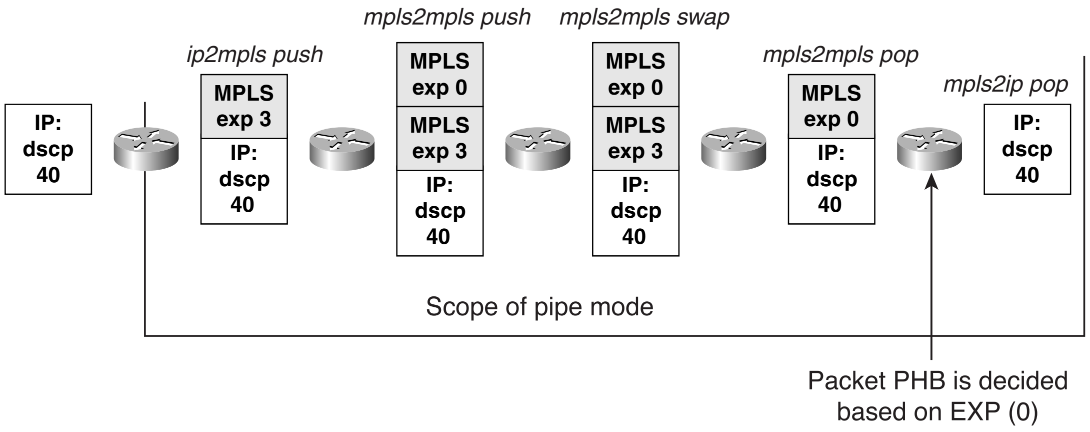
Table 6-7 shows EXP treatment in Pipe mode.

Table 6-7 *EXP Manipulation in Pipe Mode*

	Push	Swap	Pop
ip2mpls	Copies the IP Precedence into the EXP.	N/A	N/A
mpls2mpls	Copies the received EXP into the newly imposed EXP.	Copies the received EXP into the newly imposed EXP.	Copies the removed EXP into the newly revealed EXP.
mpls2ip	N/A	N/A	Doesn't modify DSCP; selects the PHB based on the EXP.

Figure 6-10 illustrates this.

Figure 6-10 *Pipe Mode Packet Handling*



When might you want to use Pipe mode? Pipe mode is useful if the ISP is the one that decides what the PHB selection is on the link immediately exiting the MPLS network. Typically, this is in a managed CPE scenario, in which the ISP does not want to extend MPLS all the way out to the CPE but wants PHB selection control over the link to the CPE.

DiffServ-Aware Traffic Engineering (DS-TE)

So far, you've seen that DiffServ service with MPLS packets is basically the same thing as with IP packets, with a few differences in configuration. Overall, MPLS packets with a given EXP setting are treated just like IP packets with a given IP Precedence setting.

There's more to MPLS TE and DiffServ than just applying IP mechanisms to MPLS packets, though. The rest of this book spends a great deal of time showing you that there's benefit in making a headend resource-aware, because the headend can then intelligently pick paths through the network for its traffic to take. However, after you add QoS, you're almost back at square one. Things are better than they were—you can steer IP traffic away from the IGP shortest path in a resource-aware fashion. What you can't do is steer traffic *per QoS*. If you have traffic destined for a particular router, all that traffic follows the same path, regardless of the DSCP/EXP settings on that packet.

To expand on this point further, suppose you are a service provider offering four classes of service: Low latency, Gold, Silver, and Bronze. Low latency being self-explanatory, suppose Gold is defined as guaranteed delivery. Edge QoS is applied to mark EXP bits to differentiate your low-latency traffic from Gold and other traffic. Now, if the forwarding is based on IGP best path alone, all your traffic, regardless of what class it belongs to, is forwarded to a downstream neighbor dictated by your routing table. Even if you use MPLS TE and **mpls traffic-eng autoroute announce**, you are still limited by the routing table and how it decides to forward traffic.

The problem with this implementation is that when you have a congested link at a downstream node along the forwarding path, even though most low-latency traffic might get through, some of the Gold traffic might get dropped. This congestion knowledge is localized at the downstream node and is not propagated back to the edge devices that send traffic down that path. As a result, your edges continue to send traffic to the same downstream router that continues to drop some of your Gold traffic. What is needed to fix this situation is *per-class call admission control*—and this is exactly what you get when you combine DiffServ and TE. Surprisingly enough, this combination is called DS-TE.

This might lead you to believe that all you need to do is mark the EXP bits according to your QoS policy and let them ride over TE tunnels, conquering all the problems.

Not quite. The problem with TE, as it's been discussed so far, is that it doesn't do admission control on a per-QoS class basis.

TE certainly offers call admission control in addition to the PHB offered by DiffServ. This takes care of the first problem—sending more traffic down a certain path than there is available bandwidth—while queuing your higher-priority traffic ahead of your low-priority traffic.

A second problem is that there might be contention between different high-priority traffic streams. For example, if you sold two voice pipes to two customers, both with a low-latency requirement, if you forward both streams down the same path which is experiencing congestion, both streams might be affected. Remember, for voice, it is better to drop a call than to get degraded service. So how does DS-TE solve this problem?

DS-TE allows you to advertise more than one pool of available resources for a given link—a global pool and things called *subpools*. A subpool is a subset of link bandwidth that is available for a specific purpose.

The current DS-TE implementation allows you to advertise one subpool. Think of this as a pool with which you can advertise resources for a separate queue. The recommended use of the DS-TE subpool is to advertise bandwidth in your low-latency queue on a link, but you can do whatever you like with this subpool. The rest of this chapter assumes that you are advertising LLQ space using the subpool.

It's important to understand that DS-TE and its subpools are, like the rest of MPLS TE, control-plane mechanisms only. If you reserve 10 Mbps of global bandwidth, it's probably not a good idea to send 100 Mbps down that LSP. DS-TE behaves the same way. Building a separate control-plane reservation for a subpool doesn't mean that any special queuing policy is enforced as a result of this reservation. The actual queuing behavior at every hop is still controlled by regular DiffServ mechanisms such as LLQ; what DS-TE buys you is purely the ability to reserve queue bandwidth, rather than just link bandwidth, in the control plane. This ability lets you build TE-LSPs that specifically reserve subpool bandwidth and carry only LLQ traffic, and in effect build a second network on top of the one you already have. You have a network of physical interfaces and global pools and a network of subpools. This lets you get better resource utilization out of your network.

Going back to the example of two voice streams that might contend for the same low-latency bandwidth along the same path, this would not happen with DS-TE. When you try to build the second LSP requesting low-latency bandwidth and that bandwidth is not available along a certain path, the LSP is not signaled. If you used dynamic path options to configure your DS-TE tunnels, using CSPF on the router might find you an alternative path that meets the subpool reservation.

But how do you configure all this?

Configuring DS-TE

The best way to understand DS-TE is to see it in action. DS-TE is easy to configure. The configuration pieces provided in this section are for a 7200 series router with an OC-3 interface. The OC-3 interface is advertising 150 MB of global RSVP bandwidth. 45 Mbps (30 percent) of this is subpool bandwidth. The OC-3 also has 45 Mbps configured in a low-latency queue. Other configurations vary by platform and according to the goal you're trying to achieve.

There are five pieces to configuring DS-TE:

- Per-link subpool bandwidth availability
- Per-link scheduling
- Headend subpool bandwidth requirements

- Headend tunnel admission control
- Tunnel preemption

Per-Link Subpool Bandwidth Availability

On each hop that you want to advertise a subpool, you configure the following command:

```
ip rsvp bandwidth interface-kbps sub-pool kbps
```

This is merely an extension of the **ip rsvp bandwidth** command discussed in earlier chapters. The *interface-kbps* parameter is the amount of bandwidth (in Kbps) on the interface to be reserved. The range is 1 to 10,000,000. **sub-pool** *kbps* is the amount of bandwidth (in Kbps) on the interface to be reserved as a portion of the total. The range is from 1 to the value of *interface-kbps*.

For this specific example of an OC-3 with 150 Mbps in the global pool and 45 Mbps in the subpool, the necessary configuration is as follows:

```
ip rsvp bandwidth 150000 sub-pool 45000
```

Per-Link Scheduling

Advertising a subpool in DS-TE doesn't change any packet queuing behavior on an interface. You need to configure the forwarding-plane LLQ mechanisms in addition to the control-plane subpool mechanisms. The subpool advertisement and LLQ capacity are usually set to the same value, but you can set them differently if you want to do fancy things such as oversubscription or undersubscription.

Per-link scheduling is just whatever LLQ mechanism exists in your platform. With MQC, the **priority** keyword builds a low-latency queue. The MQC LLQ configuration for this example is as follows:

```
class-map match-all voice
  match mpls experimental  5
policy-map llq
  class voice
    priority percent 30
interface POS3/0
 service-policy output llq
```

Headend Subpool Bandwidth Requirements

On the tunnel headend, you use the following command:

```
tunnel mpls traffic-eng bandwidth sub-pool kbps
```

This is the same thing as the **tunnel mpls traffic-eng bandwidth** *kbps* command that was covered earlier in this book, except that you are telling the headend to do its path calculation and bandwidth reservation based on the advertised subpool bandwidth.

You are allowed to have only one type of reservation per tunnel. If you try to configure a tunnel with the command **tunnel mpls traffic-eng bandwidth** followed by the command **tunnel mpls traffic-eng bandwidth sub-pool**, the **sub-pool** command overwrites the global pool command.

Headend Tunnel Admission Control

The next piece is controlling what traffic goes down the tunnel. There are three steps to this:

Step 1 Make sure no more traffic enters your network than you have sold.

Step 2 Make sure that the only traffic to enter the DS-TE tunnel is traffic that belongs there.

Step 3 Make sure your TE tunnel reservation accurately matches your traffic requirements.

The first step is a generic DiffServ architecture component; it has nothing to do with MPLS TE. You can't have any kind of QoS guarantees if you provision your network for a certain amount of traffic and then send to that network far more traffic than you have provisioned for.

The second step is where things start getting TE-specific. The general idea is that if you've decided that your DS-TE subpool is for EXP 5 traffic, you send only EXP 5 traffic down the DS-TE tunnel. Why? Because if you send nonsubpool traffic down a DS-TE tunnel, this traffic interferes with other traffic at every hop. Let's say you have a tunnel for 10 Mbps of subpool bandwidth that you're mapping to a low-latency queue for EXP 5 traffic. If you send 100 Mbps of EXP 0 traffic down this DS-TE tunnel, that EXP 0 traffic is put into the same queue as other EXP 0 traffic, but it is not reserved from the network. You'll read more about this in the section "Forwarding DS-TE Traffic Down a Tunnel."

The reason for the third step should be obvious: If you don't reserve bandwidth in accordance with what you're actually sending, and your reservations are designed to track LLQ capacity at every hop, you run the risk of overloading the LLQ and providing poor service. One way to make this adjustment is to size your DS-TE tunnels based on the amount of DS-TE tunnel bandwidth you have provisioned for. The other way is to use something such as auto-bandwidth (see Chapter 5, "Forwarding Traffic Down Tunnels") to adjust tunnel size based not on the provisioned load, but on the real-life traffic load.

Tunnel Preemption

The last thing you have to do is make sure that the subpool tunnel can preempt nonsubpool tunnels. This is done with the command **tunnel mpls traffic-eng priority**, which was covered in Chapter 3, "Information Distribution." Why do you have to allow DS-TE tunnels to preempt non-DS-TE tunnels? Because of the way subpool bandwidth is advertised. The subpool is, as the name implies, a *subset* of the global bandwidth pool on an interface, not a separate bandwidth pool. If you don't allow the DS-TE LSP to preempt a global LSP, the global LSP can reserve bandwidth that would then no longer be available for the subpool.

If the subpool bandwidth were advertised as a *separate* bandwidth pool, rather than as a *subset* of existing bandwidth, you could easily end up with bandwidth fragmentation, as the following example shows.

If you have a link configured with **ip rsvp bandwidth 150000 sub-pool 45000**, this means that you are advertising that you have 150 Mbps (150,000 Kbps) of global reservable bandwidth on the link, and 45 Mbps of that 150 Mbps is subpool bandwidth. At this point, the link is advertising the output shown in Example 6-7.

Example 6-7 *Available Bandwidth Before Any Reservations Have Been Made*

```
vxr12#show mpls traffic-eng topology 192.168.1.12

IGP Id: 0168.0001.0012.00, MPLS TE Id:192.168.1.12 Router Node  id 2
      link[0 ]:Nbr IGP Id: 0168.0001.0001.00, nbr_node_id:3, gen:1
          frag_id 0, Intf Address:2.3.4.12, Nbr Intf Address:2.3.4.1
          TE metric:10, IGP metric:10, attribute_flags:0x0
          physical_bw: 155000 (kbps), max_reservable_bw_global: 150000 (kbps)
          max_reservable_bw_sub: 45000 (kbps)

                                Global Pool       Sub Pool
               Total Allocated  Reservable        Reservable
               BW (kbps)        BW (kbps)         BW (kbps)
               ---------------  -----------       ----------
        bw[0]:            0          150000            45000
        bw[1]:            0          150000            45000
        bw[2]:            0          150000            45000
        bw[3]:            0          150000            45000
        bw[4]:            0          150000            45000
        bw[5]:            0          150000            45000
        bw[6]:            0          150000            45000
        bw[7]:            0          150000            45000
```

This shows a link with no reservations across it, a global pool of 150 Mbps, and a subpool of 45 Mbps.

Let's send three reservations across this link: a reservation for 60 Mbps from the global pool, 20 Mbps from the subpool, and 50 Mbps for the global pool, in that order. After the first reservation, the router advertises the bandwidth displayed in the output of **show mpls traffic-eng topology**, as shown in Example 6-8. All LSPs are set up with the default priority of 7.

Example 6-8 *Available Bandwidth with 60 Mbps Reserved*

```
vxr12#show mpls traffic-eng topology 192.168.1.12

IGP Id: 0168.0001.0012.00, MPLS TE Id:192.168.1.12 Router Node  id 2
      link[0 ]:Nbr IGP Id: 0168.0001.0001.00, nbr_node_id:3, gen:3
          frag_id 0, Intf Address:2.3.4.12, Nbr Intf Address:2.3.4.1
```

continues

Example 6-8 *Available Bandwidth with 60 Mbps Reserved (Continued)*

```
          TE metric:10, IGP metric:10, attribute_flags:0x0
          physical_bw: 155000 (kbps), max_reservable_bw_global: 150000 (kbps)
          max_reservable_bw_sub: 45000 (kbps)

                               Global Pool       Sub Pool
               Total Allocated   Reservable        Reservable
               BW (kbps)         BW (kbps)         BW (kbps)
               ---------------   -----------       ----------
        bw[0]:            0          150000             45000
        bw[1]:            0          150000             45000
        bw[2]:            0          150000             45000
        bw[3]:            0          150000             45000
        bw[4]:            0          150000             45000
        bw[5]:            0          150000             45000
        bw[6]:            0          150000             45000
        bw[7]:        60000           90000             45000
```

This is because 60 Mbps was reserved from the global pool (at the default setup and holding priorities of 7/7), leaving 90 Mbps available on the link. Of that 90 Mbps, 45 Mbps can be reserved as subpool bandwidth should anybody want it.

The next reservation to come across takes 20 Mbps from the subpool. This means that the available bandwidth on the interface is now as advertised in Example 6-9.

Example 6-9 *Available Bandwidth with an Additional 20 Mbps of Subpool Bandwidth Reserved*

```
vxr12#show mpls traffic-eng topology 192.168.1.12

IGP Id: 0168.0001.0012.00, MPLS TE Id:192.168.1.12 Router Node  id 2
      link[0 ]:Nbr IGP Id: 0168.0001.0001.00, nbr_node_id:3, gen:4
          frag_id 0, Intf Address:2.3.4.12, Nbr Intf Address:2.3.4.1
          TE metric:10, IGP metric:10, attribute_flags:0x0
          physical_bw: 155000 (kbps), max_reservable_bw_global: 150000 (kbps)
          max_reservable_bw_sub: 45000 (kbps)

                               Global Pool       Sub Pool
               Total Allocated   Reservable        Reservable
               BW (kbps)         BW (kbps)         BW (kbps)
               ---------------   -----------       ----------
        bw[0]:            0          150000             45000
        bw[1]:            0          150000             45000
        bw[2]:            0          150000             45000
        bw[3]:            0          150000             45000
        bw[4]:            0          150000             45000
        bw[5]:            0          150000             45000
        bw[6]:            0          150000             45000
        bw[7]:        80000           70000             25000
```

The available bandwidth on the interface is an important point to understand. 20 Mbps of additional bandwidth is reserved from the available link bandwidth. This brings the total allocated bandwidth from 60 Mbps to 80 Mbps and decreases the total available link bandwidth from 90 Mbps to 70 Mbps. It just so happens that the 20 Mbps reserved was taken from the subpool, which means that the available subpool bandwidth is now 25 Mbps. This is because the subpool is considered a subset of the global pool, rather than a whole separate pool.

The third reservation for 50 Mbps from the global pool brings the available link bandwidth down to that advertised in Example 6-10.

Example 6-10 *Available Bandwidth with an Additional 50 Mbps of Subpool Bandwidth Reserved*

```
vxr12#show mpls traffic-eng topology 192.168.1.12

IGP Id: 0168.0001.0012.00, MPLS TE Id:192.168.1.12 Router Node  id 2
      link[0 ]:Nbr IGP Id: 0168.0001.0001.00, nbr_node_id:3, gen:5
          frag_id 0, Intf Address:2.3.4.12, Nbr Intf Address:2.3.4.1
          TE metric:10, IGP metric:10, attribute_flags:0x0
          physical_bw: 155000 (kbps), max_reservable_bw_global: 150000 (kbps)
          max_reservable_bw_sub: 45000 (kbps)

                                 Global Pool       Sub Pool
               Total Allocated   Reservable        Reservable
               BW (kbps)         BW (kbps)         BW (kbps)
               ---------------   -----------       ----------
        bw[0]:            0           150000            45000
        bw[1]:            0           150000            45000
        bw[2]:            0           150000            45000
        bw[3]:            0           150000            45000
        bw[4]:            0           150000            45000
        bw[5]:            0           150000            45000
        bw[6]:            0           150000            45000
        bw[7]:       130000            20000            20000
```

The total allocated bandwidth is 130 Mbps (60 Mbps + 20 Mbps + 50 Mbps). 20 Mbps of that 130 Mbps is allocated from the subpool, and 110 Mbps (60 Mbps + 50 Mbps) is allocated from the rest of the global pool.

If you ponder this for a while, making the subpool a subset of the global pool rather than a whole separate pool makes sense. If the global pool and the subpool bandwidth were segregated, instead of advertising a pool of 150 Mbps available bandwidth (45 Mbps of which is in a subpool), the router would advertise a 105 Mbps pool and a 45 Mbps pool.

If you then have the same reservations, in the same order as before, the bandwidth reservations and availability look like what is shown in Table 6-8.

Table 6-8 *What If You Advertised Global and Subpool as Separate Pools?*

Reservation	Global Available Bandwidth	Subpool Available Bandwidth
No reservations	105 Mbps	45 Mbps
60 Mbps global	45 Mbps	45 Mbps
20 Mbps subpool	45 Mbps	25 Mbps
50 Mbps global	?????	??????

But what about the third reservation for 50 Mbps from the global bandwidth pool? It fails to get through, even though a total of 70 Mbps of bandwidth (45 + 25) is available.

Advertising DS-TE bandwidth as a subpool rather than as a separate pool means that you can reserve *up to* 45 Mbps for the subpool, but if it's not in use, other nonsubpool tunnels can use it.

If you want subpool tunnels to be able to take up to 45 Mbps of bandwidth no matter what else is reserved on the link, the subpool tunnels need to be able to preempt other tunnels on the link. However, see the section in Chapter 9, "Network Design with MPLS TE," titled, "The Packing Problem," before you start messing with tunnel priorities.

Forwarding DS-TE Traffic Down a Tunnel

Chapter 5 covered three different methods of forwarding traffic down a tunnel:

- Static routes
- Policy-based routing (PBR)
- Autoroute

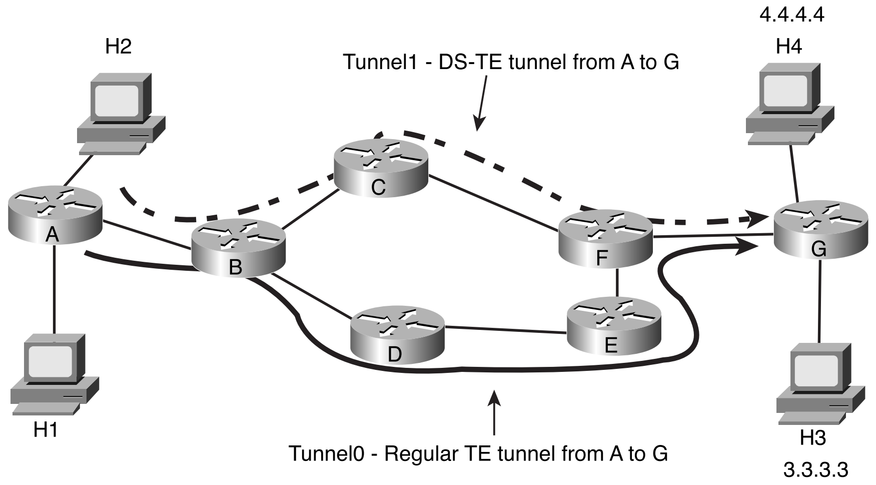
How do you forward traffic down a DS-TE tunnel? The only thing different between regular TE tunnels and DS-TE tunnels is the subpool from which they reserve bandwidth. However, there are some things you need to think about when trying to get traffic to a DS-TE tunnel.

All three of these methods work the same way on DS-TE tunnels as they do on regular TE tunnels. Out of the three, autoroute is generally the easiest method to use; it requires only one command on the headend, and all traffic destined for or behind the tail is sent down the tunnel.

The problem with autoroute and DS-TE is that autoroute is not always granular enough to do what you need to do. If you have both TE and DS-TE tunnels to the same destination, you can't autoroute on them both and have them do what you probably want.

Consider the topology shown in Figure 6-11.

Figure 6-11 *Topology in Which You Can't Enable Autoroute on a DS-TE Tunnel*


4.4.4.4
H4
H2
Tunnel1 - DS-TE tunnel from A to G
A
B
C
D
E
F
G
H1
H3
3.3.3.3
Tunnel0 - Regular TE tunnel from A to G

In Figure 6-11, H2 has VoIP traffic destined for H4, and H1 has regular data traffic destined for H3. This network has two TE tunnels. Tunnel0 is a regular TE tunnel that follows the path A→B→D→E→F→G, and Tunnel1 is a DS-TE tunnel from A to G that follows the path A→B→C→F→G. You want all traffic for H4 to go down Tunnel1 and all other traffic to go down Tunnel0.

What happens if you enable autoroute on both tunnels? Both H3 and H4 become reachable over *both* TE tunnels, as shown in Example 6-11.

Example 6-11 *H3 and H4 Can Be Reached Over Both TE Tunnels*

```
RouterA#show ip route 3.3.3.3
Routing entry for 3.3.3.3/32
  Known via "isis", distance 115, metric 50, type level-2
  Redistributing via isis
  Last update from 192.168.1.1 on Tunnel0, 00:00:19 ago
  Routing Descriptor Blocks:
  * 192.168.1.1, from 192.168.1.1, via Tunnel1
      Route metric is 50, traffic share count is 10
    192.168.1.1, from 192.168.1.1, via Tunnel0
      Route metric is 50, traffic share count is 1

RouterA#show ip route 4.4.4.4
Routing entry for 4.4.4.4/32
  Known via "isis", distance 115, metric 50, type level-2
  Redistributing via isis
  Last update from 192.168.1.1 on Tunnel0, 00:00:31 ago
  Routing Descriptor Blocks:
  * 192.168.1.1, from 192.168.1.1, via Tunnel1
      Route metric is 50, traffic share count is 10
    192.168.1.1, from 192.168.1.1, via Tunnel0
      Route metric is 50, traffic share count is 1
```

This is not what you want. The only way to solve this problem is with a static route—something like this:

```
ip route 3.3.3.3 255.255.255.255 Tunnel1
```

This gives you the routing table shown in Example 6-12.

Example 6-12 *Routing Table for Router A After Configuring a Static Route*

```
RouterA#show ip route 3.3.3.3
Routing entry for 3.3.3.3/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
  * directly connected, via Tunnel1
      Route metric is 0, traffic share count is 1
```

But what happens if you have lots of hosts that receive VoIP traffic? Static routes are a reasonable solution for a small-scale problem, but managing large numbers of static routes can be a nightmare.

You can try to solve the static route problem by aggregating your VoIP devices into shared subnets. Instead of 200 VoIP devices and a /32 route for each device, number all the VoIP devices into the same /24. Then you have only one static route.

Even then, you still have the same problem of scale. It's not always possible to summarize all your devices so neatly. Plus, you need to consider the issue of multiple sources. Consider the network shown in Figure 6-12.

Figure 6-12 *Topology in Which You Can't Enable Autoroute on a DS-TE Tunnel*

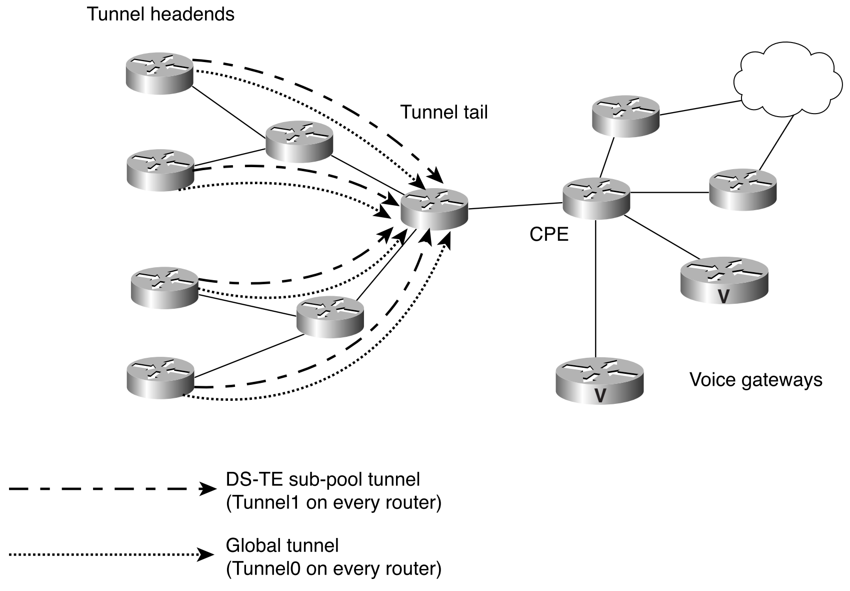


Figure 6-12 shows four routers that each have two TE tunnels terminating on the same tail node. Behind the tunnel tail is a CPE router, behind the CPE are two VoIP gateways and two routers, and behind the two routers is the rest of the customer network. You can't enable autoroute on the headends, because you only want to send traffic to the VoIP gateways down the subpool tunnels. Managing this network with static routes is messy. You need to have two static routes on each tunnel headend, and every time you add a new VoIP gateway, you need to add static routes to every headend.

Luckily, you can more easily manage this routing. You can use recursive routes and route maps to manipulate the next hop of a given route based on whether the route is for a VoIP gateway. The procedure is as follows:

Step 1 Configure a second loopback address on the tunnel tail.

Step 2 Have the CPE advertise separate routes for the VoIP routers via BGP to the tunnel tail, with a different community for the VoIP routes.

Step 3 The tunnel tail changes the next hop for the VoIP routes to be its second loopback address.

Step 4 All the tunnel headends point to a route to the second loopback address down their DS-TE subpool tunnels.

Step 5 Recursive routing ensures that the forwarding works properly.

Figure 6-13 illustrates this scenario.

Figure 6-13 *Topology in Which You Can't Enable Autoroute on a DS-TE Tunnel*

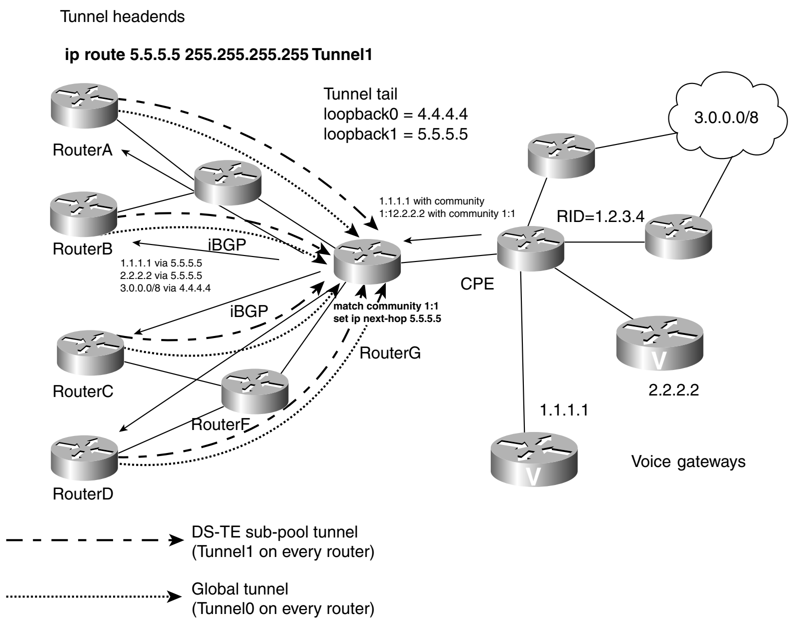


Figure 6-13 is pretty busy, so Example 6-13 shows the necessary configuration snippets.

Example 6-13 *Key Configurations for the Network Shown in Figure 6-13*

```
On the CPE:
router bgp 100
 neighbor 4.4.4.4 route-map setcom-voice out

route-map setcom-voice
  match ip address 101
  set community 1:1

access-list 101 permit ip host 1.1.1.1 host 255.255.255.255
```

```
On RouterG (the tunnel tail)
router bgp 100
  neighbor 1.2.3.4 route-map set-nh in

route-map set-nh
 match community 1
 set ip next-hop 5.5.5.5

ip community-list 1 permit 1:1
```

```
On Routers A, B, C, and D
ip route 5.5.5.5 255.255.255.255 Tunnel1
```

Doing things this way means that the only one who needs to know which routers are VoIP routers and which aren't is the CPE; the administration of any static configuration is limited to the CPE. As more VoIP and CPE routers are added off RouterG, the tunnel headends don't need to change anything. As more tunnel tails are added, the tunnel headends need to add only a single static route per DS-TE tunnel to that tail.

You can also be more flexible than using a community. RouterG can change the next hop based on any criteria, because BGP is flexible in what it supports. Rather than a community, you could change the next hop based on the destination AS, for example, or perhaps a transit AS. The choice is up to you.

Summary

This chapter covered several things—a brief look at the DiffServ architecture and its interaction with IP and MPLS packets, the MQC, and DS-TE.

Many tools are available to help you build QoS policies on your network. Here are the two main points to take away from this chapter:

- Applying the DiffServ architecture to MPLS packets is not much different from applying it to IP packets.

- DS-TE helps you solve the same problem for QoS that “regular” MPLS TE solves for link bandwidth, and in the same manner.

You can put as much or as little work as you like into your QoS policy. Whatever you can do with IP, you can do with MPLS. The resource awareness and source-based forwarding that MPLS TE gives you let you do even more with QoS in your network.