the RTP packets on port 38060. After receiving Alice's INVITE message, Bob sends an SIP response message, which resembles an HTTP response message. This response SIP message is also sent to the SIP port 5060. Bob's response includes a 200 OK as well as an indication of his IP address, his desired encoding and packetization for reception, and his port number to which the audio packets should be sent. Note that in this example Alice and Bob are going to use different audio-encoding mechanisms: Alice is asked to encode her audio with GSM whereas Bob is asked to encode his audio with PCM μ-law. After receiving Bob's response, Alice sends Bob an SIP acknowledgment message. After this SIP transaction, Bob and Alice can talk. (For visual convenience, Figure 7.12 shows Alice talking after Bob, but in truth they

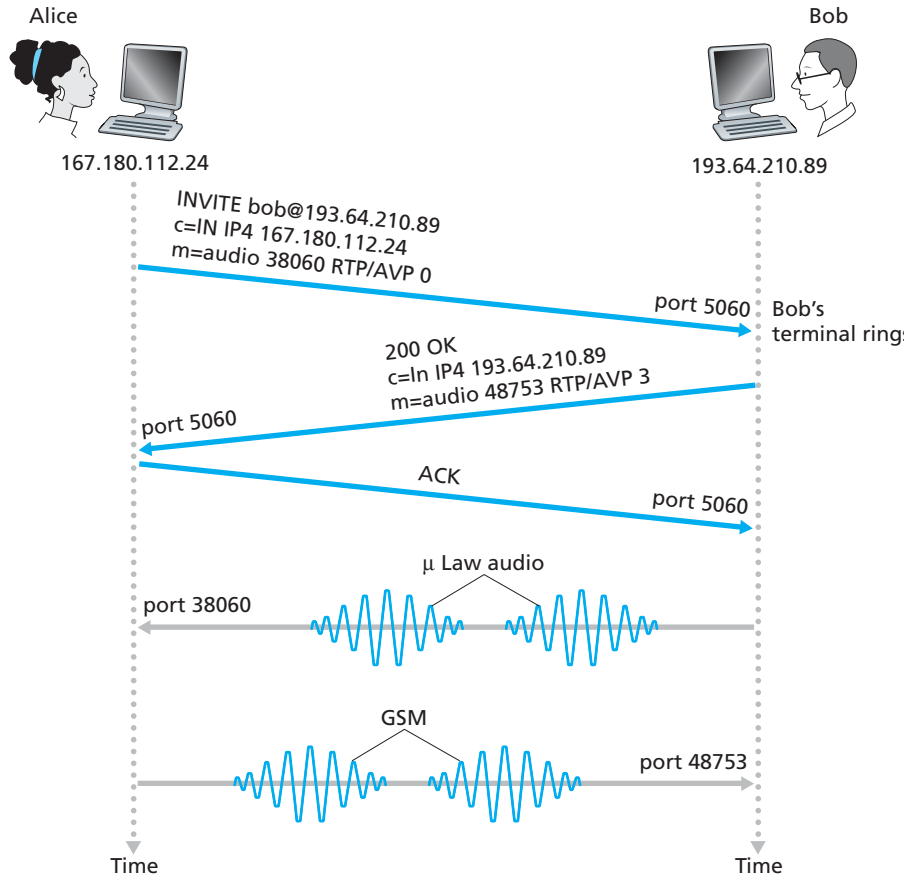


Figure 7.12 ♦ SIP call establishment when Alice knows Bob's IP address

would normally talk at the same time.) Bob will encode and packetize the audio as requested and send the audio packets to port number 38060 at IP address 167.180.112.24. Alice will also encode and packetize the audio as requested and send the audio packets to port number 48753 at IP address 193.64.210.89.

From this simple example, we have learned a number of key characteristics of SIP. First, SIP is an out-of-band protocol: The SIP messages are sent and received in sockets that are different from those used for sending and receiving the media data. Second, the SIP messages themselves are ASCII-readable and resemble HTTP messages. Third, SIP requires all messages to be acknowledged, so it can run over UDP or TCP.

In this example, let's consider what would happen if Bob does not have a PCM μ-law codec for encoding audio. In this case, instead of responding with 200 OK, Bob would likely respond with a 600 Not Acceptable and list in the message all the codecs he can use. Alice would then choose one of the listed codecs and send another INVITE message, this time advertising the chosen codec. Bob could also simply reject the call by sending one of many possible rejection reply codes. (There are many such codes, including "busy," "gone," "payment required," and "forbidden.")

SIP Addresses

In the previous example, Bob's SIP address is sip:bob@193.64.210.89. However, we expect many—if not most—SIP addresses to resemble e-mail addresses. For example, Bob's address might be sip:bob@domain.com. When Alice's SIP device sends an INVITE message, the message would include this e-mail-like address; the SIP infrastructure would then route the message to the IP device that Bob is currently using (as we'll discuss below). Other possible forms for the SIP address could be Bob's legacy phone number or simply Bob's first/middle/last name (assuming it is unique).

An interesting feature of SIP addresses is that they can be included in Web pages, just as people's e-mail addresses are included in Web pages with the mailto URL. For example, suppose Bob has a personal homepage, and he wants to provide a means for visitors to the homepage to call him. He could then simply include the URL sip:bob@domain.com. When the visitor clicks on the URL, the SIP application in the visitor's device is launched and an INVITE message is sent to Bob.

SIP Messages

In this short introduction to SIP, we'll not cover all SIP message types and headers. Instead, we'll take a brief look at the SIP INVITE message, along with a few common header lines. Let us again suppose that Alice wants to initiate a VoIP call to Bob, and this time Alice knows only Bob's SIP address, bob@domain.com, and

does not know the IP address of the device that Bob is currently using. Then her message might look something like this:

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

The INVITE line includes the SIP version, as does an HTTP request message. Whenever an SIP message passes through an SIP device (including the device that originates the message), it attaches a Via header, which indicates the IP address of the device. (We'll see soon that the typical INVITE message passes through many SIP devices before reaching the callee's SIP application.) Similar to an e-mail message, the SIP message includes a From header line and a To header line. The message includes a Call-ID, which uniquely identifies the call (similar to the message-ID in e-mail). It includes a Content-Type header line, which defines the format used to describe the content contained in the SIP message. It also includes a Content-Length header line, which provides the length in bytes of the content in the message. Finally, after a carriage return and line feed, the message contains the content. In this case, the content provides information about Alice's IP address and how Alice wants to receive the audio.

Name Translation and User Location

In the example in Figure 7.12, we assumed that Alice's SIP device knew the IP address where Bob could be contacted. But this assumption is quite unrealistic, not only because IP addresses are often dynamically assigned with DHCP, but also because Bob may have multiple IP devices (for example, different devices for his home, work, and car). So now let us suppose that Alice knows only Bob's e-mail address, bob@domain.com, and that this same address is used for SIP-based calls. In this case, Alice needs to obtain the IP address of the device that the user bob@domain.com is currently using. To find this out, Alice creates an INVITE message that begins with INVITE bob@domain.com SIP/2.0 and sends this message to an **SIP proxy**. The proxy will respond with an SIP reply that might include the IP address of the device that bob@domain.com is currently using. Alternatively, the reply might include the IP address of Bob's voicemail box, or it might include a URL of a Web page (that says "Bob is sleeping. Leave me alone!"). Also, the result returned by the proxy might depend on the caller: If the call is from Bob's wife, he

might accept the call and supply his IP address; if the call is from Bob's mother-in-law, he might respond with the URL that points to the I-am-sleeping Web page!

Now, you are probably wondering, how can the proxy server determine the current IP address for bob@domain.com? To answer this question, we need to say a few words about another SIP device, the **SIP registrar**. Every SIP user has an associated registrar. Whenever a user launches an SIP application on a device, the application sends an SIP register message to the registrar, informing the registrar of its current IP address. For example, when Bob launches his SIP application on his PDA, the application would send a message along the lines of:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

Bob's registrar keeps track of Bob's current IP address. Whenever Bob switches to a new SIP device, the new device sends a new register message, indicating the new IP address. Also, if Bob remains at the same device for an extended period of time, the device will send refresh register messages, indicating that the most recently sent IP address is still valid. (In the example above, refresh messages need to be sent every 3600 seconds to maintain the address at the registrar server.) It is worth noting that the registrar is analogous to a DNS authoritative name server: The DNS server translates fixed host names to fixed IP addresses; the SIP registrar translates fixed human identifiers (for example, bob@domain.com) to dynamic IP addresses. Often SIP registrars and SIP proxies are run on the same host.

Now let's examine how Alice's SIP proxy server obtains Bob's current IP address. From the preceding discussion we see that the proxy server simply needs to forward Alice's INVITE message to Bob's registrar/proxy. The registrar/proxy could then forward the message to Bob's current SIP device. Finally, Bob, having now received Alice's INVITE message, could send an SIP response to Alice.

As an example, consider Figure 7.13, in which jim@umass.edu, currently working on 217.123.56.89, wants to initiate a Voice-over-IP (VoIP) session with keith@upenn.edu, currently working on 197.87.54.21. The following steps are taken: (1) Jim sends an INVITE message to the umass SIP proxy. (2) The proxy does a DNS lookup on the SIP registrar upenn.edu (not shown in diagram) and then forwards the message to the registrar server. (3) Because keith@upenn.edu is no longer registered at the upenn registrar, the upenn registrar sends a redirect response, indicating that it should try keith@eurecom.fr. (4) The umass proxy sends an INVITE message to the eurecom SIP registrar. (5) The eurecom registrar knows the IP address of keith@eurecom.fr and forwards the INVITE message to the host 197.87.54.21, which is running Keith's SIP client. (6–8) An SIP response is sent back through registrars/proxies to the SIP client on 217.123.56.89. (9) Media is sent

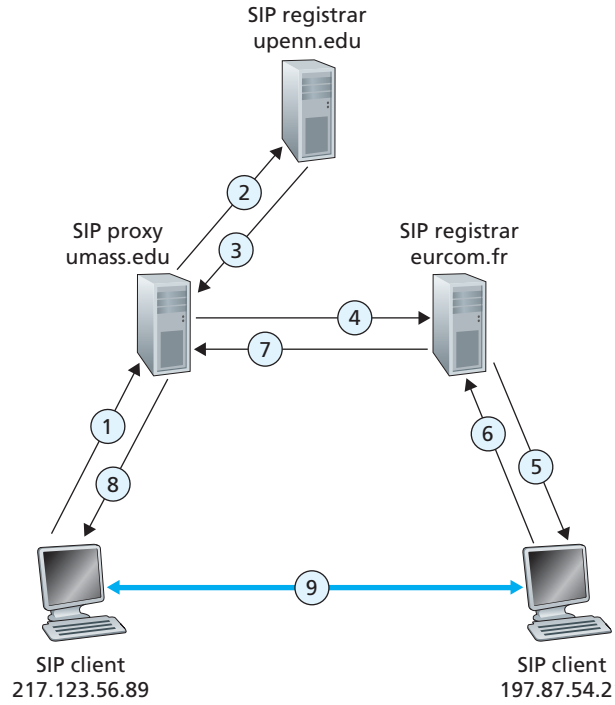


Figure 7.13 ♦ Session initiation, involving SIP proxies and registrars

directly between the two clients. (There is also an SIP acknowledgment message, which is not shown.)

Our discussion of SIP has focused on call initiation for voice calls. SIP, being a signaling protocol for initiating and ending calls in general, can be used for video conference calls as well as for text-based sessions. In fact, SIP has become a fundamental component in many instant messaging applications. Readers desiring to learn more about SIP are encouraged to visit Henning Schulzrinne's SIP Web site [Schulzrinne-SIP 2012]. In particular, on this site you will find open source software for SIP clients and servers [SIP Software 2012].

7.5 Network Support for Multimedia

In Sections 7.2 through 7.4, we learned how application-level mechanisms such as client buffering, prefetching, adapting media quality to available bandwidth, adaptive playout, and loss mitigation techniques can be used by multimedia applications

to improve a multimedia application's performance. We also learned how content distribution networks and P2P overlay networks can be used to provide a *system-level* approach for delivering multimedia content. These techniques and approaches are all designed to be used in today's best-effort Internet. Indeed, they are in use today precisely because the Internet provides only a single, best-effort class of service. But as designers of computer networks, we can't help but ask whether the *network* (rather than the applications or application-level infrastructure alone) might provide mechanisms to support multimedia content delivery. As we'll see shortly, the answer is, of course, "yes"! But we'll also see that a number of these new network-level mechanisms have yet to be widely deployed. This may be due to their complexity and to the fact that application-level techniques together with best-effort service and properly dimensioned network resources (for example, bandwidth) can indeed provide a "good-enough" (even if not-always-perfect) end-to-end multimedia delivery service.

Table 7.4 summarizes three broad approaches towards providing network-level support for multimedia applications.

- *Making the best of best-effort service.* The application-level mechanisms and infrastructure that we studied in Sections 7.2 through 7.4 can be successfully used in a well-dimensioned network where packet loss and excessive end-to-end

Approach	Granularity	Guarantee	Mechanisms	Complexity	Deployment to date
Making the best of best-effort service.	all traffic treated equally	none, or soft	application-layer support, CDNs, overlays, network-level resource provisioning	minimal	everywhere
Differentiated service	different classes of traffic treated differently	none, or soft	packet marking, policing, scheduling	medium	some
Per-connection Quality-of-Service (QoS) Guarantees	each source-destination flows treated differently	soft or hard, once flow is admitted	packet marking, policing, scheduling; call admission and signaling	light	little

Table 7.4 ♦ Three network-level approaches to supporting multimedia applications

delay rarely occur. When demand increases are forecasted, the ISPs deploy additional bandwidth and switching capacity to continue to ensure satisfactory delay and packet-loss performance [Huang 2005]. We'll discuss such **network dimensioning** further in Section 7.5.1.

- *Differentiated service.* Since the early days of the Internet, it's been envisioned that different types of traffic (for example, as indicated in the Type-of-Service field in the IP4v packet header) could be provided with different classes of service, rather than a single one-size-fits-all best-effort service. With **differentiated service**, one type of traffic might be given strict priority over another class of traffic when both types of traffic are queued at a router. For example, packets belonging to a real-time conversational application might be given priority over other packets due to their stringent delay constraints. Introducing differentiated service into the network will require new mechanisms for packet marking (indicating a packet's class of service), packet scheduling, and more. We'll cover differentiated service, and new network mechanisms needed to implement this service, in Section 7.5.2.
- *Per-connection Quality-of-Service (QoS) Guarantees.* With per-connection QoS guarantees, each instance of an application explicitly reserves end-to-end bandwidth and thus has a guaranteed end-to-end performance. A **hard guarantee** means the application will receive its requested quality of service (QoS) with certainty. A **soft guarantee** means the application will receive its requested quality of service with high probability. For example, if a user wants to make a VoIP call from Host A to Host B, the user's VoIP application reserves bandwidth explicitly in each link along a route between the two hosts. But permitting applications to make reservations and requiring the network to honor the reservations requires some big changes. First, we need a protocol that, on behalf of the applications, reserves link bandwidth on the paths from the senders to their receivers. Second, we'll need new scheduling policies in the router queues so that per-connection bandwidth reservations can be honored. Finally, in order to make a reservation, the applications must give the network a description of the traffic that they intend to send into the network and the network will need to police each application's traffic to make sure that it abides by that description. These mechanisms, when combined, require new and complex software in hosts and routers. Because per-connection QoS guaranteed service has not seen significant deployment, we'll cover these mechanisms only briefly in Section 7.5.3.

7.5.1 Dimensioning Best-Effort Networks

Fundamentally, the difficulty in supporting multimedia applications arises from their stringent performance requirements—low end-to-end packet delay, delay

jitter, and loss—and the fact that packet delay, delay jitter, and loss occur whenever the network becomes congested. A first approach to improving the quality of multimedia applications—an approach that can often be used to solve just about any problem where resources are constrained—is simply to "throw money at the problem" and thus simply avoid resource contention. In the case of networked multimedia, this means providing enough link capacity throughout the network so that network congestion, and its consequent packet delay and loss, never (or only very rarely) occurs. With enough link capacity, packets could zip through today's Internet without queuing delay or loss. From many perspectives this is an ideal situation—multimedia applications would perform perfectly, users would be happy, and this could all be achieved with no changes to Internet's best-effort architecture.

The question, of course, is how much capacity is "enough" to achieve this nirvana, and whether the costs of providing "enough" bandwidth are practical from a business standpoint to the ISPs. The question of how much capacity to provide at network links in a given topology to achieve a given level of performance is often known as **bandwidth provisioning**. The even more complicated problem of how to design a network topology (where to place routers, how to interconnect routers with links, and what capacity to assign to links) to achieve a given level of end-to-end performance is a network design problem often referred to as **network dimensioning**. Both bandwidth provisioning and network dimensioning are complex topics, well beyond the scope of this textbook. We note here, however, that the following issues must be addressed in order to predict application-level performance between two network end points, and thus provision enough capacity to meet an application's performance requirements.

- *Models of traffic demand between network end points.* Models may need to be specified at both the call level (for example, users "arriving" to the network and starting up end-to-end applications) and at the packet level (for example, packets being generated by ongoing applications). Note that workload may change over time.
- *Well-defined performance requirements.* For example, a performance requirement for supporting delay-sensitive traffic, such as a conversational multimedia application, might be that the probability that the end-to-end delay of the packet is greater than a maximum tolerable delay be less than some small value [Fraleigh 2003].
- *Models to predict end-to-end performance for a given workload model, and techniques to find a minimal cost bandwidth allocation that will result in all user requirements being met.* Here, researchers are busy developing performance models that can quantify performance for a given workload, and optimization techniques to find minimal-cost bandwidth allocations meeting performance requirements.

Given that today's best-effort Internet could (from a technology standpoint) support multimedia traffic at an appropriate performance level if it were dimensioned to do so, the natural question is why today's Internet doesn't do so. The answers are primarily economic and organizational. From an economic standpoint, would users be willing to pay their ISPs enough for the ISPs to install sufficient bandwidth to support multimedia applications over a best-effort Internet? The organizational issues are perhaps even more daunting. Note that an end-to-end path between two multimedia end points will pass through the networks of multiple ISPs. From an organizational standpoint, would these ISPs be willing to cooperate (perhaps with revenue sharing) to ensure that the end-to-end path is properly dimensioned to support multimedia applications? For a perspective on these economic and organizational issues, see [Davies 2005]. For a perspective on provisioning tier-1 backbone networks to support delay-sensitive traffic, see [Fraleigh 2003].

7.5.2 Providing Multiple Classes of Service

Perhaps the simplest enhancement to the one-size-fits-all best-effort service in today's Internet is to divide traffic into classes, and provide different levels of service to these different classes of traffic. For example, an ISP might well want to provide a higher class of service to delay-sensitive Voice-over-IP or teleconferencing traffic (and charge more for this service!) than to elastic traffic such as email or HTTP. Alternatively, an ISP may simply want to provide a higher quality of service to customers willing to pay more for this improved service. A number of residential wired-access ISPs and cellular wireless-access ISPs have adopted such tiered levels of service—with platinum-service subscribers receiving better performance than gold- or silver-service subscribers.

We're all familiar with different classes of service from our everyday lives—first-class airline passengers get better service than business-class passengers, who in turn get better service than those of us who fly economy class; VIPs are provided immediate entry to events while everyone else waits in line; elders are revered in some countries and provided seats of honor and the finest food at a table. It's important to note that such differential service is provided among aggregates of traffic, that is, among classes of traffic, not among individual connections. For example, all first-class passengers are handled the same (with no first-class passenger receiving any better treatment than any other first-class passenger), just as all VoIP packets would receive the same treatment within the network, independent of the particular end-to-end connection to which they belong. As we will see, by dealing with a small number of traffic aggregates, rather than a large number of individual connections, the new network mechanisms required to provide better-than-best service can be kept relatively simple.

The early Internet designers clearly had this notion of multiple classes of service in mind. Recall the type-of-service (ToS) field in the IPv4 header in Figure 4.13.

IEN123 [ISI 1979] describes the ToS field also present in an ancestor of the IPv4 datagram as follows: "The Type of Service [field] provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important that other traffic." More than four decades ago, the vision of providing different levels of service to different classes of traffic was clear! However, it's taken us an equally long period of time to realize this vision.

Motivating Scenarios

Let's begin our discussion of network mechanisms for providing multiple classes of service with a few motivating scenarios.

Figure 7.14 shows a simple network scenario in which two application packet flows originate on Hosts H1 and H2 on one LAN and are destined for Hosts H3 and H4 on another LAN. The routers on the two LANs are connected by a 1.5 Mbps link. Let's assume the LAN speeds are significantly higher than 1.5 Mbps, and focus on the output queue of router R1; it is here that packet delay and packet loss will occur if the aggregate sending rate of H1 and H2 exceeds 1.5 Mbps. Let's further suppose that a 1 Mbps audio application (for example, a CD-quality audio call) shares the 1.5 Mbps link between R1 and R2 with an HTTP Web-browsing application that is downloading a Web page from H2 to H4.

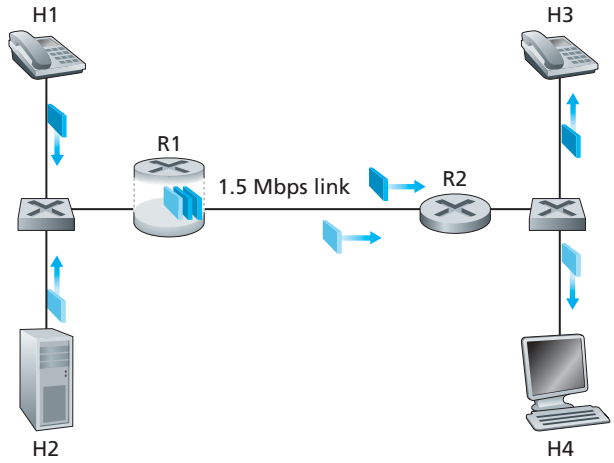


Figure 7.14 ♦ Competing audio and HTTP applications

In the best-effort Internet, the audio and HTTP packets are mixed in the output queue at R1 and (typically) transmitted in a first-in-first-out (FIFO) order. In this scenario, a burst of packets from the Web server could potentially fill up the queue, causing IP audio packets to be excessively delayed or lost due to buffer overflow at R1. How should we solve this potential problem? Given that the HTTP Web-browsing application does not have time constraints, our intuition might be to give strict priority to audio packets at R1. Under a strict priority scheduling discipline, an audio packet in the R1 output buffer would always be transmitted before any HTTP packet in the R1 output buffer. The link from R1 to R2 would look like a dedicated link of 1.5 Mbps to the audio traffic, with HTTP traffic using the R1-to-R2 link only when no audio traffic is queued. In order for R1 to distinguish between the audio and HTTP packets in its queue, each packet must be marked as belonging to one of these two classes of traffic. This was the original goal of the type-of-service (ToS) field in IPv4. As obvious as this might seem, this then is our first insight into mechanisms needed to provide multiple classes of traffic:

> **Insight 1: Packet marking** allows a router to distinguish among packets belonging to different classes of traffic.

Note that although our example considers a competing multimedia and elastic flow, the same insight applies to the case that platinum, gold, and silver classes of service are implemented—a packet-marking mechanism is still needed to indicate that class of service to which a packet belongs.

Now suppose that the router is configured to give priority to packets marked as belonging to the 1 Mbps audio application. Since the outgoing link speed is 1.5 Mbps, even though the HTTP packets receive lower priority, they can still, on average, receive 0.5 Mbps of transmission service. But what happens if the audio application starts sending packets at a rate of 1.5 Mbps or higher (either maliciously or due to an error in the application)? In this case, the HTTP packets will starve, that is, they will not receive any service on the R1-to-R2 link. Similar problems would occur if multiple applications (for example, multiple audio calls), all with the same class of service as the audio application, were sharing the link's bandwidth; they too could collectively starve the FTP session. Ideally, one wants a degree of isolation among classes of traffic so that one class of traffic can be protected from the other. This protection could be implemented at different places in the network—at each and every router, at first entry to the network, or at inter-domain network boundaries. This then is our second insight:

> **Insight 2:** It is desirable to provide a degree of **traffic isolation** among classes so that one class is not adversely affected by another class of traffic that misbehaves.

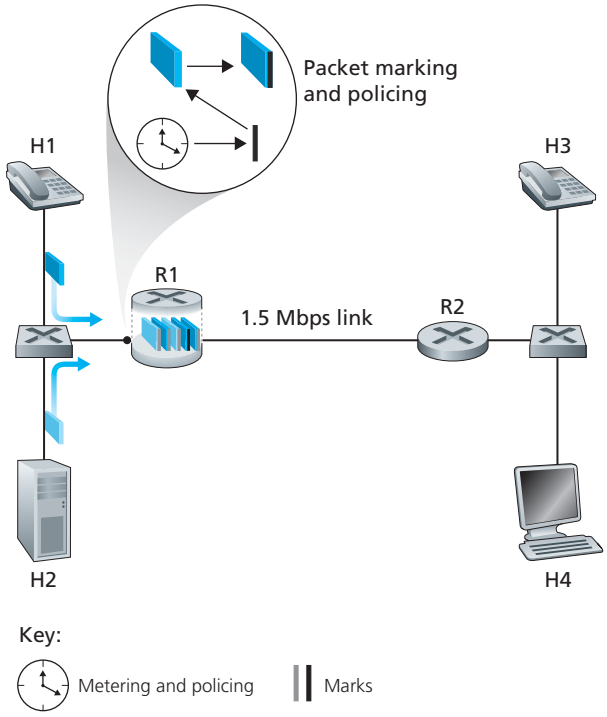


Figure 7.15 ♦ Policing (and marking) the audio and HTTP traffic classes

We'll examine several specific mechanisms for providing such isolation among traffic classes. We note here that two broad approaches can be taken. First, it is possible to perform **traffic policing**, as shown in Figure 7.15. If a traffic class or flow must meet certain criteria (for example, that the audio flow not exceed a peak rate of 1 Mbps), then a policing mechanism can be put into place to ensure that these criteria are indeed observed. If the policed application misbehaves, the policing mechanism will take some action (for example, drop or delay packets that are in violation of the criteria) so that the traffic actually entering the network conforms to the criteria. The leaky bucket mechanism that we'll examine shortly is perhaps the most widely used policing mechanism. In Figure 7.15, the packet classification and marking mechanism (Insight 1) and the policing mechanism (Insight 2) are both implemented together at the network's edge, either in the end system or at an edge router.

A complementary approach for providing isolation among traffic classes is for the link-level packet-scheduling mechanism to explicitly allocate a fixed

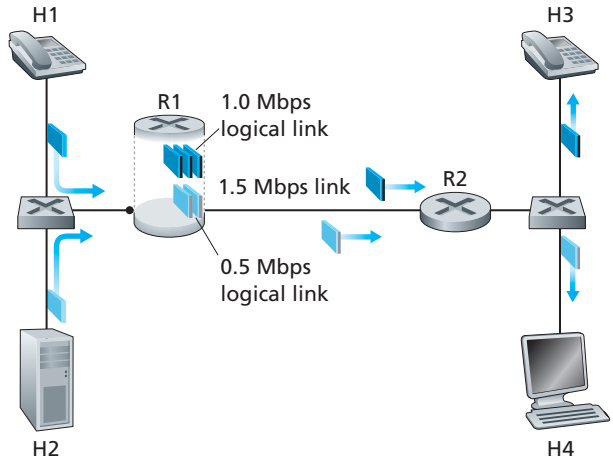


Figure 7.16 ♦ Logical isolation of audio and HTTP traffic classes

amount of link bandwidth to each class. For example, the audio class could be allocated 1 Mbps at R1, and the HTTP class could be allocated 0.5 Mbps. In this case, the audio and HTTP flows see a logical link with capacity 1.0 and 0.5 Mbps, respectively, as shown in Figure 7.16. With strict enforcement of the link-level allocation of bandwidth, a class can use only the amount of bandwidth that has been allocated; in particular, it cannot utilize bandwidth that is not currently being used by others. For example, if the audio flow goes silent (for example, if the speaker pauses and generates no audio packets), the HTTP flow would still not be able to transmit more than 0.5 Mbps over the R1-to-R2 link, even though the audio flow's 1 Mbps bandwidth allocation is not being used at that moment. Since bandwidth is a "use-it-or-lose-it" resource, there is no reason to prevent HTTP traffic from using bandwidth not used by the audio traffic. We'd like to use bandwidth as efficiently as possible, never wasting it when it could be otherwise used. This gives rise to our third insight:

> **Insight 3:** While providing isolation among classes or flows, it is desirable to use resources (for example, link bandwidth and buffers) as efficiently as possible.

Scheduling Mechanisms

Recall from our discussion in Section 1.3 and Section 4.3 that packets belonging to various network flows are multiplexed and queued for transmission at the

output buffers associated with a link. The manner in which queued packets are selected for transmission on the link is known as the **link-scheduling discipline**. Let us now consider several of the most important link-scheduling disciplines in more detail.

First-In-First-Out (FIFO)

Figure 7.17 shows the queuing model abstractions for the FIFO link-scheduling discipline. Packets arriving at the link output queue wait for transmission if the link is currently busy transmitting another packet. If there is not sufficient buffering space to hold the arriving packet, the queue's **packet-discarding policy** then determines whether the packet will be dropped (lost) or whether other packets will be removed from the queue to make space for the arriving packet. In our discussion below, we will ignore packet discard. When a packet is completely transmitted over the outgoing link (that is, receives service) it is removed from the queue.

The FIFO (also known as first-come-first-served, or FCFS) scheduling discipline selects packets for link transmission in the same order in which they arrived at the output link queue. We're all familiar with FIFO queuing from bus stops (particularly in England, where queuing seems to have been perfected) or other service centers, where arriving customers join the back of the single waiting line, remain in order, and are then served when they reach the front of the line.

Figure 7.18 shows the FIFO queue in operation. Packet arrivals are indicated by numbered arrows above the upper timeline, with the number indicating the order

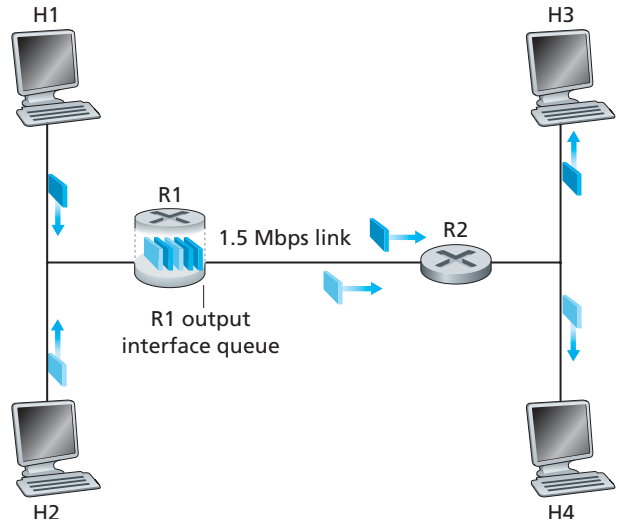


Figure 7.17 ♦ FIFO queuing abstraction

in which the packet arrived. Individual packet departures are shown below the lower timeline. The time that a packet spends in service (being transmitted) is indicated by the shaded rectangle between the two timelines. Because of the FIFO discipline, packets leave in the same order in which they arrived. Note that after the departure of packet 4, the link remains idle (since packets 1 through 4 have been transmitted and removed from the queue) until the arrival of packet 5.

Priority Queuing

Under **priority queuing**, packets arriving at the output link are classified into priority classes at the output queue, as shown in Figure 7.19. As discussed in the previous section, a packet's priority class may depend on an explicit marking that it carries in its packet header (for example, the value of the ToS bits in an IPv4 packet), its source or destination IP address, its destination port number, or other criteria. Each priority class typically has its own queue. When choosing a packet to transmit, the priority queuing discipline will transmit a packet from the highest priority class that has a nonempty queue (that is, has packets waiting for transmission). The choice among packets *in the same priority class* is typically done in a FIFO manner.

Figure 7.20 illustrates the operation of a priority queue with two priority classes. Packets 1, 3, and 4 belong to the high-priority class, and packets 2 and 5 belong to the low-priority class. Packet 1 arrives and, finding the link idle, begins transmission. During the transmission of packet 1, packets 2 and 3 arrive and are queued in the low- and high-priority queues, respectively. After the transmission of packet 1, packet 3 (a high-priority packet) is selected for transmission over packet 2 (which, even though it arrived earlier, is a low-priority packet). At the end of the transmission of packet 3, packet 2 then begins transmission. Packet 4 (a high-priority packet) arrives during the transmission of packet 2 (a low-priority packet). Under a nonpreemptive priority queuing discipline, the transmission of

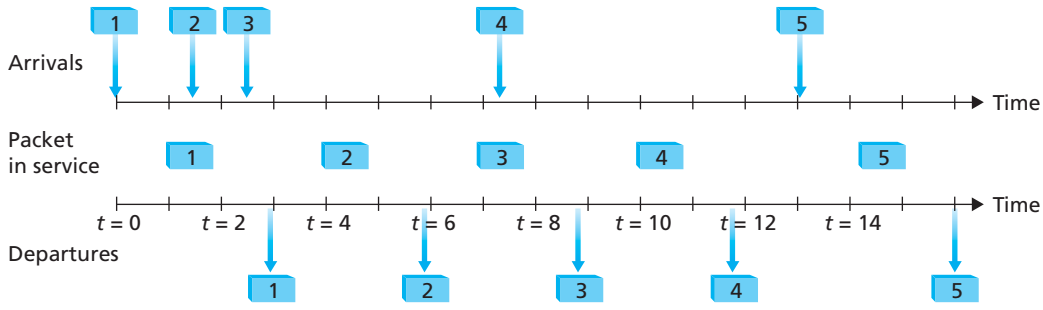


Figure 7.18 ♦ The FIFO queue in operation

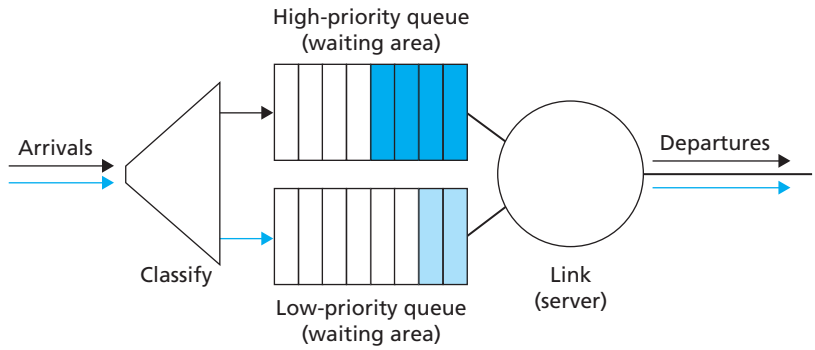


Figure 7.19 ♦ Priority queuing model

a packet is not interrupted once it has begun. In this case, packet 4 queues for transmission and begins being transmitted after the transmission of packet 2 is completed.

Round Robin and Weighted Fair Queuing (WFQ)

Under the **round robin queuing discipline**, packets are sorted into classes as with priority queuing. However, rather than there being a strict priority of service among classes, a round robin scheduler alternates service among the classes. In the simplest form of round robin scheduling, a class 1 packet is transmitted, followed by a class 2 packet, followed by a class 1 packet, followed by a class 2 packet, and so on. A so-called work-conserving queuing discipline will never allow the link to remain idle whenever there are packets (of any class) queued for

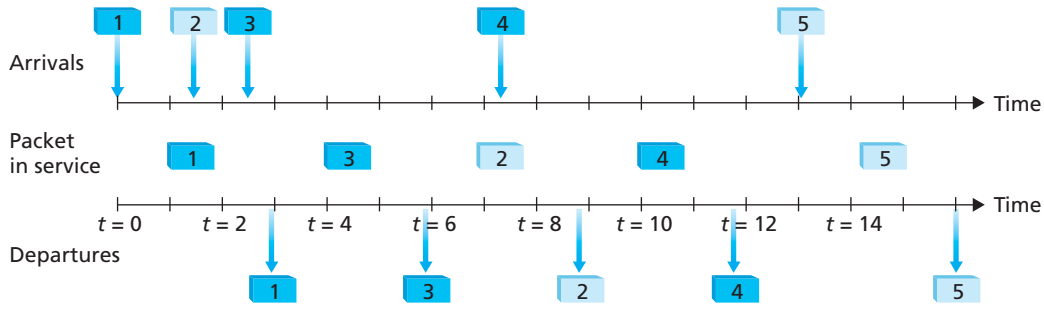


Figure 7.20 ♦ Operation of the priority queue

transmission. A **work-conserving round robin discipline** that looks for a packet of a given class but finds none will immediately check the next class in the round robin sequence.

Figure 7.21 illustrates the operation of a two-class round robin queue. In this example, packets 1, 2, and 4 belong to class 1, and packets 3 and 5 belong to the second class. Packet 1 begins transmission immediately upon arrival at the output queue. Packets 2 and 3 arrive during the transmission of packet 1 and thus queue for transmission. After the transmission of packet 1, the link scheduler looks for a class 2 packet and thus transmits packet 3. After the transmission of packet 3, the scheduler looks for a class 1 packet and thus transmits packet 2. After the transmission of packet 2, packet 4 is the only queued packet; it is thus transmitted immediately after packet 2.

A generalized abstraction of round robin queuing that has found considerable use in QoS architectures is the so-called **weighted fair queuing** (WFQ) discipline [Demers 1990; Parekh 1993]. WFQ is illustrated in Figure 7.22. Arriving packets are classified and queued in the appropriate per-class waiting area. As in round robin scheduling, a WFQ scheduler will serve classes in a circular manner—first serving class 1, then serving class 2, then serving class 3, and then (assuming there are three classes) repeating the service pattern. WFQ is also a work-conserving queuing discipline and thus will immediately move on to the next class in the service sequence when it finds an empty class queue.

WFQ differs from round robin in that each class may receive a *differential* amount of service in any interval of time. Specifically, each class, *i,* is assigned a weight, w_i. Under WFQ, during any interval of time during which there are class *i* packets to send, class *i* will then be guaranteed to receive a fraction of service equal to $w_i/(\sum w_j)$, where the sum in the denominator is taken over all classes that also have packets queued for transmission. In the worst case, even if all classes have queued packets, class *i* will still be guaranteed to receive a fraction $w_i/(\sum w_j)$ of the

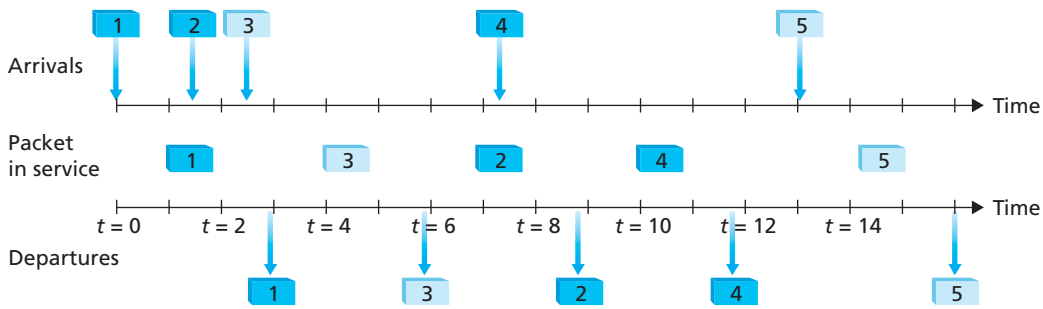


Figure 7.21 ♦ Operation of the two-class round robin queue

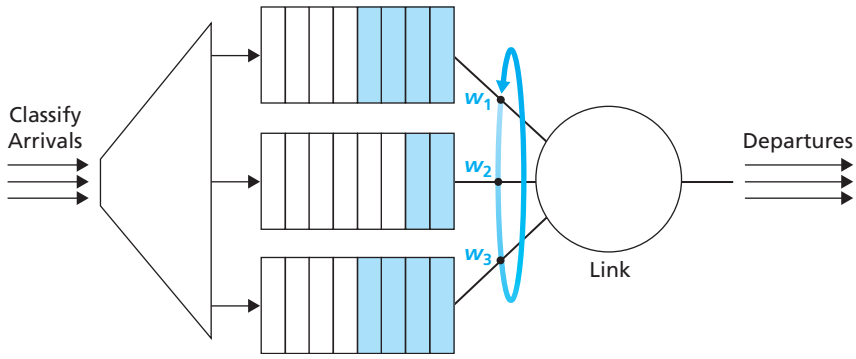


Figure 7.22 ♦ Weighted fair queuing (WFQ)

bandwidth. Thus, for a link with transmission rate R, class i will always achieve a throughput of at least $R \cdot w_i/(\sum w_j)$. Our description of WFQ has been an idealized one, as we have not considered the fact that packets are discrete units of data and a packet's transmission will not be interrupted to begin transmission of another packet; [Demers 1990] and [Parekh 1993] discuss this packetization issue. As we will see in the following sections, WFQ plays a central role in QoS architectures. It is also available in today's router products [Cisco QoS 2012].

Policing: The Leaky Bucket

One of our earlier insights was that policing, the regulation of the rate at which a class or flow (we will assume the unit of policing is a flow in our discussion below) is allowed to inject packets into the network, is an important QoS mechanism. But what aspects of a flow's packet rate should be policed? We can identify three important policing criteria, each differing from the other according to the time scale over which the packet flow is policed:

- *Average rate.* The network may wish to limit the long-term average rate (packets per time interval) at which a flow's packets can be sent into the network. A crucial issue here is the interval of time over which the average rate will be policed. A flow whose average rate is limited to 100 packets per second is more constrained than a source that is limited to 6,000 packets per minute, even though both have the same average rate over a long enough interval of time. For example, the latter constraint would allow a flow to send 1,000 packets in a given second-long interval of time, while the former constraint would disallow this sending behavior.

- *Peak rate.* While the average-rate constraint limits the amount of traffic that can be sent into the network over a relatively long period of time, a peak-rate constraint limits the maximum number of packets that can be sent over a shorter period of time. Using our example above, the network may police a flow at an average rate of 6,000 packets per minute, while limiting the flow's peak rate to 1,500 packets per second.
- *Burst size.* The network may also wish to limit the maximum number of packets (the "burst" of packets) that can be sent into the network over an extremely short interval of time. In the limit, as the interval length approaches zero, the burst size limits the number of packets that can be instantaneously sent into the network. Even though it is physically impossible to instantaneously send multiple packets into the network (after all, every link has a physical transmission rate that cannot be exceeded!), the abstraction of a maximum burst size is a useful one.

The leaky bucket mechanism is an abstraction that can be used to characterize these policing limits. As shown in Figure 7.23, a leaky bucket consists of a bucket that can hold up to b tokens. Tokens are added to this bucket as follows. New tokens, which may potentially be added to the bucket, are always being generated at a rate of r tokens per second. (We assume here for simplicity that the unit of time is a second.) If the bucket is filled with less than b tokens when a token is generated, the newly generated token is added to the bucket; otherwise the newly generated token is ignored, and the token bucket remains full with b tokens.

Let us now consider how the leaky bucket can be used to police a packet flow. Suppose that before a packet is transmitted into the network, it must first remove a

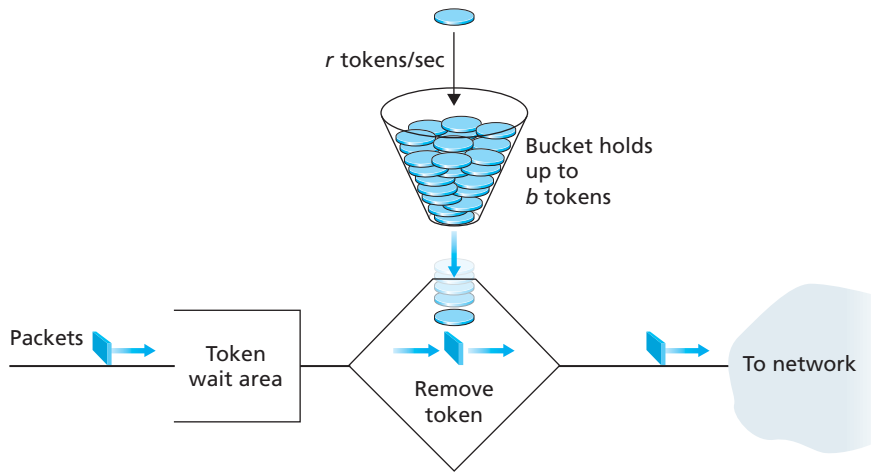


Figure 7.23 ♦ The leaky bucket policer

token from the token bucket. If the token bucket is empty, the packet must wait for a token. (An alternative is for the packet to be dropped, although we will not consider that option here.) Let us now consider how this behavior polices a traffic flow. Because there can be at most b tokens in the bucket, the maximum burst size for a leaky-bucket-policed flow is b packets. Furthermore, because the token generation rate is r, the maximum number of packets that can enter the network of *any* interval of time of length t is $rt + b$. Thus, the token-generation rate, r, serves to limit the long-term average rate at which packets can enter the network. It is also possible to use leaky buckets (specifically, two leaky buckets in series) to police a flow's peak rate in addition to the long-term average rate; see the homework problems at the end of this chapter.

Leaky Bucket + Weighted Fair Queuing = Provable Maximum Delay in a Queue

Let's close our discussion of scheduling and policing by showing how the two can be combined to provide a bound on the delay through a router's queue. Let's consider a router's output link that multiplexes n flows, each policed by a leaky bucket with parameters b_i and r_i, $i = 1, \ldots, n$, using WFQ scheduling. We use the term *flow* here loosely to refer to the set of packets that are not distinguished from each other by the scheduler. In practice, a flow might be comprised of traffic from a single end-to-end connection or a collection of many such connections, see Figure 7.24.

Recall from our discussion of WFQ that each flow, i, is guaranteed to receive a share of the link bandwidth equal to at least $R \cdot w_i/(\sum w_j)$, where R is the transmission

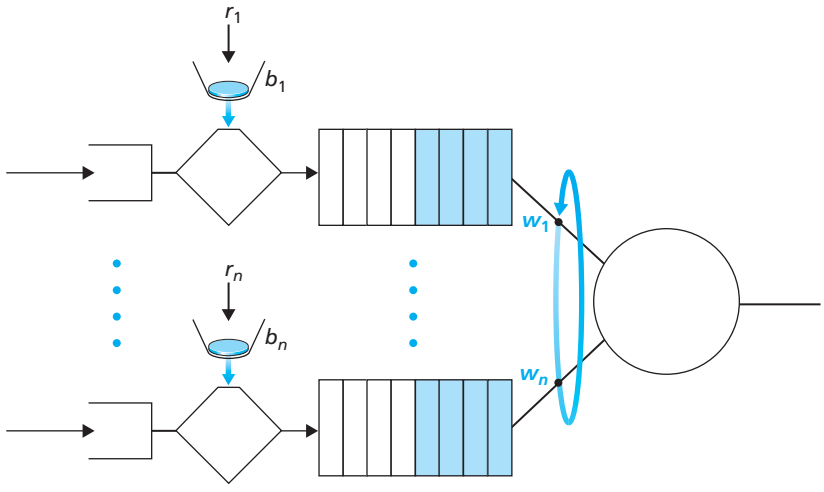


Figure 7.24 ♦ n multiplexed leaky bucket flows with WFQ scheduling

rate of the link in packets/sec. What then is the maximum delay that a packet will experience while waiting for service in the WFQ (that is, after passing through the leaky bucket)? Let us focus on flow 1. Suppose that flow 1's token bucket is initially full. A burst of b_1 packets then arrives to the leaky bucket policer for flow 1. These packets remove all of the tokens (without wait) from the leaky bucket and then join the WFQ waiting area for flow 1. Since these b_1 packets are served at a rate of at least $R \cdot w_i/(\sum w_j)$ packet/sec, the last of these packets will then have a maximum delay, $d_{\text{max,}}$ until its transmission is completed, where

$$d_{\max} = \frac{b_1}{R \cdot w_1 / \sum w_j}$$

The rationale behind this formula is that if there are b_1 packets in the queue and packets are being serviced (removed) from the queue at a rate of at least $R \cdot w_1/(\sum w_j)$ packets per second, then the amount of time until the last bit of the last packet is transmitted cannot be more than $b_1/(R \cdot w_1/(\sum w_j))$. A homework problem asks you to prove that as long as $r_1 < R \cdot w_1/(\sum w_j)$, then $d_{\max}$ is indeed the maximum delay that any packet in flow 1 will ever experience in the WFQ queue.

7.5.3 Diffserv

Having seen the motivation, insights, and specific mechanisms for providing multiple classes of service, let's wrap up our study of approaches toward proving multiple classes of service with an example—the Internet Diffserv architecture [RFC 2475; RFC Kilkki 1999]. Diffserv provides service differentiation—that is, the ability to handle different classes of traffic in different ways within the Internet in a scalable manner. The need for scalability arises from the fact that millions of simultaneous source-destination traffic flows may be present at a backbone router. We'll see shortly that this need is met by placing only simple functionality within the network core, with more complex control operations being implemented at the network's edge.

Let's begin with the simple network shown in Figure 7.25. We'll describe one possible use of Diffserv here; other variations are possible, as described in RFC 2475. The Diffserv architecture consists of two sets of functional elements:

- *Edge functions: packet classification and traffic conditioning.* At the incoming edge of the network (that is, at either a Diffserv-capable host that generates traffic or at the first Diffserv-capable router that the traffic passes through), arriving packets are marked. More specifically, the differentiated service (DS) field in the IPv4 or IPv6 packet header is set to some value [RFC 3260]. The definition of the DS field is intended to supersede the earlier definitions of the IPv4 type-of-service field and the IPv6 traffic class fields that we discussed in Chapter 4. For example, in Figure 7.25, packets being sent from H1 to H3 might be marked

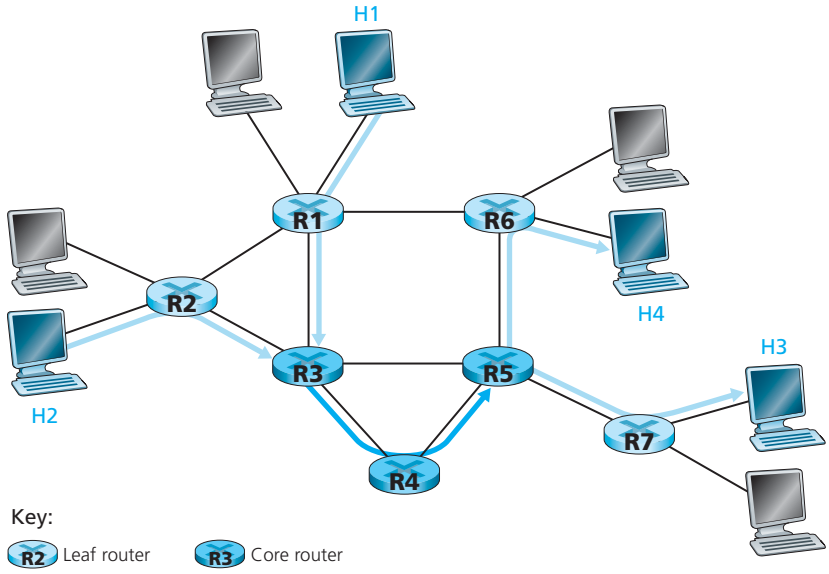


Figure 7.25 ♦ A simple Diffserv network example

at R1, while packets being sent from H2 to H4 might be marked at R2. The mark that a packet receives identifies the class of traffic to which it belongs. Different classes of traffic will then receive different service within the core network.

- *Core function: forwarding.* When a DS-marked packet arrives at a Diffserv-capable router, the packet is forwarded onto its next hop according to the so-called per-hop behavior (PHB) associated with that packet's class. The per-hop behavior influences how a router's buffers and link bandwidth are shared among the competing classes of traffic. A crucial tenet of the Diffserv architecture is that a router's per-hop behavior will be based only on packet markings, that is, the class of traffic to which a packet belongs. Thus, if packets being sent from H1 to H3 in Figure 7.25 receive the same marking as packets being sent from H2 to H4, then the network routers treat these packets as an aggregate, without distinguishing whether the packets originated at H1 or H2. For example, R3 would not distinguish between packets from H1 and H2 when forwarding these packets on to R4. Thus, the Diffserv architecture obviates the need to keep router state for individual source-destination pairs—a critical consideration in making Diffserv scalable.

An analogy might prove useful here. At many large-scale social events (for example, a large public reception, a large dance club or discothèque, a concert, or a football game), people entering the event receive a pass of one type or another: VIP passes for Very

Important People; over-21 passes for people who are 21 years old or older (for example, if alcoholic drinks are to be served); backstage passes at concerts; press passes for reporters; even an ordinary pass for the Ordinary Person. These passes are typically distributed upon entry to the event, that is, at the edge of the event. It is here at the edge where computationally intensive operations, such as paying for entry, checking for the appropriate type of invitation, and matching an invitation against a piece of identification, are performed. Furthermore, there may be a limit on the number of people of a given type that are allowed into an event. If there is such a limit, people may have to wait before entering the event. Once inside the event, one's pass allows one to receive differentiated service at many locations around the event—a VIP is provided with free drinks, a better table, free food, entry to exclusive rooms, and fawning service. Conversely, an ordinary person is excluded from certain areas, pays for drinks, and receives only basic service. In both cases, the service received within the event depends solely on the type of one's pass. Moreover, all people within a class are treated alike.

Figure 7.26 provides a logical view of the classification and marking functions within the edge router. Packets arriving to the edge router are first classified. The classifier selects packets based on the values of one or more packet header fields (for example, source address, destination address, source port, destination port, and protocol ID) and steers the packet to the appropriate marking function. As noted above, a packet's marking is carried in the DS field in the packet header.

In some cases, an end user may have agreed to limit its packet-sending rate to conform to a declared **traffic profile**. The traffic profile might contain a limit on the peak rate, as well as the burstiness of the packet flow, as we saw previously with the leaky bucket mechanism. As long as the user sends packets into the network in a way that conforms to the negotiated traffic profile, the packets receive their priority marking and are forwarded along their route to the destination. On the other hand, if the traffic profile is violated, out-of-profile packets might be marked differently, might be shaped (for example, delayed so that a maximum rate constraint would be observed), or might be dropped at the network edge. The role of the **metering function**, shown in Figure 7.26, is to compare the incoming packet flow with the negotiated traffic profile and to determine whether a packet is within the negotiated traffic profile. The actual decision about whether to immediately remark, forward, delay, or drop a packet is a policy issue determined by the network administrator and is *not* specified in the Diffserv architecture.

So far, we have focused on the marking and policing functions in the Diffserv architecture. The second key component of the Diffserv architecture involves the per-hop behavior (PHB) performed by Diffserv-capable routers. PHB is rather cryptically, but carefully, defined as "a description of the externally observable forwarding behavior of a Diffserv node applied to a particular Diffserv behavior aggregate" [RFC 2475]. Digging a little deeper into this definition, we can see several important considerations embedded within:

- A PHB can result in different classes of traffic receiving different performance (that is, different externally observable forwarding behaviors).

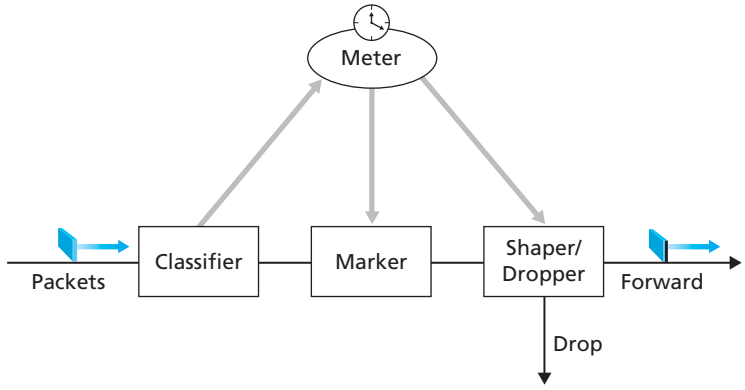


Figure 7.26 ♦ A simple Diffserv network example

- While a PHB defines differences in performance (behavior) among classes, it does not mandate any particular mechanism for achieving these behaviors. As long as the externally observable performance criteria are met, any implementation mechanism and any buffer/bandwidth allocation policy can be used. For example, a PHB would not require that a particular packet-queuing discipline (for example, a priority queue versus a WFQ queue versus a FCFS queue) be used to achieve a particular behavior. The PHB is the end, to which resource allocation and implementation mechanisms are the means.
- Differences in performance must be observable and hence measurable.

Two PHBs have been defined: an expedited forwarding (EF) PHB [RFC 3246] and an assured forwarding (AF) PHB [RFC 2597]. The **expedited forwarding** PHB specifies that the departure rate of a class of traffic from a router must equal or exceed a configured rate. The **assured forwarding** PHB divides traffic into four classes, where each AF class is guaranteed to be provided with some minimum amount of bandwidth and buffering.

Let's close our discussion of Diffserv with a few observations regarding its service model. First, we have implicitly assumed that Diffserv is deployed within a single administrative domain, but typically an end-to-end service must be fashioned from multiple ISPs sitting between communicating end systems. In order to provide end-to-end Diffserv service, all the ISPs between the end systems must not only provide this service, but most also cooperate and make settlements in order to offer end customers true end-to-end service. Without this kind of cooperation, ISPs directly selling Diffserv service to customers will find themselves repeatedly saying: "Yes, we know you paid extra, but we don't have a service agreement with the ISP that dropped and delayed your traffic. I'm sorry that there were so many gaps in your

VoIP call!" Second, if Diffserv were actually in place and the network ran at only moderate load, most of the time there would be no perceived difference between a best-effort service and a Diffserv service. Indeed, end-to-end delay is usually dominated by access rates and router hops rather than by queuing delays in the routers. Imagine the unhappy Diffserv customer who has paid more for premium service but finds that the best-effort service being provided to others almost always has the same performance as premium service!

7.5.4 Per-Connection Quality-of-Service (QoS) Guarantees: Resource Reservation and Call Admission

In the previous section, we have seen that packet marking and policing, traffic isolation, and link-level scheduling can provide one class of service with better performance than another. Under certain scheduling disciplines, such as priority scheduling, the lower classes of traffic are essentially "invisible" to the highest-priority class of traffic. With proper network dimensioning, the highest class of service can indeed achieve extremely low packet loss and delay—essentially circuit-like performance. But can the network *guarantee* that an ongoing flow in a high-priority traffic class will continue to receive such service throughout the flow's duration using only the mechanisms that we have described so far? It cannot. In this section, we'll see why yet additional network mechanisms and protocols are required when a hard service guarantee is provided to individual connections.

Let's return to our scenario from Section 7.5.2 and consider two 1 Mbps audio applications transmitting their packets over the 1.5 Mbps link, as shown in Figure 7.27. The combined data rate of the two flows (2 Mbps) exceeds the link

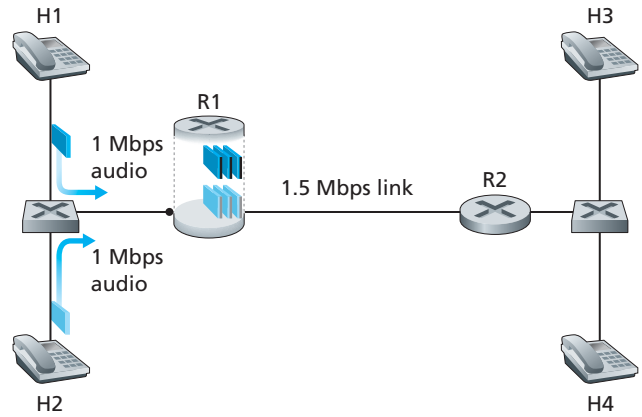


Figure 7.27 ♦ Two competing audio applications overloading the R1-to-R2 link

capacity. Even with classification and marking, isolation of flows, and sharing of unused bandwidth (of which there is none), this is clearly a losing proposition. There is simply not enough bandwidth to accommodate the needs of both applications at the same time. If the two applications equally share the bandwidth, each application would lose 25 percent of its transmitted packets. This is such an unacceptably low QoS that both audio applications are completely unusable; there's no need even to transmit any audio packets in the first place.

Given that the two applications in Figure 7.27 cannot both be satisfied simultaneously, what should the network do? Allowing both to proceed with an unusable QoS wastes network resources on application flows that ultimately provide no utility to the end user. The answer is hopefully clear—one of the application flows should be blocked (that is, denied access to the network), while the other should be allowed to proceed on, using the full 1 Mbps needed by the application. The telephone network is an example of a network that performs such call blocking—if the required resources (an end-to-end circuit in the case of the telephone network) cannot be allocated to the call, the call is blocked (prevented from entering the network) and a busy signal is returned to the user. In our example, there is no gain in allowing a flow into the network if it will not receive a sufficient QoS to be considered usable. Indeed, there is a cost to admitting a flow that does not receive its needed QoS, as network resources are being used to support a flow that provides no utility to the end user.

By explicitly admitting or blocking flows based on their resource requirements, and the source requirements of already-admitted flows, the network can guarantee that admitted flows will be able to receive their requested QoS. Implicit in the need to provide a guaranteed QoS to a flow is the need for the flow to declare its QoS requirements. This process of having a flow declare its QoS requirement, and then having the network either accept the flow (at the required QoS) or block the flow is referred to as the **call admission** process. This then is our fourth insight (in addition to the three earlier insights from Section 7.5.2) into the mechanisms needed to provide QoS.

> **Insight 4:** If sufficient resources will not always be available, and QoS is to be *guaranteed*, a call admission process is needed in which flows declare their QoS requirements and are then either admitted to the network (at the required QoS) or blocked from the network (if the required QoS cannot be provided by the network).

Our motivating example in Figure 7.27 highlights the need for several new network mechanisms and protocols if a call (an end-to-end flow) is to be guaranteed a given quality of service once it begins:

- *Resource reservation.* The only way to *guarantee* that a call will have the resources (link bandwidth, buffers) needed to meet its desired QoS is to explicitly

allocate those resources to the call—a process known in networking parlance as **resource reservation**. Once resources are reserved, the call has on-demand access to these resources throughout its duration, regardless of the demands of all other calls. If a call reserves and receives a guarantee of x Mbps of link bandwidth, and never transmits at a rate greater than x, the call will see loss- and delay-free performance.

- *Call admission.* If resources are to be reserved, then the network must have a mechanism for calls to request and reserve resources. Since resources are not infinite, a call making a call admission request will be denied admission, that is, be blocked, if the requested resources are not available. Such a call admission is performed by the telephone network—we request resources when we dial a number. If the circuits (TDMA slots) needed to complete the call are available, the circuits are allocated and the call is completed. If the circuits are not available, then the call is blocked, and we receive a busy signal. A blocked call can try again to gain admission to the network, but it is not allowed to send traffic into the network until it has successfully completed the call admission process. Of course, a router that allocates link bandwidth should not allocate more than is available at that link. Typically, a call may reserve only a fraction of the link's bandwidth, and so a router may allocate link bandwidth to more than one call. However, the sum of the allocated bandwidth to all calls should be less than the link capacity if hard quality of service guarantees are to be provided.
- *Call setup signaling.* The call admission process described above requires that a call be able to reserve sufficient resources at each and every network router on its source-to-destination path to ensure that its end-to-end QoS requirement is met. Each router must determine the local resources required by the session, consider the amounts of its resources that are already committed to other ongoing sessions, and determine whether it has sufficient resources to satisfy the per-hop QoS requirement of the session at this router without violating local QoS guarantees made to an already-admitted session. A signaling protocol is needed to coordinate these various activities—the per-hop allocation of local resources, as well as the overall end-to-end decision of whether or not the call has been able to reserve sufficient resources at each and every router on the end-to-end path. This is the job of the **call setup protocol**, as shown in Figure 7.28. The **RSVP protocol** [Zhang 1993, RFC 2210] was proposed for this purpose within an Internet architecture for providing quality-of-service guarantees. In ATM networks, the Q2931b protocol [Black 1995] carries this information among the ATM network's switches and end point.

Despite a tremendous amount of research and development, and even products that provide for per-connection quality of service guarantees, there has been almost no extended deployment of such services. There are many possible reasons. First and foremost, it may well be the case that the simple application-level mechanisms that we studied in Sections 7.2 through 7.4, combined with proper

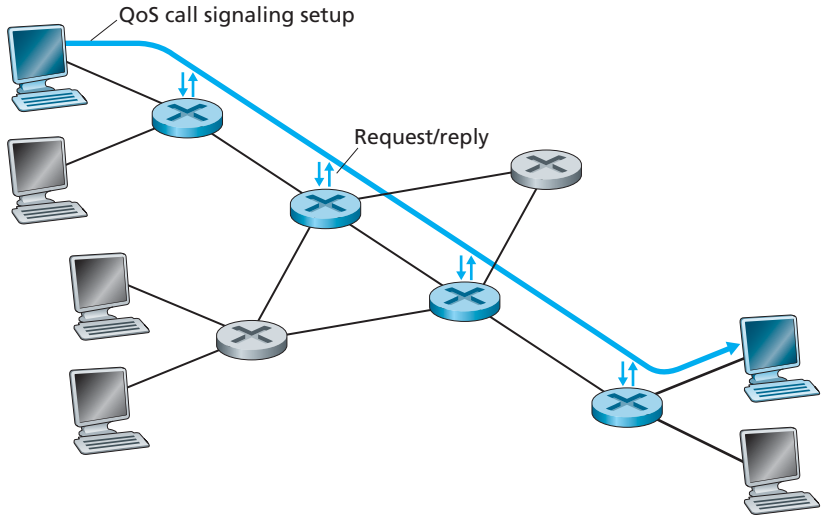


Figure 7.28 ♦ The call setup process

network dimensioning (Section 7.5.1) provide "good enough" best-effort network service for multimedia applications. In addition, the added complexity and cost of deploying and managing a network that provides per-connection quality of service guarantees may be judged by ISPs to be simply too high given predicted customer revenues for that service.

7.6 Summary

Multimedia networking is one of the most exciting developments in the Internet today. People throughout the world are spending less time in front of their radios and televisions, and are instead turning to the Internet to receive audio and video transmissions, both live and prerecorded. This trend will certainly continue as high-speed wireless Internet access becomes more and more prevalent. Moreover, with sites like YouTube, users have become producers as well as consumers of multimedia Internet content. In addition to video distribution, the Internet is also being used to transport phone calls. In fact, over the next 10 years, the Internet, along with wireless Internet access, may make the traditional circuit-switched telephone system a thing of the past. VoIP not only provides phone service inexpensively, but also provides numerous value-added services, such as video conferencing, online directory services, voice messaging, and integration into social networks such as Facebook and Google+.

In Section 7.1, we described the intrinsic characteristics of video and voice, and then classified multimedia applications into three categories: (i) streaming stored audio/video, (ii) conversational voice/video-over-IP, and (iii) streaming live audio/video.

In Section 7.2, we studied streaming stored video in some depth. For streaming video applications, prerecorded videos are placed on servers, and users send requests to these servers to view the videos on demand. We saw that streaming video systems can be classified into three categories: UDP streaming, HTTP streaming, and adaptive HTTP streaming. Although all three types of systems are used in practice, the majority of today's systems employ HTTP streaming and adaptive HTTP streaming. We observed that the most important performance measure for streaming video is average throughput. In Section 7.2 we also investigated CDNs, which help distribute massive amounts of video data to users around the world. We also surveyed the technology behind three major Internet video-streaming companies: Netflix, YouTube, and Kankan.

In Section 7.3, we examined how conversational multimedia applications, such as VoIP, can be designed to run over a best-effort network. For conversational multimedia, timing considerations are important because conversational applications are highly delay-sensitive. On the other hand, conversational multimedia applications are loss-tolerant—occasional loss only causes occasional glitches in audio/video playback, and these losses can often be partially or fully concealed. We saw how a combination of client buffers, packet sequence numbers, and timestamps can greatly alleviate the effects of network-induced jitter. We also surveyed the technology behind Skype, one of the leading voice- and video-over-IP companies. In Section 7.4, we examined two of the most important standardized protocols for VoIP, namely, RTP and SIP.

In Section 7.5, we introduced how several network mechanisms (link-level scheduling disciplines and traffic policing) can be used to provide differentiated service among several classes of traffic.

Homework Problems and Questions

Chapter 7 Review Questions

SECTION 7.1

R1. Reconstruct Table 7.1 for when Victor Video is watching a 5 Mbps video, Facebook Frank is looking at a new 150 Kbyte image every 25 seconds, and Martha Music is listening to 210 kbps audio stream.

R2. For 128 quantization levels, what is the size of each sample signal?

R3. Suppose an analog audio signal is sampled 8,000 times per second, and each sample is quantized into one of 512 levels. What would be the resulting bit rate of the PCM digital audio signal?

R4. Many Internet companies today provide streaming video, including YouTube (Google), Netflix, and Hulu. Streaming stored video has three key distinguishing features. List them.

SECTION 7.2

R5. What are advantages of client buffering?

R6. In video streaming applications, why is HTTP streaming more popular than UDP streaming?

R7. With HTTP streaming, are the TCP receive buffer and the client's application buffer the same thing? If not, how do they interact?

R8. Consider the simple model for HTTP streaming. Suppose the server sends bits at a constant rate of 2 Mbps and playback begins when 8 million bits have been received. What is the initial buffering delay t_p?

R9. What is prefetching video? How does it help?

R10. Several cluster selection strategies were described in Section 7.2.4. Which of these strategies finds a good cluster with respect to the client's LDNS? Which of these strategies finds a good cluster with respect to the client itself?

R11. Besides network-related considerations such as delay, loss, and bandwidth performance, there are many additional important factors that go into designing a cluster selection strategy. What are they?

SECTION 7.3

R12. What mechanisms are used at the receiver side to eliminate packet jitter?

R13. What are the two types of loss anticipation schemes used in VoIP?

R14. Section 7.3 describes two FEC schemes. Briefly summarize them. Both schemes increase the transmission rate of the stream by adding overhead. Does interleaving also increase the transmission rate?

SECTION 7.4

R15. What are the four main RTP header fields?

R16. What is the role of a SIP registrar? How is the role of an SIP registrar different from that of a home agent in Mobile IP?

SECTION 7.5

R17. In Section 7.5, we discussed non-preemptive priority queuing. What would be preemptive priority queuing? Does preemptive priority queuing make sense for computer networks?

R18. Give an example of a scheduling discipline that is *not* work conserving.

R19. What is the purpose of RSVP?

Problems

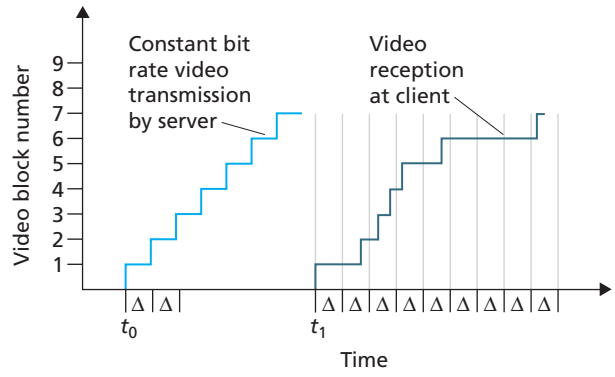
P1. Consider the figure below. Similar to our discussion of Figure 7.1, suppose that video is encoded at a fixed bit rate, and thus each video block contains video frames that are to be played out over the same fixed amount of time, $\triangle$. The server transmits the first video block at t_0, the second block at $t_0 + \triangle$, the third block at $t_0 + 2\triangle$, and so on. Once the client begins playout, each block should be played out $\triangle$ time units after the previous block.

a. Suppose that the client begins playout as soon as the first block arrives at t_1. In the figure below, how many blocks of video (including the first block) will have arrived at the client in time for their playout? Explain how you arrived at your answer.

b. Suppose that the client begins playout now at $t_1 + \triangle$. How many blocks of video (including the first block) will have arrived at the client in time for their playout? Explain how you arrived at your answer.

c. In the same scenario at (b) above, what is the largest number of blocks that is ever stored in the client buffer, awaiting playout? Explain how you arrived at your answer.

d. What is the smallest playout delay at the client, such that every video block has arrived in time for its playout? Explain how you arrived at your answer.



P2. Recall the simple model for HTTP streaming shown in Figure 7.3. Recall that B denotes the size of the client's application buffer, and Q denotes the number of bits that must be buffered before the client application begins playout. Also r denotes the video consumption rate. Assume that the server sends bits at a constant rate x whenever the client buffer is not full.

a. Suppose that $x < r$. As discussed in the text, in this case playout will alternate between periods of continuous playout and periods of freezing. Determine the length of each continuous playout and freezing period as a function of Q, r, and x.

b. Now suppose that $x > r$. At what time $t = t_f$ does the client application buffer become full?

P3. Recall the simple model for HTTP streaming shown in Figure 7.3. Suppose the buffer size is infinite but the server sends bits at variable rate $x(t)$. Specifically, suppose $x(t)$ has the following saw-tooth shape. The rate is initially zero at time $t = 0$ and linearly climbs to H at time $t = T$. It then repeats this pattern again and again, as shown in the figure below.

a. What is the server's average send rate?

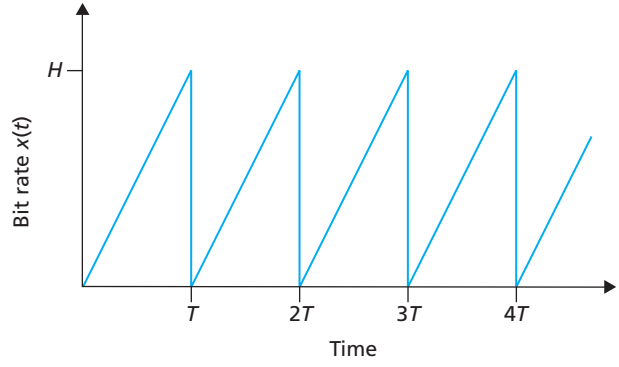
b. Suppose that $Q = 0$, so that the client starts playback as soon as it receives a video frame. What will happen?

c. Now suppose $Q > 0$. Determine as a function of Q, H, and T the time at which playback first begins.

d. Suppose $H > 2r$ and $Q = HT/2$. Prove there will be no freezing after the initial playout delay.

e. Suppose $H > 2r$. Find the smallest value of Q such that there will be no freezing after the initial playback delay.

f. Now suppose that the buffer size B is finite. Suppose $H > 2r$. As a function of Q, B, T, and H, determine the time $t = t_f$ when the client application buffer first becomes full.



P4. Consider the following in the context of prefetching. Suppose the video consumption rate is 2 Mbps but the network is capable of delivering the video from server to client at a constant rate of 2.5 Mbps. Then the client will not only be able to play out the video with a very small playout delay, but will also be able to increase the amount of buffered video data by 500 Kbits every second. In this manner, if in the future, the client receives data at a rate of less than 2 Mbps for a brief period of time, the client will be able to continue to provide continuous playback due to the reserve in its buffer. At what throughput does streaming over TCP result in minimal starvation and low buffering delays?

P5. As an example of jitter, consider two consecutive packets in our VoIP application. The sender sends the second packet 20 msecs after sending the first packet. But at the receiver, the spacing between these packets can become greater than 20 msecs. To see this, suppose the first packet arrives at a nearly empty queue at a router, but just before the second packet arrives at the queue a large number of packets from other sources arrive at the same queue. Because the first packet experiences a small queuing delay and the second packet suffers a large queuing delay at this router, the first and second packets become spaced by more than 20 msecs. Give an analogy with driving cars on roads.

P6. In the VoIP example in Section 7.3, let h be the total number of header byte added to each chunk, including UDP and IP header.

   a. Assuming an IP datagram is emitted every 40 msecs, find the transmission rate in bits per second for the datagrams generated by one side of this application.

   b. What is a typical value of h when RTP is used? How much time is required to transmit the header?

P7. Consider the procedure described in Section 7.3 for estimating average delay d_i. Suppose that $u = 0.1$. Let $r_1 - t_1$ be the most recent sample delay, let $r_2 - t_2$ be the next most recent sample delay, and so on.

   a. For a given audio application suppose four packets have arrived at the receiver with sample delays $r_4 - t_4$, $r_3 - t_3$, $r_2 - t_2$, and $r_1 - t_1$. Express the estimate of delay d in terms of the four samples.

   b. Generalize your formula for n sample delays.

   c. For the formula in Part b, let n approach infinity and give the resulting formula. Comment on why this averaging procedure is called an exponential moving average.

P8. Repeat Parts a and b in Question P7 for the estimate of average delay deviation.

P9. For the VoIP example in Section 7.3, we introduced an online procedure (exponential moving average) for estimating delay. In this problem we will

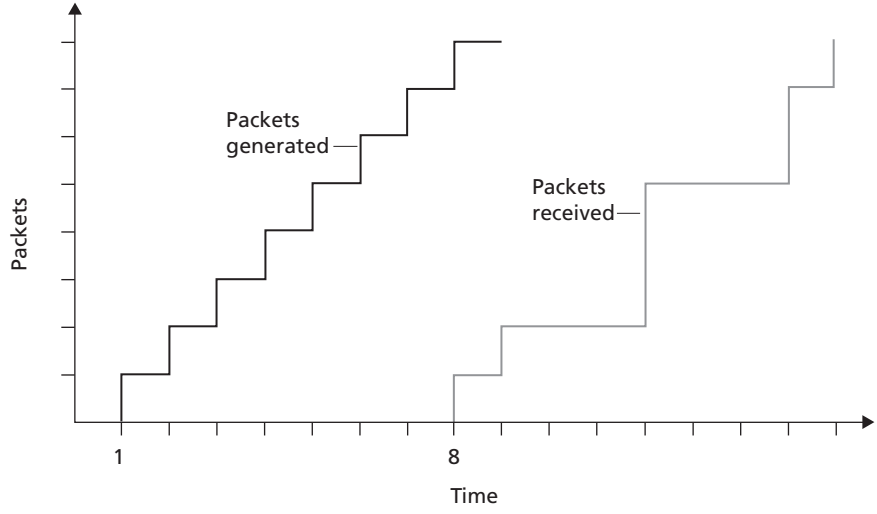
examine an alternative procedure. Let t_i be the timestamp of the ith packet received; let r_i be the time at which the ith packet is received. Let d_n be our estimate of average delay after receiving the nth packet. After the first packet is received, we set the delay estimate equal to $d_1 = r_1 - t_1$.

a. Suppose that we would like $d_n = (r_1 - t_1 + r_2 - t_2 + \ldots + r_n - t_n)/n$ for all n. Give a recursive formula for d_n in terms of d_{n-1}, r_n, and t_n.

b. Describe why for Internet telephony, the delay estimate described in Section 7.3 is more appropriate than the delay estimate outlined in Part a.

P10. With the fixed-delay strategy, the receiver attempts to play out each chunk exactly q msecs after the chunk is generated. So if a chunk is timestamped at the sender at time t, the receiver plays out the chunk at time $t + q$, assuming the chunk has arrived by that time. Packets that arrive after their scheduled playout times are discarded and considered lost. What is a good choice for q?

P11. Consider the figure below (which is similar to Figure 7.7). A sender begins sending packetized audio periodically at $t = 1$. The first packet arrives at the receiver at $t = 8$.



a. What are the delays (from sender to receiver, ignoring any playout delays) of packets 2 through 8? Note that each vertical and horizontal line segment in the figure has a length of 1, 2, or 3 time units.

b. If audio playout begins as soon as the first packet arrives at the receiver at $t = 8$, which of the first eight packets sent will *not* arrive in time for playout?

c. If audio playout begins at $t = 9$, which of the first eight packets sent will not arrive in time for playout?

d. What is the minimum playout delay at the receiver that results in all of the first eight packets arriving in time for their playout?

P12. Consider again the figure in P11, showing packet audio transmission and reception times.

a. Compute the estimated delay for packets 2 through 8, using the formula for d_i from Section 7.3.2. Use a value of $u = 0.1$.

b. Compute the estimated deviation of the delay from the estimated average for packets 2 through 8, using the formula for v_i from Section 7.3.2. Use a value of $u = 0.1$.

P13. A is at her PC and she wants to call B, who is also working at his PC. A's and B's PCs are both equipped with SIP-based software for making and receiving phone calls. Assume that A knows the IP address of B's PC. Illustrate the SIP call-establishment process.

P14. a. Consider an audio conference call in Skype with $N > 2$ participants. Suppose each participant generates a constant stream of rate r bps. How many bits per second will the call initiator need to send? How many bits per second will each of the other $N - 1$ participants need to send? What is the total send rate, aggregated over all participants?

b. Repeat part (a) for a Skype video conference call using a central server.

c. Repeat part (b), but now for when each peer sends a copy of its video stream to each of the $N - 1$ other peers.

P15. a. Suppose we send into the Internet two IP datagrams, each carrying a different UDP segment. The first datagram has source IP address A1, destination IP address B, source port P1, and destination port T. The second datagram has source IP address A2, destination IP address B, source port P2, and destination port T. Suppose that A1 is different from A2 and that P1 is different from P2. Assuming that both datagrams reach their final destination, will the two UDP datagrams be received by the same socket? Why or why not?

b. Suppose Alice, Bob, and Claire want to have an audio conference call using SIP and RTP. For Alice to send and receive RTP packets to and from Bob and Claire, is only one UDP socket sufficient (in addition to the socket needed for the SIP messages)? If yes, then how does Alice's SIP client distinguish between the RTP packets received from Bob and Claire?

P16. True or false:

a. If stored video is streamed directly from a Web server to a media player, then the application is using TCP as the underlying transport protocol.

b. When using RTP, it is possible for a sender to change encoding in the middle of a session.

c. All applications that use RTP must use port 87.

d. If an RTP session has a separate audio and video stream for each sender, then the audio and video streams use the same SSRC.

e. In differentiated services, while per-hop behavior defines differences in performance among classes, it does not mandate any particular mechanism for achieving these performances.

f. Suppose Alice wants to establish an SIP session with Bob. In her INVITE message she includes the line: m=audio 48753 RTP/AVP 3 (AVP 3 denotes GSM audio). Alice has therefore indicated in this message that she wishes to send GSM audio.

g. Referring to the preceding statement, Alice has indicated in her INVITE message that she will send audio to port 48753.

h. SIP messages are typically sent between SIP entities using a default SIP port number.

i. In order to maintain registration, SIP clients must periodically send REGISTER messages.

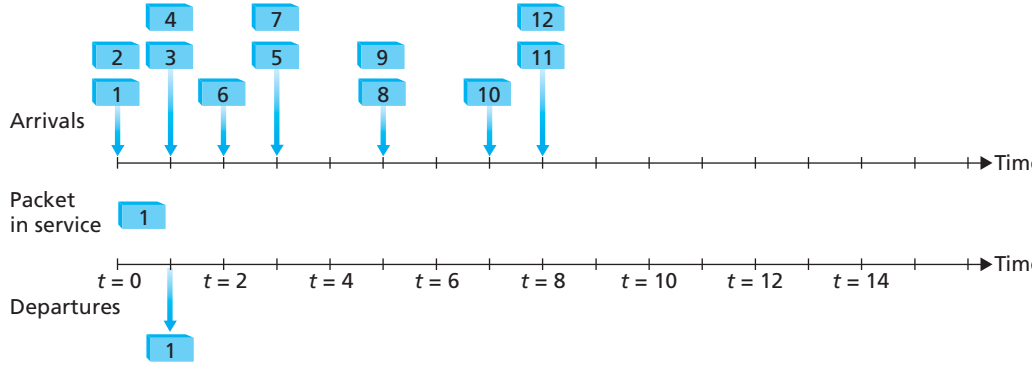
j. SIP mandates that all SIP clients support G.711 audio encoding.

P17. Suppose that the WFQ scheduling policy is applied to a buffer that supports three classes, and suppose the weights are 0.4, 0.4, and 0.2 for the three classes.

a. Suppose that each class has a large number of packets in the buffer. In what sequence might the three classes be served in order to achieve the WFQ weights? (For round robin scheduling, a natural sequence is 123123123 . . .)

b. Suppose that classes 1 and 3 have a large number of packets in the buffer, and there are no class 2 packets in the buffer. In what sequence might the three classes be served in to achieve the WFQ weights?

P18. Consider the figure below. Answer the following questions:



a. Assuming FIFO service, indicate the time at which packets 2 through 12 each leave the queue. For each packet, what is the delay between its arrival and the beginning of the slot in which it is transmitted? What is the average of this delay over all 12 packets?

b. Now assume a priority service, and assume that odd-numbered packets are high priority, and even-numbered packets are low priority. Indicate the time at which packets 2 through 12 each leave the queue. For each packet, what is the delay between its arrival and the beginning of the slot in which it is transmitted? What is the average of this delay over all 12 packets?

c. Now assume round robin service. Assume that packets 1, 2, 3, 6, 11, and 12 are from class 1, and packets 4, 5, 7, 8, 9, and 10 are from class 2. Indicate the time at which packets 2 through 12 each leave the queue. For each packet, what is the delay between its arrival and its departure? What is the average delay over all 12 packets?

d. Now assume weighted fair queueing (WFQ) service. Assume that odd-numbered packets are from class 1, and even-numbered packets are from class 2. Class 1 has a WFQ weight of 2, while class 2 has a WFQ weight of 1. Note that it may not be possible to achieve an idealized WFQ schedule as described in the text, so indicate why you have chosen the particular packet to go into service at each time slot. For each packet what is the delay between its arrival and its departure? What is the average delay over all 12 packets?

e. What do you notice about the average delay in all four cases (FIFO, RR, priority, and WFQ)?

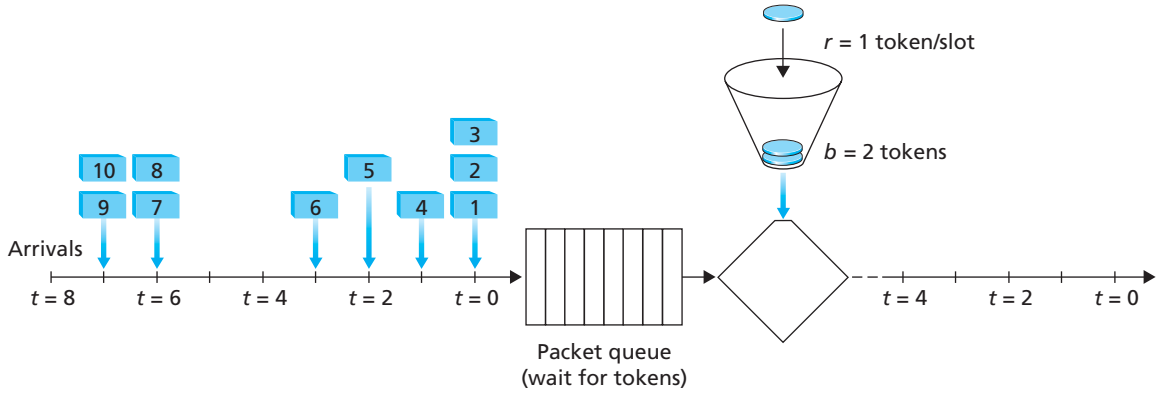
P19. Consider again the figure for P18.

a. Assume a priority service, with packets 1, 4, 5, 6, and 11 being high-priority packets. The remaining packets are low priority. Indicate the slots in which packets 2 through 12 each leave the queue.

b. Now suppose that round robin service is used, with packets 1, 4, 5, 6, and 11 belonging to one class of traffic, and the remaining packets belonging to the second class of traffic. Indicate the slots in which packets 2 through 12 each leave the queue.

c. Now suppose that WFQ service is used, with packets 1, 4, 5, 6, and 11 belonging to one class of traffic, and the remaining packets belonging to the second class of traffic. Class 1 has a WFQ weight of 1, while class 2 has a WFQ weight of 2 (note that these weights are different than in the previous question). Indicate the slots in which packets 2 through 12 each leave the queue. See also the caveat in the question above regarding WFQ service.

P20. Consider the figure below, which shows a leaky bucket policer being fed by a stream of packets. The token buffer can hold at most two tokens, and is initially full at $t = 0$. New tokens arrive at a rate of one token per slot. The output link speed is such that if two packets obtain tokens at the beginning of a time slot, they can both go to the output link in the same slot. The timing details of the system are as follows:



1. Packets (if any) arrive at the beginning of the slot. Thus in the figure, packets 1, 2, and 3 arrive in slot 0. If there are already packets in the queue, then the arriving packets join the end of the queue. Packets proceed towards the front of the queue in a FIFO manner.
2. After the arrivals have been added to the queue, if there are any queued packets, one or two of those packets (depending on the number of available tokens) will each remove a token from the token buffer and go to the output link during that slot. Thus, packets 1 and 2 each remove a token from the buffer (since there are initially two tokens) and go to the output link during slot 0.
3. A new token is added to the token buffer if it is not full, since the token generation rate is $r = 1$ token/slot.
4. Time then advances to the next time slot, and these steps repeat.

Answer the following questions:

a. For each time slot, identify the packets that are in the queue and the number of tokens in the bucket, immediately after the arrivals have been processed (step 1 above) but before any of the packets have passed through the queue and removed a token. Thus, for the $t = 0$ time slot in the example above, packets 1, 2 and 3 are in the queue, and there are two tokens in the buffer.

b. For each time slot indicate which packets appear on the output after the token(s) have been removed from the queue. Thus, for the $t = 0$ time slot in the example above, packets 1 and 2 appear on the output link from the leaky buffer during slot 0.

P21. Repeat P20 but assume that $r = 2$. Assume again that the bucket is initially full.

P22. What is network dimensioning?

P23. Consider the leaky-bucket policer that polices the average rate and burst size of a packet flow. We now want to police the peak rate, p, as well. Show how the output of this leaky-bucket policer can be fed into a second leaky bucket policer so that the two leaky buckets in series police the average rate, peak rate, and burst size. Be sure to give the bucket size and token generation rate for the second policer.

P24. A packet flow is said to conform to a leaky-bucket specification (r,b) with burst size b and average rate r if the number of packets that arrive to the leaky bucket is less than $rt + b$ packets in every interval of time of length t for all t. Will a packet flow that conforms to a leaky-bucket specification (r,b) ever have to wait at a leaky bucket policer with parameters r and b? Justify your answer.

P25. Show that as long as $r_1 < R\, w_1/(\sum w_j)$, then $d_{\max}$ is indeed the maximum delay that any packet in flow 1 will ever experience in the WFQ queue.

Programming Assignment

In this lab, you will implement a streaming video server and client. The client will use the real-time streaming protocol (RTSP) to control the actions of the server. The server will use the real-time protocol (RTP) to packetize the video for transport over UDP. You will be given Python code that partially implements RTSP and RTP at the client and server. Your job will be to complete both the client and server code. When you are finished, you will have created a client-server application that does the following:

- The client sends SETUP, PLAY, PAUSE, and TEARDOWN RTSP commands, and the server responds to the commands.
- When the server is in the playing state, it periodically grabs a stored JPEG frame, packetizes the frame with RTP, and sends the RTP packet into a UDP socket.
- The client receives the RTP packets, removes the JPEG frames, decompresses the frames, and renders the frames on the client's monitor.

The code you will be given implements the RTSP protocol in the server and the RTP depacketization in the client. The code also takes care of displaying the transmitted video. You will need to implement RTSP in the client and RTP server. This programming assignment will significantly enhance the student's understanding of RTP, RTSP, and streaming video. It is highly recommended. The assignment also suggests a number of optional exercises, including implementing the RTSP DESCRIBE command at both client and server. You can find full details of the assignment, as well as an overview of the RTSP protocol, at the Web site http://www.awl.com/kurose-ross.

AN INTERVIEW WITH...

Henning Schulzrinne

Henning Schulzrinne is a professor, chair of the Department of Computer Science, and head of the Internet Real-Time Laboratory at Columbia University. He is the co-author of RTP, RTSP, SIP, and GIST—key protocols for audio and video communications over the Internet. Henning received his BS in electrical and industrial engineering at TU Darmstadt in Germany, his MS in electrical and computer engineering at the University of Cincinnati, and his PhD in electrical engineering at the University of Massachusetts, Amherst.

What made you decide to specialize in multimedia networking?

This happened almost by accident. As a PhD student, I got involved with DARTnet, an experimental network spanning the United States with T1 lines. DARTnet was used as a proving ground for multicast and Internet real-time tools. That led me to write my first audio tool, NeVoT. Through some of the DARTnet participants, I became involved in the IETF, in the then-nascent Audio Video Transport working group. This group later ended up standardizing RTP.

What was your first job in the computer industry? What did it entail?

My first job in the computer industry was soldering together an Altair computer kit when I was a high school student in Livermore, California. Back in Germany, I started a little consulting company that devised an address management program for a travel agency—storing data on cassette tapes for our TRS-80 and using an IBM Selectric typewriter with a homebrew hardware interface as a printer.

My first real job was with AT&T Bell Laboratories, developing a network emulator for constructing experimental networks in a lab environment.

What are the goals of the Internet Real-Time Lab?

Our goal is to provide components and building blocks for the Internet as the single future communications infrastructure. This includes developing new protocols, such as GIST (for network-layer signaling) and LoST (for finding resources by location), or enhancing protocols that we have worked on earlier, such as SIP, through work on rich presence, peer-to-peer systems, next-generation emergency calling, and service creation tools. Recently, we have also looked extensively at wireless systems for VoIP, as 802.11b and 802.11n networks and maybe WiMax networks are likely to become important last-mile technologies for telephony. We are also trying to greatly improve the ability of users to diagnose faults in the complicated tangle of providers and equipment, using a peer-to-peer fault diagnosis system called DYSWIS (Do You See What I See).