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.

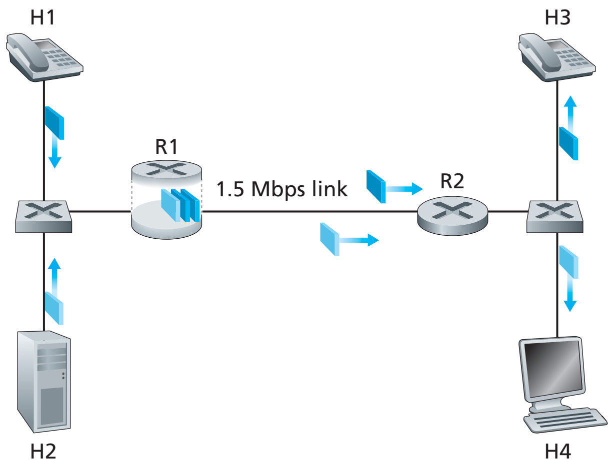

Figure 7.14 ♦ Competing audio and HTTP applications

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.

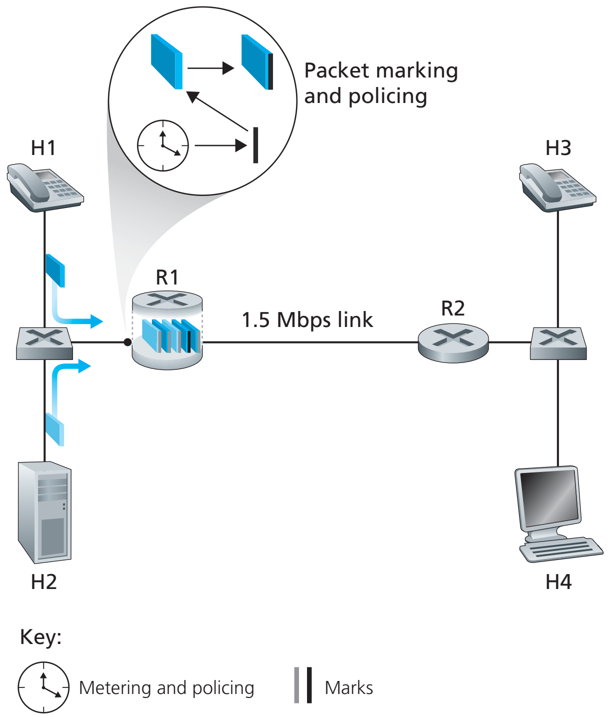

Figure 7.15 ♦ Policing (and marking) the audio and HTTP traffic classes

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

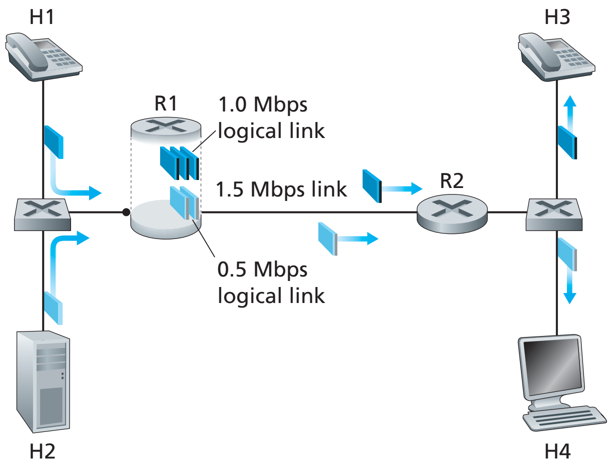

Figure 7.16 ♦ Logical isolation of audio and HTTP traffic classes

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

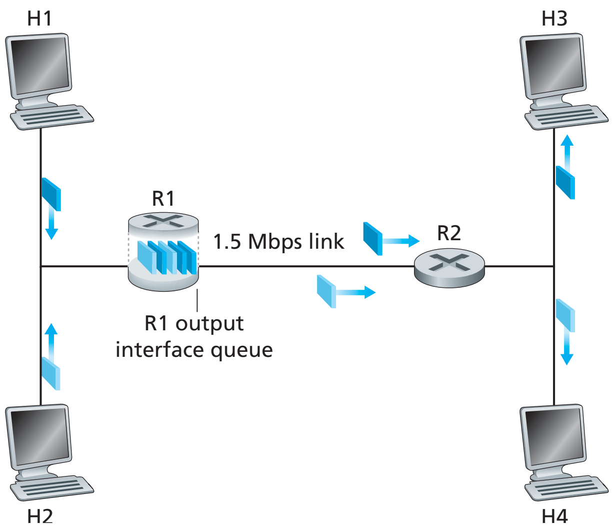

Figure 7.17 ♦ FIFO queuing abstraction

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

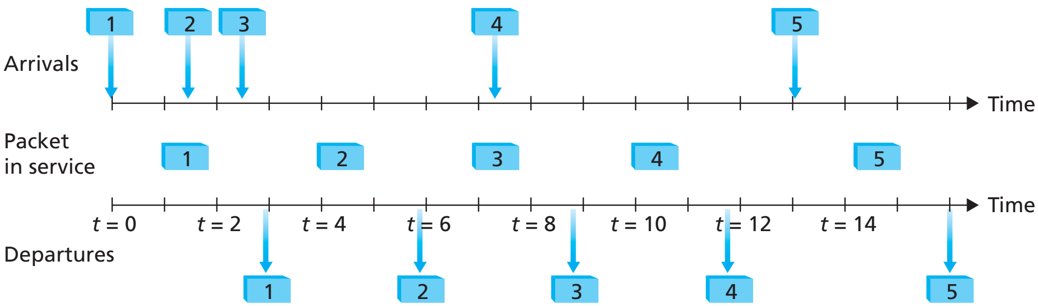

Figure 7.18 ♦ The FIFO queue in operation

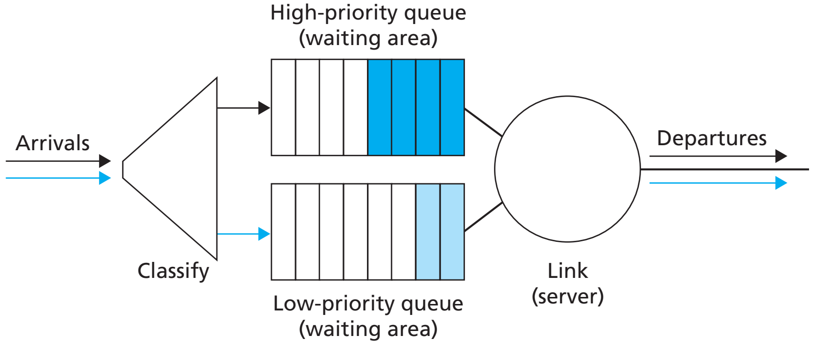

Figure 7.19 ♦ Priority queuing model

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

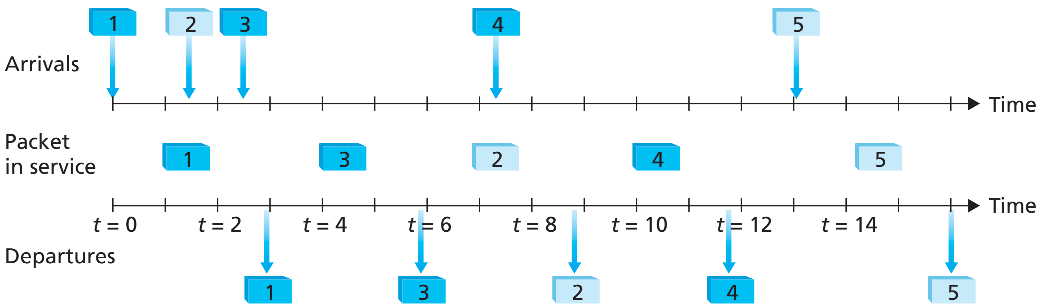

Figure 7.20 ♦ Operation of the priority queue

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

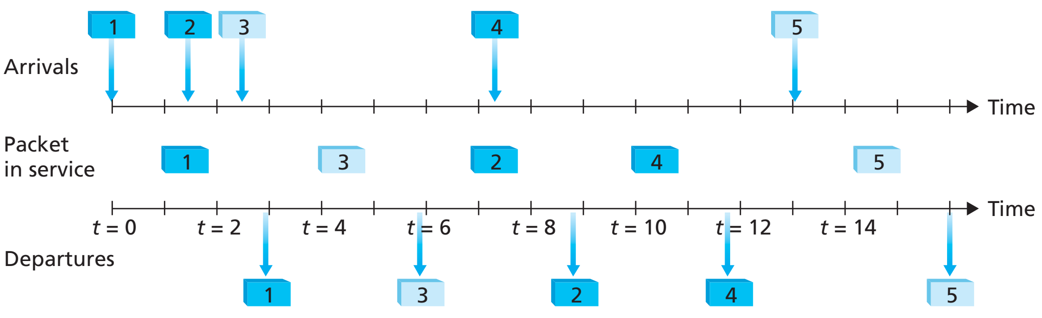

Figure 7.21 ♦ Operation of the two-class round robin queue

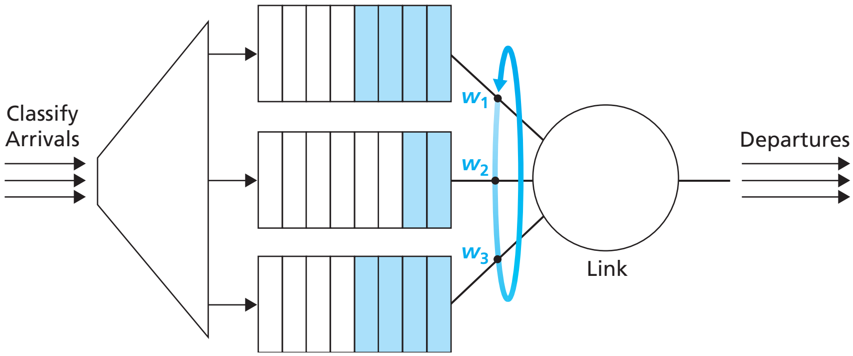

Figure 7.22 ♦ Weighted fair queuing (WFQ)

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

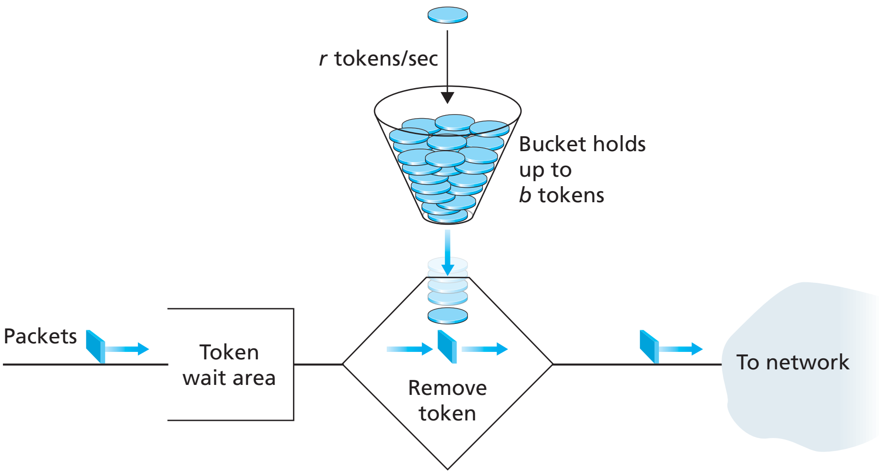

Figure 7.23 ♦ The leaky bucket policer

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

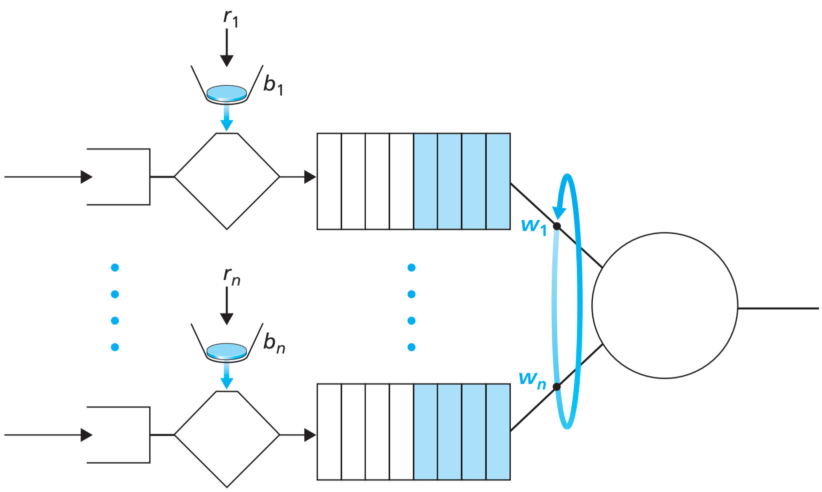

Figure 7.24 ♦ *n* multiplexed leaky bucket flows with WFQ scheduling