

make an ISP's investment in network capacity go farther. By making resources available to high-priority (and high-paying) classes of traffic whenever needed (at the expense of the lower-priority classes of traffic), the ISP can deliver a high level of performance to these high-priority classes. When these resources are not needed by the high-priority classes, they can be used by the lower-priority traffic classes (who have presumably paid less for this lower class of service).

Another concern with these advanced services is the need to police and possibly shape traffic, which may turn out to be complex and costly. One also needs to bill the services differently, most likely by volume rather than with a fixed monthly fee as currently done by most ISPs—another costly requirement for the ISP. Finally, 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, today, 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 for premium service but finds that the best-effort service being provided to others almost always has the same performance as premium service!

## 7.6 Providing Quality of Service Guarantees

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 on-going 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 can not. In this section, we'll see why yet additional network mechanisms and protocols are needed to provide quality of service *guarantees*.

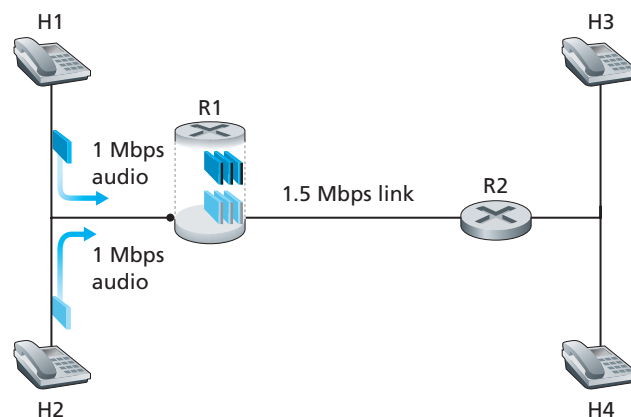
### 7.6.1 A Motivating Example

Let's return to our scenario from section 7.5.1 and consider two 1 Mbps audio applications transmitting their packets over the 1.5 Mbps link, as shown in Figure 7.31. The combined data rate of the two flows (2 Mbps) exceeds the 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 would receive only

0.75 Mbps. Looked at another way, 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 7.31 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 (i.e., 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 with 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 7.5.1) into the mechanisms needed to provide QoS.



**Figure 7.31** ♦ Two competing audio applications overloading the R1-to-R2 link

**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).

### 7.6.2 Resource Reservation, Call Admission, Call Setup

Our motivating example highlights the need for several new network mechanisms and protocols if a call (an end-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—a process known as call admission. Since resources are not infinite, a call making a call admission request will be denied admission, i.e., 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.

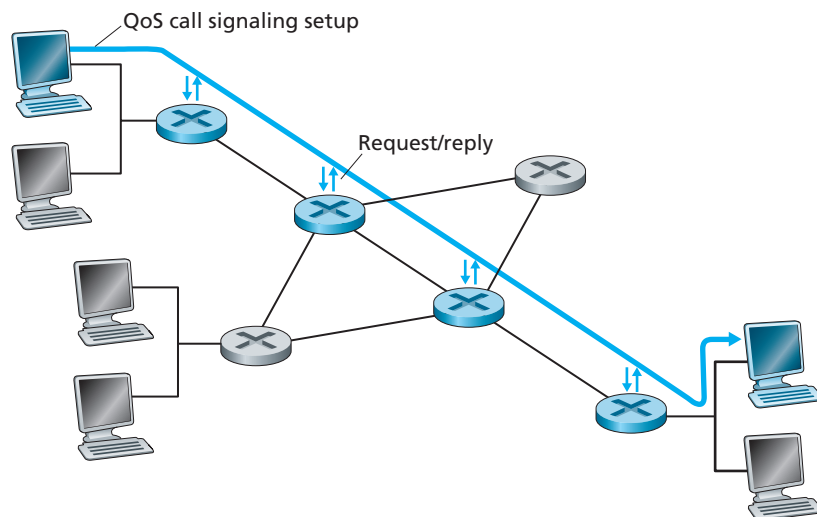
Of course, just as the restaurant manager from Section 1.3.1 should not accept reservations for more tables than the restaurant has, a router that allocates link bandwidth should not allocate more than is available at that link. Typically, a call may reserve only a fraction of the link's bandwidth, and 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.

1. *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-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**.

Figure 7.32 depicts the call setup process. Let's now consider the steps involved in call admission in more detail:

1. *Traffic characterization and specification of the desired QoS.* In order for a router to determine whether or not its resources are sufficient to meet a call's QoS requirement, that call must first declare its QoS requirement, as well as characterize the traffic that it will be sending into the network, and for which it requires a QoS guarantee. In the Internet's Intserv architecture, the so-called Rspec (R for reservation) [RFC 2215] defines the specific QoS being requested by a call; the so-called Tspec (T for traffic) [RFC 2210] characterizes the traffic the sender will be sending into the network or that the receiver will be receiving from the network, respectively. The specific form of the Rspec and Tspec will vary, depending on the service requested, as discussed below. In ATM networks, the user traffic description and the QoS parameter information



**Figure 7.32** ♦ The call setup process

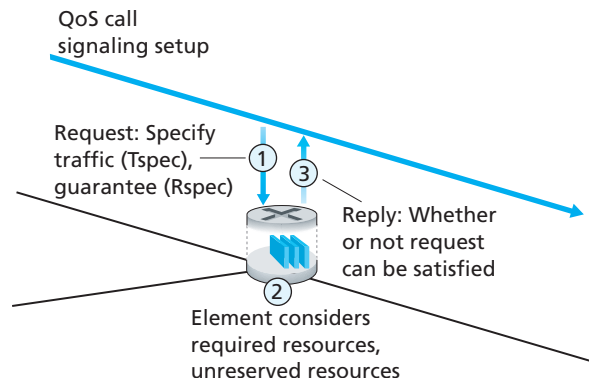
elements carry information for similar purposes as the Tspec and Rspec receptively.

2. *Signaling for call setup.* A call's traffic descriptor and QoS request must be carried to the routers at which resources will be reserved for the call. In the Internet, the RSVP protocol [RFC 2210] is used for this purpose within the Intserv architecture. In ATM networks, the Q2931b protocol carries this information among the ATM network's switches and end point.
3. *Per-element call admission.* Once a router receives the traffic specification and QoS, it must determine whether or not it can admit the call. This call admission decision will depend on the traffic specification, the requested type of service, and the existing resource commitments already made by the router to ongoing calls. Recall that in Section 7.5.3, for example, we saw how the combination of a leaky-bucket-controlled source and WFQ can be used to determine the maximum queuing delay for that source. Per-element call admission is shown in Figure 7.33.

For additional discussion of call setup and admission, see [Breslau 2000; Roberts 2004].

### 7.6.3 Guaranteed QoS in the Internet: Intserv and RSVP

The integrated services (**Intserv**) architecture is a framework developed within the IETF to provide individualized QoS guarantees to individual application sessions in the Internet. Intserv's guaranteed service specification, defined in [RFC 2212], provides firm (mathematically provable) bounds on the queuing delays that a packet will experience in a router. While the details behind guaranteed service are rather complicated, the basic idea is really quite simple. To a first approximation, a source's traffic characterization is given by a leaky bucket (see Section 7.5.2) with



**Figure 7.33** ♦ Per-element call behavior



## PRINCIPLES IN PRACTICE

### THE PRINCIPLE OF SOFT STATE

RSVP is used to install state (bandwidth reservations) in routers, and is known as a *soft-state* protocol. Broadly speaking, we associate the term *soft state* with signaling approaches in which installed state times out (and is removed) unless periodically refreshed by the receipt of a signaling message (typically from the entity that initially installed the state) indicating that the state should continue to remain installed. Since unrefreshed state will eventually time out, soft-state signaling requires neither explicit state removal nor a procedure to remove orphaned state should the state installer crash. Similarly, since state installation and refresh messages will be followed by subsequent periodic refresh messages, reliable signaling is not required. The term *soft state* was coined by Clark [Clark 1988], who described the notion of periodic state refresh messages being sent by an end system, and suggested that with such refresh messages, state could be lost in a crash and then automatically restored by subsequent refresh messages—all transparently to the end system and without invoking any explicit crash-recovery procedures:

“. . . the state information would not be critical in maintaining the desired type of service associated with the flow. Instead, that type of service would be enforced by the end points, which would periodically send messages to ensure that the proper type of service was being associated with the flow. In this way, the state information associated with the flow could be lost in a crash without permanent disruption of the service features being used. I call this concept “soft state,” and it may very well permit us to achieve our primary goals of survivability and flexibility. . . .”

Roughly speaking, then, the essence of a soft-state approach is the use of best-effort periodic state installation/refresh by the state installer and state-removal-by-timeout at the state holder. Soft-state approaches have been taken in numerous protocols, including RSVP, PIM (Section 4.7), SIP (Section 7.4.3), and IGMP (Section 4.7), and in forwarding tables in transparent bridges (Section 5.6).

*Hard-state* signaling takes the converse approach to soft state—installed state remains installed unless explicitly removed by the receipt of a state-teardown message from the state installer. Since the state remains installed unless explicitly removed, hard-state signaling requires a mechanism to remove an orphaned state that remains after the state installer has crashed or departed without removing the state. Similarly, since state installation and removal are performed only once (and without state refresh or state timeout), it is important for the state installer to know when the state has been installed or removed. Reliable (rather than best-effort) signaling protocols are thus typically associated with hard-state protocols. Roughly speaking, then, the essence of a hard-state approach is the reliable and explicit installation and removal of state information. Hard-state approaches have been taken in protocols such as ST-II [Partridge 1992, RFC 1190] and Q.2931 [ITU-T Q.2931 1994].

RSVP has provided for explicit (although optional) removal of reservations since its conception.

ACK-based reliable signaling was introduced as an extension to RSVP in [RFC 2961] and was also suggested in [Pan 1997]. RSVP has thus optionally adopted some elements of a hard-state signaling approach. For a discussion and comparison of soft-state versus hard-state protocols, see [Ji 2003].

parameters  $(r, b)$  and the requested service is characterized by the transmission rate,  $R$ , at which packets will be transmitted. In essence, a call requesting guaranteed service is requiring that the bits in its packet be guaranteed a forwarding rate of  $R$  bits/sec. Given that traffic is specified using a leaky bucket characterization, and a guaranteed rate of  $R$  is being requested, it is also possible to bound the maximum queuing delay at the router. Recall that with a leaky bucket traffic characterization, the amount of traffic (in bits) generated over any interval of length  $t$  is bounded by  $rt + b$ . Recall also from Section 7.5.2 that when a leaky bucket source is fed into a queue that guarantees that queued traffic will be serviced at least at a rate of  $R$  bits per second, the maximum queuing delay experienced by any packet will be bounded by  $b/R$ , as long as  $R$  is greater than  $r$ . A second form of Intserv service guarantee has also been defined, known as controlled load service, which specifies that a call will receive “a quality of service closely approximating the QoS that same flow would receive from an unloaded network element” [RFC 2211].

The Resource ReSerVation Protocol (RSVP) [RFC 2205; Zhang 1993] is an Internet signaling protocol that could be used to perform the call setup signaling needed by Intserv. RSVP has also been used in conjunction with Diffserv to coordinate Diffserv functions across multiple networks, and has also been extended and used as a signaling protocol in other circumstances, perhaps most notably in the form of RSVP-TE [RFC 3209] for MPLS signaling, as discussed in Section 5.8.2.

In an Intserv context, the RSVP protocol allows applications to reserve bandwidth for their data flows. It is used by a host, on the behalf of an application data flow, to request a specific amount of bandwidth from the network. RSVP is also used by the routers to forward bandwidth reservation requests. To implement RSVP, RSVP software must be present in the receivers, senders, and routers along the end-end path shown in Figure 7.32. The two principal characteristics of RSVP are:

- It provides **reservations for bandwidth in multicast trees**, with unicast being handled as a degenerate case of multicast. This is particularly important for multimedia applications such as streaming-broadcast-TV-over-IP, where many receivers may want to receive the same multimedia traffic being sent from a single source.
- It is **receiver-oriented**; that is, the receiver of a data flow initiates and maintains the resource reservation used for that flow. The innovative, receiver-centric view taken by RSVP puts receivers firmly in control of the traffic they receive, for example allowing different receivers to receive and view a multimedia multicast

at different resolutions. This contrasts with the sender-centric view of signaling adopted in ATM's Q2931b.

The RSVP standard [RFC 2205] does not specify *how* the network provides the reserved bandwidth to the data flows. It is merely a protocol that allows the applications to reserve the necessary link bandwidth. Once the reservations are in place, it is up to the routers in the Internet to actually provide the reserved bandwidth to the data flows. This provisioning would likely be done using the policing and scheduling mechanisms (leaky bucket, priority scheduling, weighted fair queuing) discussed in Section 7.5. For more information about RSVP, see [RFC 2205; Zhang 1993] and the additional online electronic material associated with this book.

## 7.7 Summary

Multimedia networking is one of the most exciting (and yet still-to-be-fully-realized) 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. As high-speed access penetrates more residences, this trend will continue—couch potatoes throughout the world will access their favorite video programs through the Internet rather than through the traditional broadcast distribution channels. In addition to audio and video distribution, the Internet is also being used to transport phone calls. In fact, over the next 10 years the Internet may render the traditional circuit-switched telephone system nearly obsolete in many countries. The Internet not only will provide phone service for less money, but will also provide numerous value-added services, such as video conferencing, online directory services, voice messaging services, and Web integration.

In Section 7.1, we classified multimedia applications into three categories: streaming stored audio and video, one-to-many transmission of real-time audio and video, and real-time interactive audio and video. We emphasized that multimedia applications are delay-sensitive and loss-tolerant—characteristics that are very different from static-content applications that are delay-tolerant and loss-intolerant. We also discussed some of the hurdles that today's best-effort Internet places before multimedia applications. We surveyed several proposals to overcome these hurdles, including simply improving the existing networking infrastructure (by adding more bandwidth, more network caches, and more CDN nodes, and by deploying multicast), adding functionality to the Internet so that applications can reserve end-to-end resources (and so that the network can honor these reservations), and finally, introducing service classes to provide service differentiation.

In Sections 7.2 through 7.4, we examined architectures and mechanisms for multimedia networking in a best-effort network. In Section 7.2, we surveyed several