Computer Networking

A Top-Down Approach Featuring the Internet

James F. Kurose ✦ Keith W. Ross

Computer Networking

A Top-Down Approach Featuring the Internet

James F. Kurose University of Massachusetts, Amherst
Keith W. Ross Institute Eurécom

Networking From a New Perspective

Computer Networking: A Top-Down Approach Featuring the Internet provides a new perspective to the study of computer networking concepts—a modern, top-down approach that starts with application-level protocols and then works down the protocol stack. Throughout the book, examples drawn from the Internet architecture show how networking principles are put into practice.

Integrating Principles and Practice

Networking is much more (and much more interesting!) than dry standards specifying message formats and protocol behaviors. Professors Kurose and Ross focus on describing the emerging principles of the field and then illustrate these principles with examples drawn from the Internet architecture. The discussion is lively, engaging, and up to date.

A Top-Down Organization with Early Emphasis on Applications

This book starts with an early discussion of application-level protocols, allowing readers to gain an intuitive feel for network protocols. The focus on application-layer paradigms and application programming interfaces allows readers to get their "hands dirty" early—studying and implementing protocols in the context of applications they use daily. Proceeding through the layered network architecture in a top-down manner, readers can focus on the network services that are needed and then, in turn, study how these services can be provided.

Comprehensive Companion Website

This book features a continuously updated website to enhance teaching and learning at **www.awl.com/kurose-ross**. This site includes material for students and instructors, including the full text with an advanced searching feature and a hyper-linked index, Java applets to help demonstrate difficult concepts, links to up-to-date material, and complete supplements for instructors.

For more information about Addison-Wesley Computer Science books visit www.awl.com/cs

"This book is a gem—Kurose and Ross take a fresh top-down approach and get the complex networking story right! It will be invaluable for students and professionals alike."

LEONARD KLEINROCK
University of California, Los Angeles

"A text on networking that is both entertaining and authoritative—a superb achievement."

FRANK KELLY
University of Cambridge

"The idea (top-down networking) is clearly innovative and, as I read on, more and more appealing."

PAUL AMER
University of Delaware

"...the relation to the Internet will make the material much easier to understand."

WU-CHI FENG
Ohio State University

Pearson Education

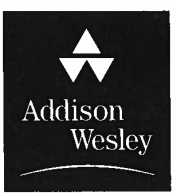
ISBN 0-201-47711-4

Computer Networking

A Top-Down Approach Featuring the Internet

James F. Kurose
University of Massachusetts, Amherst

Keith W. Ross
Institute Eurécom



Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Senior Acquisitions Editor	Susan Hartman
Assistant Editor	Lisa Kalner
Production Supervisor	Patty Mahtani
Art Editor	Helen Reebenacker
Executive Marketing Manager	Michael Hirsch
Composition	Pre-Press Company, Inc.
Technical Art	PD & PS
Copyeditor	Roberta Lewis
Proofreader	Holly McLean Aldis
Cover Design	Joyce Cosentino
Interior Design	Delgado Design
Design Manager	Regina Hagen
Cover Image	© 1999 PhotoDisc, Inc.

Access the latest information about Addison-Wesley titles from our World Wide Web site:
http://www.awl.com/cs

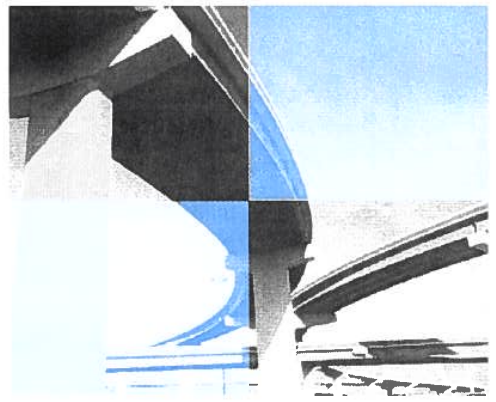
Library of Congress Cataloging-in-Publication Data
Ross, Keith W., 1956-
Computer networking: a top-down approach featuring the Internet / Keith W. Ross, James F. Kurose.
p. cm.
Includes bibliographic references and index
ISBN 0-201-47711-4
1. Internet (Computer network) I. Kurose, James F.
TK5105.875.I57 R689 2001
004.6—dc21 00-025295

1 2 3 4 5 6 7 8 9 10-MA-020100

Table of Contents

Chapter 1

Computer Networks and the Internet



1.1 ✦ What Is the Internet?

In this book we use the public Internet, a specific computer network (and one which probably most readers have used), as our principle vehicle for discussing computer networking protocols. But what is the Internet? We would like to be able to give you a one-sentence definition of the Internet, a definition that you can take home and share with your family and friends. Alas, the Internet is very complex, both in terms of its hardware and software components, as well as in the services it provides.

1.1.1 A Nuts and Bolts Description

Instead of giving a one-sentence definition, let's try a more descriptive approach. There are a couple of ways to do this. One way is to describe the nuts and bolts of the Internet, that is, the basic hardware and software components that make up the Internet. Another way is to describe the Internet in terms of a networking infrastructure that provides services to distributed applications. Let's begin with the nuts-and-bolts description, using Figure 1.1 to illustrate our discussion.

The public Internet is a world-wide **computer network,** that is, a network that interconnects millions of computing devices throughout the world. Most of these computing devices are traditional desktop PCs, Unix-based workstations, and so

universities offer entire courses on audio and video compression, and often offer a separate course on audio compression and a separate course on video compression. We therefore provide here a brief and high-level introduction to the subject.

Audio Compression in the Internet

A continuously varying analog audio signal (which could emanate from speech or music) is normally converted to a digital signal as follows:

1. The analog audio signal is first sampled at some fixed rate, for example, at 8,000 samples per second. The value of each sample is an arbitrary real number.
2. Each of the samples is then "rounded" to one of a finite number of values. This operation is referred to as "quantization." The number of finite values—called quantization values—is typically a power of 2, for example, 256 quantization values.
3. Each of the quantization values is represented by a fixed number of bits. For example, if there are 256 quantization values, then each value—and hence each sample—is represented by 1 byte. Each of the samples is converted to its bit representation. The bit representations of all the samples are concatenated together to form the digital representation of the signal.

As an example, if an analog audio signal is sampled at 8,000 samples per second and each sample is quantized and represented by 8 bits, then the resulting digital signal will have a rate of 64,000 bits per second. This digital signal can then be converted back—that is, decoded—to an analog signal for playback. However, the decoded analog signal is typically different from the original audio signal. By increasing the sampling rate and the number of quantization values, the decoded signal can approximate (and even be exactly equal to) the original analog signal. Thus, there is a clear tradeoff between the quality of the decoded signal and the storage and bandwidth requirements of the digital signal.

The basic encoding technique that we just described is called **pulse code modulation (PCM).** Speech encoding often uses PCM, with a sampling rate of 8,000 samples per second and 8 bits per sample, giving a rate of 64 Kbps. The audio compact disk (CD) also uses PCM, with a sampling rate of 44,100 samples per second with 16 bits per sample; this gives a rate of 705.6 Kbps for mono and 1.411 Mbps for stereo.

A bit rate of 1.411 Mbps for stereo music exceeds most access rates, and even 64 Kbps for speech exceeds the access rate for a dial-up modem user. For these reasons, PCM-encoded speech and music are rarely used in the Internet. Instead, compression techniques are used to reduce the bit rates of the stream. Popular compression techniques for speech include **GSM** (13 Kbps), **G.729** (8 Kbps), and **G.723.3** (both 6.4 and 5.3 Kbps), and also a large number of proprietary techniques,

including those used by RealNetworks. A popular compression technique for near CD-quality stereo music is **MPEG layer 3,** more commonly known as **MP3.** MP3 compresses the bit rate for music to 128 or 112 Kbps, and produces very little sound degradation. When an MP3 file is broken up into pieces, each piece is still playable. This headerless file format allows MP3 music files to be streamed across the Internet (assuming the playback bit rate and speed of the Internet connection are compatible). The MP3 compression standard is complex, using psychoacoustic masking, redundancy reduction, and bit reservoir buffering.

Video Compression in the Internet

A video is a sequence of images, with images typically being displayed at a constant rate, for example at 24 or 30 images per second. An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and color. There are two types of redundancy in video, both of which can be exploited for compression. Spatial redundancy is the redundancy within a given image. For example, an image that consists of mostly white space can be efficiently compressed. Temporal redundancy reflects repetition from image to subsequent image. If, for example, an image and the subsequent image are exactly the same, there is no reason to re-encode the subsequent image; it is more efficient to simply indicate during encoding that the subsequent image is exactly the same.

The MPEG compression standards are among the most popular compression techniques. These include **MPEG 1** for CD-ROM quality video (1.5 Mbps), **MPEG 2** for high-quality **DVD** video (3-6 Mbps), and **MPEG 4** for object-oriented video compression. The MPEG standard draws heavily from the JPEG standard for image compression. The **H.261** video compression standards are also very popular in the Internet. There are also numerous proprietary standards.

Readers interested in learning more about audio and video encoding are encouraged to see [Rao 1996] and [Solari 1997]. A good book on multimedia networking in general is [Crowcroft 1999].

6.2 ✦ Streaming Stored Audio and Video

In recent years, audio/video streaming has become a popular application and a major consumer of network bandwidth. This trend is likely to continue for several reasons. First, the cost of disk storage is decreasing rapidly, even faster than processing and bandwidth costs. Cheap storage will lead to a significant increase in the amount of stored/audio video in the Internet. For example, shared MP3 audio files of rock music via [Napster 2000] has become incredibly popular among college and high school

students. Second, improvements in Internet infrastructure, such as high-speed residential access (that is, cable modems and ADSL, as discussed in Chapter 1), network caching of video (see Section 2.2), and new QoS-oriented Internet protocols (see Sections 6.5–6.9) will greatly facilitate the distribution of stored audio and video. And third, there is an enormous pent-up demand for high-quality video streaming, an application that combines two existing killer communication technologies—television and the on-demand Web.

In audio/video streaming, clients request compressed audio/video files that are resident on servers. As we'll discuss in this section, these servers can be "ordinary" Web servers, or can be special streaming servers tailored for the audio/video streaming application. Upon client request, the server directs an audio/video file to the client by sending the file into a socket. Both TCP and UDP socket connections are used in practice. Before sending the audio/video file into the network, the file is segmented, and the segments are typically encapsulated with special headers appropriate for audio/video traffic. The **Real-time protocol (RTP),** discussed in Section 6.4, is a public-domain standard for encapsulating such segments. Once the client begins to receive the requested audio/video file, the client begins to render the file (typically) within a few seconds. Most existing products also provide for user interactivity, for example, pause/resume and temporal jumps within the audio/video file. This user interactivity also requires a protocol for client/server interaction. **Real-time streaming protocol (RTSP),** discussed at the end of this section, is a public-domain protocol for providing user interactivity.

Audio/video streaming is often requested by users through a Web client (that is, browser). But because audio/video playout is not integrated directly in today's Web clients, a separate **helper application** is required for playing out the audio/video. The helper application is often called a **media player,** the most popular of which are currently RealNetworks' Real Player and the Microsoft Windows Media Player. The media player performs several functions, including:

- *Decompression.* Audio/video is almost always compressed to save disk storage and network bandwidth. A media player must decompress the audio/video on the fly during playout.
- *Jitter removal.* Packet jitter is the variability of source-to-destination delays of packets within the same packet stream. Since audio and video must be played out with the same timing with which it was recorded, a receiver will buffer received packets for a short period of time to remove this jitter. We'll examine this topic in detail in Section 6.3.
- *Error correction.* Due to unpredictable congestion in the Internet, a fraction of packets in the packet stream can be lost. If this fraction becomes too large, user-perceived audio/video quality becomes unacceptable. To this end, many streaming systems attempt to recover from losses by either (1) reconstructing lost packets through the transmission of redundant packets, (2) by having the client explicitly request retransmissions of lost packets, (3) masking loss by interpolating the missing data from the received data.
- *Graphical user interface with control knobs.* This is the actual interface that the user interacts with. It typically includes volume controls, pause/resume buttons, sliders for making temporal jumps in the audio/video stream, and so on.

Plug-ins may be used to embed the user interface of the media player within the window of the Web browser. For such embeddings, the browser reserves screen space on the current Web page, and it is up to the media player to manage the screen space. But either appearing in a separate window or within the browser window (as a plug-in), the media player is a program that is being executed separately from the browser.

6.2.1 Accessing Audio and Video from a Web Server

Stored audio/video can reside either on a Web server that delivers the audio/video to the client over HTTP, or on an audio/video streaming server that delivers the audio/video over non-HTTP protocols (protocols that can be either proprietary or open standards). In this subsection, we examine delivery of audio/video from a Web server; in the next subsection, we examine delivery from a streaming server.

Consider first the case of audio streaming. When an audio file resides on a Web server, the audio file is an ordinary object in the server's file system, just as are HTML and JPEG files. When a user wants to hear the audio file, the user's host establishes a TCP connection with the Web server and sends an HTTP request for the object (see Section 2.2). Upon receiving a request, the Web server bundles the audio file in an HTTP response message and sends the response message back into the TCP connection. The case of video can be a little more tricky, because the audio and video parts of the "video" may be stored in two different files, that is, they may be two different objects in the Web server's file system. In this case, two separate HTTP requests are sent to the server (over two separate TCP connections for HTTP/1.0), and the audio and video files arrive at the client in parallel. It is up to the client to manage the synchronization of the two streams. It is also possible that the audio and video are interleaved in the same file, so that only one object need be sent to the client. To keep our discussion simple, for the case of "video" we assume that the audio and video are contained in one file.

A naive architecture for audio/video streaming is shown in Figure 6.1. In this architecture:

1. The browser process establishes a TCP connection with the Web server and requests the audio/video file with an HTTP request message.
2. The Web server sends to the browser the audio/video file in an HTTP response message.

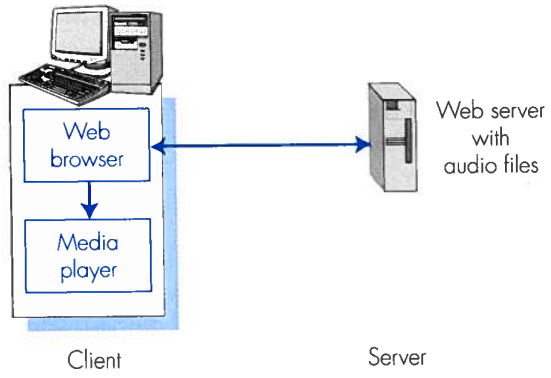


Figure 6.1 ✦ A naive implementation for audio streaming

3. The content-type header line in the HTTP response message indicates a specific audio/video encoding. The client browser examines the content-type of the response message, launches the associated media player, and passes the file to the media player.
4. The media player then renders the audio/video file.

Although this approach is very simple, it has a major drawback: The media player (that is, the helper application) must interact with the server through the intermediary of a Web browser. This can lead to many problems. In particular, when the browser is an intermediary, the entire object must be downloaded before the browser passes the object to a helper application. The resulting delay before playout can begin is typically unacceptable for audio/video clips of moderate length. For this reason, audio/video streaming implementations typically have the server send the audio/video file directly to the media player process. In other words, a direct socket connection is made between the server process and the media player process. As shown in Figure 6.2, this is typically done by making use of a **meta file,** a file that provides information (for example, URL, type of encoding) about the audio/video file that is to be streamed.

A direct TCP connection between the server and the media player is obtained as follows:

1. The user clicks on a hyperlink for an audio/video file.
2. The hyperlink does not point directly to the audio/video file, but instead to a meta file. The meta file contains the URL of the actual audio/video file. The HTTP response message that encapsulates the meta file includes a content-type header line that indicates the specific audio/video application.

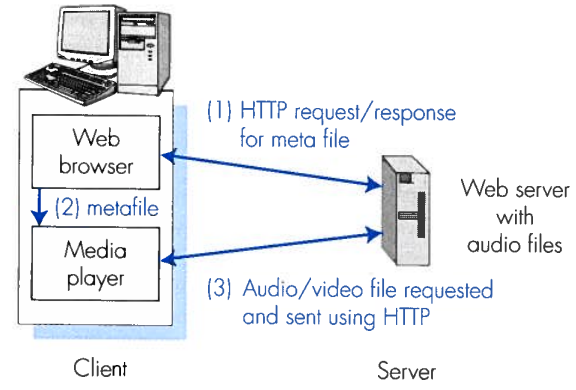


Figure 6.2 ✦ Web server sends audio/video directly to the media player

3. The client browser examines the content-type header line of the response message, launches the associated media player, and passes the entire body of the response message (that is, the meta file) to the media player.
4. The media player sets up a TCP connection directly with the HTTP server. The media player sends an HTTP request message for the audio/video file into the TCP connection.
5. The audio/video file is sent within an HTTP response message to the media player. The media player streams out the audio/video file.

The importance of the intermediate step of acquiring the meta file is clear. When the browser sees the content-type for the file, it can launch the appropriate media player, and thereby have the media player directly contact the server.

We have just learned how a meta file can allow a media player to dialogue directly with a Web server housing an audio/video. Yet many companies that sell products for audio/video streaming do not recommend the architecture we just described. This is because the architecture has the media player communicate with the server over HTTP and hence also over TCP. HTTP is often considered insufficiently rich to allow for satisfactory user interaction with the server; in particular, HTTP does not easily allow a user (through the media server) to send pause/resume, fast-forward, and temporal jump commands to the server.

6.2.2 Sending Multimedia from a Streaming Server to a Helper Application

In order to get around HTTP and/or TCP, audio/video can be stored on and sent from a streaming server to the media player. This streaming server could be a proprietary streaming server, such as those marketed by RealNetworks and Microsoft, or could

be a public-domain streaming server. With a streaming server, audio/video can be sent over UDP (rather than TCP) using application-layer protocols that may be better tailored than HTTP to audio/video streaming.

This architecture requires two servers, as shown in Figure 6.3. One server, the HTTP server, serves Web pages (including meta files). The second server, the **streaming server,** serves the audio/video files. The two servers can run on the same end system or on two distinct end systems. The steps for this architecture are similar to those described in the previous architecture. However, now the media player requests the file from a streaming server rather than from a Web server, and now the media player and streaming server can interact using their own protocols. These protocols can allow for rich user interaction with the audio/video stream.

In the architecture of Figure 6.3, there are many options for delivering the audio/video from the streaming server to the media player. A partial list of the options is given below:

1. The audio/video is sent over UDP at a constant rate equal to the drain rate at the receiver (which is the encoded rate of the audio/video). For example, if the audio is compressed using GSM at a rate of 13 Kbps, then the server clocks out the compressed audio file at 13 Kbps. As soon as the client receives compressed audio/video from the network, it decompresses the audio/video and plays it back.
2. This is the same as option 1, but the media player delays playout for 2–5 seconds in order to eliminate network-induced jitter. The client accomplishes this task by placing the compressed media that it receives from the network into a **client buffer,** as shown in Figure 6.4. Once the client has "prefetched" a few seconds of the media, it begins to drain the buffer. For this, and the previous option, the fill rate $x(t)$ is equal to the drain rate d, except when there is packet loss, in which case $x(t)$ is momentarily less than d.

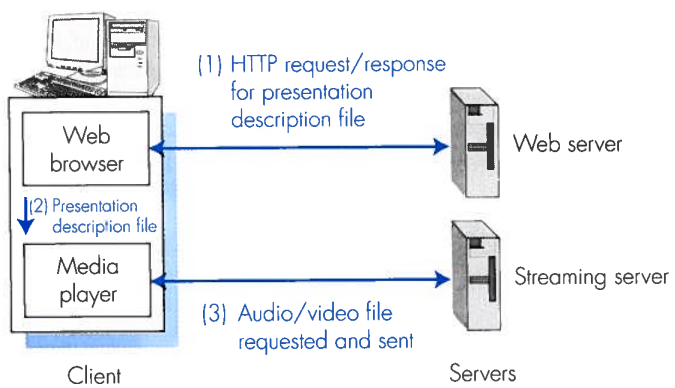


Figure 6.3 ✦ Streaming from a streaming server to a media player

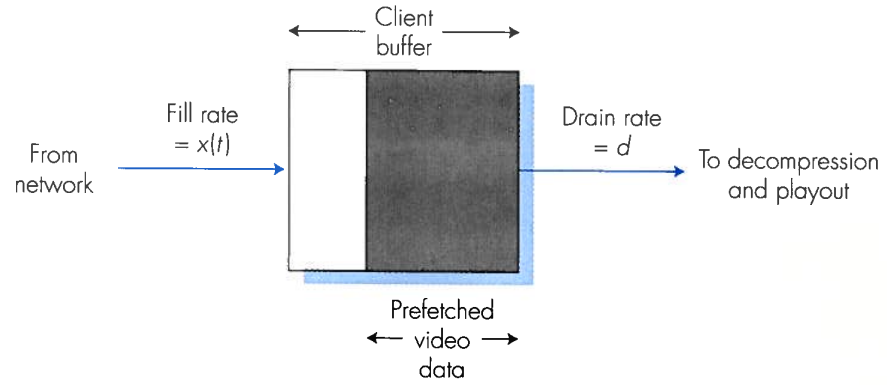


Figure 6.4 ✦ Client buffer being filled at rate $x(t)$ and drained at rate d

3. The media is sent over TCP. The server pushes the media file into the TCP socket as quickly as it can; the client (i.e., media player) reads from the TCP socket as quickly as it can, and places the compressed video into the media player buffer. After an initial 2–5 second delay, the media player reads from its buffer at a rate d and forwards the compressed media to decompression and playback. Because TCP retransmits lost packets, it has the potential to provide better sound quality than UDP. On the other hand, the fill rate $x(t)$ now fluctuates with time due to TCP congestion control and window flow control. In fact, after packet loss, TCP congestion control may reduce the instantaneous rate to less than d for long periods of time. This can empty the client buffer and introduce undesirable pauses into the output of the audio/video stream at the client.

For the third option, the behavior of $x(t)$ will very much depend on the size of the client buffer (which is not to be confused with the TCP receive buffer). If this buffer is large enough to hold all of the media file (possibly within disk storage), then TCP will make use of all the instantaneous bandwidth available to the connection, so that $x(t)$ can become much larger than d. If $x(t)$ becomes much larger than d for long periods of time, then a large portion of media is prefetched into the client, and subsequent client starvation is unlikely. If, on the other hand, the client buffer is small, then $x(t)$ will fluctuate around the drain rate d. Risk of client starvation is much larger in this case.

6.2.3 Real-Time Streaming Protocol (RTSP)

Many Internet multimedia users (particularly those who grew up with a remote TV control in hand) will want to *control* the playback of continuous media by pausing playback, repositioning playback to a future or past point of time, visual fast-forwarding playback, visual rewinding playback, and so on. This functionality is similar to what a

user has with a VCR when watching a video cassette or with a CD player when listening to a music CD. To allow a user to control playback, the media player and server need a protocol for exchanging playback control information. RTSP, defined in RFC 2326, is such a protocol.

But before getting into the details of RTSP, let us first indicate what RTSP does not do:

- RTSP does not define compression schemes for audio and video.
- RTSP does not define how audio and video is encapsulated in packets for transmission over a network; encapsulation for streaming media can be provided by RTP or by a proprietary protocol. (RTP is discussed in Section 6.4.) For example, RealMedia's G2 server and player use RTSP to send control information to each other. But the media stream itself can be encapsulated in RTP packets or in some proprietary data format.
- RTSP does not restrict how streamed media is transported; it can be transported over UDP or TCP.
- RTSP does not restrict how the media player buffers the audio/video. The audio/video can be played out as soon as it begins to arrive at the client, it can be played out after a delay of a few seconds, or it can be downloaded in its entirety before playout.

So if RTSP doesn't do any of the above, what does RTSP do? RTSP is a protocol that allows a media player to control the transmission of a media stream. As mentioned above, control actions include pause/resume, repositioning of playback, fast forward and rewind. RTSP is a so-called **out-of-band protocol.** In particular, the RTSP messages are sent out-of-band, whereas the media stream, whose packet structure is not defined by RTSP, is considered "in-band." RTSP messages use a different port number, 544, than the media stream. The RTSP specification [RFC 2326] permits RTSP messages to be sent over either TCP or UDP.

Recall from Section 2.3, that file transfer protocol (FTP) also uses the out-of-band notion. In particular, FTP uses two client/server pairs of sockets, each pair with its own port number: one client/server socket pair supports a TCP connection that transports control information; the other client/server socket pair supports a TCP connection that actually transports the file. The RTSP channel is in many ways similar to FTP's control channel.

Let us now walk through a simple RTSP example, which is illustrated in Figure 6.5. The Web browser first requests a presentation description file from a Web server. The presentation description file can have references to several continuous-media files as well as directives for synchronization of the continuous-media files. Each reference to a continuous-media file begins with the URL method, `rtsp://`. Below we provide a sample presentation file that has been adapted from [Schulzrinne 1997]. In this presentation, an audio and video stream are played in parallel and in lip sync (as part of the same "group"). For the audio stream, the

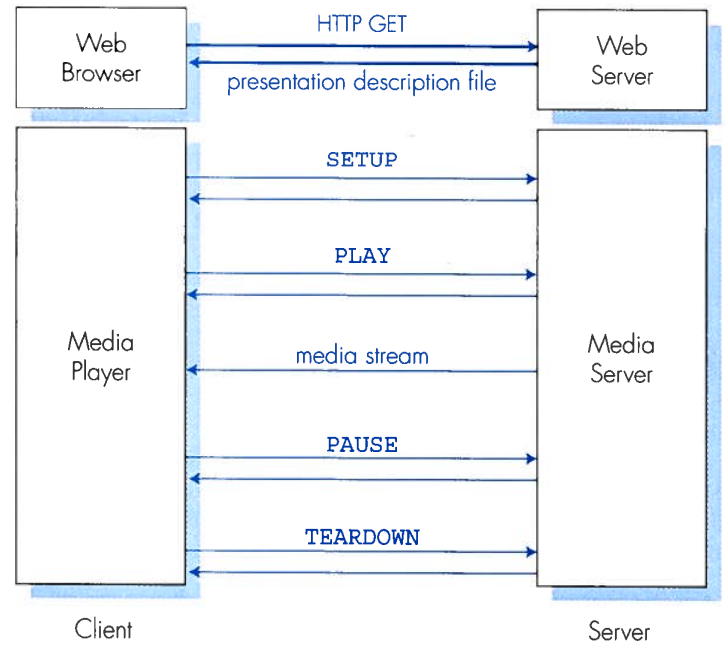


Figure 6.5 ✦ Interaction between client and server using RTSP

media player can choose ("switch") between two audio recordings, a low-fidelity recording and a high-fidelity recording.

```
<title>Twister/title>
<session>
   <group language=en lipsync>
            <switch>
                <track type=audio
                       e="PCMU/8000/1"
                       src = "rtsp://audio.example.com/twister/audio.en/lofi">
                <track type=audio
                       e="DVI4/16000/2" pt="90 DVI4/8000/1"
                       src="rtsp://audio.example.com/twister/audio.en/hifi">
            </switch>
          <track type="video/jpeg"
                         src="rtsp://video.example.com/twister/video">
   </group>
</session>
```

The Web server encapsulates the presentation description file in an HTTP response message and sends the message to the browser. When the browser receives the HTTP response message, the browser invokes a media player (that is, the helper application) based on the content-type field of the message. The presentation description file includes references to media streams, using the URL method `rtsp://`, as shown in the above sample. As shown in Figure 6.5, the player and the server then send each other a series of RTSP messages. The player sends an RTSP SETUP request, and the server sends an RTSP SETUP response. The player sends an RTSP PLAY request, say, for low-fidelity audio, and the server sends an RTSP PLAY response. At this point, the streaming server pumps the low-fidelity audio into its own in-band channel. Later, the media player sends an RTSP PAUSE request, and the server responds with an RTSP PAUSE response. When the user is finished, the media player sends an RTSP TEARDOWN request, and the server responds with an RTSP TEARDOWN response.

Each RTSP session has a session identifier, which is chosen by the server. The client initiates the session with the SETUP request, and the server responds to the request with an identifier. The client repeats the session identifier for each request, until the client closes the session with the TEARDOWN request. The following is a simplified example of an RTSP session between a client (**C:**) and a sender (**S:**).

```
C: SETUP rtsp://audio.example.com/twister/audio RTSP/1.0
   Transport: rtp/udp; compression; port=3056; mode=PLAY
S: RTSP/1.0 200 1 OK
   Session 4231
C: PLAY rtsp://audio.example.com/twister/audio.en/lofi
     RTSP/1.0
   Session: 4231
   Range: npt=0-
C: PAUSE rtsp://audio.example.com/twister/audio.en/
     lofi RTSP/1.0
   Session: 4231
   Range: npt=37
C: TEARDOWN rtsp://audio.example.com/twister/audio.en/
   lofi RTSP/1.0 Session: 4231
S: 200 3 OK
```

Notice that in this example, the player chose not to play back the complete presentation, but instead only the low-fidelity portion of the presentation. The RTSP protocol is actually capable of doing much more than described in this brief introduction. In particular, RTSP has facilities that allow clients to stream toward the server (for example, for recording). RTSP has been adopted by RealNetworks, currently the industry leader in audio/video streaming. Henning Schulzrinne makes available a Web page on RTSP [Schulzrinne 1999].

6.3 ✦ Making the Best of the Best-Effort Service: An Internet Phone Example

The Internet's network-layer protocol, IP, provides a best-effort service. That is to say that the Internet makes its best effort to move each datagram from source to destination as quickly as possible. However, best-effort service does not make any promises whatsoever on the extent of the end-to-end delay for an individual packet, or on the extent of packet jitter and packet loss within the packet stream.

Real-time interactive multimedia applications, such as Internet phone and real-time video conferencing, are acutely sensitive to packet delay, jitter, and loss. Fortunately, designers of these applications can introduce several useful mechanisms that can preserve good audio and video quality as long as delay, jitter, and loss are not excessive. In this section, we examine some of these mechanisms. To keep the discussion concrete, we discuss these mechanisms in the context of an **Internet phone application,** described below. The situation is similar for real-time video conferencing applications [Bolot 1994].

The speaker in our Internet phone application generates an audio signal consisting of alternating talk spurts and silent periods. In order to conserve bandwidth, our Internet phone application only generates packets during talk spurts. During a talk spurt the sender generates bytes at a rate of 8 Kbytes per second, and every 20 milliseconds the sender gathers bytes into chunks. Thus, the number of bytes in a chunk is (20 msecs) · (8 Kbytes/sec) = 160 bytes. A special header is attached to each chunk, the contents of which is discussed below. The chunk, along with its header, are encapsulated in a UDP segment, and then the UDP datagram is sent into the socket interface. Thus, during a talk spurt, a UDP segment is sent every 20 msec.

If each packet makes it to the receiver and has a small constant end-to-end delay, then packets arrive at the receiver periodically every 20 msec during a talk spurt. In these ideal conditions, the receiver can simply play back each chunk as soon as it arrives. But, unfortunately, some packets can be lost and most packets will not have the same end-to-end delay, even in a lightly congested Internet. For this reason, the receiver must take more care in (1) determining when to play back a chunk, and (2) determining what to do with a missing chunk.

6.3.1 The Limitations of a Best-Effort Service

We mentioned that the best-effort service can lead to packet loss, excessive end-to-end delay, and delay jitter. Let's examine these issues in more detail.

Packet Loss

Consider one of the UDP segments generated by our Internet phone application. The UDP segment is encapsulated in an IP datagram. As the datagram wanders through the network, it passes through buffers (that is, queues) in the routers in order to

signaling protocol of choice. RFC 2210 describes the use of the RSVP resource reservation protocol with the Intserv architecture.

3. *Per-element call admission.* Once a router receives the Tspec and Rspec for a session requesting a QoS guarantee, it can 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 sessions. Per-element call admission is shown in Figure 6.32.

The Intserv architecture defines two major classes of service: guaranteed service and controlled-load service. We will see shortly that each provides a very different form of a quality of service guarantee.

6.7.1 Guaranteed Quality of Service

The 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 6.6) with parameters (r,b) and the requested service is characterized by a transmission rate, R, at which packets will be transmitted. In essence, a session requesting guaranteed service is requiring that the

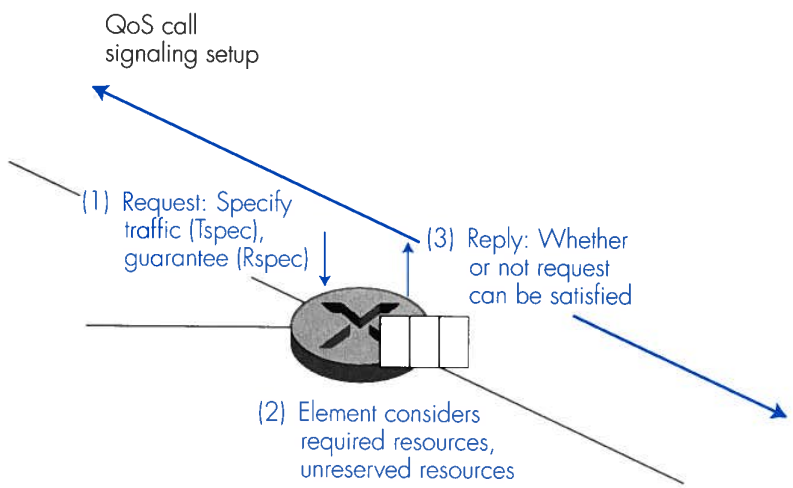


Figure 6.32 ✦ Per-element call behavior

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 6.6, 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. The actual delay bound guaranteed under the guaranteed service definition is slightly more complicated, due to packetization effects (the simple b/R bound assumes that data is in the form of a fluid-like flow rather than discrete packets), the fact that the traffic arrival process is subject to the peak rate limitation of the input link (the simple b/R bound assumes that a burst of b bits can arrive in zero time), and possible additional variations in a packet's transmission time.

6.7.2 Controlled-Load Network Service

A session receiving controlled-load service will receive "a quality of service closely approximating the QoS that same flow would receive from an unloaded network element" [RFC 2211]. In other words, the session may assume that a "very high percentage" of its packets will successfully pass through the router without being dropped and will experience a queuing delay in the router that is close to zero. Interestingly, controlled load service makes no quantitative guarantees about performance—it does not specify what constitutes a "very high percentage" of packets nor what quality of service closely approximates that of an unloaded network element.

The controlled-load service targets real-time multimedia applications that have been developed for today's Internet. As we have seen, these applications perform quite well when the network is unloaded, but rapidly degrade in performance as the network becomes more loaded.

6.8 ✦ RSVP

We learned in Section 6.7 that in order for a network to provide QoS guarantees, there must be a signaling protocol that allows applications running in hosts to reserve resources in the Internet. RSVP [RFC 2205; Zhang 1993], is such a signaling protocol for the Internet.

When people talk about *resources* in the Internet context, they usually mean link bandwidth and router buffers. To keep the discussion concrete and focused, however, we shall assume that the word *resource* is synonymous with *bandwidth*. For our pedagogic purposes, RSVP stands for bandwidth reservation protocol.

6.8.1 The Essence of RSVP

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. The two principal characteristics of RSVP are:

1. It provides **reservations for bandwidth in multicast trees** (unicast is handled as a degenerate case of multicast).
2. It is **receiver-oriented,** that is, the receiver of a data flow initiates and maintains the resource reservation used for that flow.

These two characteristics are illustrated in Figure 6.33. The diagram shows a multicast tree with data flowing from the top of the tree to hosts at the bottom of the tree. Although data originates from the sender, the reservation messages originate from the receivers. When a router forwards a reservation message upstream toward the sender, the router may merge the reservation message with other reservation messages arriving from downstream.

Before discussing RSVP in greater detail, we need to consider the notion of a **session.** As with RTP, a session can consist of multiple multicast data flows. Each sender in a session is the source of one or more data flows; for example, a sender might be the source of a video data flow and an audio data flow. Each data flow in a session has the same multicast address. To keep the discussion concrete, we assume that routers and hosts identify the session to which a packet belongs by the packet's multicast address. This assumption is somewhat restrictive; the actual RSVP specification allows for more general methods to identify a session. Within a session, the data flow to which a packet belongs also needs to be identified. This could be done, for example, with the flow identifier field in IPv6.

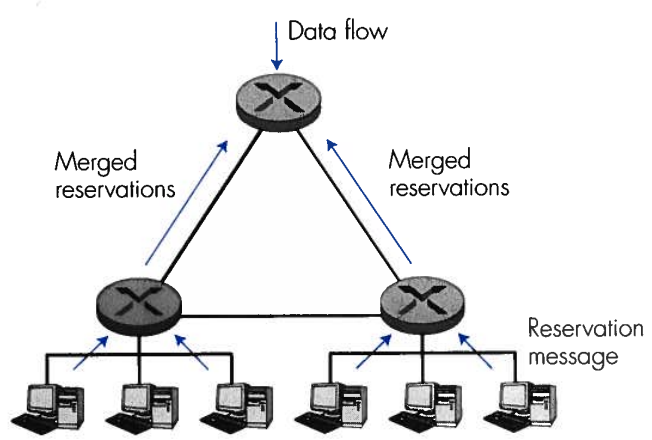


Figure 6.33 ✦ RSVP: Multicast- and receiver-oriented

What RSVP Is Not

We emphasize that 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 with the scheduling mechanisms (priority scheduling, weighted fair queuing, etc.) discussed in Section 6.6.

It is also important to understand that RSVP is not a routing protocol—it does not determine the links in which the reservations are to be made. Instead it depends on an underlying routing protocol (unicast or multicast) to determine the routes for the flows. Once the routes are in place, RSVP can reserve bandwidth in the links along these routes. (We shall see shortly that when a route changes, RSVP re-reserves resources.) Once the reservations are in place, the routers' packet schedulers must actually provide the reserved bandwidth to the data flows. Thus, RSVP is only one piece—albeit an important piece—in the QoS guarantee puzzle.

RSVP is sometimes referred to as a *signaling protocol.* By this it is meant that RSVP is a protocol that allows hosts to establish and tear down reservations for data flows. The term "signaling protocol" comes from the jargon of the circuit-switched telephony community.

Heterogeneous Receivers

Some receivers can receive a flow at 28.8 Kbps, others at 128 Kbps, and yet others at 10 Mbps or higher. This heterogeneity of the receivers poses an interesting question. If a sender is multicasting a video to a group of heterogeneous receivers, should the sender encode the video for low quality at 28.8 Kbps, for medium quality at 128 Kbps, or for high quality at 10 Mbps? If the video is encoded at 10 Mbps, then only the users with 10 Mbps access will be able to watch the video. On the other hand, if the video is encoded at 28.8 Kbps, then the 10 Mbps users will have to see a low-quality image when they know they can see something much better.

To resolve this dilemma it is often suggested that video and audio be encoded in layers. For example, a video might be encoded into two layers: a base layer and an enhancement layer. The base layer could have a rate of 20 Kbps whereas the enhancement layer could have a rate of 100 Kbps; in this manner receivers with 28.8 Kbps access could receive the low-quality base-layer image, and receivers with 128 Kbps could receive both layers to construct a high-quality image.

We note that the sender does not need to know the receiving rates of all the receivers. It only needs to know the maximum rate of all its receivers. The sender encodes the video or audio into multiple layers and sends all the layers up to the

maximum rate into multicast tree. The receivers pick out the layers that are appropriate for their receiving rates. In order to not excessively waste bandwidth in the network's links, the heterogeneous receivers must communicate to the network the rates they can handle. We shall see that RSVP gives foremost attention to the issue of reserving resources for heterogeneous receivers.

6.8.2 A Few Simple Examples

Let us first describe RSVP in the context of a concrete one-to-many multicast example. Suppose there is a source that is transmitting the video of a major sporting event into the Internet. This session has been assigned a multicast address, and the source stamps all of its outgoing packets with this multicast address. Also suppose that an underlying multicast routing protocol has established a multicast tree from the sender to four receivers as shown below; the numbers next to the receivers are the rates at which the receivers want to receive data. Let us also assume that the video is layered and encoded to accommodate this heterogeneity of receiver rates.

Crudely speaking, RSVP operates as follows for this example. Each receiver sends a **reservation message** upstream into the multicast tree. This reservation message specifies the rate at which the receiver would like to receive the data from the source. When the reservation message reaches a router, the router adjusts its packet scheduler to accommodate the reservation. It then sends a reservation upstream. The amount of bandwidth reserved upstream from the router depends on the bandwidths

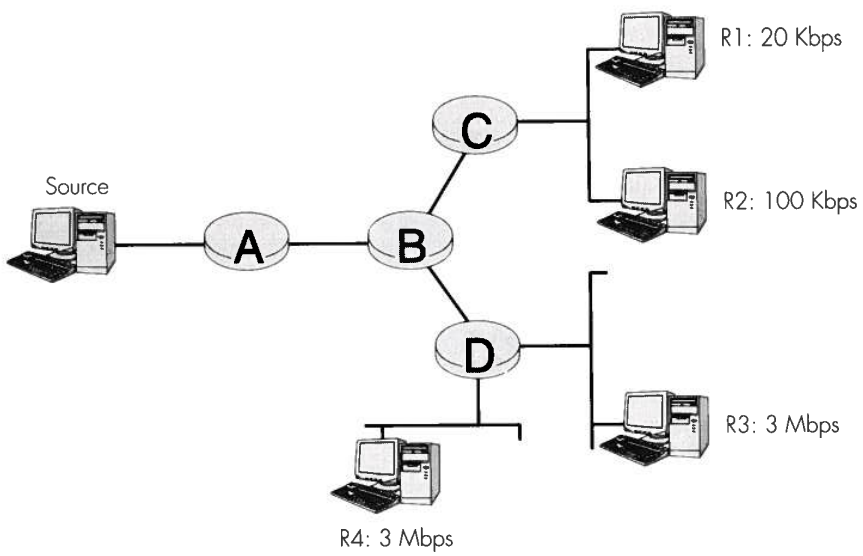


Figure 6.34 ✦ An RSVP example

reserved downstream. In the example in Figure 6.34, receivers R1, R2, R3, and R4 reserve 20 Kbps, 100 Kbps, 3 Mbps, and 3 Mbps, respectively. Thus router D's downstream receivers request a maximum of 3 Mbps. For this one-to-many transmission, Router D sends a reservation message to Router B requesting that Router B reserve 3 Mbps on the link between the two routers. Note that only 3 Mbps are reserved and not 3+3=6 Mbps; this is because receivers R3 and R4 are watching the same sporting event, so their reservations may be merged. Similarly, Router C requests that Router B reserve 100 Kbps on the link between routers B and C; the layered encoding ensures that receiver R1's 20 Kbps stream is included in the 100 Kbps stream. Once Router B receives the reservation message from its downstream routers and passes the reservations to its schedulers, it sends a new reservation message to its upstream router, Router A. This message reserves 3 Mbps of bandwidth on the link from Router A to Router B, which is again the maximum of the downstream reservations.

We see from this first example that RSVP is **receiver-oriented,** that is, the receiver of a data flow initiates and maintains the resource reservation used for that flow. Note that each router receives a reservation message from each of its downstream links in the multicast tree and sends only one reservation message into its upstream link.

As another example, suppose that four persons are participating in a video conference, as shown in Figure 6.35. Each person has three windows open on her computer to look at the other three persons. Suppose that the underlying routing protocol has established the multicast tree among the four hosts as shown in the diagram below. Finally, suppose each person wants to see each of the videos at 3 Mbps. Then

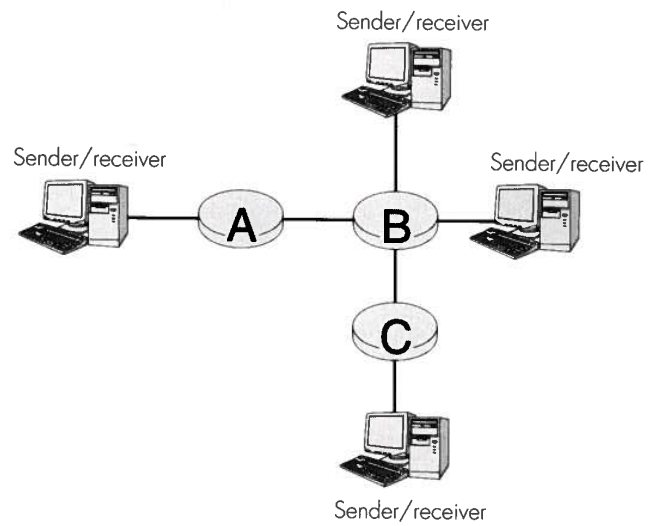


Figure 6.35 ✦ An RSVP video conference example

on each of the links in this multicast tree, RSVP would reserve 9 Mbps in one direction and 3 Mbps in the other direction. Note that RSVP does not merge reservations in this example, as each person wants to receive three distinct streams.

Now consider an audio conference among the same four persons over the same multicast tree. Suppose b bps are needed for an isolated audio stream. Because in an audio conference it is rare that more than two persons speak at the same time, it is not necessary to reserve $3 \cdot b$ bps into each receiver; $2 \cdot b$ should suffice. Thus, in this last application we can conserve bandwidth by merging reservations.

Call Admission

Just as the manager of a restaurant should not accept reservations for more tables than the restaurant has, the amount of bandwidth on a link that a router reserves should not exceed the link's capacity. Thus whenever a router receives a new reservation message, it must first determine if its downstream links on the multicast tree can accommodate the reservation. This **admission test** is performed whenever a router receives a reservation message. If the admission test fails, the router rejects the reservation and returns an error message to the appropriate receiver(s).

RSVP does not define the admission test, but it assumes that the routers perform such a test and that RSVP can interact with the test.

6.8.3 Path Messages

So far we have only discussed the RSVP reservation messages. These messages originate at the receivers and flow upstream toward the senders. **Path messages** are another important RSVP message type; they originate at the senders and flow downstream toward the receivers.

The principal purpose of the path messages is to let the routers know the links on which they should forward the reservation messages. Specifically, a path message sent within the multicast tree from a Router A to a Router B contains Router A's unicast IP address. Router B puts this address in a path-state table, and when it receives a reservation message from a downstream node it accesses the table and learns that it should send a reservation message up the multicast tree to Router A. In the future some routing protocols may supply reverse path forwarding information directly, replacing the reverse-routing function of the path state.

Along with some other information, the path messages also contain a *sender Tspec,* which defines the traffic characteristics of the data stream that the sender will generate (see Section 6.7). This Tspec can be used to prevent over-reservation.

6.8.4 Reservation Styles

Through its **reservation style,** a reservation message specifies whether merging of reservations from the same session is permissible. A reservation style also specifies

the session senders from which a receiver desires to receive data. Recall that a router can identify the sender of a datagram from the datagram's source IP address.

There are currently three reservation styles defined: *wildcard-filter style, fixed-filter style,* and *shared-explicit style.*

- *Wildcard-filter style.* When a receiver uses the wildcard-filter style in its reservation message, it is telling the network that it wants to receive all flows from all upstream senders in the session and that its bandwidth reservation is to be shared among the senders.
- *Fixed-filter style.* When a receiver uses the fixed-filter style in its reservation message, it specifies a list of senders from which it wants to receive a data flow along with a bandwidth reservation for each of these senders. These reservations are distinct, that is, they are not to be shared.
- *Shared-explicit style.* When a receiver uses the shared-explicit style in its reservation message, it specifies a list of senders from which it wants to receive a data flow along with a single bandwidth reservation. This reservation is to be shared among all the senders in the list.

Shared reservations, created by the wildcard filter and the shared-explicit styles, are appropriate for a multicast session whose sources are unlikely to transmit simultaneously. Packetized audio is an example of an application suitable for shared reservations; because a limited number of people talk at once, each receiver might issue a wildcard-filter or a shared-explicit reservation request for twice the bandwidth required for one sender (to allow for overspeaking). On the other hand, the fixed-filter reservation, which creates distinct reservations for the flows from different senders, is appropriate for video teleconferencing.

Examples of Reservation Styles

Following the RSVP Internet RFC, let's next consider a few examples of the three reservation styles. In Figure 6.36, a router has two incoming interfaces, labeled A and B, and two outgoing interfaces, labeled C and D. The many-to-many multicast session has three senders—S1, S2, and S3—and three receivers—R1, R2, and R3. Figure 6.36 also shows that interface D is connected to a LAN.

Suppose first that all of the receivers use the wildcard-filter reservation. As shown in the Figure 6.37, receivers R1, R2, and R3 want to reserve $4b$, $3b$, and $2b$, respectively, where b is a given bit rate. In this case, the router reserves 4b on interface C and $3b$ on interface D. Because of the wildcard-filter reservation, the two reservations from R2 and R3 are merged for interface D. The larger of the two reservations is used rather than the sum of reservations. The router then sends a reservation message upstream to interface A and another to interface B; each of these reservation message requests is $4b$, which is the larger of $3b$ and $4b$.

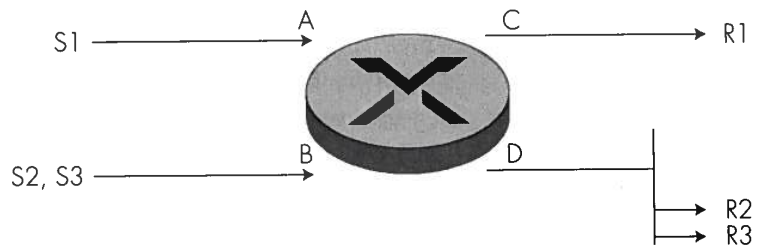


Figure 6.36 ✦ Sample scenario for RSVP reservation styles

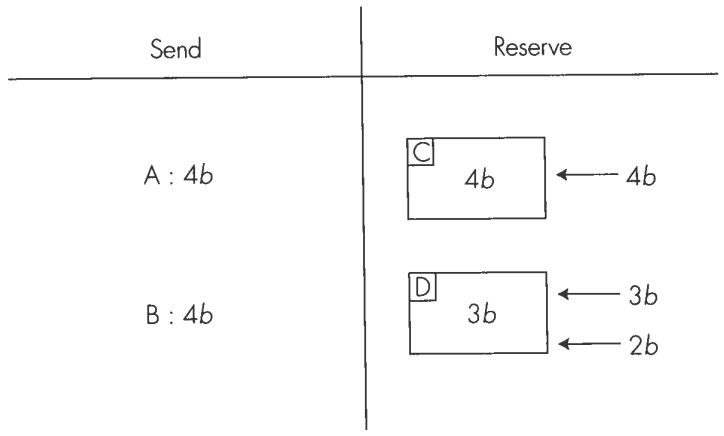


Figure 6.37 ✦ Wildcard filter reservations

Now suppose that all of the receivers use the fixed-filter reservation. As shown in Figure 6.38, receiver R1 wants to reserve 4b for source S1 and 5b for source S2; also shown in the figure are the reservation requests from R2 and R3. Because of the fixed-filter style, the router reserves two disjoint chunks of bandwidth on interface C: one chunk of 4*b* for S1 and another chunk of 5*b* for S2. Similarly, the router reserves two disjoint chunks of bandwidth on interface D: one chunk of 3*b* for S1 (the maximum of *b* and 3*b*) and one chunk of *b* for S3. On interface A, the router sends a message with a reservation for S1 of 4*b* (the maximum of 3*b* and 4*b*). On interface B, the router sends a message with a reservation of 5*b* for S2 and *b* for S3.

Finally, suppose that each of the receivers use the shared-explicit reservation. As shown in Figure 6.39, receiver R1 desires a pipe of 1*b*, which is to be shared between sources S1 and S2; receiver R2 desires a pipe of 3*b* to be shared between sources S1

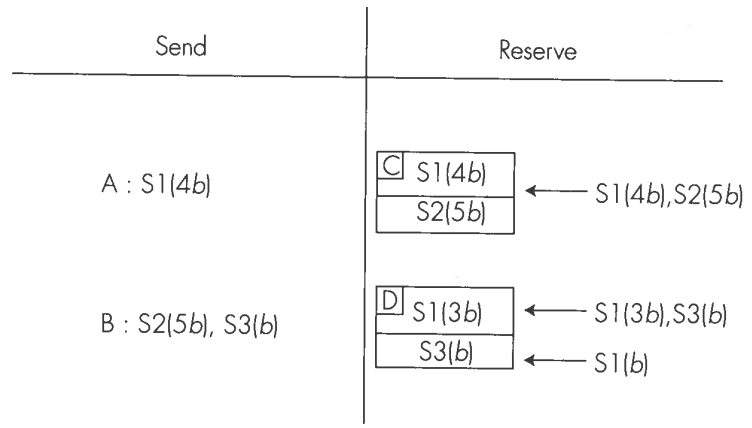


Figure 6.38 ✦ Fixed filter reservations

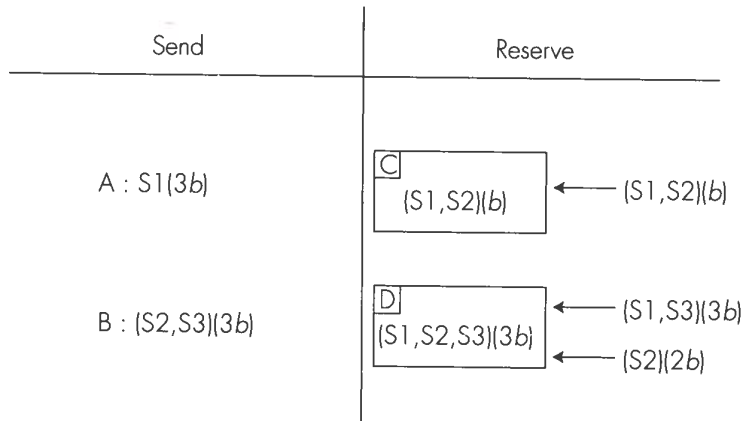


Figure 6.39 ✦ Shared-explicit reservations

and S3; and receiver R3 wants a pipe of 2*b* for source S2. Because of the shared-explicit style, the reservations from R2 and R3 are merged for interface D. Only one pipe is reserved on interface D, although it is reserved at the maximum of the reservation rates. RSVP will reserve on interface B a pipe of 3*b* to be shared by S2 and S3; note that 3*b* is the maximum of the downstream reservations for S2 and S3.

In each of the above examples the three receivers used the same reservation style. Because receivers make independent decisions, the receivers participating in a

session could use different styles. RSVP does not permit, however, reservations of different styles to be merged.

Principles in Practice

The Principle of Soft State
The reservations in the routers and hosts are maintained with soft states. By this it is meant that each reservation for bandwidth stored in a router has an associated timer. If a reservation's timer expires, then the reservation is removed. If a receiver desires to maintain a reservation, it must periodically refresh the reservation by sending reservation messages. This soft-state principle is also used by other protocols in computer networking. As we learned in Chapter 5, for the routing tables in transparent bridges, the entries are refreshed by data packets that arrive to the bridge; entries that are not refreshed are timed-out. On the other hand, if a protocol takes explicit actions to modify or release state, then the protocol makes use of hard state. Hard state is employed in virtual circuit networks (VC), in which explicit actions must be taken to adjust VC tables in switching nodes to establish and tear down VCs. ✦

6.8.5 Transport of Reservation Messages

RSVP messages are sent hop-by-hop directly over IP. Thus the RSVP message is placed in the information field of the IP datagram; the protocol number in the IP datagram is set to 46. Because IP is unreliable, RSVP messages can be lost. If an RSVP path or reservation message is lost, a replacement refresh message should arrive soon.

An RSVP reservation message that originates in a host will have the host's IP address in the source address field of the encapsulating IP datagram. It will have the IP address of the first router along the reserve path in the multicast tree in the destination address in the encapsulating IP datagram. When the IP datagram arrives at the first router, the router strips off the IP fields and passes the reservation message to the router's RSVP module. The RSVP module examines the message's multicast address (that is, session identifier) and style type, examines its current state, and then acts appropriately; for example, the RSVP module may merge the reservation with a reservation originating from another interface and then send a new reservation message to the next router upstream in the multicast tree.

Insufficient Resource

Because a reservation request that fails an admission test may embody a number of requests merged together, a reservation error must be reported to all the concerned receivers. These reservation errors are reported within **ResvError messages.** The

receivers can then reduce the amount of resource that they request and try reserving again. The RSVP standard provides mechanisms to allow the backtracking of the reservations when insufficient resources are available; unfortunately, these mechanisms add significant complexity to the RSVP protocol. Furthermore, RSVP suffers from the so-called **killer-reservation problem,** whereby a receiver requests a large reservation over and over again, each time getting its reservation rejected due to lack of sufficient resources. Because this large reservation may have been merged with smaller reservations downstream, the large reservation may be excluding smaller reservations from being established. To solve this thorny problem, RSVP uses the ResvError messages to establish additional state in routers, called blockade state. Blockade state in a router modifies the merging procedure to omit the offending reservation from the merge, allowing a smaller request to be forwarded and established. The blockade state adds yet further complexity to the RSVP protocol and its implementation.

6.9 ✦ Differentiated Services

In the previous section we discussed how RSVP can be used to reserve *per-flow* resources at routers within the network. The ability to request and reserve per-flow resources, in turn, makes it possible for the Intserv framework to provide quality-of-service guarantees to individual flows. As work on Intserv and RSVP proceeded, however, researchers involved with these efforts (for example, [Zhang 1998]) have begun to uncover some of the difficulties associated with the Intserv model and per-flow reservation of resources:

- *Scalability.* Per-flow resource reservation using RSVP implies the need for a router to process resource reservations and to maintain per-flow state for *each* flow passing though the router. With recent measurements [Thompson 1997] suggesting that even for an OC-3 speed link, approximately 256,000 source-destination pairs might be seen in one minute in a backbone router, per-flow reservation processing represents a considerable overhead in large networks.
- *Flexible service models.* The Intserv framework provides for a small number of prespecified service classes. This particular set of service classes does not allow for more qualitative or relative definitions of service distinctions (for example, "Service class A will receive preferred treatment over service class B."). These more qualitative definitions might better fit our intuitive notion of service distinction (for example, first class versus coach class in air travel; "platinum" versus "gold" versus "standard" credit cards).

These considerations have led to the recent so-called "diffserv" (Differentiated Services) activity [Diffserv 1999] within the Internet Engineering Task Force. The