CHAPTER 24

INTERNET APPLICATIONS—MULTIMEDIA

Prior to the recent explosion of sophisticated research, scientists believed that birds required no special awareness or intelligence to perform their migrations and their navigational and homing feats. Accumulated research shows that in addition to performing the difficult tasks of correcting for displacement (by storms, winds, mountains, and other hindrances), birds integrate an astonishing variety of celestial, atmospheric, and geological information to travel between their winter and summer homes. In brief, avian navigation is characterized by the ability to gather a variety of informational cues and to interpret and coordinate them so as to move closer toward a goal.

—*The Human Nature of Birds*, Theodore Barber

KEY TOPICS

- The Session Initiation Protocol (SIP) is an application-level control protocol for setting up, modifying, and terminating real-time sessions between participants over an IP data network.
- SIP uses the Session Description Protocol (SDP) to describe the media content to be used during a session.
- The Real-Time Transport Protocol (RTP) is a transport-level alternative to TCP or UDP for supporting real-time traffic.

With the increasing availability of broadband access to the Internet has come an increased interest in Web-based and Internet-based multimedia applications. The term *multimedia* refers to the use of multiple forms of information, including text, still images, audio, and video. The reader may find it useful to review Section 2.6 before proceeding.

An in-depth discussion of multimedia applications is well beyond the scope of this book. In this chapter, we focus on a few key topics. First, we look at audio and video compression, which is quite common in multimedia applications. Then we examine some of the key characteristics of real-time traffic. Next we look at SIP and its use to support voice over IP. Finally, we examine the real-time transport protocol.

24.1 AUDIO AND VIDEO COMPRESSION

In Chapter 3, we looked at some of the fundamental characteristics of both audio and video transmission. Then Chapter 5 introduced techniques such as pulse code modulation (PCM) for digitizing audio and video data for digital transmission. For

multimedia applications, it is important to make the most efficient use of transmission capacity as possible. Accordingly, much attention has been paid to the development of compression algorithms for both audio and video transmissions. This section provides an overview.

The techniques discussed in this section were standardized by the Moving Picture Experts Group (MPEG). MPEG, under the auspices of the International Organization for Standardization (ISO), has developed standards for video and associated audio in digital form, where the digital form may be stored on a variety of devices, such as CD-ROM, tapes, and writable optical disks, and transmitted on communications channels such as ISDN and LANs. The MPEG effort covers not only video compression, but also audio compression and associated system issues and formats. The premise of the MPEG effort is that a video signal can be compressed to a bit rate of about 1.5 Mbps with acceptable quality and that corresponding efficiencies are achievable for audio transmission.

Before proceeding, we introduce two terms. Data compression falls into two broad categories: lossless and lossy. With **lossless compression**, no information is lost and the decompressed data are identical to the original uncompressed data. The efficiency of lossless compression is limited to the entropy, or redundancy, of the data source. In other words, compression is limited to the process of eliminating some or all of the redundancy in a bit stream, so that no information is lost. With **lossy compression**, the decompressed data may be an acceptable approximation (according to some fidelity criterion) to the original uncompressed data. For example, for image or video compression, the criterion may be that the decompressed image is indistinguishable from the original to the human eye. In what follows, we will see that lossy compression is used for both audio and video. However, in the case of audio, the fidelity of the output is so high that, for all practical purposes, the compression is lossless.

Audio Compression

The first step in the development of an audio compression algorithm is to digitize the audio signal, using a technique such as PCM. It is important to note that PCM or a similar technique does in fact provide a measure of compression. Recall from Chapter 5 that the sampling theorem states that if a signal $f(t)$ is sampled at regular intervals of time and at a rate higher than twice the highest signal frequency, then the samples contain all the information of the original signal. The function $f(t)$ may be reconstructed from these samples by the use of a lowpass filter. For this technique to reproduce the original signal, the samples must have analog values that have infinite precision; this is known as pulse amplitude modulation (PAM). To create a digital signal, each sample is quantized to a value that can be represented by a fixed number of bits, producing pulse code modulation (PCM). A PCM-encoded signal produces only an approximation of the original signal. If unlimited fidelity were required, then an unlimited number of bits would be needed for each sample. The fact that a fixed, finite number of bits is used per sample results, in effect, in compression.

Taking a simple-minded approach, further compression could be achieved by reducing the frequency of sampling or reducing the number of bits per sample. However, there is another approach that can produce significant compression and

at the same time retain a fidelity that is equivalent to lossless compression. Such an approach is taken in the MPEG standard.

The MPEG standard for audio compression is quite complex and beyond our scope. In fact, the standard provides for three layers of compression. Layer 3, popularly known as MP3, provides a compression ratio of 10 : 1. In what follows, we look at the basic approach used in all of the MPEG audio compression algorithms.

Effective audio compression takes into account the physiology of human hearing. The compression algorithm exploits a phenomenon known as simultaneous auditory masking. This masking is an effect produced by the way the human nervous system perceives sound. In essence, if two signals are sufficiently near to one another and one tone is stronger, the weaker signal is simply not perceived at all; the human hearing apparatus masks it. Thus, what the percipient hears is exactly the same whether the weaker tone is there or not.

Figure 24.1a shows, in general terms, how masking is used to encode audio signals. The input is partitioned into time frames ranging in duration from 2 to 50 ms. A time-frequency analysis module decomposes each frame. At its simplest, this module determines the amplitude in each of a sequence of narrow frequency subbands; more complex analysis algorithms are typical. In any case, the output of this module is a set of parameters that define the acoustic signal in that particular time frame and that can be quantized. In parallel, a psychoacoustic module analyzes the time frame for masking effects and other properties that can be exploited to achieve compression. Based on this analysis, a bit allocation module decides how to apportion the total number of code bits available for the quantization of the subband signals. The resulting quantized signal is then fed into a lossless coding module that eliminates any redundancies in the digital signal to achieve maximum compression.

Figure 24.1b shows the inverse operation performed at a destination system to reproduce the original audio signal. The unpacking module recovers the quantized signal by inverting the lossless compression. The resulting signal is then processed to produce the audio output.

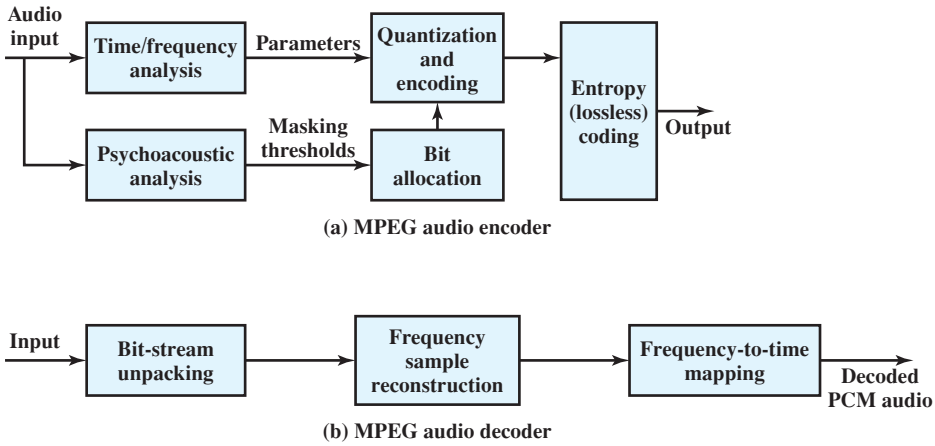
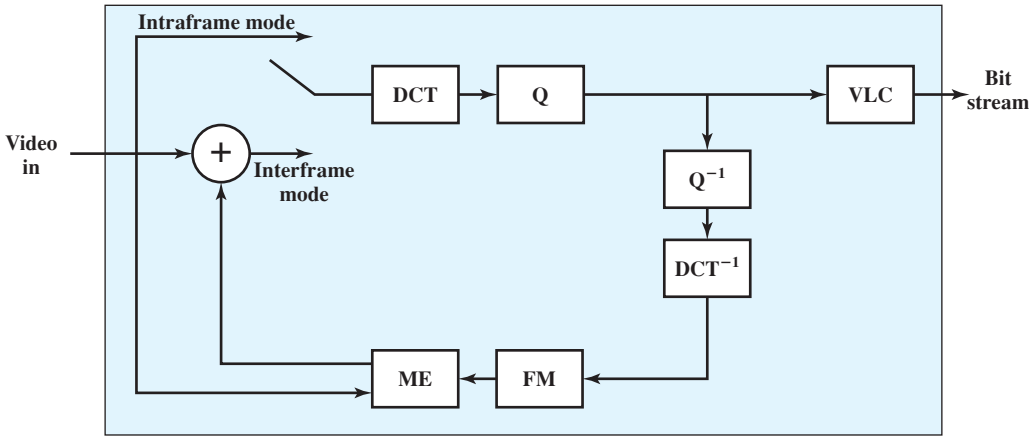


Figure 24.1 MPEG Audio Compression and Decompression



DCT: Discrete cosine transform
Q: quantizer
VLC: variable-length coder
FM: frame memory
ME: motion estimator

Figure 24.2 MPEG Block Diagram

Video Compression

A moving picture is simply a succession of still pictures. Accordingly, one can achieve a degree of compression by independently compressing each still picture in the sequence that makes up the moving picture. But much more can be done. Even in a moving picture with a lot of action, the differences between adjacent still pictures are generally small compared to the amount of information in a single still picture. This suggests that an encoding of the differences between adjacent still pictures is a fruitful approach to compression; this is a tool used in MPEG.

Overview of the Video Compression Algorithm Figure 24.2 illustrates the MPEG video compression scheme. The input to the MPEG compression module is a sequence of video frames. Each frame is processed separately, being treated as a single still image. While operating on a single frame, the MPEG coder is in intraframe mode. In this mode, the algorithm performs the following steps:

1. **Preliminary scaling and color conversion.** Each frame is converted into a standardized representation known as Source Input Format (SIF), and color information is translated into a scheme known as YUV.
2. **Color subsampling.** Brightness is the dominant component of color seen by the human eye, while hue is less important. Accordingly, the algorithm is able to reduce the hue information by about 75% with little effect on subjective fidelity.
3. **Discrete cosine transformation (DCT).** This process maps each 8×8 block of points (pixels) into a set of numbers similar to a Fourier transform of the block. In essence, the DCT provides a frequency domain representation of the image.

This transformation does not result in any compression but provides suitable input for later stages.

4. **Quantization.** The DCT values are quantized into a finite number of possible values (similar to pulse-code modulation quantization). The more quantization levels that are used, the greater the picture fidelity, but the less the amount of compression.
5. **Run-length encoding.** The quantized DCT values are represented using a run-length encoding technique.
6. **Huffman coding.** The data stream from the preceding step is compressed using Huffman coding, a lossless compression technique that assigns the most common bit sequences from the preceding step to symbols that are as short as possible.

Although significant compression can be achieved by simply processing a video signal as a sequence of still images, as just described, this approach fails to exploit the considerable redundancy present in all video sequences. Typically, many of the pixels will change very little or not at all from one frame to the next, or the change simply involves the movement of a pattern of pixels from one location on a frame to a nearby location on the next frame. The MPEG studies indicate an additional compression on the order of a factor of 3 [GALL91] by exploiting these redundancies in an interframe mode.

For interframe mode, similar blocks of pixels common to two or more successive frames are replaced by a pointer that references one of the blocks. The major complication has to do with the order of the frames. Sometimes it is convenient to refer to a block in a preceding frame. At other times it is convenient to refer to a block in a future frame. In this latter case, the encoder replaces the block with a pointer and also reverses the order of the frame. The decompression routine must put the frames back in proper order prior to display.

Figure 24.3 shows the interframe technique in a very general way. After the DCT and quantization phases in the processing of a frame, the frame goes through the reverse process (dequantization, inverse DCT) in order to recover a frame that is identical to that which will be recovered by the decompression algorithm. This frame is then stored and used in the interframe mode to compare to succeeding frames.

In designing the video compression algorithm, the MPEG study group identified a number of features that are important in order to meet the range of applications of MPEG. Two of these are relevant to our discussion:

- **Random access:** A compressed video bit stream should be accessible at any point in the sequence of video frames, and any frame should be decodable in a limited amount of time. This implies the existence of access frames, which are frames coded only in intraframe mode and that therefore can be decoded without reference to other frames.
- **Fast forward/reverse searches:** It should be possible to scan a compressed bit stream and, using the appropriate access frames, display selected frames to obtain a fast forward or fast reverse effect. This feature is essentially a more demanding form of the random access feature.

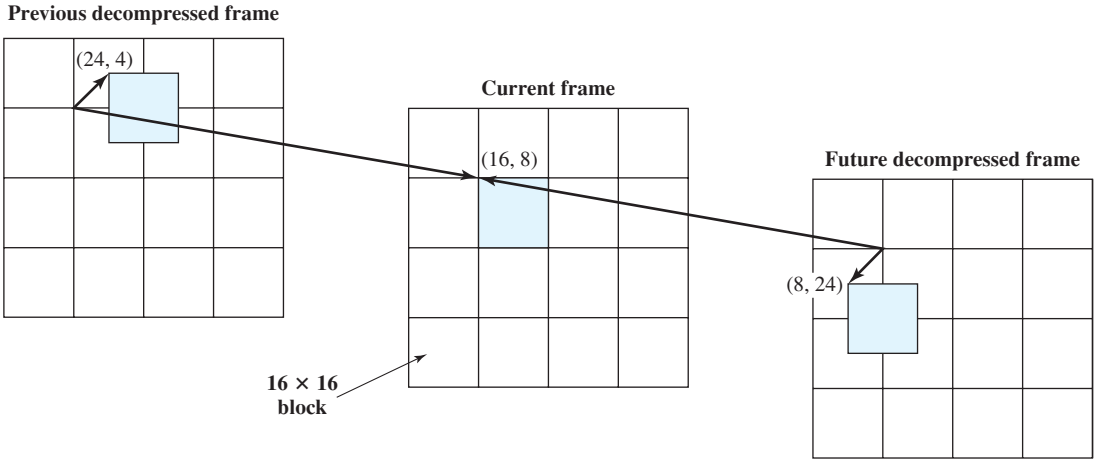


Figure 24.3 Block Motion Compensation

Motion Compensation The foundation of MPEG interframe compression is motion compensation. The idea behind motion compensation is that a portion of an image in one frame will be the same as or very similar to an equal-sized portion in a nearby frame. The MPEG scheme makes use of two forms of motion compensation: prediction and interpolation.

Prediction MPEG makes use of blocks of 16 × 16 pixels, called *macroblocks* (in contrast to the smaller 8 × 8 blocks used in the DCT coding), for purposes of motion compensation. A frame processed in prediction mode is divided into its macroblocks, each of which is encoded separately. The encoding is done with reference to an *anchor frame* that precedes the current frame.

Each macroblock in the current frame is to be represented by a pointer, called the motion vector, to that macroblock in the anchor frame that most closely matches this macroblock. The motion vector gives the displacement of the macroblock in the current frame with respect to its match in the anchor frame. This is shown in the left two-thirds of Figure 24.3. In this example, each video frame consist of 64 × 64 pixels grouped into 16 macroblocks. The shaded portion of the current frame is the macroblock whose upper left-hand pixel is in the *x-y* position (16, 8). The match for this block in the previous frame is in position (24, 4). The short arrowed line in the left-hand frame represents the motion vector, which in this case is (8, −4).

Key aspects of predictive coding:

1. The matching macroblock in the previous frame need not be on a 16-pixel boundary.
2. Matching is not done against a previous source video frame but rather against a video frame that has been through compression and decompression because the decompression module does not have access to the source video frames but only to decompressed versions of the original frames.

Having determined the matching block from the preceding frame, the MPEG algorithm records the motion vector and the prediction error, which is 16×16 matrix of differences between the current macroblock, in frame *c*, and the reference macroblock, in frame *r:*

$$E_c(x, y) = I_c(x, y) - I_r[(x, y) + M_{rc}]$$

where $E_c(x, y)$ is the prediction error, $I_i(x, y)$ is the value of the pixel located at position (x, y) in frame *i,* and M_{ij} is the motion vector for frame *j* relative to frame *i.*

Thus, the current frame is mapped into a matrix of prediction error values, one for each pixel position, and of motion vector values, one for each macroblock. The prediction error matrix will have many zero values. This matrix is encoded using the DCT-quantization technique and should yield a higher compression ratio than simply encoding the original pixel matrix.

The MPEG standard does not dictate how the matching process is to be done. Typically, the motion vector for a macroblock is obtained by minimizing a cost function that measures the difference between a macroblock and each predictor candidate. The calculation can be expressed as

$$\underset{m \in M}{\text{MIN}}\left[\sum_{(x,y) \in B_i} C[I_c(x, y) - I_r((x, y) + m)]\right]$$

where

B_i = a macroblock in the current frame I_c

m = the displacement vector with respect to the reference frame I_r

M = the search range in the reference frame

C = cost function

The value of m that minimizes the preceding expression is used as the motion vector M_{rc} for this block. The search range could encompass only small displacements or could range up to the entire frame size.

Interpolation Although prediction results in higher compression ratios than a simple frame-by-frame compression, more can be done. In particular, MPEG allows some video frames to be encoded using two reference frames, one in the past and one in the future. This approach, called *bidirectional interpolation*, results in higher compression ratios than prediction based on one reference frame.

To see why bidirectional interpolation can improve results, consider a scene that is moving with respect to the picture frame at a rate of one half pixel per frame. If we attempt to predict a macroblock in the current frame based on the immediately preceding frame, no exact matching block will be found. Similarly, no exact match to the macroblock will be found in the immediately following frame. However, an average of the best match from the preceding and following frames provides an exact prediction, so that the error matrix is all zeroes.

Figure 24.3 illustrates the technique used in bidirectional interpolation. The current frame, referred to as a B frame, is processed against two reference frames, one before and one after this frame in time. Each macroblock can be encoded using a block from the preceding frame (forward prediction), the following frame

Table 24.1 Prediction Modes for Macroblock in B Picture

Mode	Predictor
Forward predicted	$\hat{I}_1(z) = \hat{I}_0(z + M_{01})$
Backward predicted	$\hat{I}_1(z) = \hat{I}_2(z + M_{21})$
Average	$\hat{I}_1(z) = \frac{\hat{I}_0(z + M_{01}) + \hat{I}_2(z + M_{21})}{2}$

Note: z = the vector (x, y).

(backward prediction), or one block from each reference frame (averaging), whichever gives the minimum error matrix. Table 24.1 summarizes the calculations for each option, with frame 1 being the current frame, frame 0 the preceding reference frame, and frame 2 the following reference frame.

In the case of bidirectional interpolation, more information must be encoded. As with predicted frames, a matrix of differences is produced and then encoded using DCT. In addition, each macroblock is encoded with an indication of the prediction mode (forward, backward, average) and one or two motion vectors.

Frame Ordering Three types of frames are defined in MPEG:

- **Intraframe (I):** Encoded in JPEG style as an independent still image
- **Predicted (P):** Encoded with reference to the preceding anchor frame
- **Bidirectional interpolated (B):** Encoded with reference to the preceding and the following anchor frames

The relative frequency of these types of frames within a video stream is a configurable parameter and must satisfy several tradeoffs. First, there is the need to satisfy the requirements for random access and fast forward/reverse searches, described earlier. These requirements place a lower bound on the fraction of I frames in the encoded stream. Second, there is a tradeoff between computational complexity and the number of B frames: More B frames means more computation. Finally, B frames can only be processed with respect to I and P frames; that is, one B frame cannot serve as a reference frame for another B frame. Therefore, the higher the fraction of B frames, the greater the average distance between a B frame and its references and the less the correlation between the B frame and its reference frames.

The rules for encoding are as follows. Each I frame is encoded using intraframe coding only. Each P frame is encoded based on the most recent preceding I or P frame, whichever is closest. Each B frame is encoded with the closest preceding and following I or P frames.

Picture frames in MPEG are organized into groups. Each group consists of a single I frame followed by a number of P frames and B frames. Because a B frame cannot be decoded until both its preceding and following reference frames are decoded, the members of a group are reorganized so that each B frame follows both of its reference frames. Figure 24.4 provides an example. The first six frames form a group. Frame 4 is stored after frame 1, because it is used as the forward prediction

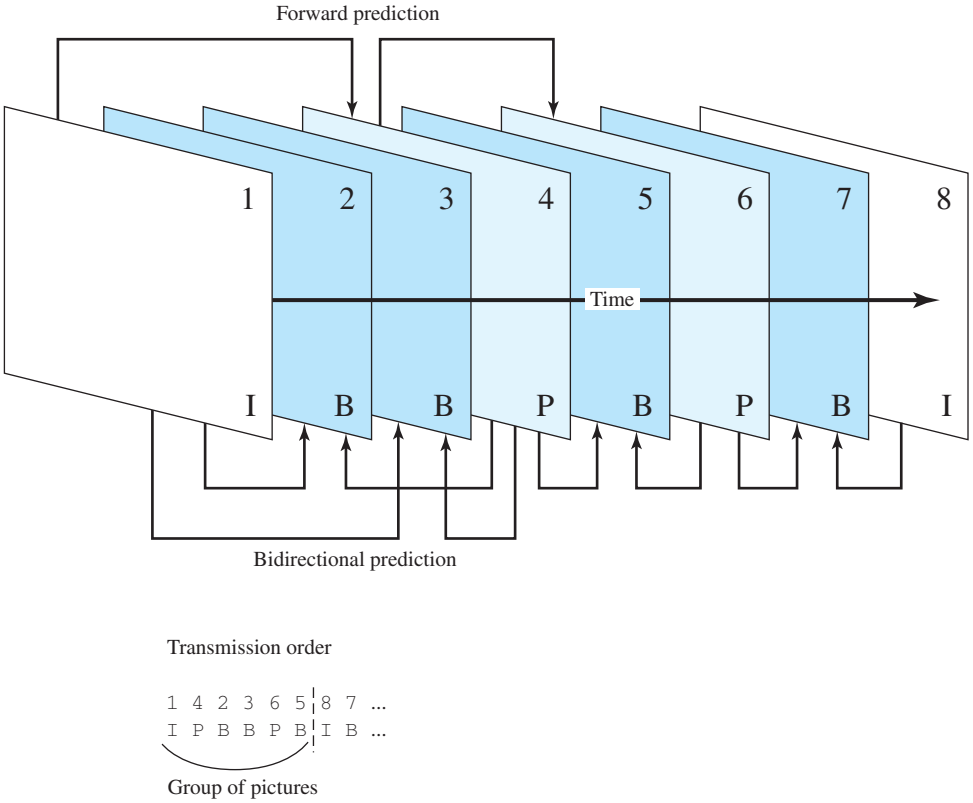


Figure 24.4 Example of Temporal Picture Structure

frame for B frames 2 and 3. Frames 5 and 6 are interchanged for the same reason. B frame 7 is recorded as part of the next group because it is encoded after I frame 8.

24.2 REAL-TIME TRAFFIC

The widespread deployment of high-speed LANs and WANs and the increase in the line capacity on the Internet and other internets has opened up the possibility of using IP-based networks for the transport of real-time traffic. However, it is important to recognize that the requirements of real-time traffic differ from those of high-speed but non-real-time traffic.

With traditional internet applications, such as file transfer, electronic mail, and client/server applications including the Web, the performance metrics of interest are generally throughput and delay. There is also a concern with reliability, and mechanisms are used to make sure that no data are lost, corrupted, or misordered during transit. By contrast, real-time applications are more concerned with timing issues. In most cases, there is a requirement that data be delivered at a constant rate equal to the sending rate. In other cases, a deadline is associated with each block of data, such that the data are not usable after the deadline has expired.

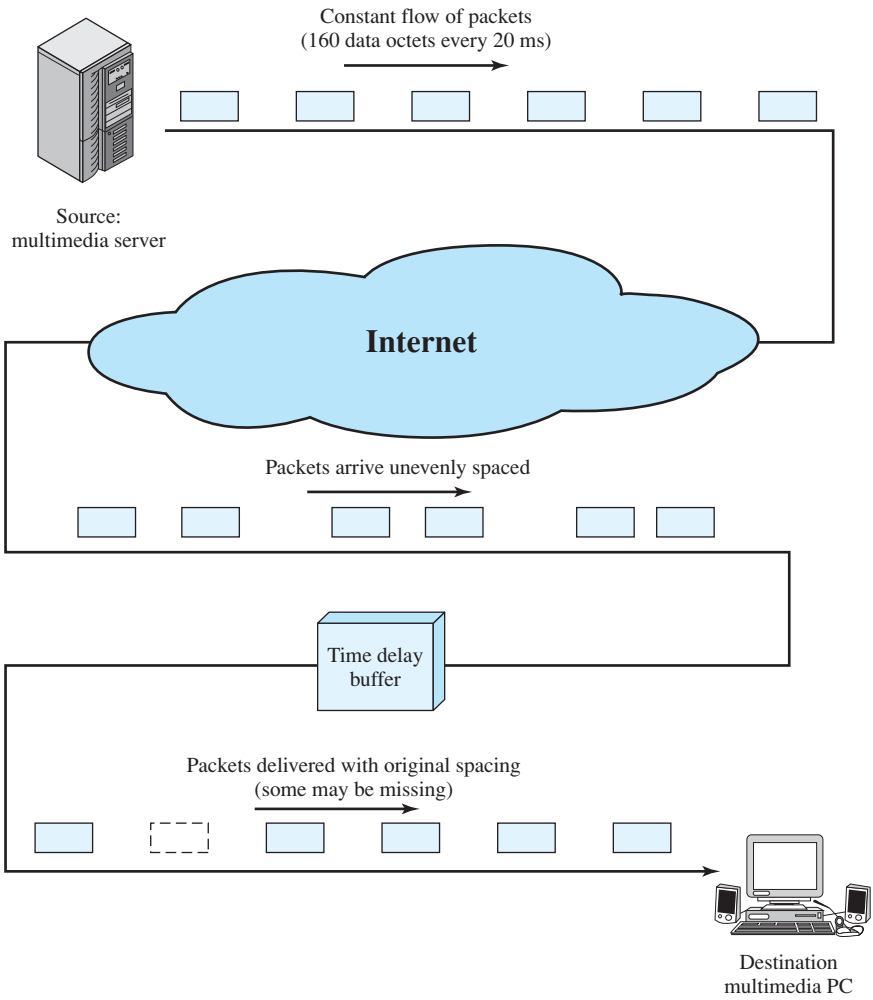
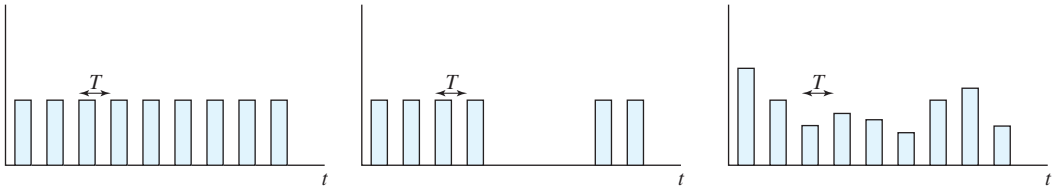


Figure 24.5 Real-Time Traffic

Real-Time Traffic Characteristics

Figure 24.5 illustrates a typical real-time environment. Here, a server is generating audio to be transmitted at 64 kbps. The digitized audio is transmitted in packets containing 160 octets of data, so that one packet is issued every 20 ms. These packets are passed through an internet and delivered to a multimedia PC, which plays the audio in real time as it arrives. However, because of the variable delay imposed by the Internet, the interarrival times between packets are not maintained at a fixed 20 ms at the destination. To compensate for this, the incoming packets are buffered, delayed slightly, and then released at a constant rate to the software that generates the audio.

The compensation provided by the delay buffer is limited. To understand this, we need to define the concept of *delay jitter*, which is the maximum variation in



(a) Continuous data source (b) Voice source with silent intervals (c) Compressed video source

Figure 24.6 Real-Time Packet Transmission (based on [ARAS94])

delay experienced by packets in a single session. For example, if the minimum end-to-end delay seen by any packet is 1 ms and the maximum is 6 ms, then the delay jitter is 5 ms. As long as the time delay buffer delays incoming packets by at least 5 ms, then the output of the buffer will include all incoming packets. However, if the buffer delayed packets only by 4 ms, then any incoming packets that had experienced a relative delay of more than 4 ms (an absolute delay of more than 5 ms) would have to be discarded so as not to be played back out of order.

The description of real-time traffic so far implies a series of equal-size packets generated at a constant rate. This is not always the profile of the traffic. Figure 24.6 illustrates some of the common possibilities:

- **Continuous data source:** Fixed-size packets are generated at fixed intervals. This characterizes applications that are constantly generating data, have few redundancies, and that are too important to compress in a lossy way. Examples are air traffic control radar and real-time simulations.
- **On/off source:** The source alternates between periods when fixed-size packets are generated at fixed intervals and periods of inactivity. A voice source, such as in telephony or audio conferencing, fits this profile.
- **Variable packet size:** The source generates variable-length packets at uniform intervals. An example is digitized video in which different frames may experience different compression ratios for the same output quality level.

Requirements for Real-Time Communication

[ARAS94] lists the following as desirable properties for real-time communication:

- Low jitter
- Low latency
- Ability to easily integrate non-real-time and real-time services
- Adaptable to dynamically changing network and traffic conditions
- Good performance for large networks and large numbers of connections
- Modest buffer requirements within the network
- High effective capacity utilization
- Low overhead in header bits per packet
- Low processing overhead per packet within the network and at the end system

These requirements are difficult to meet in a wide area IP-based network or internet. Neither TCP nor UDP by itself is appropriate. We will see that RTP provides a reasonable foundation for addressing these issues.

Hard versus Soft Real-Time Applications

A distinction needs to be made between hard and soft real-time communication applications. Soft real-time applications can tolerate the loss of some portion of the communicated data, while hard real-time applications have zero loss tolerance. In general, soft real-time applications impose fewer requirements on the network, and it is therefore permissible to focus on maximizing network utilization, even at the cost of some lost or misordered packets. In hard real-time applications, a deterministic upper bound on jitter and high reliability take precedence over network utilization considerations.

24.3 VOICE OVER IP AND MULTIMEDIA SUPPORT—SIP

The Session Initiation Protocol (SIP), defined in RFC 3261, is an application-level control protocol for setting up, modifying, and terminating real-time sessions between participants over an IP data network. The key driving force behind SIP is to enable Internet telephony, also referred to as voice over IP (VoIP). SIP can support any type of single media or multimedia session, including teleconferencing.

SIP supports five facets of establishing and terminating multimedia communications:

- **User location:** Users can move to other locations and access their telephony or other application features from remote locations.
- **User availability:** Determination of the willingness of the called party to engage in communications.
- **User capabilities:** Determination of the media and media parameters to be used.
- **Session setup:** Setup up point-to-point and multiparty calls, with agreed session parameters.
- **Session management:** Including transfer and termination of sessions, modifying session parameters, and invoking services.

SIP employs design elements developed for earlier protocols. SIP is based on an HTTP-like request/response transaction model. Each transaction consists of a client request that invokes a particular method, or function, on the server and at least one response. SIP uses most of the header fields, encoding rules, and status codes of HTTP. This provides a readable text-based format for displaying information. SIP also uses concepts similar to the recursive and iterative searches of DNS. SIP incorporates the use of a Session Description Protocol (SDP), which defines session content using a set of types similar to those used in MIME.

SIP Components and Protocols

An SIP network can be viewed of consisting of components defined on two dimensions: client/server and individual network elements. RFC 3261 defines **client** and **server** as follows:

- **Client:** A client is any network element that sends SIP requests and receives SIP responses. Clients may or may not interact directly with a human user. User agent clients and proxies are clients.
- **Server:** A server is a network element that receives requests in order to service them and sends back responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and registrars.

The individual elements of a standard SIP network are as follows:

- **User Agent:** Resides in every SIP end station. It acts in two roles:

  —User agent client (UAC): Issues SIP requests

  —User agent server (UAS): Receives SIP requests and generates a response that accepts, rejects, or redirects the request
- **Redirect Server:** Used during session initiation to determine the address of the called device. The redirect server returns this information to the calling device, directing the UAC to contact an alternate URI. This is analogous to iterative searches in DNS.
- **Proxy Server:** An intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity closer to the targeted user. Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it. This is analogous to recursive searches in DNS.
- **Registrar:** A server that accepts REGISTER requests and places the information it receives (the SIP address and associated IP address of the registering device) in those requests into the location service for the domain it handles.
- **Location Service:** A location service is used by a SIP redirect or proxy server to obtain information about a callee's possible location(s). For this purpose, the location service maintains a database of SIP-address/IP-address mappings.

The various servers are defined in RFC 3261 as logical devices. They may be implemented as separate servers configured on the Internet or they may be combined into a single application that resides in a physical server.

Figure 24.7 shows how some of the SIP components relate to one another and the protocols that are employed. A user agent acting as a client (in this case UAC alice) uses SIP to set up a session with a user agent that will act as a server (in this case UAS bob). The session initiation dialogue uses SIP and involves one or more proxy servers to forward requests and responses between the two user agents. The user agents also make use of the Session Description Protocol (SDP), which is used to describe the media session.

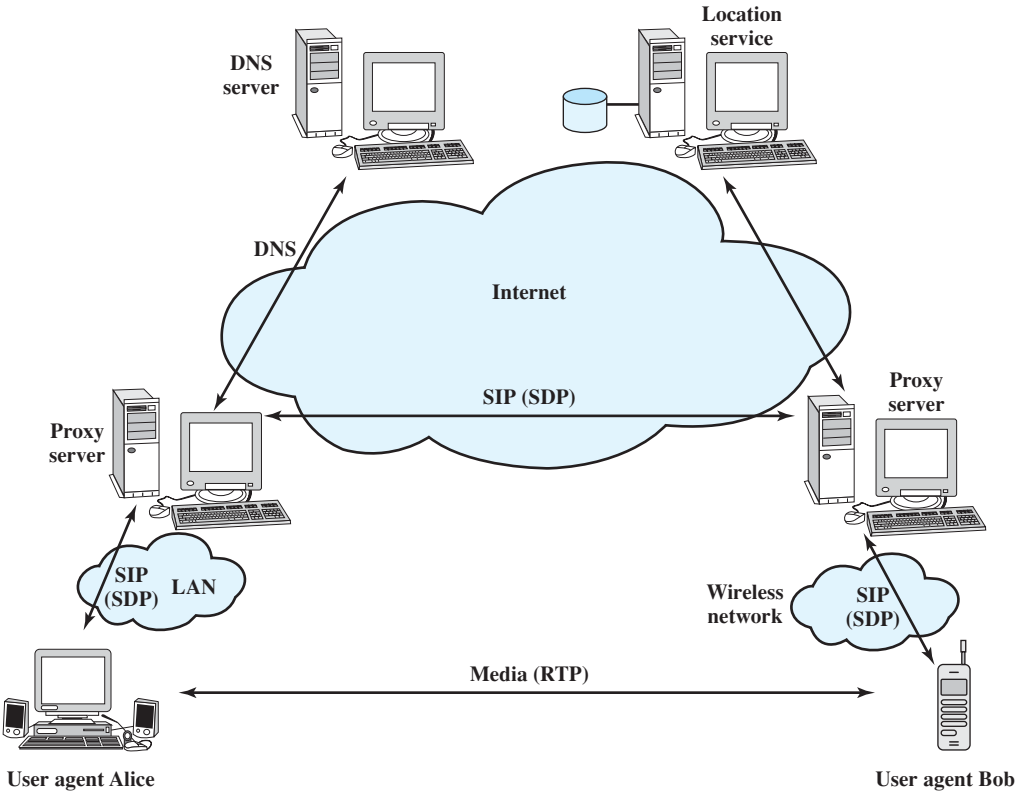


Figure 24.7 SIP Components and Protocols

The proxy servers may also act as redirect servers as needed. If redirection is done, a proxy server will need to consult the location service database, which may be collocated with a proxy server or not. The communication between the proxy server and the location service is beyond the scope of the SIP standard. DNS is also an important part of SIP operation. Typically, a UAC will make a request using the domain name of the UAS, rather than an IP address. A proxy server will need to consult a DNS server to find a proxy server for the target domain.

SIP typically runs on top of UDP for performance reasons, and provides its own reliability mechanisms, but may also use TCP. If a secure, encrypted transport mechanism is desired, SIP messages may alternatively be carried over the Transport Layer Security (TLS) protocol, described in Chapter 21.

Associated with SIP is the Session Description Protocol (SDP), defined in RFC 2327. SIP is used to invite one or more participants to a session, while the SDP-encoded body of the SIP message contains information about what media encodings (e.g., voice, video) the parties can and will use. Once this information is exchanged and acknowledged, all participants are aware of the participants' IP addresses, available transmission capacity, and media type. Then data transmission begins, using an appropriate transport protocol. Typically, the Real-Time Transport

Protocol (RTP), described subsequently, is used. Throughout the session, participants can make changes to session parameters, such as new media types or new parties to the session, using SIP messages.

SIP Uniform Resource Identifier

A resource within a SIP network is identified by a Uniform Resource Identifier (URI). Examples of communications resources include the following:

- A user of an online service
- An appearance on a multiline phone
- A mailbox on a messaging system
- A telephone number at a gateway service
- A group (such as "sales" or "helpdesk") in an organization

SIP URIs have a format based on email address formats, namely user@domain. There are two common schemes. An ordinary SIP URI is of the form

sip:bob@biloxi.com

The URI may also include a password, port number, and related parameters. If secure transmission is required, "sip:" is replaced by "sips:". In the latter case, SIP messages are transported over TLS.

Examples of Operation

The SIP specification is quite complex; the main document, RFC 3261, is 269 pages long. To give some feel for its operation, we present a few examples.

Figure 24.8 shows an unsuccessful attempt by user Alice to establish a session with user Bob, whose URI is bob@biloxi.com.[1] Alice's UAC is configured to communicate with a proxy server (the outbound server) in its domain and begins by sending an INVITE message to the proxy server that indicates its desire to invite Bob's UAS into a session (1); the server acknowledges the request (2). Although Bob's UAS is identified by its URI, the outbound proxy server needs to take into account the possibility that Bob is not currently available or that Bob has moved. Accordingly, the outbound proxy server should forward the INVITE request to the proxy server that is responsible for the domain biloxi.com. The outbound proxy thus consults a local DNS server to obtain the IP address of the biloxi.com proxy server (3), by asking for the SRV resource record (Table 23.2) that contains information on the proxy server for **biloxi.com**.

The DNS server responds (4) with the IP address of the biloxi.com proxy server (the inbound server). Alice's proxy server can now forward the INVITE message to the inbound proxy server (5), which acknowledges the message (6). The inbound proxy server now consults a location server to determine the location of Bob (7), and the location server responds that Bob is not signed in, and therefore not available for SIP messages (8). This information is communicated back to the outbound proxy server (9, 10) and then to Alice (11, 12).

[1]Figures 24.8 through 24.11 are adapted from ones developed by Professor H. Charles Baker of Southern Methodist University.

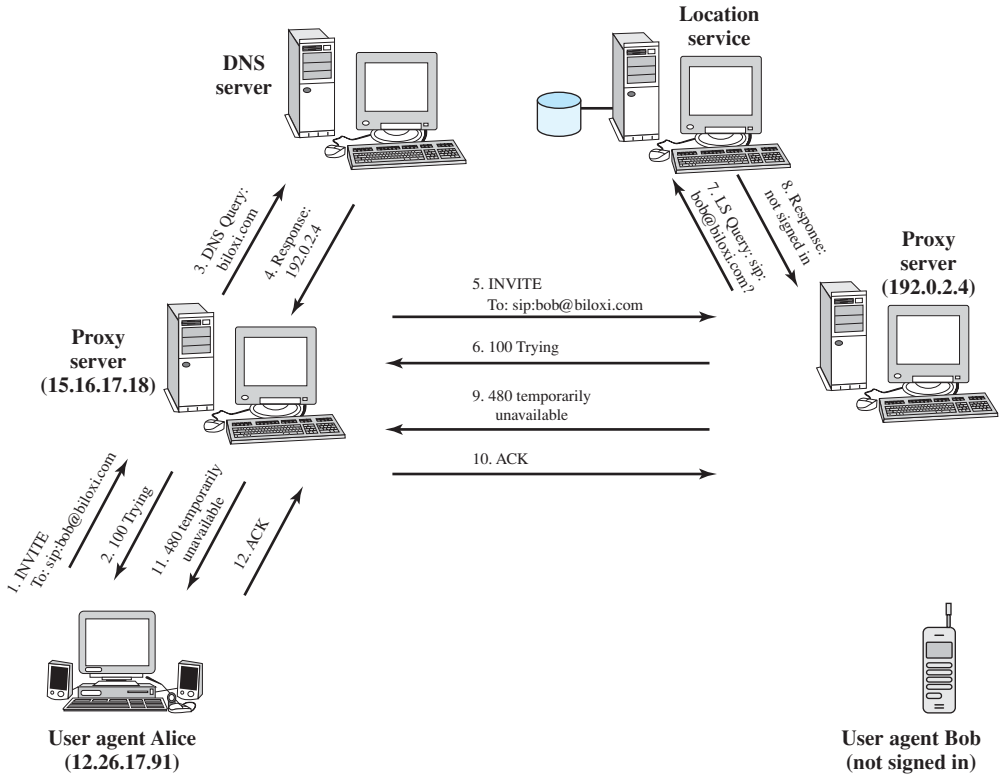


Figure 24.8 SIP Call Setup Attempt Scenario

The next example (Figure 24.9) makes use of two message types that are not yet part of the SIP standard but that are documented in RFC 2848 and are likely to be incorporated in a later revision of SIP. These message types support telephony applications. At the end of the preceding example, Alice was informed that Bob was not available. Alice's UAC then issues a SUBSCRIBE message (1), indicating that it wants to be informed when Bob is available. This request is forwarded through the two proxies in our example to a PINT (PSTN-Internet Networking)[2] server (2, 3). A PINT server acts as a gateway between an IP network from which comes a request to place a telephone call and a telephone network that executes the call by connecting to the destination telephone. In this example, we assume that the PINT server logic is collocated with the location service. It could also be the case that Bob is attached to the Internet rather than a PSTN, in which case the equivalent of PINT logic is needed to handle SUBSCRIBE requests. In this example, we assume that latter and assume that the PINT functionality is implemented in the location service. In any case, the location service authorizes subscription by returning an OK message (4), which is passed back to Alice (5, 6). The location service then immediately sends a NOTIFY message with Bob's current status of not signed in (7, 8, 9), which Alice's UAC acknowledges (10, 11, 12).

[2]PSTN is the public switched telephone network.

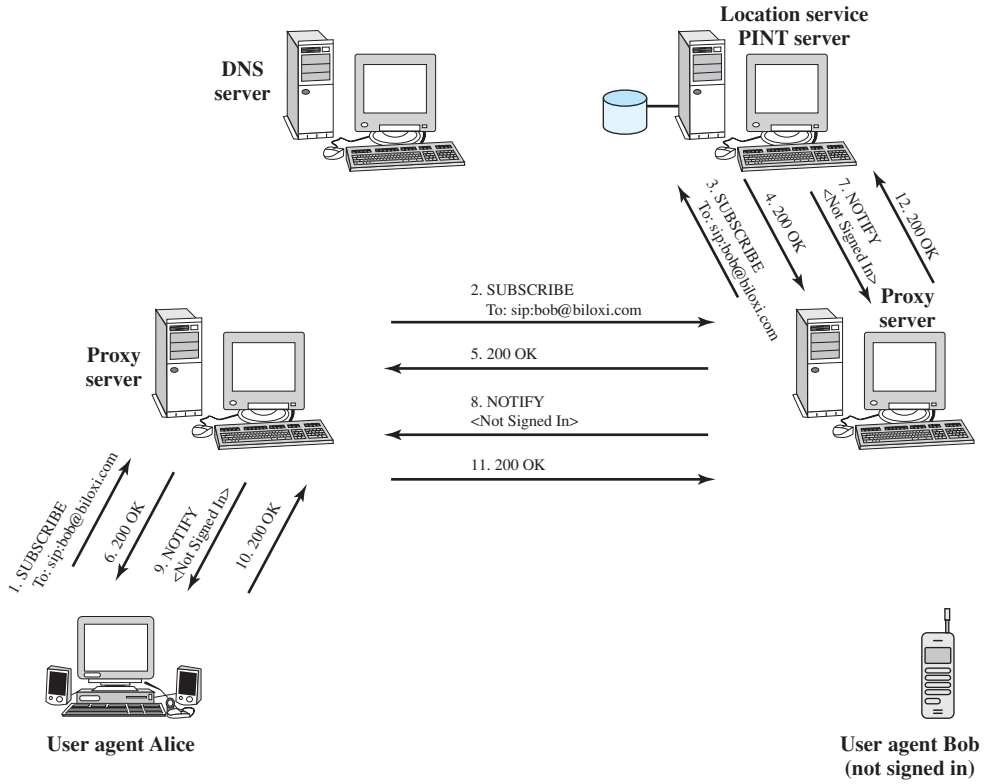


Figure 24.9 SIP Presence Example

Figure 24.10 continues the example of Figure 24.9. Bob signs on by sending a REGISTER message to the proxy in its domain (1). The proxy updates the database at the location service to reflect registration (2). The update is confirmed to the proxy (3), which confirms the registration to Bob (4). The PINT functionality learns of Bob's new status from the location server (here we assume that they are collocated) and sends a NOTIFY message containing the new status of Bob (5), which is forwarded to Alice (6, 7). Alice's UAC acknowledges receipt of the notification (8, 9, 10).

Now that Bob is registered, Alice can try again to establish a session, as shown in Figure 24.11. This figure shows the same flow as Figure 24.8, with a few differences. We assume that Alice's proxy server has cached the IP address of the proxy server for domain biloxi.com, and therefore need not consult the DNS server. A ringing response is sent from Bob back to Alice (8, 9, 10) while the UAS at Bob is alerting the local media application (e.g., telephony). When the media application accepts the call, Bob's UAS sends back an OK response to Alice (11, 12, 13).

Finally, Alice's UAC sends an acknowledgement message to Bob's UAS to confirm the reception of the final response (14). In this example, the ACK is sent directly from Alice to Bob, bypassing the two proxies. This occurs because the endpoints have learned each other's address from the INVITE/200 (OK) exchange, which was not known when the initial INVITE was sent. The media session has now begun, and Alice and Bob can exchange data over an RTP connection.

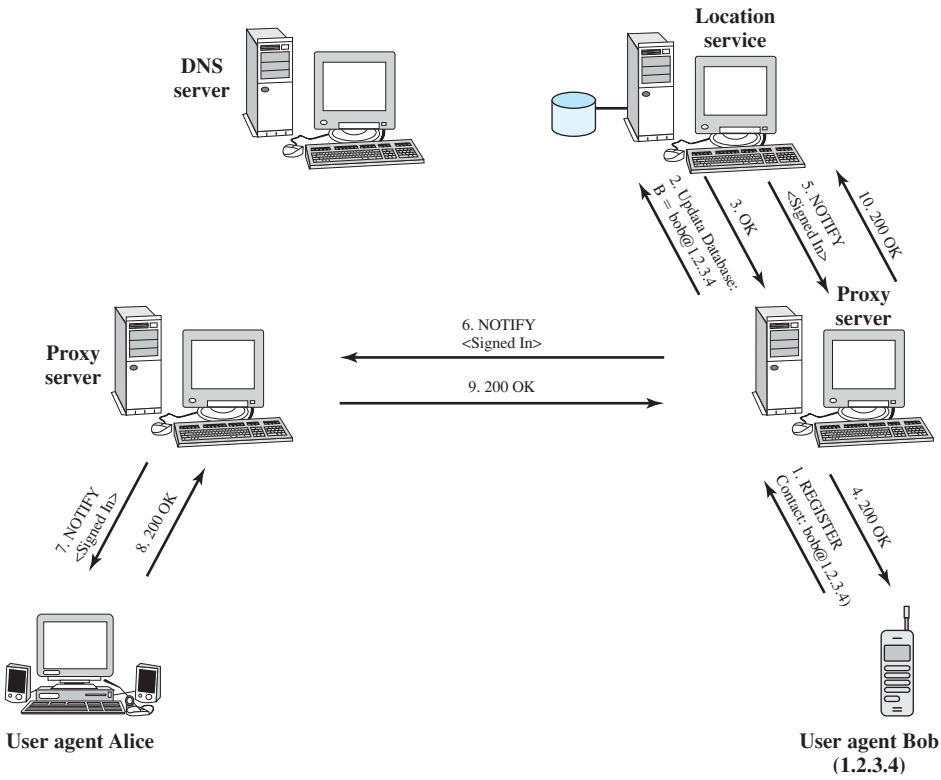


Figure 24.10 SIP Registration and Notification Example

SIP Messages

As was mentioned, SIP is a text-based protocol with a syntax similar to that of HTTP. There are two different types of SIP messages, requests and responses. The format difference between the two types of messages is seen in the first line. The first line of a request has a **method**, defining the nature of the request and a Request-URI, indicating where the request should be sent. The first line of a response has a **response code**. All messages include a header, consisting of a number of lines, each line beginning with a header label. A message can also contain a body, such as an SDP media description.

SIP Requests RFC 3261 defines the following methods:

- **REGISTER:** Used by a user agent to notify a SIP network of its current IP address and the URLs for which it would like to receive calls
- **INVITE:** Used to establish a media session between user agents
- **ACK:** Confirms reliable message exchanges
- **CANCEL:** Terminates a pending request, but does not undo a completed call
- **BYE:** Terminates a session between two users in a conference

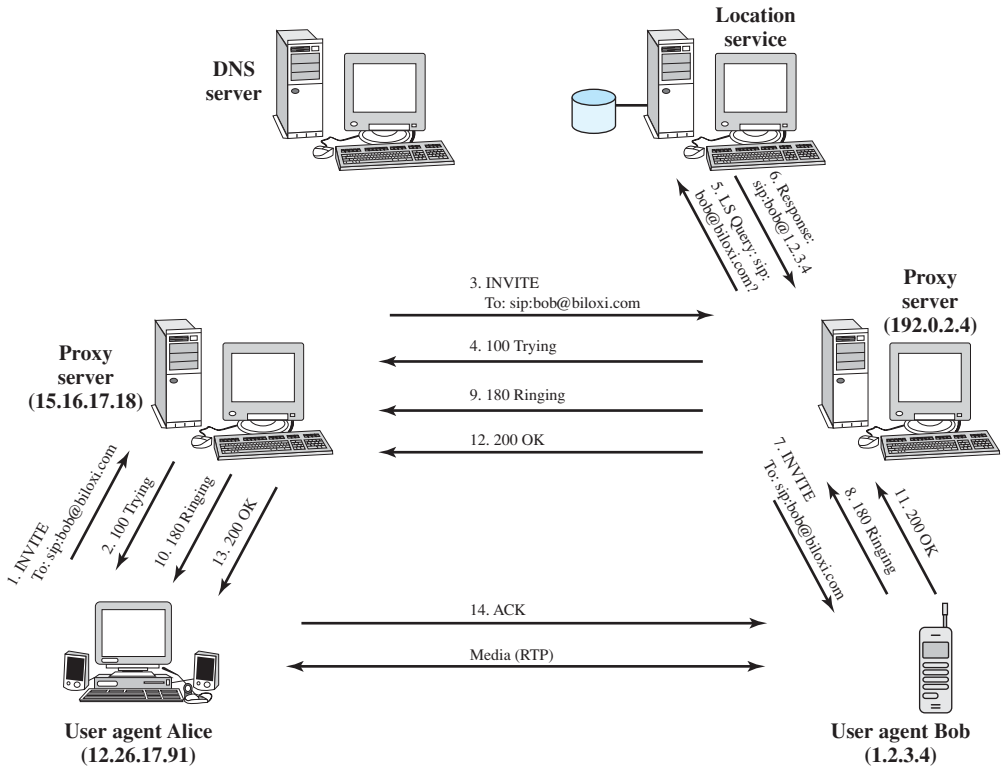


Figure 24.11 SIP Successful Call Setup

- **OPTIONS:** Solicits information about the capabilities of the callee, but does not set up a call

For example, the header of message (1) in Figure 24.11 might look like this:

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP 12.26.17.91:5060
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@12.26.17.91
CSeq: 314159 INVITE
Contact: <sip:alice@atlanta.com>
Content-Type: application/sdp
Content-Length: 142

The boldface type used for header labels is not typical but is used here for clarity. The first line contains the method name (**INVITE**), a SIP URI, and the version

number of SIP that is used. The lines that follow are a list of header fields. This example contains the minimum required set.

The **Via** headers show the path the request has taken in the SIP network (source and intervening proxies) and are used to route responses back along the same path. In message (1), there is only one Via header, inserted by Alice. The Via line contains the IP address (12.26.17.91), port number (5060), and transport protocol (UDP) that Alice wants Bob to use in his response. Subsequent proxies add additional Via headers.

Max-Forwards serves to limit the number of hops a request can make on the way to its destination. It consists of an integer that is decremented by one by each proxy that forwards the request. If the Max-Forwards value reaches 0 before the request reaches its destination, it will be rejected with a 483 (Too Many Hops) error response.

To contains a display name (Bob) and a SIP or SIPS URI (sip:bob@biloxi.com) toward which the request was originally directed. **From** also contains a display name (Alice) and a SIP or SIPS URI (sip:alice@atlanta.com) that indicate the originator of the request. This header field also has a tag parameter containing a random string (1928301774) that was added to the URI by the UAC. It is used to identify the session.

Call-ID contains a globally unique identifier for this call, generated by the combination of a random string and the host name or IP address. The combination of the To tag, From tag, and Call-ID completely defines a peer-to-peer SIP relationship between Alice and Bob and is referred to as a dialog.

CSeq or Command Sequence contains an integer and a method name. The CSeq number is initialized at the start of a call (314159 in this example), incremented for each new request within a dialog, and is a traditional sequence number. The CSeq is used to distinguish a retransmission from a new request.

The **Contact** header contains a SIP URI for direct communication between UAs. While the Via header field tells other elements where to send the response, the Contact header field tells other elements where to send future requests for this dialog.

The **Content-Type** indicates the type of the message body. **Content-Length** gives the length in octets of the message body.

SIP Responses The response types defined in RFC 3261 are in the following categories:

- **Provisional (1xx):** Request received and being processed.
- **Success (2xx):** The action was successfully received, understood, and accepted.
- **Redirection (3xx):** Further action needs to be taken in order to complete the request.
- **Client Error (4xx):** The request contains bad syntax or cannot be fulfilled at this server.
- **Server Error (5xx):** The server failed to fulfill an apparently valid request.
- **Global Failure (6xx):** The request cannot be fulfilled at any server.

For example, the header of message (11) in Figure 24.11 might look like this:

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com

Via: SIP/2.0/UDP 12.26.17.91:5060
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@12.26.17.91
CSeq: 314159 INVITE
Contact: <sip:bob@biloxi.com>
Content-Type: application/sdp
Content-Length: 131

The first line contains the version number of SIP that is used and the response code and name. The lines that follow are a list of header fields. The Via, To, From, Call-ID, and CSeq header fields are copied from the INVITE request. (There are three Via header field values—one added by Alice's SIP UAC, one added by the atlanta.com proxy, and one added by the biloxi.com proxy.) Bob's SIP phone has added a tag parameter to the To header field. This tag will be incorporated by both endpoints into the dialog and will be included in all future requests and responses in this call.

Session Description Protocol

The Session Description Protocol (SDP), defined in RFC 2327, describes the content of sessions, including telephony, Internet radio, and multimedia applications. SDP includes information about the following [SCHU99]:

- **Media streams:** A session can include multiple streams of differing content. SDP currently defines audio, video, data, control, and application as stream types, similar to the MIME types used for Internet mail (Table 22.3).
- **Addresses:** Indicates the destination addresses, which may be a multicast address, for a media stream.
- **Ports:** For each stream, the UDP port numbers for sending and receiving are specified.
- **Payload types:** For each media stream type in use (e.g., telephony), the payload type indicates the media formats that can be used during the session.
- **Start and stop times:** These apply to broadcast sessions, like a television or radio program. The start, stop, and repeat times of the session are indicated.
- **Originator:** For broadcast sessions, the originator is specified, with contact information. This may be useful if a receiver encounters technical difficulties.

24.4 REAL-TIME TRANSPORT PROTOCOL (RTP)

The most widely used transport-level protocol is TCP. Although TCP has proven its value in supporting a wide range of distributed applications, it is not suited for use with real-time distributed applications. By a real-time distributed application, we mean one in which a source is generating a stream of data at a constant rate, and one or more destinations must deliver that data to an application at the same constant rate. Examples of such applications include audio and video

conferencing, live video distribution (not for storage but for immediate play), shared workspaces, remote medical diagnosis, telephony, command and control systems, distributed interactive simulations, games, and real-time monitoring. A number of features of TCP disqualify it for use as the transport protocol for such applications:

1. TCP is a point-to-point protocol that sets up a connection between two endpoints. Therefore, it is not suitable for multicast distribution.
2. TCP includes mechanisms for retransmission of lost segments, which then arrive out of order. Such segments are not usable in most real-time applications.
3. TCP contains no convenient mechanism for associating timing information with segments, which is another real-time requirement.

The other widely used transport protocol, UDP, does not exhibit the first two characteristics listed but, like TCP, does not provide timing information. By itself, UDP does not provide any general-purpose tools useful for real-time applications.

Although each real-time application could include its own mechanisms for supporting real-time transport, there are a number of common features that warrant the definition of a common protocol. A protocol designed for this purpose is the Real-Time Transport Protocol (RTP), defined in RFC 1889. RTP is best suited to soft real-time communication. It lacks the necessary mechanisms to support hard real-time traffic.

This section provides an overview of RTP. We begin with a discussion of real-time transport requirements. Next, we examine the philosophical approach of RTP. The remainder of the section is devoted to the two protocols that make up RTP: the first is simply called RTP and is a data transfer protocol; the other is a control protocol known as RTCP (RTP Control Protocol).

RTP Protocol Architecture

In RTP, there is close coupling between the RTP functionality and the application-layer functionality. Indeed, RTP is best viewed as a framework that applications can use directly to implement a single protocol. Without the application-specific information, RTP is not a full protocol. On the other hand, RTP imposes a structure and defines common functions so that individual real-time applications are relieved of part of their burden.

RTP follows the principles of protocol architecture design outlined in a paper by Clark and Tennenhouse [CLAR90]. The two key concepts presented in that paper are application-level framing and integrated layer processing.

Application-Level Framing In a traditional transport protocol, such as TCP, the responsibility for recovering from lost portions of data is performed transparently at the transport layer. [CLAR90] lists two scenarios in which it might be more appropriate for recovery from lost data to be performed by the application:

1. The application, within limits, may accept less than perfect delivery and continue unchecked. This is the case for real-time audio and video. For such applications, it may be necessary to inform the source in more general terms about the quality of the delivery rather than to ask for retransmission. If too much data are

being lost, the source might perhaps move to a lower-quality transmission that places lower demands on the network, increasing the probability of delivery.

2. It may be preferable to have the application rather than the transport protocol provide data for retransmission. This is useful in the following contexts:
   (a) The sending application may recompute lost data values rather than storing them.
   (b) The sending application can provide revised values rather than simply retransmitting lost values, or send new data that "fix" the consequences of the original loss.

To enable the application to have control over the retransmission function, Clark and Tennenhouse propose that lower layers, such as presentation and transport, deal with data in units that the application specifies. The application should break the flow of data into application-level data units (ADUs), and the lower layers must preserve these ADU boundaries as they process the data. The application-level frame is the unit of error recovery. Thus, if a portion of an ADU is lost in transmission, the application will typically be unable to make use of the remaining portions. In such a case, the application layer will discard all arriving portions and arrange for retransmission of the entire ADU, if necessary.

Integrated Layer Processing In a typical layered protocol architecture, such as TCP/IP or OSI, each layer of the architecture contains a subset of the functions to be performed for communications, and each layer must logically be structured as a separate module in end systems. Thus, on transmission, a block of data flows down through and is sequentially processed by each layer of the architecture. This structure restricts the implementer from invoking certain functions in parallel or out of the layered order to achieve greater efficiency. Integrated layer processing, as proposed in [CLAR90], captures the idea that adjacent layers may be tightly coupled and that the implementer should be free to implement the functions in those layers in a tightly coupled manner.

The idea that a strict protocol layering may lead to inefficiencies has been propounded by a number of researchers. For example, [CROW92] examined the inefficiencies of running a remote procedure call (RPC) on top of TCP and suggested a tighter coupling of the two layers. The researchers argued that the integrated layer processing approach is preferable for efficient data transfer.

Figure 24.12 illustrates the manner in which RTP realizes the principle of integrated layer processing. RTP is designed to run on top of a connectionless transport protocol such as UDP. UDP provides the basic port addressing functionality of the transport layer. RTP contains further transport-level functions, such as sequencing. However, RTP by itself is not complete. It is completed by modifications and/or additions to the RTP headers to include application-layer functionality. The figure indicates that several different standards for encoding video data can be used in conjunction with RTP for video transmission.

RTP Data Transfer Protocol

We first look at the basic concepts of the RTP data transfer protocol and then examine the protocol header format. Throughout this section, the term *RTP* will refer to the RTP data transfer protocol.

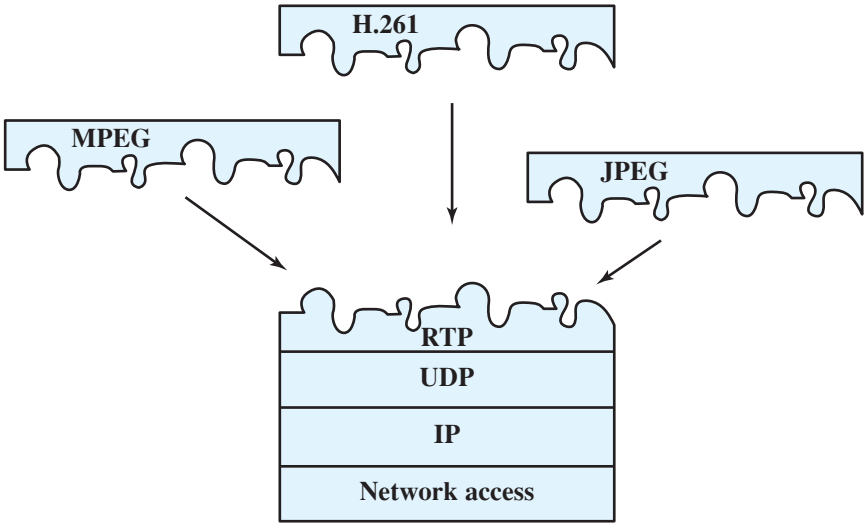


Figure 24.12 RTP Protocol Architecture [THOM96]

RTP Concepts RTP supports the transfer of real-time data among a number of participants in a session. A session is simply a logical association among two or more RTP entities that is maintained for the duration of the data transfer. A session is defined by

- **RTP port number:** The destination port address is used by all participants for RTP transfers. If UDP is the lower layer, this port number appears in the Destination Port field (see Figure 2.3) of the UDP header.
- **RTCP port number:** The destination port address is used by all participants for RTCP transfers.
- **Participant IP addresses:** This can either be a multicast IP address, so that the multicast group defines the participants, or a set of unicast IP addresses.

The process of setting up a session is beyond the scope of RTP and RTCP.

Although RTP can be used for unicast real-time transmission, its strength lies in its ability to support multicast transmission. For this purpose, each RTP data unit includes a source identifier that identifies which member of the group generated the data. It also includes a timestamp so that the proper timing can be re-created on the receiving end using a delay buffer. RTP also identifies the payload format of the data being transmitted.

RTP allows the use of two kinds of RTP relays: translators and mixers. First we need to define the concept of relay. A relay operating at a given protocol layer is an intermediate system that acts as both a destination and a source in a data transfer. For example, suppose that system A wishes to send data to system B but cannot do so directly. Possible reasons are that B may be behind a firewall or B may not be able to use the format transmitted by A. In such a case, A may be able to send the data to an intermediate relay R. R accepts the data unit, makes any necessary changes or performs any necessary processing, and then transmits the data to B.

A **mixer** is an RTP relay that receives streams of RTP packets from one or more sources, combines these streams, and forwards a new RTP packet stream to one or more destinations. The mixer may change the data format or simply perform the mixing function. Because the timing among the multiple inputs is not typically synchronized, the mixer provides the timing information in the combined packet stream and identifies itself as the source of synchronization.

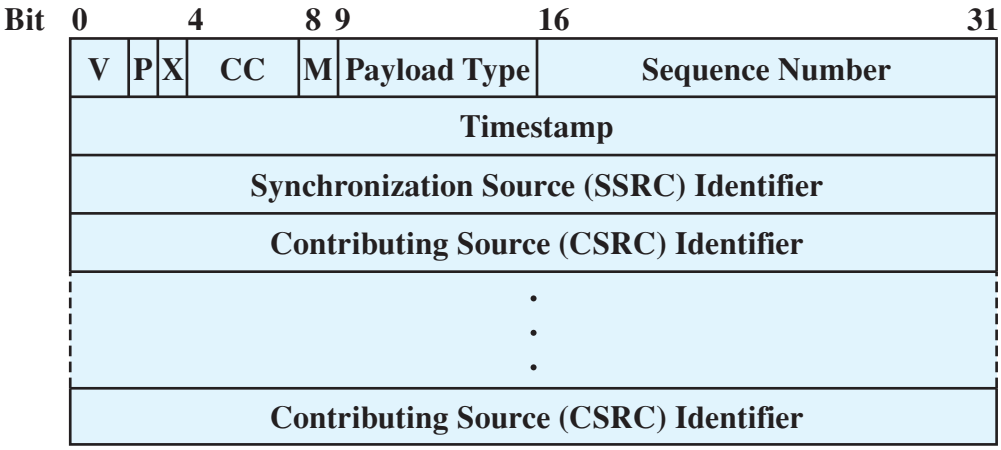
An example of the use of a mixer is to combine of a number of on/off sources such as audio. Suppose that a number of systems are members of an audio session and each generates its own RTP stream. Most of the time only one source is active, although occasionally more than one source will be "speaking" at the same time. A new system may wish to join the session, but its link to the network may not be of sufficient capacity to carry all of the RTP streams. Instead, a mixer could receive all of the RTP streams, combine them into a single stream, and retransmit that stream to the new session member. If more than one incoming stream is active at one time, the mixer would simply sum their PCM values. The RTP header generated by the mixer includes the identifier(s) of the source(s) that contributed to the data in each packet.

The **translator** is a simpler device that produces one or more outgoing RTP packets for each incoming RTP packet. The translator may change the format of the data in the packet or use a different lower-level protocol suite to transfer from one domain to another. Examples of translator use are as follows:

- A potential recipient may not be able to handle a high-speed video signal used by the other participants. The translator converts the video to a lower-quality format requiring a lower data rate.
- An application-level firewall may prevent the forwarding of RTP packets. Two translators are used, one on each side of the firewall, with the outside one tunneling all multicast packets received through a secure connection to the translator inside the firewall. The inside translator then sends out RTP packets to a multicast group protected by the firewall.
- A translator can replicate an incoming multicast RTP packet and send it to a number of unicast destinations.

RTP Fixed Header Each RTP packet includes a fixed header and may also include additional application-specific header fields. Figure 24.13 shows the fixed header. The first 12 octets (shaded portion) are always present and consist of the following fields:

- **Version (2 bits):** Current version is 2.
- **Padding (1 bit):** Indicates whether padding octets appear at the end of the payload. If so, the last octet of the payload contains a count of the number of padding octets. Padding is used if the application requires that the payload be an integer multiple of some length, such as 32 bits.
- **Extension (1 bit):** If set, the fixed header is followed by exactly one extension header, which is used for experimental extensions to RTP.
- **CSRC Count (4 bits):** The number of CSRC (contributing source) identifiers that follow the fixed header.



V = Version
P = Padding
X = Extension
CC = CSRC count
M = Marker

Figure 24.13 RTP Header

- **Marker (1 bit):** The interpretation of the marker bit depends on the payload type; it is typically used to indicate a boundary in the data stream. For video, it is set to mark the end of a frame. For audio, it is set to mark the beginning of a talk spurt.
- **Payload Type (7 bits):** Identifies the format of the RTP payload, which follows the RTP header.
- **Sequence Number (16 bits):** Each source starts with a random sequence number, which is incremented by one for each RTP data packet sent. This allows for loss detection and packet sequencing within a series of packets with the same timestamp. A number of consecutive packets may have the same timestamp if they are logically generated at the same time; an example is several packets belonging to the same video frame.
- **Timestamp (32 bits):** Corresponds to the generation instant of the first octet of data in the payload. The time units of this field depend on the payload type. The values must be generated from a local clock at the source.
- **Synchronization Source Identifier:** A randomly generated value that uniquely identifies the source within a session.

Following the fixed header, there may be one or more of the following field:

- **Contributing Source Identifier:** Identifies a contributing source for the payload. These identifiers are supplied by a mixer.

The Payload Type field identifies the media type of the payload and the format of the data, including the use of compression or encryption. In a steady state, a source should only use one payload type during a session but may change the payload type in response to changing conditions, as discovered by RTCP. Table 24.2 summarizes the payload types defined in RFC 1890.

Table 24.2 Payload Types for Standard Audio and Video Encodings (RFC 1890)

0	PCMU audio	16–23	unassigned audio
1	1016 audio	24	unassigned video
2	G721 audio	25	CelB video
3	GSM audio	26	JPEG video
4	unassigned audio	27	unassigned
5	DV14 audio (8 kHz)	28	nv video
6	DV14 audio (16 kHz)	29–30	unassigned video
7	LPC audio	31	H261 video
8	PCMA audio	32	MPV video
9	G722 audio	33	MP2T video
10	L16 audio (stereo)	34–71	unassigned
11	L16 audio (mono)	72–76	reserved
12–13	unassigned audio	77–95	unassigned
14	MPA audio	96–127	dynamic
15	G728 audio		

RTP Control Protocol (RTCP)

The RTP data transfer protocol is used only for the transmission of user data, typically in multicast fashion among all participants in a session. A separate control protocol (RTCP) also operates in a multicast fashion to provide feedback to RTP data sources as well as all session participants. RTCP uses the same underlying transport service as RTP (usually UDP) and a separate port number. Each participant periodically issues an RTCP packet to all other session members. RFC 1889 outlines four functions performed by RTCP:

- **Quality of service (QoS) and congestion control:** RTCP provides feedback on the quality of data distribution. Because RTCP packets are multicast, all session members can assess how well other members are performing and receiving. Sender reports enable receivers to estimate data rates and the quality of the transmission. Receiver reports indicate any problems encountered by receivers, including missing packets and excessive jitter. For example, an audio-video application might decide to reduce the rate of transmission over low-speed links if the traffic quality over the links is not high enough to support the current rate. The feedback from receivers is also important in diagnosing distribution faults. By monitoring reports from all session recipients, a network manager can tell whether a problem is specific to a single user or more widespread.
- **Identification:** RTCP packets carry a persistent textual description of the RTCP source. This provides more information about the source of data packets than the random SSRC identifier and enables a user to associate multiple streams from different sessions. For example, separate sessions for audio and video may be in progress.
- **Session size estimation and scaling:** To perform the first two functions, all participants send periodic RTCP packets. The rate of transmission of such packets must be scaled down as the number of participants increases. In a session with

few participants, RTCP packets are sent at the maximum rate of one every five seconds. RFC 1889 includes a relatively complex algorithm by which each participant limits its RTCP rate on the basis of the total session population. The objective is to limit RTCP traffic to less than 5% of total session traffic.

- **Session control:** RTCP optionally provides minimal session control information. An example is a participant identification to be displayed in the user interface.

An RTCP transmission consists of a number of separate RTCP packets bundled in a single UDP datagram (or other lower-level data unit). The following packet types are defined in RFC 1889:

- Sender Report (SR)
- Receiver Report (RR)
- Source Description (SDES)
- Goodbye (BYE)
- Application Specific

Figure 24.14 depicts the formats of these packet types. Each type begins with a 32-bit word containing the following fields:

- **Version (2 bits):** Current version is 2.
- **Padding (1 bit):** If set, indicates that this packet contains padding octets at the end of the control information. If so, the last octet of the padding contains a count of the number of padding octets.
- **Count (5 bits):** The number of reception report blocks contained in an SR or RR packet (RC), or the number of source items contained in an SDES or BYE packet.
- **Packet Type (8 bits):** Identifies RTCP packet type.
- **Length (16 bits):** Length of this packet in 32 bit words, minus one.

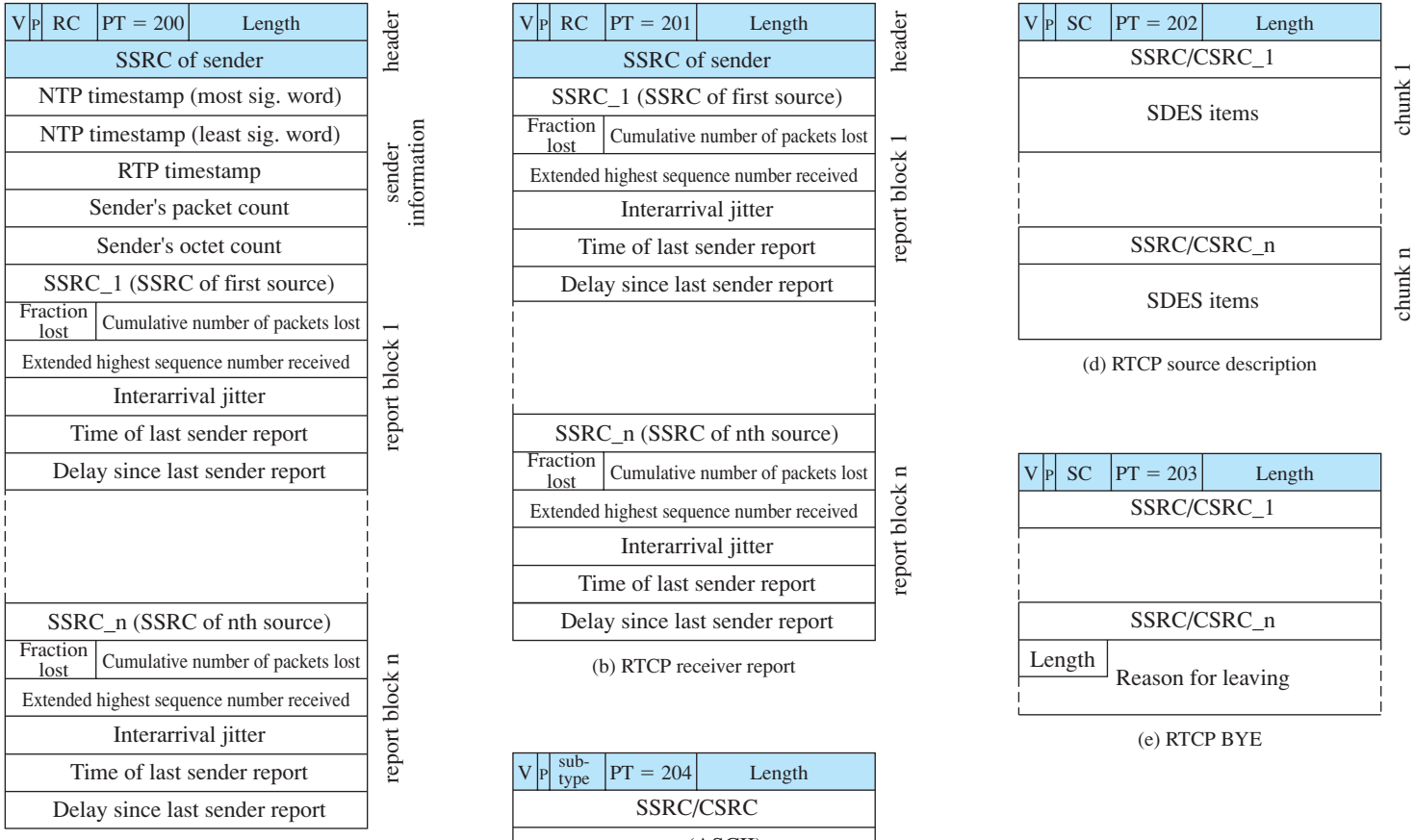
In addition, the Sender Report and Receiver Report packets contain the following field:

- **Synchronization Source Identifier:** Identifies the source of this RTCP packet

We now turn to a description of each packet type.

Sender Report (SR) RTCP receivers provide reception quality feedback using a Sender Report or a Receiver Report, depending on whether the receiver is also a sender during this session. Figure 24.14a shows the format of a Sender Report. The Sender Report consists of a header, already described; a sender information block; and zero or more reception report blocks. The sender information block includes the following fields:

- **NTP Timestamp (64 bits):** The absolute wall clock time when this report was sent; this is an unsigned fixed-point number with the integer part in the first 32 bits and the fractional part in the last 32 bits. This may be used by the sender in combination with timestamps returned in receiver reports to measure round-trip time to those receivers.



(a) RTCP sender report

(b) RTCP receiver report

(c) RTCP application-defined packet

(d) RTCP source description

(e) RTCP BYE

Figure 24.14 RTCP Formats

- **RTP Timestamp (32 bits):** This is a the relative time used to create timestamps in RTP data packets. This lets recipients place this report in the appropriate time sequence with RTP data packets from this source.
- **Sender's Packet Count (32 bits):** Total number of RTP data packets transmitted by this sender so far in this session.
- **Sender's Octet Count (32 bits):** Total number of RTP payload octets transmitted by this sender so far in this session.

Following the sender information block are zero or more reception report blocks. One reception block is included for each source from which this participant has received data during this session. Each block includes the following fields:

- **SSRC_n (32 bits):** Identifies the source referred to by this report block.
- **Fraction Lost (8 bits):** The fraction of RTP data packets from SSRC_n lost since the previous SR or RR packet was sent.
- **Cumulative Number of Packets Lost (24 bits):** Total number of RTP data packets from SSRC_n lost during this session.
- **Extended Highest Sequence Number Received (32 bits):** The least significant 16 bits record the highest RTP data sequence number received from SSRC_n. The most significant 16 bits record the number of times the sequence number has wrapped back to zero.
- **Interarrival Jitter (32 bits):** An estimate of the jitter experienced on RTP data packets from SSRC_n, explained later.
- **Last SR Timestamp (32 bits):** The middle 32 bits of the NTP timestamp in the last SR packet received from SSRC_n. This captures the least significant half of the integer and the most significant half of the fractional part of the timestamp and should be adequate.
- **Delay Since Last SR (32 bits):** The delay, expressed in units of 2^{-16} seconds, between receipt of the last SR packet from SSRC_n and the transmission of this report block. These last two fields can be used by a source to estimate round-trip time to a particular receiver.

Recall that delay jitter was defined as the maximum variation in delay experienced by packets in a single session. There is no simple way to measure this quantity at the receiver, but it is possible to estimate the average jitter in the following way. At a particular receiver, define the following parameters for a given source:

$S(I)$ = Timestamp from RTP data packet I.

$R(I)$ = Time of arrival for RTP data packet I. expressed in RTP timestamp units. The receiver must use the same clock frequency (increment interval) as the source but need not synchronize time values with the source.

$D(I)$ = The difference between the interarrival time at the receiver and the spacing between adjacent RTP data packets leaving the source.

$J(I)$ = Estimated average interarrival jitter up to the receipt of RTP data packet I.

Table 24.3 SDES Types (RFC 1889)

Value	Name	Description
0	END	End of SDES list
1	CNAME	Canonical name: unique among all participants within one RTP session
2	NAME	Real user name of the source
3	EMAIL	E-mail address
4	PHONE	Telephone number
5	LOC	Geographic location
6	TOOL	Name of application generating the stream
7	NOTE	Transient message describing the current state of the source
8	PRIV	Private experimental or application-specific extensions

The value of $D(I)$ is calculated as

$$D(I) = (R(I) - R(I - 1)) - (S(I) - S(I - 1))$$

Thus, $D(I)$ measures how much the spacing between arriving packets differs from the spacing between transmitted packets. In the absence of jitter, the spacings will be the same and $D(I)$ will have a value of 0. The interarrival jitter is calculated continuously as each data packet I is received, according to the formula

$$J(I) = \frac{15}{16}J(I - 1) + \frac{1}{16}|D(I)|$$

In this equation, $J(I)$ is calculated as an exponential average[3] of observed values of $D(I)$. Only a small weight is given to the most recent observation, so that temporary fluctuations do not invalidate the estimate.

The values in the Sender Report enable senders, receivers, and network managers to monitor conditions on the network as they relate to a particular session. For example, packet loss values give an indication of persistent congestion, while the jitter measures transient congestion. The jitter measure may provide a warning of increasing congestion before it leads to packet loss.

Receiver Report (RR) The format for the Receiver Report (Figure 24.14b) is the same as that for a Sender Report, except that the Packet Type field has a different value and there is no sender information block.

Source Description (SDES) The Source Description packet (Figure 24.14d) is used by a source to provide more information about itself. The packet consists of a 32-bit header followed by zero or more chunks, each of which contains information describing this source. Each chunk begins with an identifier for this source or for a contributing source. This is followed by a list of descriptive items. Table 24.3 lists the types of descriptive items defined in RFC 1889.

[3]For comparison, see Equation (20.3).

Goodbye (BYE) The BYE packet indicates that one or more sources are no longer active. This confirms to receivers that a prolonged silence is due to departure rather than network failure. If a BYE packet is received by a mixer, it is forwarded with the list of sources unchanged. The format of the BYE packet consists of a 32-bit header followed by one or more source identifiers. Optionally, the packet may include a textual description of the reason for leaving.

Application-Defined Packet This packet is intended for experimental use for functions and features that are application specific. Ultimately, an experimental packet type that proves generally useful may be assigned a packet type number and become part of the standardized RTCP.

24.5 RECOMMENDED READING AND WEB SITES

[GALL91] is a good overview of MPEG. [CHIA98] is a brief survey of all the MPEG standards. [KOEN99] provides an overview of MPEG-4; [BATT99] and [BATT00] are a more detailed treatment. [NACK99a] and [NACK99b] cover MPEG-7 in detail.

A good technical treatment of the algorithms in this chapter is [SAYO06]. [GOOD02] and [SCHU99] discuss SIP in the context of VoIP. [DIAN02] looks at SIP in the context of the support of multimedia services over the Internet.

BATT99 Battista, S.; Casalio, F.; and Lande, C. "MPEG-4: A Multimedia Standard for the Third Millennium, Part 1." *IEEE Multimedia*, October–December 1999.

BATT00 Battista, S.; Casalio, F.; and Lande, C. "MPEG-4: A Multimedia Standard for the Third Millennium, Part 2." *IEEE Multimedia*, January–March 2000.

CHIA98 Chiariglione, L. "The Impact of MPEG Standards on Multimedia Industry." *Proceedings of the IEEE*, June 1998.

DIAN02 Dianda, J.; Gurbani, V.; and Jones, M. "Session Initiation Protocol Services Architecture." *Bell Labs Technical Journal*, Volume 7, Number 1, 2002.

GALL91 Gall, D. "MPEG: A Video Compression Standard for Multimedia Applications." *Communications of the ACM*, April 1991.

GOOD02 Goode, B. "Voice Over Internet Protocol (VoIP)." *Proceedings of the IEEE*, September 2002.

KOEN99 Koenen, R. "MPEG-4: Multimedia for Our Time." *IEEE Spectrum*, February 1999.

NACK99a Nack, F., and Lindsay, A. "Everything You Wanted to Know about MPEG-7, Part 1."*IEEE Multimedia*, July–September 1999.

NACK99b Nack, F., and Lindsay, A. "Everything You Wanted to Know about MPEG-7, Part 2."*IEEE Multimedia*, October–December 1999.

SAYO06 Sayood, K. *Introduction to Data Compression.* New York: Elsevier, 2006.

SCHU99 Schulzrinne, H., and Rosenberg, J. "The IETF Internet Telephony Architecture and Protocols." *IEEE Network*, May/June 1999.

Recommended Web Sites:

- **MPEG Pointers and Resources:** An exhaustive list of links to MPEG-related sites, including products, software, video files, announcements, FAQs, and technical information.
- **SIP Forum:** Nonprofit organization to promote SIP. Site contains product information, white papers, and other useful information and links.
- **SIP Working Group:** Chartered by IETF to develop standards related to SIP. The Web site includes all relevant RFCs and Internet drafts.
- **Audio/Video Transport Working Group:** Chartered by IETF to develop standards related to RTP. The Web site includes all relevant RFCs and Internet drafts.
- **About RTP:** Web site devoted to RTP developments, including technical and industry developments.

24.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

lossless compression	Session Description Protocol (SDP)	SIP proxy server
lossy compression	Session Initiation Protocol (SIP)	SIP redirect server
MPEG	SIP location service	SIP registrar
Real-Time Transport Protocol (RTP)	SIP method	voice over IP (VoIP)
RTP Control Protocol (RTCP)		

Review Questions

24.1 What is the distinction between lossy and lossless compression?
24.2 What are the five key services provided by SIP?
24.3 List and briefly define the major components in an SIP network.
24.4 What is the Session Description Protocol?
24.5 What are some desirable properties for real-time communications?
24.6 What is the difference between hard and soft real-time applications?
24.7 What is the purpose of RTP?
24.8 What is the difference between RTP and RTCP?

Problems

24.1 In the MPEG block diagram shown in Figure 24.2, interframe processing involves comparing the current frame to a processed copy of preceding frames ($DCT(Q(Q^{-1}(DCT^{-1}(F)))))$). Why not do the comparison between input frames?

24.2 A single video source transmits 30 frames per second, each containing 2 Mbits of data. The data experiences a delay jitter of 1 s. What size of delay buffer is required at the destination to eliminate the jitter?

24.3 Argue the effectiveness, or lack thereof, of using RTP as a means of alleviating network congestion for multicast traffic.

24.4 In RTP, senders periodically transmit a sender report message that provides an absolute timestamp (the NTP Timestamp). The use of this absolute timestamp is essential to synchronize multiple streams, such as a video and an audio channel. Why can't RTP's Timestamp field be used for that purpose?

24.5 Illustrate how the last two fields in an RTCP SR or RR receiver report block can be used to calculate round-trip propagation time.