7.4 Protocols for Real-Time Interactive Applications

Real-time interactive applications, including Internet phone and video conferencing, promise to drive much of the future Internet growth. It is therefore not surprising that standards bodies, such as the IETF and ITU, have been busy for many years (and continue to be busy!) at hammering out standards for this class of applications. With the appropriate standards in place for real-time interactive applications, independent companies will be able to create new and compelling products that interoperate with each other. In this section we examine RTP, SIP, and H.323 for real-time interactive applications. All three sets of standards are enjoying widespread implementation in industry products.

7.4.1 RTP

In the previous section we learned that the sender side of a multimedia application appends header fields to the audio/video chunks before passing them to the transport layer. These header fields include sequence numbers and timestamps. Since most multimedia networking applications can make use of sequence numbers and timestamps, it is convenient to have a standardized packet structure that includes fields for audio/video data, sequence number, and timestamp, as well as other potentially useful fields. RTP, defined in RFC 3550, is such a standard. RTP can be used for transporting common formats such as PCM, GSM, and MP3 for sound and MPEG and H.263 for video. It can also be used for transporting proprietary sound and video formats. Today, RTP enjoys widespread implementation in hundreds of products and research prototypes. It is also complementary to other important real-time interactive protocols, including SIP and H.323.

In this section we provide an introduction to RTP and to its companion protocol, RTCP. We also encourage you to visit Henning Schulzrinne's RTP site [Schulzrinne-RTP 2007], which provides a wealth of information on the subject.

Also, you may want to visit the RAT site [RAT 2007], which documents an Internet phone application that uses RTP.

RTP Basics

RTP typically runs on top of UDP. The sending side encapsulates a media chunk within an RTP packet, then encapsulates the packet in a UDP segment, and then hands the segment to IP. The receiving side extracts the RTP packet from the UDP segment, then extracts the media chunk from the RTP packet, and then passes the chunk to the media player for decoding and rendering.

As an example, consider the use of RTP to transport voice. Suppose the voice source is PCM-encoded (that is, sampled, quantized, and digitized) at 64 kbps. Further suppose that the application collects the encoded data in 20-msec chunks, that is, 160 bytes in a chunk. The sending side precedes each chunk of the audio data with an **RTP header** that includes the type of audio encoding, a sequence number, and a timestamp. The RTP header is normally 12 bytes. The audio chunk along with the RTP header form the **RTP packet**. The RTP packet is then sent into the UDP socket interface. At the receiver side, the application receives the RTP packet from its socket interface. The application extracts the audio chunk from the RTP packet and uses the header fields of the RTP packet to properly decode and play back the audio chunk.

If an application incorporates RTP—instead of a proprietary scheme to provide payload type, sequence numbers, or timestamps—then the application will more easily interoperate with other networked multimedia applications. For example, if two different companies develop Internet phone software and they both incorporate RTP into their product, there may be some hope that a user using one of the Internet phone products will be able to communicate with a user using the other Internet phone product. In Section 7.4.3 we'll see that RTP is often used in conjunction with the Internet telephony standards.

It should be emphasized that RTP does not provide any mechanism to ensure timely delivery of data or provide other quality-of-service (QoS) guarantees; it does not even guarantee delivery of packets or prevent out-of-order delivery of packets. Indeed, RTP encapsulation is seen only at the end systems. Routers do not distinguish between IP datagrams that carry RTP packets and IP datagrams that don't.

RTP allows each source (for example, a camera or a microphone) to be assigned its own independent RTP stream of packets. For example, for a video conference between two participants, four RTP streams could be opened—two streams for transmitting the audio (one in each direction) and two streams for transmitting the video (again, one in each direction). However, many popular encoding techniques—including MPEG 1 and MPEG 2—bundle the audio and video into a single stream during the encoding process. When the audio and video are bundled by the encoder, then only one RTP stream is generated in each direction.

RTP packets are not limited to unicast applications. They can also be sent over one-to-many and many-to-many multicast trees. For a many-to-many multicast

session, all of the session's senders and sources typically use the same multicast group for sending their RTP streams. RTP multicast streams belonging together, such as audio and video streams emanating from multiple senders in a video conference application, belong to an **RTP session**.

RTP Packet Header Fields

As shown in Figure 7.10, the four main RTP packet header fields are the payload type, sequence number, timestamp, and source identifier fields.

The payload type field in the RTP packet is 7 bits long. For an audio stream, the payload type field is used to indicate the type of audio encoding (for example, PCM, adaptive delta modulation, linear predictive encoding) that is being used. If a sender decides to change the encoding in the middle of a session, the sender can inform the receiver of the change through this payload type field. The sender may want to change the encoding in order to increase the audio quality or to decrease the RTP stream bit rate. Table 7.2 lists some of the audio payload types currently supported by RTP.

For a video stream, the payload type is used to indicate the type of video encoding (for example, motion JPEG, MPEG 1, MPEG 2, H.261). Again, the sender can change video encoding on the fly during a session. Table 7.3 lists some of the video payload types currently supported by RTP. The other important fields are the following:

- *Sequence number field.* The sequence number field is 16 bits long. The sequence number increments by one for each RTP packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. For example, if the receiver side of the application receives a stream of RTP packets with a gap between sequence numbers 86 and 89, then the receiver knows that packets 87 and 88 are missing. The receiver can then attempt to conceal the lost data.
- *Timestamp field.* The timestamp field is 32 bits long. It reflects the sampling instant of the first byte in the RTP data packet. As we saw in the preceding section, the receiver can use timestamps to remove packet jitter introduced in the network and to provide synchronous playout at the receiver. The timestamp is derived from a sampling clock at the sender. As an example, for audio the timestamp clock increments by one for each sampling period (for example, each 125 μsec for an 8 kHz sampling clock); if the audio application generates chunks consisting of 160 encoded samples, then the timestamp increases by 160 for each RTP packet when the source is active. The timestamp clock continues to increase at a constant rate even if the source is inactive.

Figure 7.10 ♦ RTP header fields

Payload-Type Number	Audio Format	Sampling Rate	Rate
0	PCM μ-law	8 kHz	64 kbps
1	1016	8 kHz	4.8 kbps
3	GSM	8 kHz	13 kbps
7	LPC	8 kHz	2.4 kbps
9	G.722	16 kHz	48–64 kbps
14	MPEG Audio	90 kHz	—
15	G.728	8 kHz	16 kbps

Table 7.2 ♦ Audio payload types supported by RTP

- *Synchronization source identifier (SSRC).* The SSRC field is 32 bits long. It identifies the source of the RTP stream. Typically, each stream in an RTP session has a distinct SSRC. The SSRC is not the IP address of the sender, but instead is a number that the source assigns randomly when the new stream is started. The probability that two streams get assigned the same SSRC is very small. Should this happen, the two sources pick a new SSRC value.

Developing Software Applications with RTP

There are two approaches to developing an RTP-based networked application. The first approach is for the application developer to incorporate RTP by hand—that is, actually to write the code that performs RTP encapsulation at the sender side and RTP unraveling at the receiver side. The second approach is for the application developer to use existing RTP libraries (for C programmers) and Java classes (for

Payload-Type Number	Video Format
26	Motion JPEG
31	H.261
32	MPEG 1 video
33	MPEG 2 video

Table 7.3 ♦ Some video payload types supported by RTP

Java programmers), which perform the encapsulation and unraveling for the application. Since you may be itching to write your first multimedia networking application using RTP, let us now elaborate a little on these two approaches. (The programming assignment at the end of this chapter will guide you through the creation of an RTP application.) We'll do this in the context of unicast communication (rather than for multicast).

Recall from Chapter 2 that the UDP API requires the sending process to set, for each UDP segment it sends, the destination IP address and the destination port number before popping the packet into the UDP socket. The UDP segment will then wander through the Internet and (if the segment is not lost due to, for example, router buffer overflow) eventually arrive at the door of the receiving process for the application. This door is fully addressed by the destination IP address and the destination port number. In fact, any IP datagram containing this destination IP address and destination port number will be directed to the receiving process's UDP door. (The UDP API also lets the application developer set the UDP source port number; however, this value has no effect on which process the segment is sent to.) It is important to note that RTP does not mandate a specific port number. When the application developer creates an RTP application, the developer specifies the port numbers for the two sides of the application.

As part of the programming assignment for this chapter, you will write an RTP server that encapsulates stored video frames within RTP packets. You will do this by hand; that is, your application will grab a video frame, add the RTP headers to the frame to create an RTP packet, and then pass the RTP frame to the UDP socket. To do this, you will need to create placeholder fields for the various RTP headers, including a sequence number field and a timestamp field. And for each RTP packet that is created, you will have to set the sequence number and the timestamp appropriately. You will explicitly code all of these RTP operations into the sender side of your application. As shown in Figure 7.11, your API to the network will be the standard UDP socket API.

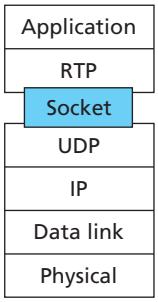

Figure 7.11 ♦ RTP is part of the application and lies above the UDP socket.

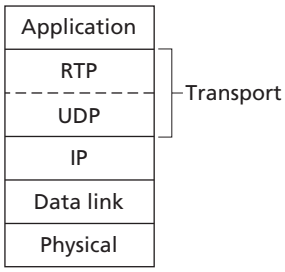

Figure 7.12 ♦ RTP can be viewed as a sublayer of the transport layer.

An alternative approach (not done in the programming assignment) is to use a Java RTP class (or a C RTP library for C programmers) to implement the RTP operations. With this approach, as shown in Figure 7.12, the application developer is given the impression that RTP is part of the transport layer, with an RTP/UDP API between the application layer and the transport layer. Without getting into the nitty-gritty details (as they are class/library-dependent), when sending a chunk of media into the API, the sending side of the application needs to provide the interface with the media chunk itself, a payload-type number, an SSRC, and a timestamp, along with a destination port number and an IP destination address. We mention here that the Java Media Framework (JMF) includes a complete RTP implementation.

7.4.2 RTP Control Protocol (RTCP)

RFC 3550 also specifies RTCP, a protocol that a networked multimedia application can use in conjunction with RTP. As shown in the multicast scenario in Figure 7.13, RTCP packets are transmitted by each participant in an RTP session to all other participants in the session using IP multicast. For an RTP session, typically there is a single multicast address and all RTP and RTCP packets belonging to the session use the multicast address. RTP and RTCP packets are distinguished from each other through the use of distinct port numbers. (The RTCP port number is set to be equal to the RTP port number plus one.)

RTCP packets do not encapsulate chunks of audio or video. Instead, RTCP packets are sent periodically and contain sender and/or receiver reports that announce statistics that can be useful to the application. These statistics include number of packets sent, number of packets lost, and interarrival jitter. The RTP specification [RFC 3550] does not dictate what the application should do with this feedback information; this is up to the application developer. Senders can use the feedback information, for example, to modify their transmission rates. The feedback

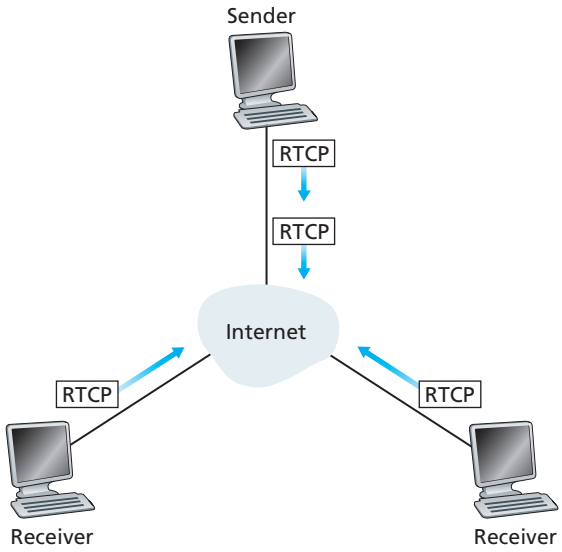

Figure 7.13 ♦ Both senders and receivers send RTCP messages.

information can also be used for diagnostic purposes; for example, receivers can determine whether problems are local, regional, or global.

RTCP Packet Types

For each RTP stream that a receiver receives as part of a session, the receiver generates a reception report. The receiver aggregates its reception reports into a single RTCP packet. The packet is then sent into the multicast tree that connects all the session's participants. The reception report includes several fields, the most important of which are listed below.

- The SSRC of the RTP stream for which the reception report is being generated.
- The fraction of packets lost within the RTP stream. Each receiver calculates the number of RTP packets lost divided by the number of RTP packets sent as part of the stream. If a sender receives reception reports indicating that the receivers are receiving only a small fraction of the sender's transmitted packets, it can switch to a lower encoding rate, with the aim of decreasing network congestion and improving the reception rate.
- The last sequence number received in the stream of RTP packets.
- The interarrival jitter, which is a smoothed estimate of the variation in the interarrival time between successive packets in the RTP stream.

For each RTP stream that a sender is transmitting, the sender creates and transmits RTCP sender report packets. These packets include information about the RTP stream, including:

- The SSRC of the RTP stream
- The timestamp and wall clock time of the most recently generated RTP packet in the stream
- The number of packets sent in the stream
- The number of bytes sent in the stream

Sender reports can be used to synchronize different media streams within an RTP session. For example, consider a video conferencing application for which each sender generates two independent RTP streams, one for video and one for audio. The timestamps in these RTP packets are tied to the video and audio sampling clocks, and are not tied to the *wall clock time* (i.e., real time). Each RTCP sender report contains, for the most recently generated packet in the associated RTP stream, the timestamp of the RTP packet and the wall clock time when the packet was created. Thus the RTCP sender report packets associate the sampling clock with the real-time clock. Receivers can use this association in RTCP sender reports to synchronize the playout of audio and video.

For each RTP stream that a sender is transmitting, the sender also creates and transmits source description packets. These packets contain information about the source, such as the e-mail address of the sender, the sender's name, and the application that generates the RTP stream. It also includes the SSRC of the associated RTP stream. These packets provide a mapping between the source identifier (that is, the SSRC) and the user/host name.

RTCP packets are stackable; that is, receiver reception reports, sender reports, and source descriptors can be concatenated into a single packet. The resulting packet is then encapsulated into a UDP segment and forwarded into the multicast tree.

RTCP Bandwidth Scaling

You may have observed that RTCP has a potential scaling problem. Consider, for example, an RTP session that consists of one sender and a large number of receivers. If each of the receivers periodically generates RTCP packets, then the aggregate transmission rate of RTCP packets can greatly exceed the rate of RTP packets sent by the sender. Observe that the amount of RTP traffic sent into the multicast tree does not change as the number of receivers increases, whereas the amount of RTCP traffic grows linearly with the number of receivers. To solve this scaling problem, RTCP modifies the rate at which a participant sends RTCP packets into the multicast tree as a function of the number of participants in the session. Also, since each

participant sends control packets to everyone else, each participant can estimate the total number of participants in the session [Friedman 1999].

RTCP attempts to limit its traffic to 5 percent of the session bandwidth. For example, suppose there is one sender, which is sending video at a rate of 2 Mbps. Then RTCP attempts to limit its traffic to 5 percent of 2 Mbps, or 100 kbps, as follows. The protocol gives 75 percent of this rate, or 75 kbps, to the receivers; it gives the remaining 25 percent of the rate, or 25 kbps, to the sender. The 75 kbps devoted to the receivers is equally shared among the receivers. Thus, if there are R receivers, then each receiver gets to send RTCP traffic at a rate of $75/R$ kbps, and the sender gets to send RTCP traffic at a rate of 25 kbps. A participant (a sender or receiver) determines the RTCP packet transmission period by dynamically calculating the average RTCP packet size (across the entire session) and dividing the average RTCP packet size by its allocated rate. In summary, the period for transmitting RTCP packets for a sender is

$$T = \frac{\text{number of senders}}{.25 \cdot .05 \cdot \text{session bandwidth}} (\text{avg. RTCP packet size})$$

And the period for transmitting RTCP packets for a receiver is

$$T = \frac{\text{number of receivers}}{.75 \cdot .05 \cdot \text{session bandwidth}} (\text{avg. RTCP packet size})$$

7.4.3 SIP

Imagine a world in which, when you are working on your PC, your phone calls arrive over the Internet to your PC. When you get up and start walking around, your new phone calls are automatically routed to your PDA. And when you are driving in your car, your new phone calls are automatically routed to some Internet appliance in your car. In this same world, while participating in a conference call, you can access an address book to call and invite other participants into the conference. The other participants may be at their PCs, or walking with their PDAs, or driving their cars—no matter where they are, your invitation is transparently routed to them. In this same world, when you browse an individual's homepage, there will be a link "Call Me"; clicking on this link establishes an Internet phone session between your PC and the owner of the homepage (wherever that person might be).

In this world, there is no longer a circuit-switched telephone network. Instead, all calls pass over the Internet—from end to end. In this same world, companies no longer use private branch exchanges (PBXs), that is, local circuit switches for handling intracompany telephone calls. Instead, the intracompany phone traffic flows over the company's high-speed LAN.

All of this may sound like science fiction. And, of course, today's circuit-switched networks and PBXs are not going to disappear completely in the near future [Jiang 2001]. Nevertheless, protocols and products exist to turn this vision into a reality. Among the most promising protocols in this direction is the Session

Initiation Protocol (SIP), defined in [RFC 3261]. SIP is a lightweight protocol that does the following:

- It provides mechanisms for establishing calls between a caller and a callee over an IP network. It allows the caller to notify the callee that it wants to start a call. It allows the participants to agree on media encodings. It also allows participants to end calls.
- It provides mechanisms for the caller to determine the current IP address of the callee. Users do not have a single, fixed IP address because they may be assigned addresses dynamically (using DHCP) and because they may have multiple IP devices, each with a different IP address.
- It provides mechanisms for call management, such as adding new media streams during the call, changing the encoding during the call, inviting new participants during the call, call transfer, and call holding.

Setting Up a Call to a Known IP Address

To understand the essence of SIP, it is best to take a look at a concrete example. In this example, Alice is at her PC and she wants to call Bob, who is also working at his PC. Alice's and Bob's PCs are both equipped with SIP-based software for making and receiving phone calls. In this initial example, we'll assume that Alice knows the IP address of Bob's PC. Figure 7.14 illustrates the SIP call-establishment process.

In Figure 7.14, we see that an SIP session begins when Alice sends Bob an INVITE message, which resembles an HTTP request message. This INVITE message is sent over UDP to the well-known port 5060 for SIP. (SIP messages can also be sent over TCP.) The INVITE message includes an identifier for Bob (bob@193.64.210.89), an indication of Alice's current IP address, an indication that Alice desires to receive audio, which is to be encoded in format AVP 0 (PCM encoded μ-law) and encapsulated in RTP, and an indication that she wants to receive the RTP packets on port 38060. After receiving Alice's INVITE message, Bob sends an SIP response message, which resembles an HTTP response message. This response SIP message is also sent to the SIP port 5060. Bob's response includes a 200 OK as well as an indication of his IP address, his desired encoding and packetization for reception, and his port number to which the audio packets should be sent. Note that in this example Alice and Bob are going to use different audio-encoding mechanisms: Alice is asked to encode her audio with GSM whereas Bob is asked to encode his audio with PCM μ-law. After receiving Bob's response, Alice sends Bob an SIP acknowledgment message. After this SIP transaction, Bob and Alice can talk. (For visual convenience, Figure 7.14 shows Alice talking after Bob, but in truth they would normally talk at the same time.) Bob will encode and packetize the audio as requested and send the audio packets to port number 38060 at IP address

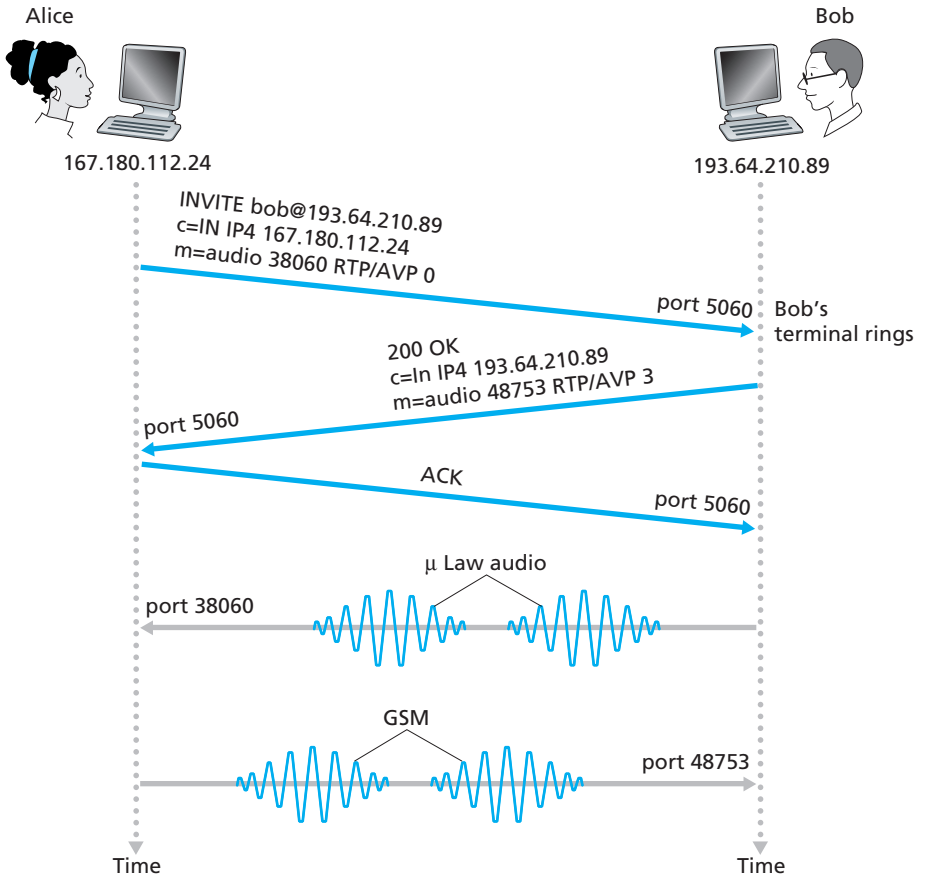

Figure 7.14 ♦ SIP call establishment when Alice knows Bob's IP address

167.180.112.24. Alice will also encode and packetize the audio as requested and send the audio packets to port number 48753 at IP address 193.64.210.89.

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

In this example, let's consider what would happen if Bob does not have a PCM μ-law codec for encoding audio. In this case, instead of responding with 200 OK, Bob would likely respond with a 600 Not Acceptable and list in the message all the codecs he can use. Alice would then choose one of the listed codecs and send another INVITE message, this time advertising the chosen codec. Bob could also simply

reject the call by sending one of many possible rejection reply codes. (There are many such codes, including "busy," "gone," "payment required," and "forbidden.")

SIP Addresses

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

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

SIP Messages

In this short introduction to SIP, we'll not cover all SIP message types and headers. Instead, we'll take a brief look at the SIP INVITE message, along with a few common header lines. Let us again suppose that Alice wants to initiate an IP phone call to Bob, and this time Alice knows only Bob's SIP address, bob@domain.com, and does not know the IP address of the device that Bob is currently using. Then her message might look something like this:

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

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

The INVITE line includes the SIP version, as does an HTTP request message. Whenever an SIP message passes through an SIP device (including the device that originates the message), it attaches a Via header, which indicates the IP address of the device. (We'll see soon that the typical INVITE message passes through many SIP

devices before reaching the callee's SIP application.) Similar to an e-mail message, the SIP message includes a From header line and a To header line. The message includes a Call-ID, which uniquely identifies the call (similar to the message-ID in e-mail). It includes a Content-Type header line, which defines the format used to describe the content contained in the SIP message. It also includes a Content-Length header line, which provides the length in bytes of the content in the message. Finally, after a carriage return and line feed, the message contains the content. In this case, the content provides information about Alice's IP address and how Alice wants to receive the audio.

Name Translation and User Location

In the example in Figure 7.14, we assumed that Alice's SIP device knew the IP address where Bob could be contacted. But this assumption is quite unrealistic, not only because IP addresses are often dynamically assigned with DHCP, but also because Bob may have multiple IP devices (for example, different devices for his home, work, and car). So now let us suppose that Alice knows only Bob's e-mail address, bob@domain.com, and that this same address is used for SIP-based calls. In this case, Alice needs to obtain the IP address of the device that the user bob@domain.com is currently using. To find this out, Alice creates an INVITE message that begins with INVITE bob@domain.com SIP/2.0 and sends this message to an **SIP proxy**. The proxy will respond with an SIP reply that might include the IP address of the device that bob@domain.com is currently using. Alternatively, the reply might include the IP address of Bob's voicemail box, or it might include a URL of a Web page (that says "Bob is sleeping. Leave me alone!"). Also, the result returned by the proxy might depend on the caller: if the call is from Bob's wife, he might accept the call and supply his IP address; if the call is from Bob's mother-in-law, he might respond with the URL that points to the I-am-sleeping Web page!

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

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

Bob's registrar keeps track of Bob's current IP address. Whenever Bob switches to a new SIP device, the new device sends a new register message, indicating the

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

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

As an example, consider Figure 7.15, in which jim@umass.edu, currently working on 217.123.56.89, wants to initiate a Voice over IP (VoIP) session with keith@upenn.edu, currently working on 197.87.54.21. The following steps are taken: (1) Jim sends an INVITE message to the umass SIP proxy. (2) The proxy does a DNS lookup on the SIP registrar upenn.edu (not shown in diagram) and then

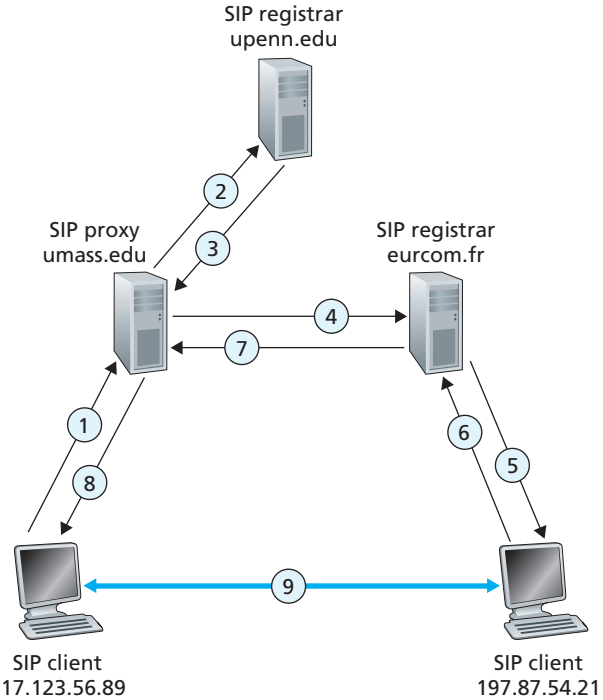

Figure 7.15 ♦ Session initiation, involving SIP proxies and registrars

forwards the message to the registrar server. (3) Because keith@upenn.edu is no longer registered at the upenn registrar, the upenn registrar sends a redirect response, indicating that it should try keith@eurecom.fr. (4) The umass proxy sends an INVITE message to the eurecom SIP registrar. (5) The eurecom registrar knows the IP address of keith@eurecom.fr and forwards the INVITE message to the host 197.87.54.21, which is running Keith's SIP client. (6–8) An SIP response is sent back through registrars/proxies to the SIP client on 217.123.56.89. (9) Media is sent directly between the two clients. (There is also an SIP acknowledgment message, which is not shown.)

Our discussion of SIP has focused on call initiation for voice calls. SIP, being a signaling protocol for initiating and ending calls in general, can be used for video conference calls as well as for text-based sessions. In fact, SIP has become a fundamental component in many instant messaging applications. Readers desiring to learn more about SIP are encouraged to visit Henning Schulzrinne's SIP Web site [Schulzrinne-SIP 2007]. In particular, on this site you will find open source software for SIP clients and servers [SIP Software 2007].

7.4.4 H.323

As an alternative to SIP, H.323 is a popular standard for real-time audio and video conferencing among end systems on the Internet. As shown in Figure 7.16, the standard also covers how end systems attached to the Internet communicate with

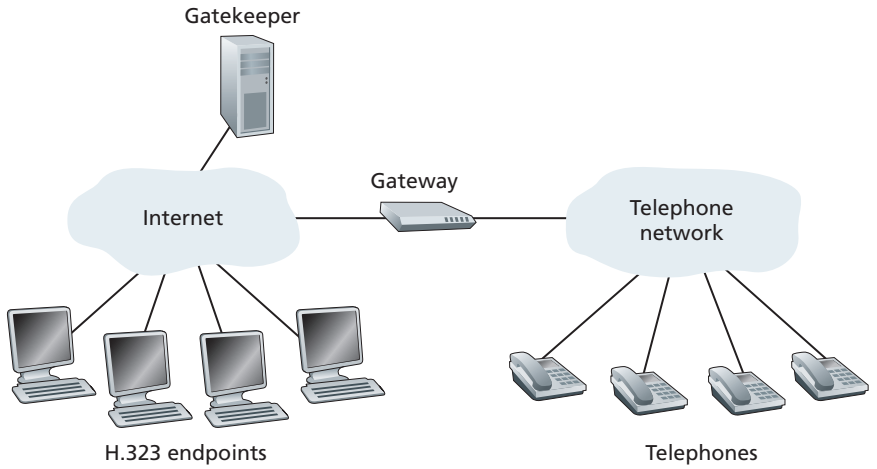

Figure 7.16 ♦ H.323 end systems attached to the Internet can communicate with telephones attached to a circuit-switched telephone network.

telephones attached to ordinary circuit-switched telephone networks. (SIP does this as well, although we did not discuss it.) The H.323 gatekeeper is a device similar to an SIP registrar.

The H.323 standard is an umbrella specification that includes the following specifications:

- A specification for how end points negotiate common audio/video encodings. Because H.323 supports a variety of audio and video encoding standards, a protocol is needed to allow the communicating end points to agree on a common encoding.
- A specification for how audio and video chunks are encapsulated and sent over the network. In particular, H.323 mandates RTP for this purpose.
- A specification for how end points communicate with their respective gatekeepers.
- A specification for how Internet phones communicate through a gateway with ordinary phones in the PSTN.

Minimally, each H.323 end point *must* support the G.711 speech compression standard. G.711 uses PCM to generate digitized speech at either 56 kbps or 64 kbps. Although H.323 requires every end point to be voice capable (through G.711), video capabilities are optional. Because video support is optional, manufacturers of terminals can sell simpler speech terminals as well as more complex terminals that support both audio and video. Video capabilities for an H.323 end point are optional. However, if an end point does support video, then it must (at the very least) support the QCIF H.261 (176 x 144 pixels) video standard.

H.323 is a comprehensive umbrella standard, which, in addition to the standards and protocols described above, mandates an H.245 control protocol, a Q.931 signaling channel, and an RAS protocol for registration with the gatekeeper.

We conclude this section by highlighting some of the most important differences between H.323 and SIP.

- H.323 is a complete, vertically integrated suite of protocols for multimedia conferencing: signaling, registration, admission control, transport, and codecs.
- SIP, on the other hand, addresses only session initiation and management and is a single component. SIP works with RTP but does not mandate it. It works with G.711 speech codecs and QCIF H.261 video codecs but does not mandate them. It can be combined with other protocols and services.
- H.323 comes from the ITU (telephony), whereas SIP comes from the IETF and borrows many concepts from the Web, DNS, and Internet e-mail.
- H.323, being an umbrella standard, is large and complex. SIP uses the KISS principle: keep it simple, stupid.

For an excellent discussion of H.323, SIP, and VoIP in general, see [Hersent 2000].