PART II

MEDIA TRANSPORT USING RTP

3

THE REAL-TIME TRANSPORT PROTOCOL

- Fundamental Design Philosophies of RTP
- Standard Elements of RTP
- Related Standards
- Future Standards Development

This chapter describes the design of the RTP framework starting with the philosophy and background of the design, gives an overview of the applicable standards, and explains how those standards interrelate. It concludes with a discussion of possible future directions for the development of those standards.

Fundamental Design Philosophies of RTP

The challenge facing the designers of RTP was to build a mechanism for robust, real-time media delivery above an unreliable transport layer. They achieved this goal with a design that follows

the twin philosophies of *application-level framing* and the *end-to-end principle*.

Application-Level Framing

The concepts behind application-level framing were first elucidated by Clark and Tennenhouse[65] in 1990. Their central thesis is that only the application has sufficient knowledge of its data to make an informed decision about how that data should be transported. The implication is that a transport protocol should accept data in application-meaningful units (application data units, ADUs) and expose the details of their delivery as much as possible so that the application can make an appropriate response if an error occurs. The application partners with the transport, cooperating to achieve reliable delivery.

Application-level framing comes from the recognition that there are many ways in which an application can recover from network problems, and that the correct approach depends on both the application and the scenario in which it is being used. In some cases it is necessary to retransmit an exact copy of the lost data. In others, a lower-fidelity copy may be used, or the data may have been superseded, so the replacement is different from the original. Alternatively, the loss can be ignored if the data was of only transient interest. These choices are possible only if the application interacts closely with the transport.

The goal of application-level framing is somewhat at odds with the design of TCP, which hides the lossy nature of the underlying IP network to achieve reliable delivery at the expense of timeliness. It does, however, fit well with UDP-based transport and with the characteristics of real-time media. As noted in Chapter 2, Voice and Video Communication over Packet Networks, real-time audio and visual media is often loss tolerant but has strict timing bounds. By using application-level framing with UDP-based transport, we are able to accept losses where necessary, but we also have the flexibility to use the full spectrum of recovery techniques, such as retransmission and forward error correction, where appropriate.

These techniques give an application great flexibility to react to network problems in a suitable manner, rather than being constrained by the dictates of a single transport layer.

A network that is designed according to the principles of application-level framing should not be specific to a particular application. Rather it should expose the limitations of a generic transport layer so that the application can cooperate with the network in achieving the best possible delivery. Application-level framing implies a weakening of the strict layers defined by the OSI reference model. It is a pragmatic approach, acknowledging the importance of layering, but accepting the need to expose some details of the lower layers.

The philosophy of application-level framing implies smart, network-aware applications that are capable of reacting to problems.

The End-to-End Principle

The other design philosophy adopted by RTP is the end-to-end principle.[70] It is one of two approaches to designing a system that must communicate reliably across a network. In one approach, the system can pass responsibility for the correct delivery of data along with that data, thus ensuring reliability hop by hop. In the other approach, the responsibility for data can remain with the endpoints, ensuring reliability end-to-end even if the individual hops are unreliable. It is this second end-to-end approach that permeates the design of the Internet, with both TCP and RTP following the end-to-end principle.

The main consequence of the end-to-end principle is that intelligence tends to bubble up toward the top of the protocol stack. If the systems that make up the network path never take responsibility for the data, they can be simple and do not need to be robust. They may discard data that they cannot deliver, because the endpoints will recover without their help. The end-to-end principle implies that intelligence is at the endpoints, not within the network.

The result is a design that implies smart, network-aware endpoints and a dumb network. This design is well suited to the Internet—perhaps the ultimate dumb network—but does require significant work on the part of an application designer. It is also a design unlike that of many other networks. The traditional telephone network, for example, adopts the model of an intelligent network and dumb endpoints, and the MPEG transport model allows dumb receivers with smart senders. This difference in design changes the style of the applications, placing greater emphasis on receiver design and making sender and receiver more equal partners in the transmission.

Achieving Flexibility

The RTP framework was designed to be sufficient for many scenarios, with little additional protocol support. In large part this design was based around the lightweight sessions model for video conferencing.[76] In this scenario the RTP control protocol provides all the necessary session management functions, and all that is needed to join the session is the IP address and the mapping from media definitions to RTP payload type identifiers. This model also works well for one to many scenarios—for example, Internet radio, where the feedback provided by the control protocol gives the source an estimate of the audience size and reception quality.

For unicast voice telephony, some have argued that RTP provides unnecessary features, and is heavyweight and inefficient for highly compressed voice frames. In practice, with the use of header compression this is not a strong argument, and the features provided enable extension to multimedia and multiparty sessions with ease. Still others—for example, the digital cinema community—have argued that RTP is underspecified for their needs and should include stronger quality-of-service and security support, more detailed statistics, and so on.

The strength of RTP is that it provides a unifying framework for real-time audio/video transport, satisfying most applications directly, yet being malleable for those applications that stretch its limits.

Standard Elements of RTP

The primary standard for audio/video transport in IP networks is the Real-time Transport Protocol (RTP), along with associated profiles and payload formats. RTP was developed by the Audio/Video Transport working group of the Internet Engineering Task Force (IETF), and it has since been adopted by the International Telecommunications Union (ITU) as part of its H.323 series of recommendations, and by several other standards organizations.

RTP provides a framework for the transport of real-time media and needs to be profiled for particular uses before it is complete. The RTP profile for audio and video conferences with minimal control was standardized along with RTP, and several more profiles are under development. Each profile is accompanied by several payload format specifications, each of which describes the transport of a particular media format.

The RTP Specification

RTP was published as an IETF proposed standard (RFC 1889) in January 1996,[6] and its revision for draft standard status is almost complete.[50] The first revision of ITU recommendation H.323 included a verbatim copy of the RTP specification; later revisions reference the current IETF standard.

In the IETF standards process,[8] a specification undergoes a development cycle in which multiple *Internet drafts* are produced as the details of the design are worked out. When the design is complete, it is published as a *proposed standard* RFC. A proposed standard is generally considered stable, with all known design issues worked out, and suitable for implementation. If that proposed standard proves useful, and if there are independent and interoperable implementations of each feature of that standard, it can then be advanced to *draft standard* status (possibly involving changes to correct any problems found in the proposed standard). Finally, after extensive experi-

ence, it may be published as a full *standard* RFC. Advancement beyond proposed standard status is a significant hurdle that many protocols never achieve.

RTP typically sits on top of UDP/IP transport, enhancing that transport with loss detection and reception quality reporting, provision for timing recovery and synchronization, payload and source identification, and marking of significant events within the media stream. Most implementations of RTP are part of an application or library that is layered above the UDP/IP sockets interface provided by the operating system. This is not the only possible design, though, and nothing in the RTP protocol requires UDP or IP. For example, some implementations layer RTP above TCP/IP, and others use RTP on non-IP networks, such as Asynchronous Transfer Mode (ATM) networks.

There are two parts to RTP: the *data transfer protocol* and an associated *control protocol*. The RTP data transfer protocol manages delivery of real-time data, such as audio and video, between end systems. It defines an additional level of framing for the media payload, incorporating a sequence number for loss detection, timestamp to enable timing recovery, payload type and source identifiers, and a marker for significant events within the media stream. Also specified are rules for timestamp and sequence number usage, although these rules are somewhat dependent on the profile and payload format in use, and for multiplexing multiple streams within a session. The RTP data transfer protocol is discussed further in Chapter 4.

The RTP control protocol (RTCP) provides reception quality feedback, participant identification, and synchronization between media streams. RTCP runs alongside RTP and provides periodic reporting of this information. Although data packets are typically sent every few milliseconds, the control protocol operates on the scale of seconds. The information sent in RTCP is necessary for synchronization between media streams—for example, for lip

synchronization between audio and video—and can be useful for adapting the transmission according to reception quality feedback, and for identifying the participants. The RTP control protocol is discussed further in Chapter 5.

RTP supports the notion of *mixers* and *translators*, middle boxes that can operate on the media as it flows between endpoints. These may be used to translate an RTP session between different lower-layer protocols—for example, bridging between participants on IPv4 and IPv6 networks, or bringing a unicast-only participant into a multicast group. They can also adapt a media stream in some way—for example, transcoding the data format to reduce the bandwidth, or mixing multiple streams together.

It is hard to place RTP in the OSI reference model. It performs many of the tasks typically assigned to a transport-layer protocol, yet it is not a complete transport in itself. RTP also performs some tasks of the session layer (spanning disparate transport connections and managing participant identification in a transport-neutral manner) and of the presentation layer (defining standard representations for media data).

RTP Profiles

It is important to be aware of the limits of the RTP protocol specification because it is deliberately incomplete in two ways. First, the standard does not specify algorithms for media playout and timing regeneration, synchronization between media streams, error concealment and correction, or congestion control. These are properly the province of the application designer, and because different applications have different needs, it would be foolish for the standard to mandate a single behavior. It does, of course, provide the necessary information for these algorithms to operate when they have been specified. Later chapters will discuss application design and the trade-offs inherent in providing these features.

Second, some details of the transport are left open to modification by profiles and payload format definitions. These include features

such as the resolution of the timestamps, marking of interesting events within a media stream, and use of the payload type field. The features that can be specified by RTP profiles include the following:

- Mapping between the payload type identifier in the RTP header and the payload format specifications (which describe how individual media codecs are to be used with RTP). Each profile will reference multiple payload formats and may indicate how particular signaling protocols (for example, SDP[15]) are used to describe the mapping.
- The size of the payload type identifier field in the RTP header, and the number of bits used to mark events of interest within a media stream.
- Additions to the fixed RTP data transfer protocol header, if that header proves insufficient for a particular class of application.
- The reporting interval for the RTP control protocol—for example, to make feedback more timely at the expense of additional overhead.
- Limitations on the RTCP packet types that are to be used, if some of the information provided is not useful to that class of applications. In addition, a profile may define extensions to RTCP to report additional information.
- Additional security mechanisms—for example, new encryption and authentication algorithms.
- Mapping of RTP and RTCP onto lower-layer transport protocols.

At the time of this writing, there is a single RTP profile: the RTP profile for audio and video conferences with minimal control. This profile was published as a proposed standard (RFC 1890) along with the RTP specification in January 1996,[7] and its revision for draft standard status is almost complete.[49] Several new profiles are under development. Those likely to be available soon include

profiles specifying additional security,[55] as well as feedback and repair mechanisms.[44]

RTP Payload Formats

The final piece of the RTP framework is the payload formats, defining how particular media types are transported within RTP. Payload formats are referenced by RTP profiles, and they may also define certain properties of the RTP data transfer protocol.

The relation between an RTP payload format and profile is primarily one of namespace, although the profile may also specify some general behavior for payload formats. The namespace relates the payload type identifier in the RTP packets to the payload format specifications, allowing an application to relate the data to a particular media codec. In some cases the mapping between payload type and payload format is static; in others the mapping is dynamic via an out-of-band control protocol. For example, the RTP profile for audio and video conferences with minimal control[7] defines a set of static payload type assignments, and a mechanism for mapping between a MIME type identifying a payload format, and a payload type identifier using the Session Description Protocol (SDP).

The relation between a payload format and the RTP data transfer protocol is twofold: A payload format will specify the use of certain RTP header fields, and it may define an additional payload header. The output produced by a media codec is translated into a series of RTP data packets—some parts mapping onto the RTP header, some into a payload header, and most into the payload data. The complexity of this mapping process depends on the design of the codec and on the degree of error resilience required. In some cases the mapping is simple; in others it is more complex.

At its simplest, a payload format defines only the mapping between media clock and RTP timestamp, and mandates that each frame of codec output is placed directly into an RTP packet for

transport. An example of this is the payload format for G.722.1 audio.[36] Unfortunately, this is not sufficient in many cases because many codecs were developed without reference to the needs of a packet delivery system and need to be adapted to this environment. Others were designed for packet networks but require additional header information. In these cases the payload format specification defines an additional payload header, to be placed after the main RTP header, and rules for generation of that header.

Many payload formats have been defined, matching the diversity of codecs that are in use today, and many more are under development. At the time of this writing, the following audio payload formats are in common use, although this is by no means an exhaustive list: G.711, G.723.1, G.726, G.728, G.729, GSM, QCELP, MP3, and DTMF.[30,34,38,49] The commonly used video payload formats include H.261, H.263, and MPEG.[9,12,22]

There are also payload formats that specify error correction schemes. For example, RFC 2198 defines an audio redundancy encoding scheme,[10] and RFC 2733 defines a generic forward error correction scheme based on parity coding.[32] In these payload formats there is an additional layer of indirection, the codec output is mapped onto RTP packets, and those packets themselves are mapped to produce an error-resilient transport. Error correction is discussed in more detail in Chapter 9, Error Correction.

Optional Elements

Two optional pieces of the RTP framework are worth mentioning at this stage: header compression and multiplexing.

Header compression is a means by which the overhead of the RTP and UDP/IP headers can be reduced on a per-link basis. It is used on bandwidth-constrained links—for example, cellular and dial-up links—and can reduce the 40-byte combination of RTP/UDP/IP headers to 2 bytes, at the expense of additional processing by the systems on the ends of the compressed link. Header compression is discussed further in Chapter 11.

Multiplexing is the means by which multiple related RTP sessions are combined into one. Once again, the motivation is to reduce overheads, except this time the procedure operates end-to-end. Multiplexing is discussed in Chapter 12, Multiplexing and Tunneling.

Both header compression and multiplexing can be considered to be part of the RTP framework. Unlike the profiles and payload formats, they are clearly special-purpose, optional parts of the system, and many implementations don't use either feature.

Related Standards

In addition to the RTP framework, a complete system will typically require the use of various other protocols and standards for call setup and control, session description, multiparty communication, and signaling quality-of-service requirements. Although this book does not cover the use of such protocols in detail, in this section it provides pointers to their specification and further reading.

The complete multimedia protocol stack is illustrated in Figure 3.1, showing the relationship between the RTP framework and the

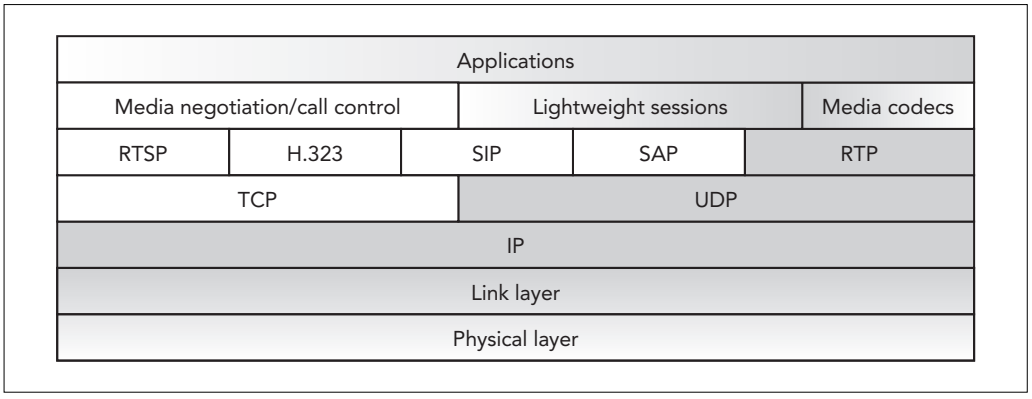

FIGURE 3.1

The Multimedia Protocol Stack

supporting setup and control protocols. The protocols and functions that are discussed in this book are highlighted.

Call Setup and Control

Various call setup, control, and advertisement protocols can be used to start an RTP session, depending on the application scenario:

- For the purpose of starting an interactive session, be it a voice telephony call or a video conference, there are two standards. The original standard in this area was ITU recommendation H.323,[62] and more recently the IETF has defined the Session Initiation Protocol (SIP).[28,111]
- For the purpose of starting a noninteractive session—for example, video-on-demand—the main standard is the Real-Time Streaming Protocol (RTSP).[14]
- The original use of RTP was with IP multicast and the lightweight sessions model of conferencing. This design used the Session Announcement Protocol (SAP)[35] and IP multicast to announce ongoing sessions, such as seminars and TV broadcasts, that were open to the public.

The requirements for these protocols are quite distinct, in terms of the number of participants in the session and the coupling between those participants. Some sessions are very loosely coupled, with only limited membership control and knowledge of the participants. Others are closely managed, with explicit permission required to join, talk, listen, and watch.

These different requirements have led to very different protocols being designed for each scenario, with tremendous ongoing work in this area. RTP deliberately does not include session initiation and control functions, making it suitable for a wide range of applications.

As an application designer, you will have to implement some form of session initiation, call setup, or call control in addition to the media transport provided by RTP.

Session Description

Common to all setup and announcement protocols is the need for a means of describing the session. One commonly used protocol in this area is the Session Description Protocol (SDP),[15] although other mechanisms may be used.

Regardless of the format of the session description, certain information is always needed. It is necessary to convey the transport addresses on which the media flows, the format of the media, the RTP payload formats and profile that are to be used, the times when the session is active, and the purpose of the session.

SDP bundles this information into a text file format, which is human-readable and can be easily parsed. In some cases this file is passed directly to an RTP application, giving it enough information to join a session directly. In others, the session description forms a basis for negotiation, part of a call setup protocol, before a participant can enter a tightly controlled conference call.

SDP is discussed in more detail in Chapter 4, RTP Data Transfer Protocol.

Quality of Service

Although RTP is designed to operate over the best-effort service provided by IP, it is sometimes useful to be able to reserve network resources, giving enhanced quality of service to the RTP flows. Once again, this is not a service provided by RTP, and it is necessary to enlist the help of another protocol. At the time of this writing, there is no commonly accepted "best practice" for resource reservation on the Internet. Two standard frameworks exist, Integrated Services and Differentiated Services, with only limited deployment of each.

The Integrated Services framework provides for strict quality-of-service guarantees, through the use of the Resource ReSerVation Protocol (RSVP).[11] Routers are required to partition the available capacity into service classes, and to account for capacity used by

the traffic. Before starting to transmit, a host must signal its requirements to the routers, which permit the reservation to succeed only if sufficient capacity is available in the desired service class. Provided that all the routers respect the service classes and do not overcommit resources, this requirement prevents overloading of the links, providing guaranteed quality of service. The available classes of service include guaranteed service (giving an assured level of bandwidth, a firm end-to-end delay bound, and no congestive packet loss) and controlled load (providing a service equivalent to that of a lightly loaded best-effort network).

The Integrated Services framework and RSVP suffer from the need to make reservations for every flow, and from the difficulty in aggregating these reservations. As a result, scaling RSVP to large numbers of heterogeneous reservations is problematic because of the amount of state that must be kept at the routers, and this constraint has limited its deployment.

The Differentiated Services framework[23,24] takes a somewhat different approach to quality of service. Instead of providing end-to-end resource reservations and strict performance guarantees, it defines several per-hop queuing behaviors, which it selects by setting the type-of-service field in the IP header of each packet. These per-hop behaviors enable a router to prioritize certain types of traffic to give low probability of loss or delay, but because a router cannot control the amount of traffic admitted to the network, there is no absolute guarantee that performance bounds are met. The advantage of the Differentiated Services framework is that it does not require complex signaling, and the state requirements are much smaller than those for RSVP. The disadvantage is that it provides statistical guarantees only.

The combination of the Integrated and Differentiated Services frameworks is powerful, and future networks may combine them. RSVP can be used to signal application requirements to the edge routers, with those routers then mapping these requirements onto Differentiated Services traffic classes. The combination allows the edge routers to reject excessive traffic, improving the guarantees

that can be offered by a Differentiated Services network, while keeping the state required for RSVP out of the core of the network.

Both frameworks have their place, but neither has achieved critical mass at the time of this writing. It is likely, but by no means certain, that future networks will employ some form of quality of service. Until then we are left with the task of making applications perform well in the best-effort network that is currently deployed.

Future Standards Development

With the revision of RTP for draft standard status, there are no known unresolved issues with the protocol specification, and RTP itself is not expected to change in the foreseeable future. This does not mean that the standards work is finished, though. New payload formats are always under development, and work on new profiles will extend RTP to encompass new functionality (for example, the profiles for secure RTP and enhanced feedback).

In the long term, we expect the RTP framework to evolve along with the network itself. Future changes in the network may also affect RTP, and we expect new profiles to be developed to take advantage of any changes. We also expect a continual series of new payload format specifications, to keep up with changes in codec technology and to provide new error resilience schemes.

Finally, we can expect considerable changes in the related protocols for call setup and control, resource reservation, and quality of service. These protocols are newer than RTP, and they are currently undergoing rapid development, implying that changes here will likely be more substantial than changes to RTP, its profile, and payload formats.

Summary

RTP provides a flexible framework for delivery of real-time media, such as audio and video, over IP networks. Its core philosophies—application-level framing and the end-to-end principle—make it well suited to the unique environment of IP networks.

This chapter has provided an overview of the RTP specification, profiles, and payload formats. Related standards cover call setup, control and advertisement, and resource reservation.

The two parts of RTP introduced in this chapter—the data transfer protocol and the control protocol—are covered in detail in the next two chapters.