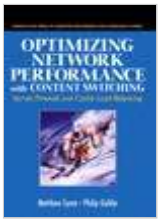
Understanding Application Layer Protocols

By Matthew Syme and Philip Goldie

Mar 5, 2004

This chapter is from the book



Optimizing Network Performance with Content Switching: Server, Firewall and Cache Load Balancing: Server, Firewall, and Cache Load Balancing

Learn More Buy

Real Time Streaming Protocol (*RTSP*)

In the modern Internet, applications are required to deliver value. One of the biggest conundrums in recent years has been the battle to actually make the Internet a viable platform for making money. As we'll see throughout the course of this book, one of the biggest drivers for delivering on the "Gold Rush" promise of Internet technologies is *content.* Making content attractive to end consumers to the point where they are willing to pay is a big challenge and one that has been aided by the delivery of Application layer protocols such as *RTSP*, which enables the delivery of real-time video and audio in variable qualities. The other Application layer protocols we've looked at so far in this chapter work in a request/response manner, whereby the client asks for some piece of content, the content is delivered using *TCP* or *UDP*, and then the client application can display the content to the user. While these mechanisms are suitable for a large number of applications in the Internet, there also exists a requirement to deliver content, be it images, audio, video, or a combination of all three, in real time. Imagine if a user were to try to watch a full-screen video file of a one-hour movie using *HTTP* or *FTP* as the Application layer protocol. The movie file might be several hundred megabytes, if not several gigabytes, in size. Even with modern broadband services deliverable to the home, this type of large file size does not fit well in the "download then play" model we saw previously.

RTSP uses a combination of reliable transmission over *TCP* (used for control) and best-efforts delivery over *UDP* (used for content) to stream content to users. By this, we mean that the file delivery can start and the client-side application can begin displaying the audio and video content before the complete file has arrived. In terms of our one-hour movie example, this means that the client can request a movie file and watch a "live" feed similar to how one would watch a TV. Along with this "on demand" type service, *RTSP* also enables the delivery of live broadcast content that would not be possible with traditional download and play type mechanisms.

The Components of *RTSP* Delivery

During our look at *RTSP*, we'll use the term to describe a number of protocols that work together in delivering content to the user.

RTSP

RTSP is the control protocol for the delivery of multimedia content across *IP* networks. It is based typically on *TCP* for reliable delivery and has a very similar operation and syntax to *HTTP*. *RTSP* is used by the client application to communicate to the server information such as the media file being requested, the type of application the client is using, the mechanism of delivery of the file (unicast or multicast, *UDP* or *TCP*), and other important control information commands such as DESCRIBE, SETUP, and PLAY. The actual multimedia content is not typically delivered over the *RTSP* connection(s), although it can be interleaved if required. *RTSP* is analogous to the remote control of the streaming protocols.

Real Time Transport Protocol (*RTP*)

RTP is the protocol used for the actual transport and delivery of the real-time audio and video data. As the delivery of the actual data for audio and video is typically delay sensitive, the lighter weight *UDP* protocol is used as the Layer 4 delivery mechanism, although *TCP* might also be used in environments that suffer higher packet loss. The *RTP* flow when delivering the content is unidirectional from the server to the client. One interesting part of the *RTP* operation is that the source port used by the server when sending the *UDP* data is *always* even—although it is dynamically assigned. The destination port (i.e., the *UDP* port on which the client is listening) is chosen by the client and communicated over the *RTSP* control connection.

Real Time Control Protocol (*RTCP*)

RTCP is a complimentary protocol to *RTP* and is a bidirectional *UDP*-based mechanism to allow the client to communicate stream-quality information back to the object server. The *RTCP UDP* communication *always* uses the next *UDP* source port up from that used by the *RTP* stream, and consequently is *always* odd. Figure 3-7 shows how the three protocols work together.

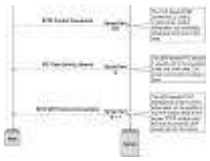


Figure 3-7. The three main application protocols used in real-time streaming.

RTSP Operation

The *RTSP* protocol is very similar in structure and specifically syntax to *HTTP*. Both use the same *URL* structure to describe an object, with *RTSP* using the rtsp:// scheme rather than the http://. *RTSP*, however, introduces a number of additional headers (such as DESCRIBE, SETUP, and PLAY) and also allows data transport out-of-band and over a different protocol, such as *RTP* described earlier. The best way to understand how the components described previously work together to deliver an audio/video stream is to look at an example. The basic steps involved in the process are as follows:

1. The client establishes a *TCP* connection to the servers, typically on *TCP* port 554, the well-known port for *RTSP*.
2. The client will then commence issuing a series of *RTSP* header commands that have a similar format to *HTTP*, each of which is acknowledged by the server. Within these *RTSP* commands, the client will describe to the server details of the session requirements, such as the version of *RTSP* it supports, the transport to be used for the data flow, and any associated *UDP* or *TCP* port information. This information is passed using the DESCRIBE and SETUP headers and is augmented on the server response with a Session ID that the client, and any transitory proxy devices, can use to identify the stream in further exchanges.
3. Once the negotiation of transport parameters has been completed, the client will issue a PLAY command to instruct the server to commence delivery of the *RTP* data stream.
4. Once the client decides to close the stream, a TEARDOWN command is issued along with the Session ID instructing the server to cease the *RTP* delivery associated with that ID.

Example—*RTSP* with *UDP*-Based *RTP* Delivery

Let's consider an example interaction where the client and server will use a combination of *TCP*-based *RTSP* and *UDP*-based *RTP* and *RTCP* to deliver and view a video stream. In the first step, the client will establish a *TCP* connection to port 554 on the server and issue an OPTIONS command showing the protocol version used for the session. The server acknowledges this with a 200 OK message, similar to *HTTP*.

```
C->S  OPTIONS rtsp://video.foocorp.com:554 RTSP/1.0
Cseq: 1

S->C  RTSP/1.0 200 OK
      Cseq: 1
```

Next, the client issues a DESCRIBE command that indicates to the server the *URL* of the media file being requested. The server responds with another 200 OK acknowledgment and includes a full media description of the content, which is presented in either Session Description Protocol (*SDP*) or Multimedia and Hypermedia Experts Group (*MHEG*) format.

```
C->S  DESCRIBE rtsp://video.foocorp.com:554/streams/example.rm RTSP/1.0
      Cseq:2

S->C  RTSP/1.0 200 OK
      Cseq: 2
      Content-Type: application/sdp
Content-Length: 210
      <SDP Data...>
```

In the third stage of the *RTSP* negotiation, the client issues a SETUP command that identifies to the server the transport mechanisms, in order of preference, the client wants to use. We won't list all of the available transport options here (the RFC obviously contains an exhaustive list), but we'll see the client request *RTP* over *UDP* on ports 5067 and 5068 for the data transport. The server responds with confirmation of the *RTP* over *UDP* transport mechanism and the client-side ports and includes the unique Session ID and server port information.

```
C->S  SETUP rtsp://video.foocorp.com:554/streams/example.rm RTSP/1.0
      Cseq: 3
      Transport: rtp/udp;unicast;client_port=5067-5068

S->C  RTSP/1.0 200 OK
      Cseq: 3
      Session: 12345678
      Transport: rtp/udp;client_port=5067-5068;server_port=6023-6024
```

Finally, the client is now ready to commence the receipt of the data stream and issues a PLAY command. This simply contains the *URL* and Session ID previously provided by the server. The server acknowledges this PLAY command, and the *RTP* stream from the server to client will begin.

```
C->S  PLAY rtsp://video.foocorp.com:554/streams/example.rm RTSP/1.0
      Cseq: 4
      Session: 12345678

S->C  RTSP/1.0 200 OK
      Cseq: 4
```

Once the client decides that the stream can be stopped, a TEARDOWN command is issued over the *RTSP* connection referenced only by the Session ID. The server again acknowledges this and the *RTP* delivery will cease.

```
C->S  TEARDOWN rtsp://video.foocorp.com:554/streams/example.rm RTSP/1.0
      Cseq: 5
      Session: 12345678

S->C  RTSP/1.0 200 OK
      Cseq: 5
```

Figure 3-8 shows this example in a simplified graphic form.

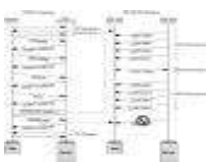


Figure 3-8. An example of RTSP in action with the video and audio data being delivered over a separate UDP-based RTP stream.

Other Options for Data Delivery

In certain scenarios, the best-effort, dynamic port methods of *UDP*-based *RTP*, as described previously, are not suitable. Some environments might consider the allocation of dynamic source and destination *UDP* ports through firewalls to be something they can live happily without. Moreover, just the nature of the Layer 1 and Layer 2 transport mechanisms underlying the data delivery might not be suited to nonguaranteed *UDP* traffic. In either instance, *RTSP* allows for the negotiation of the *RTP* delivery of the media data to be interleaved into the existing *TCP* connection.

When interleaving, the client-to-server SETUP command has the following format:

```
C->S  SETUP rtsp://video.foocorp.com:554/streams/example.rm RTSP/1.0
      Cseq: 3
      Transport: rtp/avp/tcp; interleaved=0-1
```

The changeover in the preceding example is in the transport description. First, the transport mechanisms have changed to show that the *RTP* delivery must be over *TCP* rather than *UDP*. Second, the addition of the `interleaved` option shows that the *RTP* data should be interleaved and use channel identifiers 0 and 1—0 will be used for the *RTP* data and 1 will be used for the *RTCP* messages. To confirm the transport setup, the server will respond with confirmation and a Session ID as before:

```
S->C  RTSP/1.0 200 OK
      Cseq: 3
      Session: 12345678
      Transport: rtp/ avp/tcp; interleaved=0-1
```

The *RTP* and *RTCP* data can now be transmitted over the existing *RTSP TCP* connection with the server using the 0 and 1 identifiers to represent the relevant channel.

One further delivery option for *RTP* and *RTCP* under *RTSP* is to wrap the delivery of all media streaming components inside traditional *HTTP* frame formats. This removes most barriers presented when using streaming media through firewalled environments, as even the most stringent administrator will typically allow *HTTP* traffic to traverse perimeter security. While *HTTP* and *RTSP* interleaved delivery of the streamed media data will make the content available to the widest possible audience, when you consider the overhead of wrapping all *RTP* data inside either an existing *TCP* stream or, worse still, inside *HTTP*, it is the least efficient method for delivery. To enable the streaming media client browser to cope with the different options described previously, most offer the client users the ability to configure their preferred delivery mechanism or mechanisms, and the timeout that should be imposed in failing between them. What you will see from a client perspective is that the client application will first request that the stream be delivered using *RTP* in *UDP*, and if the stream does not arrive within *x* seconds (as it is potentially being blocked by an intermediate firewall), it will fail back to using *RTP* interleaved in the existing *RTSP* connection.

RTSP and *RTP*—Further Reading

For further information on the *RTSP* and *RTP* protocols, RFCs 2326 and 1889, respectively, are a good source.