Using RTSP with Firewalls, Proxies, and Other Intermediary Network Devices

Version 2.0/rev.2

Introduction

This white paper provides information to developers and implementers about the incorporation of Real Time Streaming Protocol (RTSP) into firewalls, proxies, and other intermediary devices. The paper highlights the specific protocol messages that must be examined to reallocate or deallocate ports, and the pertinent RTSP message syntax.

This paper is organized in the following sections:

- RTSP Data Packet Formats
- RTSP Message Formats
- Protocol Semantics
- RTSP and SMIL
- Tips and Tricks
- Conclusion
- Appendixes

Terms used in this document (additional references provided in Appendix B):

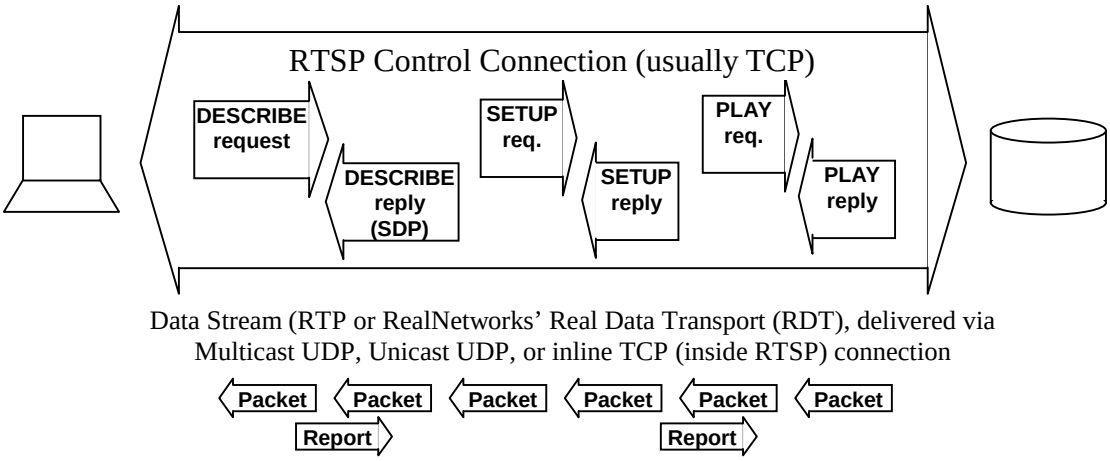
- **Real Time Streaming Protocol (RTSP)** An application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video, using the Transmission Control Protocol (TCP) or the User Data Protocol (UDP). All known RTSP servers to date are TCP-based, though the specification has provisions for a UDP-based version.

- **Session Description Protocol (SDP)** A media description format intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation.

- **Real Time Transport Protocol (RTP)** A UDP packet format and set of conventions which provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services.

- **Hypertext Transfer Protocol (HTTP)** An application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods.

An example of how RTSP works with other protocols is shown below.



The RTSP protocol is intentionally similar in syntax and operation to HTTP/1.1. HTTP extension mechanisms can in most cases also be added to RTSP. However, RTSP differs in a number of important aspects from HTTP:

- RTSP introduces a number of new methods and has a different protocol identifier.
- An RTSP server needs to maintain state by default in almost all cases, as opposed to the stateless nature of HTTP. An RTSP proxy does not necessarily need to keep state information.
- Both an RTSP server and client can issue requests.
- With RTSP, data can be carried out-of-band by a different protocol (e.g., RDT, RTP).
- RTSP is defined to use ISO 10646 (UTF-8) rather than ISO 8859-1, consistent with current HTML internationalization efforts.
- The RTSP Request-URI always contains the absolute URI. Due to backward compatibility with a historical blunder, HTTP/1.1 carries only the absolute path in the request and puts the host name in a separate header field.

RTSP Data Packet Formats

In general, an RTSP server can use any type of packet format for sending media data to an RTSP client. The packet format used to send media data is of little interest to a proxy since all necessary network port information is contained in the RTSP messages. At this time, the G2 RealServer uses one of two packet formats for sending media data to an RTSP client:

- Standard Real Time Transport Protocol, RTP
- RealNetworks' Real Data Transport, RDT

The data packets can be transported using multicast UDP, unicast UDP or inline TCP (data interleaved with the RTSP control stream).

Standard RTP

The RTSP client sets up three network channels with the RTSP server when media data is delivered using the RTP over UDP, as shown in Figure 1. A full-duplex TCP connection is used for control and negotiation. A simplex UDP channel is used for media data delivery using the RTP packet format. A full-duplex UDP channel called RTCP is used to provide synchronization information to the client and packet loss information to the server.

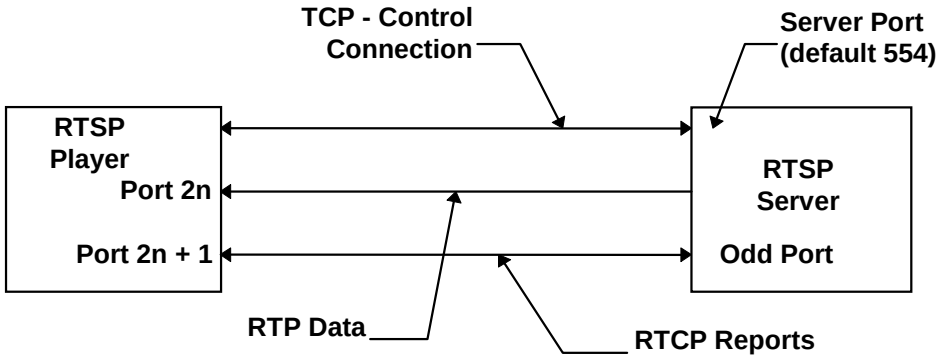


Figure 1 - RTSP Client / Server Communications : Standard UDP Mode

Note that the RTP port must be an even numbered port, while the RTCP port **must** be the next consecutive port. Therefore, the RTCP port is always an odd number. The standard TCP connection port for a RTSP server is 554.

RealNetworks' RDT

When data is delivered using RDT, the RTSP client sets up three network connections with the RTSP Server as shown in Figure 2. A full-duplex TCP connection is used for control and negotiation. A simplex UDP path from the RTSP server to the client is used for media data delivery. A second simplex UDP path from the client to the server is used to request that the server resend lost UDP media data packets.

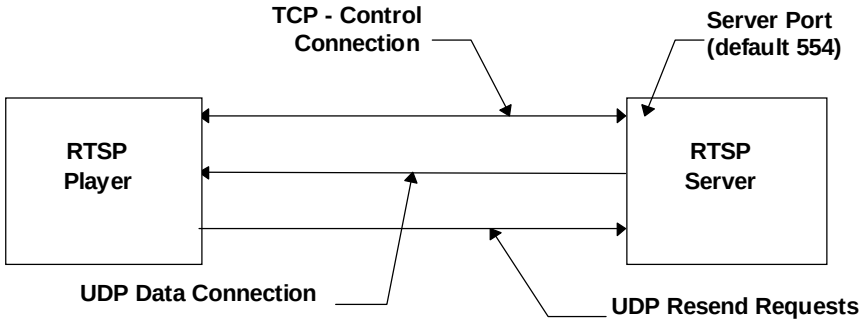


Figure 2 - RTSP Client / Server Communications: RealNetworks RDT Mode

TCP-Only

Media data may be made into packets using RTP or RDT over TCP. In this scenario , a single full-duplex TCP connection is used for both control and for media data delivery from the RTSP server to the client, as shown in Figure 3. The data stream is interleaved with the RTSP control stream. See section “Data Streamed Over TCP” for information.

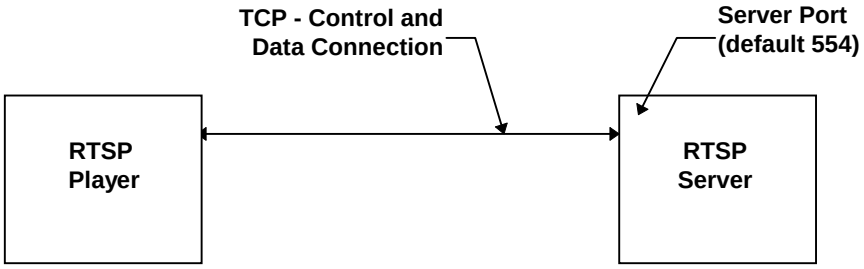


Figure 3 -RTSP Client / Server Communications: TCP-Only Mode

RTSP Message Formats

The RTSP message formats share a similar syntax to HTTP messages. The general syntax for an RTSP method is:

```
{method name} {URL} {protocol version}CRLF
{parameters}
```

An example of an RTSP request follows:

```
DESCRIBE http://foo.com/bar.rm RTSP/1.0
CSeq: 312
Accept: application/sdp, application/mheg
```

This is a request for an RTSP server to send a description of the media content, http://foo.com/bar.rm, using either Session Description Protocol (SDP) or Multimedia and Hypermedia Experts Group (MHEG) formats.

An RTSP message may also contain a body. The general syntax for a method with a body is:

```
{method name} {URL} {protocol version}CRLF
{MIME header field}CRLF
...
{MIME header field}CRLF
CRLF
{optional body, depending on the presence of a "Content-length"}
```

The following example contains a description of the media referenced by the request URL, rtsp://foo.bar.com/bar.rm, using the SDP format.

```
ANNOUNCE rtsp://foo.bar.com/bar.rm RTSP/1.0
CSeq: 312
Date: 9 Sep 1998 13:00:00 GMT
Session: 45991232
Content-Type: application/sdp
Content-Length: 332
```



```
v=0
o=efutz 1928384477 1928386879 IN IP4 127.15.32.2
s=A Short Story
i=A short narrative depicting the early days of the Internet
u=http://www.yo.com/efutz/sdp.04.ps
e=efutz@yo.com (Elmer Futz)
c=IN IP4 225.2.14.10/127
t=3928384899 3928493389
a=recvonly
m=audio 8756 RTP/AVP 0
m=video 3487 RTP/AVP 31
```

Each RTSP request is followed by a response message. The general syntax for a response message is:

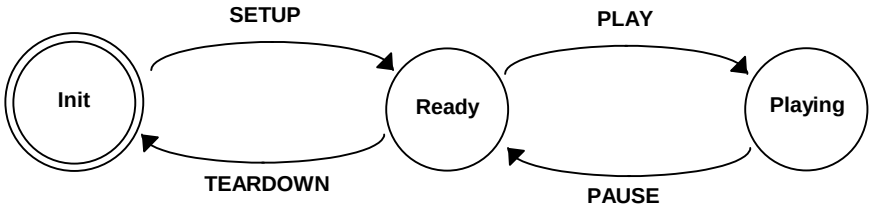
```
{protocol version} {status code} {reason-phrase}CRLF
{parameters}
```

A typical response message may look like the following:

```
RTSP/1.0 200 OK
CSeq: 312
```

Protocol Semantics

Unlike HTTP, RTSP does have a certain amount of state associated with it. Here is a simple state machine that describes the minimal state in RTSP:



There are other methods (DESCRIBE, SET_PARAMETER, OPTIONS) that have no effect on state. It is not necessary for proxies to keep track of an RTSP session's state.

There are only two RTSP methods that provide a proxy with all of the port information necessary to open up, map and close ports. These methods are SETUP and TEARDOWN. The client sends the SETUP method to the server. The data in the SETUP method specifies the set of delivery transports the client can handle. Note that a client is capable of supporting multiple transports for data delivery (e.g., RTP and RDT over UDP--unicast or multicast--or TCP). The server selects one transport mechanism from the list of transports supplied by the client and returns this information in the setup response message.

The client sends the TEARDOWN method to the server to stop media data delivery. When the TEARDOWN method is received, the ports associated with this session can now be deallocated.

Appendix A contains sample RTSP sessions demonstrating SETUP and TEARDOWN transactions.

The Setup Method

Firewall systems and other intermediate network devices can use the information exchanged in the setup transaction to retrieve specific data such as the transport mechanism and port numbers for translation.

Different types of network intermediaries need different types of translation:

- Application-level proxies need to translate both client and server ports
- Network address translators usually only need to remap client ports
- Packet-filters and stateful-inspection filters do not need to do any translation at all

The general syntax of the setup method is:

```
SETUP {URL} {protocol version}CRLF
{sequence number}
{Transport:} {options}
```

The transport header specifies the transport options acceptable to the client for data transmission. The transport options are separated by commas and listed in order of preference, and consist of a transport specifier and associated parameter settings. The parameter settings are separated by semicolons. See RFC 2326, section 12.39 for details.

The syntax for the transport specifier is:

```
transport/profile/lower-transport
```

The default for the lower-transport is specific to the profile:

- For standard RTP, the transport specifier is: rtp/avp (audio/video profile; See the companion Internet Draft, draft-ietf-avt-profile).
- The lower transport can be TCP or UDP. UDP is the default lower-transport if none is given.

Thus, rtp/avp, rtp/avp/udp, and rtp/avp/tcp are all valid transport specifiers.

- For RDT, the transport specifier is: x-real-rdt and the deprecated transport specifier: x-pn-tng.
- The lower transport can be TCP, UDP or mcast for multicast. UDP is the default lower-transport if none is given.

Thus, x-real-rdt, x-real-rdt/udp, x-real-rdt/tcp, x-pn-tng/udp, and x-real-rdt/mcast are all valid transport specifiers.

A transport header may contain transport options that are qualified using different parameters. The following example lists three transport options:

- The first option is RTP using multicasted UDP
- The second option is RTP using UDP

- The third option is RDT over UDP

```
Transport: rtp/avp;multicast;ttl=127;mode=play,
rtp/avp;unicast;client_port=6970-6971;mode=play,x-real-
rdt/udp;client_port=6970;mode=play
```

The following example shows a complete client server SETUP message.

```
C->S  SETUP rtsp://192.168.60.27:6060/sony3/start.smi/streamid=0 RTSP/1.0
      CSeq: 3
      Transport: rtp/avp;unicast;client_port=6970-6971;mode=play,x-real-
rdt/udp;client_port=6970;mode=play

S->C  RTSP/1.0 200 OK
      CSeq: 3
      Session: 968367008-2
      Transport: rtp/avp;client_port=6970-6971;server_port=7970-7971
```

In the transaction above, the client requests to receive data using RTP on client ports 6970 and 6971 or RDT using UDP on client port 6970. The server response message selects RTP as the data delivery transport and supplies the server port numbers of 7970 and 7971. The server response also contains a unique session ID. A proxy should keep track of ports associated with this session ID since the session ID is the only data supplied in the TEARDOWN message.

The proxy should keep a list of all ports that are given in the client server SETUP request. On receipt of the server reply message, the proxy should free all unselected ports. To reiterate, the proxy should assign all ports listed in the SETUP request and then strip out unneeded ports based on the server's response message.

A proxy is permitted to strip out transports in the client SETUP request that it cannot or will not handle. However, the risk in doing this is that the server may not accept the remaining transports, where it might have accepted one of the transports that was stripped out by the proxy.

Note that the RTSP SETUP transport header connection information takes precedence over the SDP connection field data. SDP is one format that can be used to describe the presentation requested. The SDP contains a "c=" connection field (see section 6 of SDP RFC 2327) which may contain redundant information found in the RTSP SETUP transport header. See RTSP spec section C.1.7 for more information. The SDP connection information should not be used.

The Teardown Method

Firewall systems and other intermediate network devices can use the information exchanged in the TEARDOWN method to deallocate ports associated with this session. The client sends this message to the server to stop media data delivery.

The general syntax of the TEARDOWN method is:

```
TEARDOWN {URL} {protocol version} CRLF
{sequence number}
{Session:} {session-id}
```


The following is an example of a TEARDOWN transaction. Note that the session ID is specified in this message. This unique ID should be used to associate this session with the list of allocated ports defined in the SETUP transaction.

```
C->S  TEARDOWN rtsp://192O.168.27:6060/sony3/test.rm RTSP/1.0
      CSeq: 63
      Session: 968367008-2

S->C  RTSP/1.0 200 OK
      CSeq: 63
```

RTSP, SMIL and Application Level Proxies

The RealSystem G2 provides support for Synchronized Multimedia Integration Language (SMIL). SMIL is a layout language that allows easy creation of multimedia presentations consisting of multiple elements of music, voice, images, text, video, and graphics in a common, synchronized timeline. SMIL is a Proposed Specification of the World Wide Web Consortium (W3C).

The following text illustrates a simple SMIL file that contains two media sources, one.rm and two.rm. Media source one.rm is streamed from realserver1.company.com and media source two.rm is streamed from realserver2.company.com.

```
<smil>
  <body>
    <audio src="rtsp://realserver1.company.com/one.rm"/>
    <audio src="rtsp://realserver2.company.com/two.rm"/>
  </body>
</smil>
```

SMIL presentations can add complexity to application-level proxies. The RealPlayer uses a single control channel for all servers referenced in the SMIL file when the RealPlayer is configured to use an application-level proxy. Figure 4 shows the client server connections when the RealPlayer is configured to use a proxy and the SMIL file contains two sources, where each source is retrieved from different RealServers.

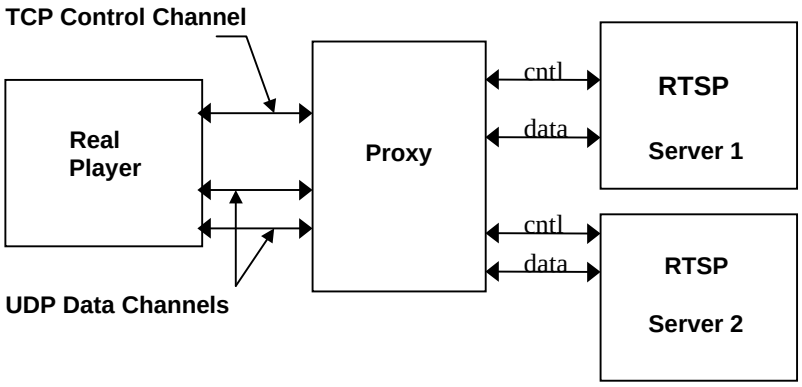


Figure 4 -RTSP Client / Server Communications with Two Servers in SMIL File.

In this scenario, all RTSP messages to each RealServer are transferred from the client to the proxy on a single TCP control channel. The backend of the proxy must create and maintain a control and data connection to each unique RealServer.

When the RealPlayer is in TCP-only mode, the client/server communication is as shown in Figure 5.

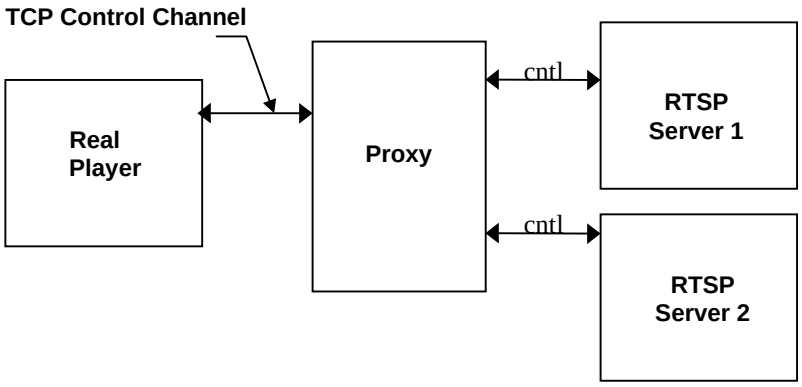


Figure 5 -RTSP Client / Server Communications with Two Servers in SMIL File Using TCP Only.

In this scenario, all RTSP control messages and media data for both servers are transmitted over a single control channel from the client to the proxy. The data is interleaved into the control channel. The proxy backend creates a control stream to each server. See the section “Data Streamed Over TCP” for more details on the packaging of the data.

An application level proxy should be aware of three potential conditions that may arise between an RTSP client and server, when a single control channel is used for multiple server connections, as may be the case when using SMIL.

- Cseq numbers will not increase monotonically across all servers. The client creates the RTSP Cseq number for messages originating from the client. When the client is using a single control channel for communication to multiple servers, the client may generate Cseq numbers that increase monotonically across all servers rather than per server.

- Cseq numbers will not increase monitonically from servers. The server creates RTSP Cseq numbers for messages originating from each server. The proxy should remap these numbers to numbers that are unique and increase monotonically.

- It is possible for RTSP servers to generate identical session ids. If this situation occurs, the client will not be able to determine what server sent a particular message since the session id is used to uniquely identify client server messages.

- In TCP-only mode, the interleaved data contains channel numbers to identify the RTP or RDT channels. The RTSP servers may generate the same channel

numbers for their respective data streams. These numbers must be unique to be comprehensible to the client. See the section on “Data Streamed Over TCP” for more detailed information.

The table below depicts the first three scenarios listed above. The Cseq numbers between the client and server1 are 1, 3, and 5 for the first three RTSP messages sent from the client to the server. The sequence numbers should be monotonically increasing. Therefore, the proxy must remap the Cseq numbers to be 1, 2, and 3 for client messages to server1 and remap the Cseq numbers of 2, 4, and 6 to 1, 2, and 3 for client messages to server2.

RTSP Message	C->S1	S1->C	C->S2	S2->C
OPTIONS	Cseq: 1		Cseq: 2	
DESCRIBE	Cseq: 3		Cseq: 4	
SETUP	Cseq: 5	Session: 345	Cseq: 6	Session: 345
SET PARAMETER		Cseq: 1		Cseq: 1

The Cseq numbers for messages initiated by the server (e.g., SET PARAMETER) may be identical. The proxy should remap these numbers to be unique and increasing before sending the message to the client.

Finally, in the table above, the session ids returned from the servers are identical. The proxy should remap the session ids to unique identifiers before sending them to the client.

Tips and Tricks

RTSP was designed to be as simple as possible. However, the task of implementing an RTSP client or server can be very complex. This section focuses on RTSP issues and nuances that are useful to designers of RTSP firewalls, proxies and other intermediary devices.

Packet Fragmentation

Make no assumptions about RTSP message boundaries relating to TCP packet boundaries. One TCP packet may contain one, part of one, or more than one RTSP message. A proxy or other device must assemble RTSP messages from the TCP packets.

Stream Ports

Note that RTP requires that the port number specified must be an even number and that the RTCP port must be the subsequent number, which is an odd number. RTP and RTCP **must** be on contiguous ports. RDT does not have this restriction.

The port ranges specified in the transport header (RFC 2326, section 12.39) are inclusive.

Data Streamed Over TCP

Media data and control information can be streamed over a full-duplex TCP connection using RTSP. RTSP contains syntax for interleaving the RTSP control stream with the data stream. This is called embedded (interleaved) binary data. Interleaved binary data is only used when RTSP is carried over TCP. See RFC, section 10.12 for details.

The RTP or RDT packets are encapsulated by an ASCII dollar sign (24 hexidecimal), followed by a one-byte binary channel identifier, followed by the length of the encapsulated binary data as a binary, two-byte integer in network byte order, followed by the upper-layer protocol layer and stream data. Since one TCP packet may contain one, part of one, or more than one RTSP message or data packet, a proxy or other device must assemble RTSP messages and data packets on message or packet boundaries. These repackaged messages and data packets should be forwarded to the destination system.

The following paragraphs illustrate a client server session using interleaved binary data.

The client posts a setup method to the server as follows:

```
SETUP rtsp://foo.com/bar.rm RTSP/1.0
CSeq: 2
Transport: RTP/AVP/TCP; interleaved=0-1
```

The message above states that the transport type requested is RTP over TCP and includes the interleaved parameter. The interleaved parameter's argument specifies the channel number used in the $ statement. Note that two interleave channels are specified: 0 and 1. The channel 0 is used to transmit the data stream and channel 1 is used to transmit RTCP messages.

The server responds with a session ID.

```
RTSP/1.0 200 OK
CSeq: 2
Date: 09 Sep 1998 13:00:01 GMT
Transport: RTP/AVP/TCP;interleaved=0-1
Session: 12345678
```

The client starts the session by sending a PLAY method to the server.

```
PLAY rtsp://foo.com/bar.rm RTSP/1.0
CSeq: 3
Session: 12345678
```

The server responds as follows:

```
RTSP/2.0 200 OK
CSeq: 3
Session: 12345678
Date: 09 Sep 1998 13:01:00 GMT
RTP-Info: url=rtsp://foo.com/bar.rm;seq=245366;rtptime=782343219
```

The server now begins to send the interleaved binary data as follows:

```
$\000{2 byte length}{"length" bytes including RTP header}
```

This above statement represents an RTP packet.

RTCP packets are transmitted by specifying the channel identifier as 1.

```
$\001{2 byte length}{“length” bytes including RTCP packet}
```

Client UDP Port Range

The RealPlayer client uses UDP ports 6970-7170 as the default port range for data packet delivery. These ranges are configurable in the RealPlayer client’s Preferences dialog. The user has the option to specify a specific UDP port or a range of ports.

Conclusion

Adding RTSP support to your system requires examining the following aspects of this protocol:

- Extracting transport information from the SETUP transaction between client and server for port allocation
- Collecting TEARDOWN messages to deallocate ports
- Handling TCP-only connections with care by paying special attention to packet fragmentation

Finally, viewing the actual packets transferred between the client and server in various client/server scenarios (e.g., RTP, RDT, TCP only, multicast) is highly recommended. Knowledge of the various RTSP messages will ensure that the implementation of RTSP support is robust.

Appendix A: Sample RTSP Sessions

Only the relevant fields, of interest to proxy makers, have been included in this sample interaction. Other key fields have been omitted or replaced with italicized paraphrasing.

Sample 1: Single Stream

```
C->S  OPTIONS rtsp://192.168.60.27:6060 RTSP/1.0
      CSeq: 1

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

This is the first message in the session that identifies the protocol version and of the session.

```
C->S  DESCRIBE rtsp://192.168.60.27:6060/sony3/first.rm RTSP/1.0
      CSeq: 2

S->C  RTSP/1.0 200 OK
      CSeq: 2
      Content-length: 197

      {197 bytes of SDP}
```

The DESCRIBE transaction constitutes the media initialization phase of an RTSP session. The client is requesting a description of the media content associated with the first.rm RealMedia file.

```
C->S  SETUP rtsp://192.168.60.27:6060/sony3/first.rm RTSP/1.0
      CSeq: 3
      Transport: x-real-rdt/udp;client_port=6970;mode=play,rtp/avp;
          unicast;client_port=6970-6971;mode=play

S->C  RTSP/1.0 200 OK
      CSeq: 3
      Session: 968367008-2
      Transport: x-real-rdt/udp;client_port=6970;server_port=6970
```

The SETUP transaction specifies the transport mechanisms to use when delivering the requested media.

```
C->S  PLAY rtsp://192.168.60.27:6060/sony3/first.rm RTSP/1.0
      CSeq: 4
      Session: 968367008-2
```

The client asks the server to begin sending the contents of the first.rm RealMedia file using the transport mechanism specified in the SETUP response message.

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

Server response to the PLAY message.

```
C->S  TEARDOWN rtsp://192.168.60.27:6060/sony3/first.rm RTSP/1.0
      CSeq: 5
      Session: 968367008-2
```

The client requests that the server stop the delivery of data associated with this session.


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

Sample 2: Multiple Streams

```
C->S  OPTIONS rtsp://192.168.60.27:6060 RTSP/1.0
      CSeq: 1

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

Begin session with the OPTIONS method.

```
C->S  DESCRIBE rtsp://192.168.60.27:6060/sony3/multi.smi RTSP/1.0
      CSeq: 2

S->C  RTSP/1.0 200 OK
      CSeq: 2
      Content-length: 197

      {197 bytes of SDP}
```

The client requests a description of the media content associated with the multi.smi SMIL file. The SMIL stream references a RealText and RealVideo stream.

```
C->S  SETUP rtsp://192.168.60.27:6060/sony3/multi.smi RTSP/1.0
      CSeq: 3
      Transport: x-real-rdt/udp;client_port=6970;mode=play,rtp/avp;
          unicast;client_port=6970-6971;mode=play

S->C  RTSP/1.0 200 OK
      CSeq: 3
      Session: 968367008-2
      Transport: x-real-rdt/udp;client_port=6970;server_port=6970
```

The server selects the transport from the SETUP message.

```
C->S  PLAY rtsp://192.168.60.27:6060/sony3/multi.smi RTSP/1.0
      CSeq: 4
      Session: 968367008-2
```

The client asks the server to begin sending the contents of the multi.smi SMIL file using the transport mechanism specified in the SETUP response message.

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

C->S  DESCRIBE rtsp://192.168.60.27:6060/sony3/text1.rtx RTSP/1.0
      CSeq: 5

C->S  DESCRIBE rtsp://192.168.60.27:6060/sony3/video.rm RTSP/1.0
      CSeq: 6
```

The above DESCRIBE messages represent the media streams referenced in the SMIL file.

```
S->C  RTSP/1.0 200 OK
      CSeq: 5
      Content-length: 197

      {197 bytes of SDP}

S->C  RTSP/1.0 200 OK
      CSeq: 6
      Content-length: 197
```



```
        {197 bytes of SDP}
```

The server responds with SDP descriptions of each media stream requested.

```
C->S  SETUP rtsp://192.168.60.27:6060/sony3/text1.rtx RTSP/1.0
      CSeq: 7
      Transport: x-real-rdt/udp;client_port=6972;mode=play,rtp/avp;
           unicast;client_port=6972-6973;mode=play

C->S  SETUP rtsp://192.168.60.27:6060/sony3/video.rm RTSP/1.0
      CSeq: 8
      Transport: x-real-rdt/udp;client_port=6974;mode=play,rtp/avp;
           unicast;client_port=6974-6975;mode=play
```

The client now negotiates the transport mechanism for each media stream, text1.rtx and video.rm, with the server.

```
S->C  RTSP/1.0 200 OK
      CSeq: 7
      Session: 968367009-3
      Transport: x-real-rdt/udp;client_port=6972;server_port=6972

S->C  RTSP/1.0 200 OK
      CSeq: 8
      Session: 968367010-4
      Transport: x-real-rdt/udp;client_port=6974;server_port=6974


C->S  PLAY rtsp://192.168.60.27:6060/sony3/text1.rtx RTSP/1.0
      CSeq: 9
      Session: 968367009-3
```

The client now requests the server to begin sending the data associated with this session. The following set of messages from the client to the server are requests for the data associated with the specified URL.

```
C->S  PLAY rtsp://192.168.60.27:6060/sony3/video.rm RTSP/1.0
      CSeq: 10
      Session: 968367010-4

S->C  RTSP/1.0 200 OK
      CSeq: 9

S->C  RTSP/1.0 200 OK
      CSeq: 10


C->S  TEARDOWN rtsp://192.168.60.27:6060/sony3/multi.smi RTSP/1.0
      CSeq: 11
      Session: 968367009-3

C->S  TEARDOWN rtsp://192.168.60.27:6060/sony3/multi.smi RTSP/1.0
      CSeq: 12
      Session: 968367010-4

C->S  TEARDOWN rtsp://192.168.60.27:6060/sony3/multi.smi RTSP/1.0
      CSeq: 13
      Session: 968367008-2
```

The client requests that the server stop the delivery of data associated with each of the sessions.


```
S->C  RTSP/1.0 200 OK
      CSeq: 11

S->C  RTSP/1.0 200 OK
      CSeq: 12

S->C  RTSP/1.0 200 OK
      CSeq: 13
```

Appendix B: References

The following are the standards referenced in this document:

Real Time Streaming Protocol (RTSP; RFC 2326)
`ftp://ftp.isi.edu/in-notes/rfc2326.txt`

This specification references the HTTP spec by using the following notation: [H3.2.1]. This example refers to section 3.2.1 in the HTTP RFC.

Session Description Protocol (SDP; RFC 2327)
`ftp://ftp.isi.edu/in-notes/rfc2327.txt`

Real Time Transport Protocol (RTP; RFC 1889)
`ftp://ftp.isi.edu/in-notes/rfc1889.txt`

Hypertext Transfer Protocol (HTTP; RFC 2068)
`ftp://ftp.isi.edu/in-notes/rfc2068.txt`

Additional information on RTSP can be found on our website:
`http://www.real.com/technology/rtsp/index.html`

For more information on SMIL, see the W3C web site:

http://www.w3.org/SMIL

http://www.real.com/technology/smil/specs.html

