                 Real Time Streaming Protocol (RTSP)

Status of this Memo

Copyright Notice

Abstract

   The Real Time Streaming Protocol, or RTSP, is 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. Sources of data can include both live data feeds and stored
   clips. This protocol is intended to control multiple data delivery
   sessions, provide a means for choosing delivery channels such as UDP,
   multicast UDP and TCP, and provide a means for choosing delivery
   mechanisms based upon RTP (RFC 1889).

Table of Contents

1 Introduction

1.1 Purpose

   The Real-Time Streaming Protocol (RTSP) establishes and controls
   either a single or several time-synchronized streams of continuous
   media such as audio and video. It does not typically deliver the
   continuous streams itself, although interleaving of the continuous
   media stream with the control stream is possible (see Section 10.12).
   In other words, RTSP acts as a "network remote control" for
   multimedia servers.

   The set of streams to be controlled is defined by a presentation
   description. This memorandum does not define a format for a
   presentation description.

   There is no notion of an RTSP connection; instead, a server maintains
   a session labeled by an identifier. An RTSP session is in no way tied
   to a transport-level connection such as a TCP connection. During an
   RTSP session, an RTSP client may open and close many reliable
   transport connections to the server to issue RTSP requests.
   Alternatively, it may use a connectionless transport protocol such as
   UDP.

   The streams controlled by RTSP may use RTP [1], but the operation of
   RTSP does not depend on the transport mechanism used to carry
   continuous media. The protocol is intentionally similar in syntax
   and operation to HTTP/1.1 [2] so that extension mechanisms to HTTP
   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.
     * Both an RTSP server and client can issue requests.
     * Data is carried out-of-band by a different protocol. (There is an
       exception to this.)
     * RTSP is defined to use ISO 10646 (UTF-8) rather than ISO 8859-1,
       consistent with current HTML internationalization efforts [3].
     * The Request-URI always contains the absolute URI. Because of
       backward compatibility with a historical blunder, HTTP/1.1 [2]
       carries only the absolute path in the request and puts the host
       name in a separate header field.

   This makes "virtual hosting" easier, where a single host with one
   IP address hosts several document trees.

The protocol supports the following operations:

Retrieval of media from media server:
     The client can request a presentation description via HTTP or
     some other method. If the presentation is being multicast, the
     presentation description contains the multicast addresses and
     ports to be used for the continuous media. If the presentation
     is to be sent only to the client via unicast, the client
     provides the destination for security reasons.

Invitation of a media server to a conference:
     A media server can be "invited" to join an existing
     conference, either to play back media into the presentation or
     to record all or a subset of the media in a presentation. This
     mode is useful for distributed teaching applications. Several
     parties in the conference may take turns "pushing the remote
     control buttons."

Addition of media to an existing presentation:
     Particularly for live presentations, it is useful if the
     server can tell the client about additional media becoming
     available.

RTSP requests may be handled by proxies, tunnels and caches as in
HTTP/1.1 [2].

1.2 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in RFC 2119 [4].

1.3 Terminology

Some of the terminology has been adopted from HTTP/1.1 [2]. Terms not
listed here are defined as in HTTP/1.1.

Aggregate control:
     The control of the multiple streams using a single timeline by
     the server. For audio/video feeds, this means that the client
     may issue a single play or pause message to control both the
     audio and video feeds.

Conference:
     a multiparty, multimedia presentation, where "multi" implies
     greater than or equal to one.

Client:
        The client requests continuous media data from the media
        server.

Connection:
        A transport layer virtual circuit established between two
        programs for the purpose of communication.

Container file:
        A file which may contain multiple media streams which often
        comprise a presentation when played together. RTSP servers may
        offer aggregate control on these files, though the concept of
        a container file is not embedded in the protocol.

Continuous media:
        Data where there is a timing relationship between source and
        sink; that is, the sink must reproduce the timing relationship
        that existed at the source. The most common examples of
        continuous media are audio and motion video. Continuous media
        can be real-time (interactive), where there is a "tight"
        timing relationship between source and sink, or streaming
        (playback), where the relationship is less strict.

Entity:
        The information transferred as the payload of a request or
        response. An entity consists of metainformation in the form of
        entity-header fields and content in the form of an entity-
        body, as described in Section 8.

Media initialization:
        Datatype/codec specific initialization. This includes such
        things as clockrates, color tables, etc. Any transport-
        independent information which is required by a client for
        playback of a media stream occurs in the media initialization
        phase of stream setup.

Media parameter:
        Parameter specific to a media type that may be changed before
        or during stream playback.

Media server:
        The server providing playback or recording services for one or
        more media streams. Different media streams within a
        presentation may originate from different media servers. A
        media server may reside on the same or a different host as the
        web server the presentation is invoked from.

   Media server indirection:
          Redirection of a media client to a different media server.

   (Media) stream:
          A single media instance, e.g., an audio stream or a video
          stream as well as a single whiteboard or shared application
          group. When using RTP, a stream consists of all RTP and RTCP
          packets created by a source within an RTP session. This is
          equivalent to the definition of a DSM-CC stream([5]).

   Message:
          The basic unit of RTSP communication, consisting of a
          structured sequence of octets matching the syntax defined in
          Section 15 and transmitted via a connection or a
          connectionless protocol.

   Participant:
          Member of a conference. A participant may be a machine, e.g.,
          a media record or playback server.

   Presentation:
          A set of one or more streams presented to the client as a
          complete media feed, using a presentation description as
          defined below. In most cases in the RTSP context, this implies
          aggregate control of those streams, but does not have to.

   Presentation description:
          A presentation description contains information about one or
          more media streams within a presentation, such as the set of
          encodings, network addresses and information about the
          content. Other IETF protocols such as SDP (RFC 2327 [6]) use
          the term "session" for a live presentation. The presentation
          description may take several different formats, including but
          not limited to the session description format SDP.

   Response:
          An RTSP response. If an HTTP response is meant, that is
          indicated explicitly.

   Request:
          An RTSP request. If an HTTP request is meant, that is
          indicated explicitly.

   RTSP session:
          A complete RTSP "transaction", e.g., the viewing of a movie.
          A session typically consists of a client setting up a
          transport mechanism for the continuous media stream (SETUP),
          starting the stream with PLAY or RECORD, and closing the

stream with TEARDOWN.

   Transport initialization:
        The negotiation of transport information (e.g., port numbers,
        transport protocols) between the client and the server.

1.4 Protocol Properties

   RTSP has the following properties:

   Extendable:
        New methods and parameters can be easily added to RTSP.

   Easy to parse:
        RTSP can be parsed by standard HTTP or MIME parsers.

   Secure:
        RTSP re-uses web security mechanisms. All HTTP authentication
        mechanisms such as basic (RFC 2068 [2, Section 11.1]) and
        digest authentication (RFC 2069 [8]) are directly applicable.
        One may also reuse transport or network layer security
        mechanisms.

   Transport-independent:
        RTSP may use either an unreliable datagram protocol (UDP) (RFC
        768 [9]), a reliable datagram protocol (RDP, RFC 1151, not
        widely used [10]) or a reliable stream protocol such as TCP
        (RFC 793 [11]) as it implements application-level reliability.

   Multi-server capable:
        Each media stream within a presentation can reside on a
        different server. The client automatically establishes several
        concurrent control sessions with the different media servers.
        Media synchronization is performed at the transport level.

   Control of recording devices:
        The protocol can control both recording and playback devices,
        as well as devices that can alternate between the two modes
        ("VCR").

   Separation of stream control and conference initiation:
        Stream control is divorced from inviting a media server to a
        conference. The only requirement is that the conference
        initiation protocol either provides or can be used to create a
        unique conference identifier. In particular, SIP [12] or H.323
        [13] may be used to invite a server to a conference.

Suitable for professional applications:
        RTSP supports frame-level accuracy through SMPTE time stamps
        to allow remote digital editing.

Presentation description neutral:
        The protocol does not impose a particular presentation
        description or metafile format and can convey the type of
        format to be used. However, the presentation description must
        contain at least one RTSP URI.

Proxy and firewall friendly:
        The protocol should be readily handled by both application and
        transport-layer (SOCKS [14]) firewalls. A firewall may need to
        understand the SETUP method to open a "hole" for the UDP media
        stream.

HTTP-friendly:
        Where sensible, RTSP reuses HTTP concepts, so that the
        existing infrastructure can be reused. This infrastructure
        includes PICS (Platform for Internet Content Selection
        [15,16]) for associating labels with content. However, RTSP
        does not just add methods to HTTP since the controlling
        continuous media requires server state in most cases.

Appropriate server control:
        If a client can start a stream, it must be able to stop a
        stream. Servers should not start streaming to clients in such
        a way that clients cannot stop the stream.

Transport negotiation:
        The client can negotiate the transport method prior to
        actually needing to process a continuous media stream.

Capability negotiation:
        If basic features are disabled, there must be some clean
        mechanism for the client to determine which methods are not
        going to be implemented. This allows clients to present the
        appropriate user interface. For example, if seeking is not
        allowed, the user interface must be able to disallow moving a
        sliding position indicator.

  An earlier requirement in RTSP was multi-client capability.
  However, it was determined that a better approach was to make sure
  that the protocol is easily extensible to the multi-client
  scenario. Stream identifiers can be used by several control
  streams, so that "passing the remote" would be possible. The
  protocol would not address how several clients negotiate access;
  this is left to either a "social protocol" or some other floor

   control mechanism.

1.5 Extending RTSP

   Since not all media servers have the same functionality, media
   servers by necessity will support different sets of requests. For
   example:

     * A server may only be capable of playback thus has no need to
       support the RECORD request.
     * A server may not be capable of seeking (absolute positioning) if
       it is to support live events only.
     * Some servers may not support setting stream parameters and thus
       not support GET_PARAMETER and SET_PARAMETER.

   A server SHOULD implement all header fields described in Section 12.

   It is up to the creators of presentation descriptions not to ask the
   impossible of a server. This situation is similar in HTTP/1.1 [2],
   where the methods described in [H19.6] are not likely to be supported
   across all servers.

   RTSP can be extended in three ways, listed here in order of the
   magnitude of changes supported:

     * Existing methods can be extended with new parameters, as long as
       these parameters can be safely ignored by the recipient. (This is
       equivalent to adding new parameters to an HTML tag.) If the
       client needs negative acknowledgement when a method extension is
       not supported, a tag corresponding to the extension may be added
       in the Require: field (see Section 12.32).
     * New methods can be added. If the recipient of the message does
       not understand the request, it responds with error code 501 (Not
       implemented) and the sender should not attempt to use this method
       again. A client may also use the OPTIONS method to inquire about
       methods supported by the server. The server SHOULD list the
       methods it supports using the Public response header.
     * A new version of the protocol can be defined, allowing almost all
       aspects (except the position of the protocol version number) to
       change.

1.6 Overall Operation

   Each presentation and media stream may be identified by an RTSP URL.
   The overall presentation and the properties of the media the
   presentation is made up of are defined by a presentation description
   file, the format of which is outside the scope of this specification.
   The presentation description file may be obtained by the client using

HTTP or other means such as email and may not necessarily be stored
on the media server.

For the purposes of this specification, a presentation description is
assumed to describe one or more presentations, each of which
maintains a common time axis. For simplicity of exposition and
without loss of generality, it is assumed that the presentation
description contains exactly one such presentation. A presentation
may contain several media streams.

The presentation description file contains a description of the media
streams making up the presentation, including their encodings,
language, and other parameters that enable the client to choose the
most appropriate combination of media. In this presentation
description, each media stream that is individually controllable by
RTSP is identified by an RTSP URL, which points to the media server
handling that particular media stream and names the stream stored on
that server. Several media streams can be located on different
servers; for example, audio and video streams can be split across
servers for load sharing. The description also enumerates which
transport methods the server is capable of.

Besides the media parameters, the network destination address and
port need to be determined. Several modes of operation can be
distinguished:

Unicast:
        The media is transmitted to the source of the RTSP request,
        with the port number chosen by the client. Alternatively, the
        media is transmitted on the same reliable stream as RTSP.

Multicast, server chooses address:
        The media server picks the multicast address and port. This is
        the typical case for a live or near-media-on-demand
        transmission.

Multicast, client chooses address:
        If the server is to participate in an existing multicast
        conference, the multicast address, port and encryption key are
        given by the conference description, established by means
        outside the scope of this specification.

1.7 RTSP States

RTSP controls a stream which may be sent via a separate protocol,
independent of the control channel. For example, RTSP control may
occur on a TCP connection while the data flows via UDP. Thus, data
delivery continues even if no RTSP requests are received by the media

server. Also, during its lifetime, a single media stream may be
controlled by RTSP requests issued sequentially on different TCP
connections. Therefore, the server needs to maintain "session state"
to be able to correlate RTSP requests with a stream. The state
transitions are described in Section A.

Many methods in RTSP do not contribute to state. However, the
following play a central role in defining the allocation and usage of
stream resources on the server: SETUP, PLAY, RECORD, PAUSE, and
TEARDOWN.

SETUP:
    Causes the server to allocate resources for a stream and start
    an RTSP session.

PLAY and RECORD:
    Starts data transmission on a stream allocated via SETUP.

PAUSE:
    Temporarily halts a stream without freeing server resources.

TEARDOWN:
    Frees resources associated with the stream. The RTSP session
    ceases to exist on the server.

    RTSP methods that contribute to state use the Session header
    field (Section 12.37) to identify the RTSP session whose state
    is being manipulated. The server generates session identifiers
    in response to SETUP requests (Section 10.4).

1.8 Relationship with Other Protocols

RTSP has some overlap in functionality with HTTP. It also may
interact with HTTP in that the initial contact with streaming content
is often to be made through a web page. The current protocol
specification aims to allow different hand-off points between a web
server and the media server implementing RTSP. For example, the
presentation description can be retrieved using HTTP or RTSP, which
reduces roundtrips in web-browser-based scenarios, yet also allows
for standalone RTSP servers and clients which do not rely on HTTP at
all.

However, RTSP differs fundamentally from HTTP in that data delivery
takes place out-of-band in a different protocol. HTTP is an
asymmetric protocol where the client issues requests and the server
responds. In RTSP, both the media client and media server can issue
requests. RTSP requests are also not stateless; they may set
parameters and continue to control a media stream long after the

request has been acknowledged.

   Re-using HTTP functionality has advantages in at least two areas,
   namely security and proxies. The requirements are very similar, so
   having the ability to adopt HTTP work on caches, proxies and
   authentication is valuable.

While most real-time media will use RTP as a transport protocol, RTSP
is not tied to RTP.

RTSP assumes the existence of a presentation description format that
can express both static and temporal properties of a presentation
containing several media streams.

2 Notational Conventions

Since many of the definitions and syntax are identical to HTTP/1.1,
this specification only points to the section where they are defined
rather than copying it. For brevity, [HX.Y] is to be taken to refer
to Section X.Y of the current HTTP/1.1 specification (RFC 2068 [2]).

All the mechanisms specified in this document are described in both
prose and an augmented Backus-Naur form (BNF) similar to that used in
[H2.1]. It is described in detail in RFC 2234 [17], with the
difference that this RTSP specification maintains the "1#" notation
for comma-separated lists.

In this memo, we use indented and smaller-type paragraphs to provide
background and motivation. This is intended to give readers who were
not involved with the formulation of the specification an
understanding of why things are the way that they are in RTSP.

3 Protocol Parameters

3.1 RTSP Version

   [H3.1] applies, with HTTP replaced by RTSP.

3.2 RTSP URL

   The "rtsp" and "rtspu" schemes are used to refer to network resources
   via the RTSP protocol. This section defines the scheme-specific
   syntax and semantics for RTSP URLs.

   rtsp_URL =   ("rtsp:" | "rtspu:")
                 "//" host [":" port] [abs_path]
   host   =   <A legal Internet host domain name of IP address
                 (in dotted decimal form), as defined by Section 2.1

```
                    of RFC 1123 \cite{rfc1123}>
    port        =   *DIGIT
```

    abs_path is defined in [H3.2.1].

       Note that fragment and query identifiers do not have a well-defined
       meaning at this time, with the interpretation left to the RTSP
       server.

    The scheme rtsp requires that commands are issued via a reliable
    protocol (within the Internet, TCP), while the scheme rtspu identifies
    an unreliable protocol (within the Internet, UDP).

    If the port is empty or not given, port 554 is assumed. The semantics
    are that the identified resource can be controlled by RTSP at the
    server listening for TCP (scheme "rtsp") connections or UDP (scheme
    "rtspu") packets on that port of host, and the Request-URI for the
    resource is rtsp_URL.

    The use of IP addresses in URLs SHOULD be avoided whenever possible
    (see RFC 1924 [19]).

    A presentation or a stream is identified by a textual media
    identifier, using the character set and escape conventions [H3.2] of
    URLs (RFC 1738 [20]). URLs may refer to a stream or an aggregate of
    streams, i.e., a presentation. Accordingly, requests described in
    Section 10 can apply to either the whole presentation or an individual
    stream within the presentation. Note that some request methods can
    only be applied to streams, not presentations and vice versa.

    For example, the RTSP URL:
       rtsp://media.example.com:554/twister/audiotrack

    identifies the audio stream within the presentation "twister", which
    can be controlled via RTSP requests issued over a TCP connection to
    port 554 of host media.example.com.

    Also, the RTSP URL:
       rtsp://media.example.com:554/twister

    identifies the presentation "twister", which may be composed of
    audio and video streams.

    This does not imply a standard way to reference streams in URLs.
    The presentation description defines the hierarchical relationships
    in the presentation and the URLs for the individual streams. A
    presentation description may name a stream "a.mov" and the whole
    presentation "b.mov".

   The path components of the RTSP URL are opaque to the client and do
   not imply any particular file system structure for the server.

      This decoupling also allows presentation descriptions to be used
      with non-RTSP media control protocols simply by replacing the
      scheme in the URL.

3.3 Conference Identifiers

   Conference identifiers are opaque to RTSP and are encoded using
   standard URI encoding methods (i.e., LWS is escaped with %). They can
   contain any octet value. The conference identifier MUST be globally
   unique. For H.323, the conferenceID value is to be used.

 conference-id =   1*xchar

      Conference identifiers are used to allow RTSP sessions to obtain
      parameters from multimedia conferences the media server is
      participating in. These conferences are created by protocols
      outside the scope of this specification, e.g., H.323 [13] or SIP
      [12]. Instead of the RTSP client explicitly providing transport
      information, for example, it asks the media server to use the
      values in the conference description instead.

3.4 Session Identifiers

   Session identifiers are opaque strings of arbitrary length. Linear
   white space must be URL-escaped. A session identifier MUST be chosen
   randomly and MUST be at least eight octets long to make guessing it
   more difficult. (See Section 16.)

      session-id   =   1*(ALPHA | DIGIT | safe)

3.5 SMPTE Relative Timestamps

   A SMPTE relative timestamp expresses time relative to the start of
   the clip. Relative timestamps are expressed as SMPTE time codes for
   frame-level access accuracy. The time code has the format
   hours:minutes:seconds:frames.subframes, with the origin at the start
   of the clip. The default smpte format is "SMPTE 30 drop" format, with
   frame rate is 29.97 frames per second. Other SMPTE codes MAY be
   supported (such as "SMPTE 25") through the use of alternative use of
   "smpte time". For the "frames" field in the time value can assume
   the values 0 through 29. The difference between 30 and 29.97 frames
   per second is handled by dropping the first two frame indices (values
   00 and 01) of every minute, except every tenth minute. If the frame
   value is zero, it may be omitted. Subframes are measured in
   one-hundredth of a frame.

```
   smpte-range  =    smpte-type "=" smpte-time "-" [ smpte-time ]
   smpte-type   =    "smpte" | "smpte-30-drop" | "smpte-25"
                              ; other timecodes may be added
   smpte-time   =    1*2DIGIT ":" 1*2DIGIT ":" 1*2DIGIT [ ":" 1*2DIGIT ]
                     [ "." 1*2DIGIT ]
```

```
   Examples:
     smpte=10:12:33:20-
     smpte=10:07:33-
     smpte=10:07:00-10:07:33:05.01
     smpte-25=10:07:00-10:07:33:05.01
```

3.6 Normal Play Time

   Normal play time (NPT) indicates the stream absolute position
   relative to the beginning of the presentation. The timestamp consists
   of a decimal fraction. The part left of the decimal may be expressed
   in either seconds or hours, minutes, and seconds. The part right of
   the decimal point measures fractions of a second.

   The beginning of a presentation corresponds to 0.0 seconds. Negative
   values are not defined. The special constant now is defined as the
   current instant of a live event. It may be used only for live events.

   NPT is defined as in DSM-CC: "Intuitively, NPT is the clock the
   viewer associates with a program. It is often digitally displayed on
   a VCR. NPT advances normally when in normal play mode (scale = 1),
   advances at a faster rate when in fast scan forward (high positive
   scale ratio), decrements when in scan reverse (high negative scale
   ratio) and is fixed in pause mode. NPT is (logically) equivalent to
   SMPTE time codes." [5]

```
   npt-range    =    ( npt-time "-" [ npt-time ] ) | ( "-" npt-time )
   npt-time     =    "now" | npt-sec | npt-hhmmss
   npt-sec      =    1*DIGIT [ "." *DIGIT ]
   npt-hhmmss   =    npt-hh ":" npt-mm ":" npt-ss [ "." *DIGIT ]
   npt-hh       =    1*DIGIT    ; any positive number
   npt-mm       =    1*2DIGIT   ; 0-59
   npt-ss       =    1*2DIGIT   ; 0-59
```

```
   Examples:
     npt=123.45-125
     npt=12:05:35.3-
     npt=now-
```

     The syntax conforms to ISO 8601. The npt-sec notation is optimized
     for automatic generation, the ntp-hhmmss notation for consumption
     by human readers. The "now" constant allows clients to request to

receive the live feed rather than the stored or time-delayed
version. This is needed since neither absolute time nor zero time
are appropriate for this case.

3.7 Absolute Time

Absolute time is expressed as ISO 8601 timestamps, using UTC (GMT).
Fractions of a second may be indicated.

```
utc-range    =    "clock" "=" utc-time "-" [ utc-time ]
utc-time     =    utc-date "T" utc-time "Z"
utc-date     =    8DIGIT                      ; < YYYYMMDD >
utc-time     =    6DIGIT [ "." fraction ]   ; < HHMMSS.fraction >
```

Example for November 8, 1996 at 14h37 and 20 and a quarter seconds
UTC:

```
19961108T143720.25Z
```

3.8 Option Tags

Option tags are unique identifiers used to designate new options in
RTSP. These tags are used in Require (Section 12.32) and Proxy-
Require (Section 12.27) header fields.

Syntax:

```
option-tag   =   1*xchar
```

The creator of a new RTSP option should either prefix the option with
a reverse domain name (e.g., "com.foo.mynewfeature" is an apt name
for a feature whose inventor can be reached at "foo.com"), or
register the new option with the Internet Assigned Numbers Authority
(IANA).

3.8.1 Registering New Option Tags with IANA

When registering a new RTSP option, the following information should
be provided:

   * Name and description of option. The name may be of any length,
     but SHOULD be no more than twenty characters long. The name MUST
     not contain any spaces, control characters or periods.
   * Indication of who has change control over the option (for
     example, IETF, ISO, ITU-T, other international standardization
     bodies, a consortium or a particular company or group of
     companies);

        * A reference to a further description, if available, for example
          (in order of preference) an RFC, a published paper, a patent
          filing, a technical report, documented source code or a computer
          manual;
        * For proprietary options, contact information (postal and email
          address);

4 RTSP Message

   RTSP is a text-based protocol and uses the ISO 10646 character set in
   UTF-8 encoding (RFC 2279 [21]). Lines are terminated by CRLF, but
   receivers should be prepared to also interpret CR and LF by
   themselves as line terminators.

      Text-based protocols make it easier to add optional parameters in a
      self-describing manner. Since the number of parameters and the
      frequency of commands is low, processing efficiency is not a
      concern. Text-based protocols, if done carefully, also allow easy
      implementation of research prototypes in scripting languages such
      as Tcl, Visual Basic and Perl.

      The 10646 character set avoids tricky character set switching, but
      is invisible to the application as long as US-ASCII is being used.
      This is also the encoding used for RTCP. ISO 8859-1 translates
      directly into Unicode with a high-order octet of zero. ISO 8859-1
      characters with the most-significant bit set are represented as
      1100001x 10xxxxxx. (See RFC 2279 [21])

   RTSP messages can be carried over any lower-layer transport protocol
   that is 8-bit clean.

   Requests contain methods, the object the method is operating upon and
   parameters to further describe the method. Methods are idempotent,
   unless otherwise noted. Methods are also designed to require little
   or no state maintenance at the media server.

4.1 Message Types

   See [H4.1]

4.2 Message Headers

   See [H4.2]

4.3 Message Body

   See [H4.3]

4.4 Message Length

   When a message body is included with a message, the length of that
   body is determined by one of the following (in order of precedence):

   1.   Any response message which MUST NOT include a message body
           (such as the 1xx, 204, and 304 responses) is always terminated
           by the first empty line after the header fields, regardless of
           the entity-header fields present in the message. (Note: An
           empty line consists of only CRLF.)

   2.   If a Content-Length header field (section 12.14) is present,
           its value in bytes represents the length of the message-body.
           If this header field is not present, a value of zero is
           assumed.

   3.   By the server closing the connection. (Closing the connection
           cannot be used to indicate the end of a request body, since
           that would leave no possibility for the server to send back a
           response.)

   Note that RTSP does not (at present) support the HTTP/1.1 "chunked"
   transfer coding(see [H3.6]) and requires the presence of the
   Content-Length header field.

      Given the moderate length of presentation descriptions returned,
      the server should always be able to determine its length, even if
      it is generated dynamically, making the chunked transfer encoding
      unnecessary. Even though Content-Length must be present if there is
      any entity body, the rules ensure reasonable behavior even if the
      length is not given explicitly.

5 General Header Fields

   See [H4.5], except that Pragma, Transfer-Encoding and Upgrade headers
   are not defined:

      general-header   =   Cache-Control   ; Section 12.8
                         | Connection   ; Section 12.10
                         | Date   ; Section 12.18
                         | Via   ; Section 12.43

6 Request

   A request message from a client to a server or vice versa includes,
   within the first line of that message, the method to be applied to
   the resource, the identifier of the resource, and the protocol
   version in use.

```
        Request       =       Request-Line          ; Section 6.1
                      *(      general-header        ; Section 5
                      |       request-header         ; Section 6.2
                      |       entity-header )        ; Section 8.1
                              CRLF
                              [ message-body ]       ; Section 4.3
```

6.1 Request Line

```
  Request-Line = Method SP Request-URI SP RTSP-Version CRLF
```

```
   Method        =         "DESCRIBE"             ; Section 10.2
                 |         "ANNOUNCE"             ; Section 10.3
                 |         "GET_PARAMETER"        ; Section 10.8
                 |         "OPTIONS"              ; Section 10.1
                 |         "PAUSE"                ; Section 10.6
                 |         "PLAY"                 ; Section 10.5
                 |         "RECORD"               ; Section 10.11
                 |         "REDIRECT"             ; Section 10.10
                 |         "SETUP"                ; Section 10.4
                 |         "SET_PARAMETER"        ; Section 10.9
                 |         "TEARDOWN"             ; Section 10.7
                 |         extension-method
```

```
  extension-method = token
```

```
  Request-URI = "*" | absolute_URI
```

```
  RTSP-Version = "RTSP" "/" 1*DIGIT "." 1*DIGIT
```

6.2 Request Header Fields

```
  request-header  =         Accept                ; Section 12.1
                  |         Accept-Encoding        ; Section 12.2
                  |         Accept-Language        ; Section 12.3
                  |         Authorization          ; Section 12.5
                  |         From                   ; Section 12.20
                  |         If-Modified-Since      ; Section 12.23
                  |         Range                  ; Section 12.29
                  |         Referer                ; Section 12.30
                  |         User-Agent             ; Section 12.41
```

   Note that in contrast to HTTP/1.1 [2], RTSP requests always contain
   the absolute URL (that is, including the scheme, host and port)
   rather than just the absolute path.

HTTP/1.1 requires servers to understand the absolute URL, but
clients are supposed to use the Host request header. This is purely
needed for backward-compatibility with HTTP/1.0 servers, a
consideration that does not apply to RTSP.

The asterisk "*" in the Request-URI means that the request does not
apply to a particular resource, but to the server itself, and is only
allowed when the method used does not necessarily apply to a
resource. One example would be:

    OPTIONS * RTSP/1.0

7 Response

[H6] applies except that HTTP-Version is replaced by RTSP-Version.
Also, RTSP defines additional status codes and does not define some
HTTP codes. The valid response codes and the methods they can be used
with are defined in Table 1.

After receiving and interpreting a request message, the recipient
responds with an RTSP response message.

```
Response      =       Status-Line          ; Section 7.1
              *(      general-header       ; Section 5
              |       response-header      ; Section 7.1.2
              |       entity-header )       ; Section 8.1
                      CRLF
                      [ message-body ]      ; Section 4.3
```

7.1 Status-Line

The first line of a Response message is the Status-Line, consisting
of the protocol version followed by a numeric status code, and the
textual phrase associated with the status code, with each element
separated by SP characters. No CR or LF is allowed except in the
final CRLF sequence.

    Status-Line =   RTSP-Version SP Status-Code SP Reason-Phrase CRLF

7.1.1 Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the
attempt to understand and satisfy the request. These codes are fully
defined in Section 11. The Reason-Phrase is intended to give a short
textual description of the Status-Code. The Status-Code is intended
for use by automata and the Reason-Phrase is intended for the human
user. The client is not required to examine or display the Reason-
Phrase.

The first digit of the Status-Code defines the class of response. The
last two digits do not have any categorization role. There are 5
values for the first digit:

  * 1xx: Informational - Request received, continuing process
  * 2xx: Success - The action was successfully received, understood,
    and accepted
  * 3xx: Redirection - Further action must be taken in order to
    complete the request
  * 4xx: Client Error - The request contains bad syntax or cannot be
    fulfilled
  * 5xx: Server Error - The server failed to fulfill an apparently
    valid request

The individual values of the numeric status codes defined for
RTSP/1.0, and an example set of corresponding Reason-Phrase's, are
presented below. The reason phrases listed here are only recommended
- they may be replaced by local equivalents without affecting the
protocol. Note that RTSP adopts most HTTP/1.1 [2] status codes and
adds RTSP-specific status codes starting at x50 to avoid conflicts
with newly defined HTTP status codes.

```
   Status-Code  =     "100"      ; Continue
                |     "200"      ; OK
                |     "201"      ; Created
                |     "250"      ; Low on Storage Space
                |     "300"      ; Multiple Choices
                |     "301"      ; Moved Permanently
                |     "302"      ; Moved Temporarily
                |     "303"      ; See Other
                |     "304"      ; Not Modified
                |     "305"      ; Use Proxy
                |     "400"      ; Bad Request
                |     "401"      ; Unauthorized
                |     "402"      ; Payment Required
                |     "403"      ; Forbidden
                |     "404"      ; Not Found
                |     "405"      ; Method Not Allowed
                |     "406"      ; Not Acceptable
                |     "407"      ; Proxy Authentication Required
                |     "408"      ; Request Time-out
                |     "410"      ; Gone
                |     "411"      ; Length Required
                |     "412"      ; Precondition Failed
                |     "413"      ; Request Entity Too Large
                |     "414"      ; Request-URI Too Large
                |     "415"      ; Unsupported Media Type
                |     "451"      ; Parameter Not Understood
                |     "452"      ; Conference Not Found
                |     "453"      ; Not Enough Bandwidth
                |     "454"      ; Session Not Found
                |     "455"      ; Method Not Valid in This State
                |     "456"      ; Header Field Not Valid for Resource
                |     "457"      ; Invalid Range
                |     "458"      ; Parameter Is Read-Only
                |     "459"      ; Aggregate operation not allowed
                |     "460"      ; Only aggregate operation allowed
                |     "461"      ; Unsupported transport
                |     "462"      ; Destination unreachable
                |     "500"      ; Internal Server Error
                |     "501"      ; Not Implemented
                |     "502"      ; Bad Gateway
                |     "503"      ; Service Unavailable
                |     "504"      ; Gateway Time-out
                |     "505"      ; RTSP Version not supported
                |     "551"      ; Option not supported
                |     extension-code
```

    extension-code =   3DIGIT

    Reason-Phrase =   *<TEXT, excluding CR, LF>

    RTSP status codes are extensible. RTSP applications are not required
    to understand the meaning of all registered status codes, though such
    understanding is obviously desirable. However, applications MUST
    understand the class of any status code, as indicated by the first
    digit, and treat any unrecognized response as being equivalent to the
    x00 status code of that class, with the exception that an
    unrecognized response MUST NOT be cached. For example, if an
    unrecognized status code of 431 is received by the client, it can
    safely assume that there was something wrong with its request and
    treat the response as if it had received a 400 status code. In such
    cases, user agents SHOULD present to the user the entity returned
    with the response, since that entity is likely to include human-
    readable information which will explain the unusual status.

    Code   reason

    100   Continue   all

    200   OK   all
    201   Created   RECORD
    250   Low on Storage Space   RECORD

    300   Multiple Choices   all
    301   Moved Permanently   all
    302   Moved Temporarily   all
    303   See Other   all
    305   Use Proxy   all

```
400         Bad Request                  all
401         Unauthorized                 all
402         Payment Required             all
403         Forbidden                    all
404         Not Found                    all
405         Method Not Allowed           all
406         Not Acceptable               all
407         Proxy Authentication Required    all
408         Request Timeout              all
410         Gone                         all
411         Length Required              all
412         Precondition Failed          DESCRIBE, SETUP
413         Request Entity Too Large     all
414         Request-URI Too Long         all
415         Unsupported Media Type       all
451         Invalid parameter            SETUP
452         Illegal Conference Identifier    SETUP
453         Not Enough Bandwidth         SETUP
454         Session Not Found            all
455         Method Not Valid In This State   all
456         Header Field Not Valid       all
457         Invalid Range                PLAY
458         Parameter Is Read-Only       SET_PARAMETER
459         Aggregate Operation Not Allowed  all
460         Only Aggregate Operation Allowed all
461         Unsupported Transport        all
462         Destination Unreachable      all

500         Internal Server Error        all
501         Not Implemented              all
502         Bad Gateway                  all
503         Service Unavailable          all
504         Gateway Timeout              all
505         RTSP Version Not Supported   all
551         Option not support           all
```

        Table 1: Status codes and their usage with RTSP methods

7.1.2 Response Header Fields

   The response-header fields allow the request recipient to pass
   additional information about the response which cannot be placed in
   the Status-Line. These header fields give information about the
   server and about further access to the resource identified by the
   Request-URI.

```
response-header  =     Location            ; Section 12.25
                 |     Proxy-Authenticate  ; Section 12.26
                 |     Public              ; Section 12.28
                 |     Retry-After         ; Section 12.31
                 |     Server              ; Section 12.36
                 |     Vary                ; Section 12.42
                 |     WWW-Authenticate    ; Section 12.44
```

Response-header field names can be extended reliably only in
combination with a change in the protocol version. However, new or
experimental header fields MAY be given the semantics of response-
header fields if all parties in the communication recognize them to
be response-header fields. Unrecognized header fields are treated as
entity-header fields.

8 Entity

Request and Response messages MAY transfer an entity if not otherwise
restricted by the request method or response status code. An entity
consists of entity-header fields and an entity-body, although some
responses will only include the entity-headers.

In this section, both sender and recipient refer to either the client
or the server, depending on who sends and who receives the entity.

8.1 Entity Header Fields

Entity-header fields define optional metainformation about the
entity-body or, if no body is present, about the resource identified
by the request.

```
entity-header        =     Allow               ; Section 12.4
                     |     Content-Base        ; Section 12.11
                     |     Content-Encoding    ; Section 12.12
                     |     Content-Language    ; Section 12.13
                     |     Content-Length      ; Section 12.14
                     |     Content-Location    ; Section 12.15
                     |     Content-Type        ; Section 12.16
                     |     Expires             ; Section 12.19
                     |     Last-Modified       ; Section 12.24
                     |     extension-header
   extension-header  =     message-header
```

The extension-header mechanism allows additional entity-header fields
to be defined without changing the protocol, but these fields cannot
be assumed to be recognizable by the recipient. Unrecognized header
fields SHOULD be ignored by the recipient and forwarded by proxies.

8.2 Entity Body

   See [H7.2]

9 Connections

   RTSP requests can be transmitted in several different ways:

      * persistent transport connections used for several
        request-response transactions;
      * one connection per request/response transaction;
      * connectionless mode.

   The type of transport connection is defined by the RTSP URI (Section
   3.2). For the scheme "rtsp", a persistent connection is assumed,
   while the scheme "rtspu" calls for RTSP requests to be sent without
   setting up a connection.

   Unlike HTTP, RTSP allows the media server to send requests to the
   media client. However, this is only supported for persistent
   connections, as the media server otherwise has no reliable way of
   reaching the client. Also, this is the only way that requests from
   media server to client are likely to traverse firewalls.

9.1 Pipelining

   A client that supports persistent connections or connectionless mode
   MAY "pipeline" its requests (i.e., send multiple requests without
   waiting for each response). A server MUST send its responses to those
   requests in the same order that the requests were received.

9.2 Reliability and Acknowledgements

   Requests are acknowledged by the receiver unless they are sent to a
   multicast group. If there is no acknowledgement, the sender may
   resend the same message after a timeout of one round-trip time (RTT).
   The round-trip time is estimated as in TCP (RFC 1123) [18], with an
   initial round-trip value of 500 ms. An implementation MAY cache the
   last RTT measurement as the initial value for future connections.

   If a reliable transport protocol is used to carry RTSP, requests MUST
   NOT be retransmitted; the RTSP application MUST instead rely on the
   underlying transport to provide reliability.

      If both the underlying reliable transport such as TCP and the RTSP
      application retransmit requests, it is possible that each packet
      loss results in two retransmissions. The receiver cannot typically
      take advantage of the application-layer retransmission since the

transport stack will not deliver the application-layer
retransmission before the first attempt has reached the receiver.
If the packet loss is caused by congestion, multiple
retransmissions at different layers will exacerbate the congestion.

If RTSP is used over a small-RTT LAN, standard procedures for
optimizing initial TCP round trip estimates, such as those used in
T/TCP (RFC 1644) [22], can be beneficial.

The Timestamp header (Section 12.38) is used to avoid the
retransmission ambiguity problem [23, p. 301] and obviates the need
for Karn's algorithm.

Each request carries a sequence number in the CSeq header (Section
12.17), which is incremented by one for each distinct request
transmitted. If a request is repeated because of lack of
acknowledgement, the request MUST carry the original sequence number
(i.e., the sequence number is not incremented).

Systems implementing RTSP MUST support carrying RTSP over TCP and MAY
support UDP. The default port for the RTSP server is 554 for both UDP
and TCP.

A number of RTSP packets destined for the same control end point may
be packed into a single lower-layer PDU or encapsulated into a TCP
stream. RTSP data MAY be interleaved with RTP and RTCP packets.
Unlike HTTP, an RTSP message MUST contain a Content-Length header
whenever that message contains a payload. Otherwise, an RTSP packet
is terminated with an empty line immediately following the last
message header.

10 Method Definitions

The method token indicates the method to be performed on the resource
identified by the Request-URI. The method is case-sensitive. New
methods may be defined in the future. Method names may not start with
a $ character (decimal 24) and must be a token. Methods are
summarized in Table 2.

method	direction	object	requirement
DESCRIBE	C->S	P,S	recommended
ANNOUNCE	C->S, S->C	P,S	optional
GET_PARAMETER	C->S, S->C	P,S	optional
OPTIONS	C->S, S->C	P,S	required (S->C: optional)
PAUSE	C->S	P,S	recommended
PLAY	C->S	P,S	required
RECORD	C->S	P,S	optional
REDIRECT	S->C	P,S	optional
SETUP	C->S	S	required
SET_PARAMETER	C->S, S->C	P,S	optional
TEARDOWN	C->S	P,S	required

Table 2: Overview of RTSP methods, their direction, and what
objects (P: presentation, S: stream) they operate on

Notes on Table 2: PAUSE is recommended, but not required in that a
fully functional server can be built that does not support this
method, for example, for live feeds. If a server does not support a
particular method, it MUST return "501 Not Implemented" and a client
SHOULD not try this method again for this server.

10.1 OPTIONS

The behavior is equivalent to that described in [H9.2]. An OPTIONS
request may be issued at any time, e.g., if the client is about to
try a nonstandard request. It does not influence server state.

Example:

```
C->S:   OPTIONS * RTSP/1.0
        CSeq: 1
        Require: implicit-play
        Proxy-Require: gzipped-messages

S->C:   RTSP/1.0 200 OK
        CSeq: 1
        Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
```

Note that these are necessarily fictional features (one would hope
that we would not purposefully overlook a truly useful feature just
so that we could have a strong example in this section).

10.2 DESCRIBE

   The DESCRIBE method retrieves the description of a presentation or
   media object identified by the request URL from a server. It may use
   the Accept header to specify the description formats that the client
   understands. The server responds with a description of the requested
   resource. The DESCRIBE reply-response pair constitutes the media
   initialization phase of RTSP.

   Example:

     C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
           CSeq: 312
           Accept: application/sdp, application/rtsl, application/mheg

     S->C: RTSP/1.0 200 OK
           CSeq: 312
           Date: 23 Jan 1997 15:35:06 GMT
           Content-Type: application/sdp
           Content-Length: 376

           v=0
           o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
           s=SDP Seminar
           i=A Seminar on the session description protocol
           u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
           e=mjh@isi.edu (Mark Handley)
           c=IN IP4 224.2.17.12/127
           t=2873397496 2873404696
           a=recvonly
           m=audio 3456 RTP/AVP 0
           m=video 2232 RTP/AVP 31
           m=whiteboard 32416 UDP WB
           a=orient:portrait

   The DESCRIBE response MUST contain all media initialization
   information for the resource(s) that it describes. If a media client
   obtains a presentation description from a source other than DESCRIBE
   and that description contains a complete set of media initialization
   parameters, the client SHOULD use those parameters and not then
   request a description for the same media via RTSP.

   Additionally, servers SHOULD NOT use the DESCRIBE response as a means
   of media indirection.

     Clear ground rules need to be established so that clients have an
     unambiguous means of knowing when to request media initialization
     information via DESCRIBE, and when not to. By forcing a DESCRIBE

      response to contain all media initialization for the set of streams
      that it describes, and discouraging use of DESCRIBE for media
      indirection, we avoid looping problems that might result from other
      approaches.

      Media initialization is a requirement for any RTSP-based system,
      but the RTSP specification does not dictate that this must be done
      via the DESCRIBE method. There are three ways that an RTSP client
      may receive initialization information:

      * via RTSP's DESCRIBE method;
      * via some other protocol (HTTP, email attachment, etc.);
      * via the command line or standard input (thus working as a browser
        helper application launched with an SDP file or other media
        initialization format).

      In the interest of practical interoperability, it is highly
      recommended that minimal servers support the DESCRIBE method, and
      highly recommended that minimal clients support the ability to act
      as a "helper application" that accepts a media initialization file
      from standard input, command line, and/or other means that are
      appropriate to the operating environment of the client.

10.3 ANNOUNCE

      The ANNOUNCE method serves two purposes:

      When sent from client to server, ANNOUNCE posts the description of a
      presentation or media object identified by the request URL to a
      server. When sent from server to client, ANNOUNCE updates the session
      description in real-time.

      If a new media stream is added to a presentation (e.g., during a live
      presentation), the whole presentation description should be sent
      again, rather than just the additional components, so that components
      can be deleted.

      Example:

        C->S: ANNOUNCE rtsp://server.example.com/fizzle/foo RTSP/1.0
              CSeq: 312
              Date: 23 Jan 1997 15:35:06 GMT
              Session: 47112344
              Content-Type: application/sdp
              Content-Length: 332

              v=0
              o=mhandley 2890844526 2890845468 IN IP4 126.16.64.4

```
         s=SDP Seminar
         i=A Seminar on the session description protocol
         u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
         e=mjh@isi.edu (Mark Handley)
         c=IN IP4 224.2.17.12/127
         t=2873397496 2873404696
         a=recvonly
         m=audio 3456 RTP/AVP 0
         m=video 2232 RTP/AVP 31

   S->C: RTSP/1.0 200 OK
         CSeq: 312
```

10.4 SETUP

   The SETUP request for a URI specifies the transport mechanism to be
   used for the streamed media. A client can issue a SETUP request for a
   stream that is already playing to change transport parameters, which
   a server MAY allow. If it does not allow this, it MUST respond with
   error "455 Method Not Valid In This State". For the benefit of any
   intervening firewalls, a client must indicate the transport
   parameters even if it has no influence over these parameters, for
   example, where the server advertises a fixed multicast address.

      Since SETUP includes all transport initialization information,
      firewalls and other intermediate network devices (which need this
      information) are spared the more arduous task of parsing the
      DESCRIBE response, which has been reserved for media
      initialization.

   The Transport header specifies the transport parameters acceptable to
   the client for data transmission; the response will contain the
   transport parameters selected by the server.

```
    C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
          CSeq: 302
          Transport: RTP/AVP;unicast;client_port=4588-4589

    S->C: RTSP/1.0 200 OK
          CSeq: 302
          Date: 23 Jan 1997 15:35:06 GMT
          Session: 47112344
          Transport: RTP/AVP;unicast;
            client_port=4588-4589;server_port=6256-6257
```

   The server generates session identifiers in response to SETUP
   requests. If a SETUP request to a server includes a session
   identifier, the server MUST bundle this setup request into the

   existing session or return error "459 Aggregate Operation Not
   Allowed" (see Section 11.3.10).

10.5 PLAY

   The PLAY method tells the server to start sending data via the
   mechanism specified in SETUP. A client MUST NOT issue a PLAY request
   until any outstanding SETUP requests have been acknowledged as
   successful.

   The PLAY request positions the normal play time to the beginning of
   the range specified and delivers stream data until the end of the
   range is reached. PLAY requests may be pipelined (queued); a server
   MUST queue PLAY requests to be executed in order. That is, a PLAY
   request arriving while a previous PLAY request is still active is
   delayed until the first has been completed.

      This allows precise editing.

   For example, regardless of how closely spaced the two PLAY requests
   in the example below arrive, the server will first play seconds 10
   through 15, then, immediately following, seconds 20 to 25, and
   finally seconds 30 through the end.

      C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
            CSeq: 835
            Session: 12345678
            Range: npt=10-15

      C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
            CSeq: 836
            Session: 12345678
            Range: npt=20-25

      C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
            CSeq: 837
            Session: 12345678
            Range: npt=30-

   See the description of the PAUSE request for further examples.

   A PLAY request without a Range header is legal. It starts playing a
   stream from the beginning unless the stream has been paused. If a
   stream has been paused via PAUSE, stream delivery resumes at the
   pause point. If a stream is playing, such a PLAY request causes no
   further action and can be used by the client to test server liveness.

The Range header may also contain a time parameter. This parameter
specifies a time in UTC at which the playback should start. If the
message is received after the specified time, playback is started
immediately. The time parameter may be used to aid in synchronization
of streams obtained from different sources.

For a on-demand stream, the server replies with the actual range that
will be played back. This may differ from the requested range if
alignment of the requested range to valid frame boundaries is
required for the media source. If no range is specified in the
request, the current position is returned in the reply. The unit of
the range in the reply is the same as that in the request.

After playing the desired range, the presentation is automatically
paused, as if a PAUSE request had been issued.

The following example plays the whole presentation starting at SMPTE
time code 0:10:20 until the end of the clip. The playback is to start
at 15:36 on 23 Jan 1997.

```
C->S: PLAY rtsp://audio.example.com/twister.en RTSP/1.0
      CSeq: 833
      Session: 12345678
      Range: smpte=0:10:20-;time=19970123T153600Z

S->C: RTSP/1.0 200 OK
      CSeq: 833
      Date: 23 Jan 1997 15:35:06 GMT
      Range: smpte=0:10:22-;time=19970123T153600Z
```

For playing back a recording of a live presentation, it may be
desirable to use clock units:

```
C->S: PLAY rtsp://audio.example.com/meeting.en RTSP/1.0
      CSeq: 835
      Session: 12345678
      Range: clock=19961108T142300Z-19961108T143520Z

S->C: RTSP/1.0 200 OK
      CSeq: 835
      Date: 23 Jan 1997 15:35:06 GMT
```

A media server only supporting playback MUST support the npt format
and MAY support the clock and smpte formats.

10.6 PAUSE

    The PAUSE request causes the stream delivery to be interrupted
    (halted) temporarily. If the request URL names a stream, only
    playback and recording of that stream is halted. For example, for
    audio, this is equivalent to muting. If the request URL names a
    presentation or group of streams, delivery of all currently active
    streams within the presentation or group is halted. After resuming
    playback or recording, synchronization of the tracks MUST be
    maintained. Any server resources are kept, though servers MAY close
    the session and free resources after being paused for the duration
    specified with the timeout parameter of the Session header in the
    SETUP message.

    Example:

      C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
            CSeq: 834
            Session: 12345678

      S->C: RTSP/1.0 200 OK
            CSeq: 834
            Date: 23 Jan 1997 15:35:06 GMT

    The PAUSE request may contain a Range header specifying when the
    stream or presentation is to be halted. We refer to this point as the
    "pause point". The header must contain exactly one value rather than
    a time range. The normal play time for the stream is set to the pause
    point. The pause request becomes effective the first time the server
    is encountering the time point specified in any of the currently
    pending PLAY requests. If the Range header specifies a time outside
    any currently pending PLAY requests, the error "457 Invalid Range" is
    returned. If a media unit (such as an audio or video frame) starts
    presentation at exactly the pause point, it is not played or
    recorded. If the Range header is missing, stream delivery is
    interrupted immediately on receipt of the message and the pause point
    is set to the current normal play time.

    A PAUSE request discards all queued PLAY requests. However, the pause
    point in the media stream MUST be maintained. A subsequent PLAY
    request without Range header resumes from the pause point.

    For example, if the server has play requests for ranges 10 to 15 and
    20 to 29 pending and then receives a pause request for NPT 21, it
    would start playing the second range and stop at NPT 21. If the pause
    request is for NPT 12 and the server is playing at NPT 13 serving the
    first play request, the server stops immediately. If the pause
    request is for NPT 16, the server stops after completing the first

play request and discards the second play request.

As another example, if a server has received requests to play ranges
10 to 15 and then 13 to 20 (that is, overlapping ranges), the PAUSE
request for NPT=14 would take effect while the server plays the first
range, with the second PLAY request effectively being ignored,
assuming the PAUSE request arrives before the server has started
playing the second, overlapping range. Regardless of when the PAUSE
request arrives, it sets the NPT to 14.

If the server has already sent data beyond the time specified in the
Range header, a PLAY would still resume at that point in time, as it
is assumed that the client has discarded data after that point. This
ensures continuous pause/play cycling without gaps.

10.7 TEARDOWN

The TEARDOWN request stops the stream delivery for the given URI,
freeing the resources associated with it. If the URI is the
presentation URI for this presentation, any RTSP session identifier
associated with the session is no longer valid. Unless all transport
parameters are defined by the session description, a SETUP request
has to be issued before the session can be played again.

Example:
  C->S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/1.0
        CSeq: 892
        Session: 12345678
  S->C: RTSP/1.0 200 OK
        CSeq: 892

10.8 GET_PARAMETER

The GET_PARAMETER request retrieves the value of a parameter of a
presentation or stream specified in the URI. The content of the reply
and response is left to the implementation. GET_PARAMETER with no
entity body may be used to test client or server liveness ("ping").

Example:

  S->C: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
        CSeq: 431
        Content-Type: text/parameters
        Session: 12345678
        Content-Length: 15

        packets_received
        jitter

```
C->S: RTSP/1.0 200 OK
      CSeq: 431
      Content-Length: 46
      Content-Type: text/parameters

      packets_received: 10
      jitter: 0.3838
```

The "text/parameters" section is only an example type for
parameter. This method is intentionally loosely defined with the
intention that the reply content and response content will be
defined after further experimentation.

10.9 SET_PARAMETER

   This method requests to set the value of a parameter for a
   presentation or stream specified by the URI.

   A request SHOULD only contain a single parameter to allow the client
   to determine why a particular request failed. If the request contains
   several parameters, the server MUST only act on the request if all of
   the parameters can be set successfully. A server MUST allow a
   parameter to be set repeatedly to the same value, but it MAY disallow
   changing parameter values.

   Note: transport parameters for the media stream MUST only be set with
   the SETUP command.

   Restricting setting transport parameters to SETUP is for the
   benefit of firewalls.

   The parameters are split in a fine-grained fashion so that there
   can be more meaningful error indications. However, it may make
   sense to allow the setting of several parameters if an atomic
   setting is desirable. Imagine device control where the client does
   not want the camera to pan unless it can also tilt to the right
   angle at the same time.

   Example:

```
C->S: SET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
      CSeq: 421
      Content-length: 20
      Content-type: text/parameters

      barparam: barstuff

S->C: RTSP/1.0 451 Invalid Parameter
```

```
            CSeq: 421
            Content-length: 10
            Content-type: text/parameters

            barparam
```

   The "text/parameters" section is only an example type for
   parameter. This method is intentionally loosely defined with the
   intention that the reply content and response content will be
   defined after further experimentation.

10.10 REDIRECT

   A redirect request informs the client that it must connect to another
   server location. It contains the mandatory header Location, which
   indicates that the client should issue requests for that URL. It may
   contain the parameter Range, which indicates when the redirection
   takes effect. If the client wants to continue to send or receive
   media for this URI, the client MUST issue a TEARDOWN request for the
   current session and a SETUP for the new session at the designated
   host.

   This example request redirects traffic for this URI to the new server
   at the given play time:

```
      S->C: REDIRECT rtsp://example.com/fizzle/foo RTSP/1.0
            CSeq: 732
            Location: rtsp://bigserver.com:8001
            Range: clock=19960213T143205Z-
```

10.11 RECORD

   This method initiates recording a range of media data according to
   the presentation description. The timestamp reflects start and end
   time (UTC). If no time range is given, use the start or end time
   provided in the presentation description. If the session has already
   started, commence recording immediately.

   The server decides whether to store the recorded data under the
   request-URI or another URI. If the server does not use the request-
   URI, the response SHOULD be 201 (Created) and contain an entity which
   describes the status of the request and refers to the new resource,
   and a Location header.

   A media server supporting recording of live presentations MUST
   support the clock range format; the smpte format does not make sense.

   In this example, the media server was previously invited to the
   conference indicated.

      C->S: RECORD rtsp://example.com/meeting/audio.en RTSP/1.0
            CSeq: 954
            Session: 12345678
            Conference: 128.16.64.19/32492374

10.12 Embedded (Interleaved) Binary Data

   Certain firewall designs and other circumstances may force a server
   to interleave RTSP methods and stream data. This interleaving should
   generally be avoided unless necessary since it complicates client and
   server operation and imposes additional overhead. Interleaved binary
   data SHOULD only be used if RTSP is carried over TCP.

   Stream data such as RTP packets is encapsulated by an ASCII dollar
   sign (24 hexadecimal), followed by a one-byte channel identifier,
   followed by the length of the encapsulated binary data as a binary,
   two-byte integer in network byte order. The stream data follows
   immediately afterwards, without a CRLF, but including the upper-layer
   protocol headers. Each $ block contains exactly one upper-layer
   protocol data unit, e.g., one RTP packet.

   The channel identifier is defined in the Transport header with the
   interleaved parameter(Section 12.39).

   When the transport choice is RTP, RTCP messages are also interleaved
   by the server over the TCP connection. As a default, RTCP packets are
   sent on the first available channel higher than the RTP channel. The
   client MAY explicitly request RTCP packets on another channel. This
   is done by specifying two channels in the interleaved parameter of
   the Transport header(Section 12.39).

     RTCP is needed for synchronization when two or more streams are
     interleaved in such a fashion. Also, this provides a convenient way
     to tunnel RTP/RTCP packets through the TCP control connection when
     required by the network configuration and transfer them onto UDP
     when possible.

      C->S: SETUP rtsp://foo.com/bar.file RTSP/1.0
            CSeq: 2
            Transport: RTP/AVP/TCP;interleaved=0-1

      S->C: RTSP/1.0 200 OK
            CSeq: 2
            Date: 05 Jun 1997 18:57:18 GMT
            Transport: RTP/AVP/TCP;interleaved=0-1

```
             Session: 12345678

   C->S: PLAY rtsp://foo.com/bar.file RTSP/1.0
         CSeq: 3
         Session: 12345678

   S->C: RTSP/1.0 200 OK
         CSeq: 3
         Session: 12345678
         Date: 05 Jun 1997 18:59:15 GMT
         RTP-Info: url=rtsp://foo.com/bar.file;
           seq=232433;rtptime=972948234

   S->C: $\000{2 byte length}{"length" bytes data, w/RTP header}
   S->C: $\000{2 byte length}{"length" bytes data, w/RTP header}
   S->C: $\001{2 byte length}{"length" bytes  RTCP packet}
```

11 Status Code Definitions

   Where applicable, HTTP status [H10] codes are reused. Status codes
   that have the same meaning are not repeated here. See Table 1 for a
   listing of which status codes may be returned by which requests.

11.1 Success 2xx

11.1.1 250 Low on Storage Space

   The server returns this warning after receiving a RECORD request that
   it may not be able to fulfill completely due to insufficient storage
   space. If possible, the server should use the Range header to
   indicate what time period it may still be able to record. Since other
   processes on the server may be consuming storage space
   simultaneously, a client should take this only as an estimate.

11.2 Redirection 3xx

   See [H10.3].

   Within RTSP, redirection may be used for load balancing or
   redirecting stream requests to a server topologically closer to the
   client. Mechanisms to determine topological proximity are beyond the
   scope of this specification.

11.3 Client Error 4xx

11.3.1 405 Method Not Allowed

   The method specified in the request is not allowed for the resource
   identified by the request URI. The response MUST include an Allow
   header containing a list of valid methods for the requested resource.
   This status code is also to be used if a request attempts to use a
   method not indicated during SETUP, e.g., if a RECORD request is
   issued even though the mode parameter in the Transport header only
   specified PLAY.

11.3.2 451 Parameter Not Understood

   The recipient of the request does not support one or more parameters
   contained in the request.

11.3.3 452 Conference Not Found

   The conference indicated by a Conference header field is unknown to
   the media server.

11.3.4 453 Not Enough Bandwidth

   The request was refused because there was insufficient bandwidth.
   This may, for example, be the result of a resource reservation
   failure.

11.3.5 454 Session Not Found

   The RTSP session identifier in the Session header is missing,
   invalid, or has timed out.

11.3.6 455 Method Not Valid in This State

   The client or server cannot process this request in its current
   state. The response SHOULD contain an Allow header to make error
   recovery easier.

11.3.7 456 Header Field Not Valid for Resource

   The server could not act on a required request header. For example,
   if PLAY contains the Range header field but the stream does not allow
   seeking.

11.3.8 457 Invalid Range

   The Range value given is out of bounds, e.g., beyond the end of the
   presentation.

11.3.9 458 Parameter Is Read-Only

   The parameter to be set by SET_PARAMETER can be read but not
   modified.

11.3.10 459 Aggregate Operation Not Allowed

   The requested method may not be applied on the URL in question since
   it is an aggregate (presentation) URL. The method may be applied on a
   stream URL.

11.3.11 460 Only Aggregate Operation Allowed

   The requested method may not be applied on the URL in question since
   it is not an aggregate (presentation) URL. The method may be applied
   on the presentation URL.

11.3.12 461 Unsupported Transport

   The Transport field did not contain a supported transport
   specification.

11.3.13 462 Destination Unreachable

   The data transmission channel could not be established because the
   client address could not be reached. This error will most likely be
   the result of a client attempt to place an invalid Destination
   parameter in the Transport field.

11.3.14 551 Option not supported

   An option given in the Require or the Proxy-Require fields was not
   supported. The Unsupported header should be returned stating the
   option for which there is no support.

12 Header Field Definitions

   HTTP/1.1 [2] or other, non-standard header fields not listed here
   currently have no well-defined meaning and SHOULD be ignored by the
   recipient.

   Table 3 summarizes the header fields used by RTSP. Type "g"
   designates general request headers to be found in both requests and
   responses, type "R" designates request headers, type "r" designates
   response headers, and type "e" designates entity header fields.
   Fields marked with "req." in the column labeled "support" MUST be
   implemented by the recipient for a particular method, while fields
   marked "opt." are optional. Note that not all fields marked "req."
   will be sent in every request of this type. The "req." means only
   that client (for response headers) and server (for request headers)
   MUST implement the fields. The last column lists the method for which
   this header field is meaningful; the designation "entity" refers to
   all methods that return a message body. Within this specification,
   DESCRIBE and GET_PARAMETER fall into this class.

Header	type	support	methods
Accept	R	opt.	entity
Accept-Encoding	R	opt.	entity
Accept-Language	R	opt.	all
Allow	r	opt.	all
Authorization	R	opt.	all
Bandwidth	R	opt.	all
Blocksize	R	opt.	all but OPTIONS, TEARDOWN
Cache-Control	g	opt.	SETUP
Conference	R	opt.	SETUP
Connection	g	req.	all
Content-Base	e	opt.	entity
Content-Encoding	e	req.	SET_PARAMETER
Content-Encoding	e	req.	DESCRIBE, ANNOUNCE
Content-Language	e	req.	DESCRIBE, ANNOUNCE
Content-Length	e	req.	SET_PARAMETER, ANNOUNCE
Content-Length	e	req.	entity
Content-Location	e	opt.	entity
Content-Type	e	req.	SET_PARAMETER, ANNOUNCE
Content-Type	r	req.	entity
CSeq	g	req.	all
Date	g	opt.	all
Expires	e	opt.	DESCRIBE, ANNOUNCE
From	R	opt.	all
If-Modified-Since	R	opt.	DESCRIBE, SETUP
Last-Modified	e	opt.	entity
Proxy-Authenticate			
Proxy-Require	R	req.	all
Public	r	opt.	all
Range	R	opt.	PLAY, PAUSE, RECORD
Range	r	opt.	PLAY, PAUSE, RECORD
Referer	R	opt.	all
Require	R	req.	all
Retry-After	r	opt.	all
RTP-Info	r	req.	PLAY
Scale	Rr	opt.	PLAY, RECORD
Session	Rr	req.	all but SETUP, OPTIONS
Server	r	opt.	all
Speed	Rr	opt.	PLAY
Transport	Rr	req.	SETUP
Unsupported	r	req.	all
User-Agent	R	opt.	all
Via	g	opt.	all
WWW-Authenticate	r	opt.	all

   Overview of RTSP header fields

12.1 Accept

   The Accept request-header field can be used to specify certain
   presentation description content types which are acceptable for the
   response.

      The "level" parameter for presentation descriptions is properly
      defined as part of the MIME type registration, not here.

   See [H14.1] for syntax.

   Example of use:
      Accept: application/rtsl, application/sdp;level=2

12.2 Accept-Encoding

      See [H14.3]

12.3 Accept-Language

   See [H14.4]. Note that the language specified applies to the
   presentation description and any reason phrases, not the media
   content.

12.4 Allow

   The Allow response header field lists the methods supported by the
   resource identified by the request-URI. The purpose of this field is
   to strictly inform the recipient of valid methods associated with the
   resource. An Allow header field must be present in a 405 (Method not
   allowed) response.

   Example of use:
      Allow: SETUP, PLAY, RECORD, SET_PARAMETER

12.5 Authorization

      See [H14.8]

12.6 Bandwidth

   The Bandwidth request header field describes the estimated bandwidth
   available to the client, expressed as a positive integer and measured
   in bits per second. The bandwidth available to the client may change
   during an RTSP session, e.g., due to modem retraining.

```
   Bandwidth = "Bandwidth" ":" 1*DIGIT
```

```
   Example:
     Bandwidth: 4000
```

12.7 Blocksize

   This request header field is sent from the client to the media server
   asking the server for a particular media packet size. This packet
   size does not include lower-layer headers such as IP, UDP, or RTP.
   The server is free to use a blocksize which is lower than the one
   requested. The server MAY truncate this packet size to the closest
   multiple of the minimum, media-specific block size, or override it
   with the media-specific size if necessary. The block size MUST be a
   positive decimal number, measured in octets. The server only returns
   an error (416) if the value is syntactically invalid.

12.8 Cache-Control

   The Cache-Control general header field is used to specify directives
   that MUST be obeyed by all caching mechanisms along the
   request/response chain.

   Cache directives must be passed through by a proxy or gateway
   application, regardless of their significance to that application,
   since the directives may be applicable to all recipients along the
   request/response chain. It is not possible to specify a cache-
   directive for a specific cache.

   Cache-Control should only be specified in a SETUP request and its
   response. Note: Cache-Control does not govern the caching of
   responses as for HTTP, but rather of the stream identified by the
   SETUP request. Responses to RTSP requests are not cacheable, except
   for responses to DESCRIBE.

```
   Cache-Control             =    "Cache-Control" ":" 1#cache-directive
   cache-directive           =    cache-request-directive
                                |  cache-response-directive
   cache-request-directive   =    "no-cache"
                                |  "max-stale"
                                |  "min-fresh"
                                |  "only-if-cached"
                                |  cache-extension
   cache-response-directive  =    "public"
                                |  "private"
                                |  "no-cache"
                                |  "no-transform"
                                |  "must-revalidate"
```

```
                          |    "proxy-revalidate"
                          |    "max-age" "=" delta-seconds
                          |    cache-extension
     cache-extension      =    token [ "=" ( token | quoted-string ) ]
```

   no-cache:
          Indicates that the media stream MUST NOT be cached anywhere.
          This allows an origin server to prevent caching even by caches
          that have been configured to return stale responses to client
          requests.

   public:
          Indicates that the media stream is cacheable by any cache.

   private:
          Indicates that the media stream is intended for a single user
          and MUST NOT be cached by a shared cache. A private (non-
          shared) cache may cache the media stream.

   no-transform:
          An intermediate cache (proxy) may find it useful to convert
          the media type of a certain stream. A proxy might, for
          example, convert between video formats to save cache space or
          to reduce the amount of traffic on a slow link. Serious
          operational problems may occur, however, when these
          transformations have been applied to streams intended for
          certain kinds of applications. For example, applications for
          medical imaging, scientific data analysis and those using
          end-to-end authentication all depend on receiving a stream
          that is bit-for-bit identical to the original entity-body.
          Therefore, if a response includes the no-transform directive,
          an intermediate cache or proxy MUST NOT change the encoding of
          the stream. Unlike HTTP, RTSP does not provide for partial
          transformation at this point, e.g., allowing translation into
          a different language.

   only-if-cached:
          In some cases, such as times of extremely poor network
          connectivity, a client may want a cache to return only those
          media streams that it currently has stored, and not to receive
          these from the origin server. To do this, the client may
          include the only-if-cached directive in a request. If it
          receives this directive, a cache SHOULD either respond using a
          cached media stream that is consistent with the other
          constraints of the request, or respond with a 504 (Gateway
          Timeout) status. However, if a group of caches is being
          operated as a unified system with good internal connectivity,
          such a request MAY be forwarded within that group of caches.

max-stale:
        Indicates that the client is willing to accept a media stream
        that has exceeded its expiration time. If max-stale is
        assigned a value, then the client is willing to accept a
        response that has exceeded its expiration time by no more than
        the specified number of seconds. If no value is assigned to
        max-stale, then the client is willing to accept a stale
        response of any age.

min-fresh:
        Indicates that the client is willing to accept a media stream
        whose freshness lifetime is no less than its current age plus
        the specified time in seconds. That is, the client wants a
        response that will still be fresh for at least the specified
        number of seconds.

must-revalidate:
        When the must-revalidate directive is present in a SETUP
        response received by a cache, that cache MUST NOT use the
        entry after it becomes stale to respond to a subsequent
        request without first revalidating it with the origin server.
        That is, the cache must do an end-to-end revalidation every
        time, if, based solely on the origin server's Expires, the
        cached response is stale.)

12.9 Conference

This request header field establishes a logical connection between a
pre-established conference and an RTSP stream. The conference-id must
not be changed for the same RTSP session.

Conference = "Conference" ":" conference-id Example:
  Conference: 199702170042.SAA08642@obiwan.arl.wustl.edu%20Starr

A response code of 452 (452 Conference Not Found) is returned if the
conference-id is not valid.

12.10 Connection

See [H14.10]

12.11 Content-Base

See [H14.11]

12.12 Content-Encoding

See [H14.12]

12.13 Content-Language

   See [H14.13]

12.14 Content-Length

   This field contains the length of the content of the method (i.e.
   after the double CRLF following the last header). Unlike HTTP, it
   MUST be included in all messages that carry content beyond the header
   portion of the message. If it is missing, a default value of zero is
   assumed. It is interpreted according to [H14.14].

12.15 Content-Location

   See [H14.15]

12.16 Content-Type

   See [H14.18]. Note that the content types suitable for RTSP are
   likely to be restricted in practice to presentation descriptions and
   parameter-value types.

12.17 CSeq

   The CSeq field specifies the sequence number for an RTSP request-
   response pair. This field MUST be present in all requests and
   responses. For every RTSP request containing the given sequence
   number, there will be a corresponding response having the same
   number. Any retransmitted request must contain the same sequence
   number as the original (i.e. the sequence number is not incremented
   for retransmissions of the same request).

12.18 Date

   See [H14.19].

12.19 Expires

   The Expires entity-header field gives a date and time after which the
   description or media-stream should be considered stale. The
   interpretation depends on the method:

   DESCRIBE response:
        The Expires header indicates a date and time after which the
        description should be considered stale.

A stale cache entry may not normally be returned by a cache (either a
proxy cache or an user agent cache) unless it is first validated with
the origin server (or with an intermediate cache that has a fresh
copy of the entity). See section 13 for further discussion of the
expiration model.

The presence of an Expires field does not imply that the original
resource will change or cease to exist at, before, or after that
time.

The format is an absolute date and time as defined by HTTP-date in
[H3.3]; it MUST be in RFC1123-date format:

Expires = "Expires" ":" HTTP-date

An example of its use is

   Expires: Thu, 01 Dec 1994 16:00:00 GMT

RTSP/1.0 clients and caches MUST treat other invalid date formats,
especially including the value "0", as having occurred in the past
(i.e., "already expired").

To mark a response as "already expired," an origin server should use
an Expires date that is equal to the Date header value. To mark a
response as "never expires," an origin server should use an Expires
date approximately one year from the time the response is sent.
RTSP/1.0 servers should not send Expires dates more than one year in
the future.

The presence of an Expires header field with a date value of some
time in the future on a media stream that otherwise would by default
be non-cacheable indicates that the media stream is cacheable, unless
indicated otherwise by a Cache-Control header field (Section 12.8).

12.20 From

See [H14.22].

12.21 Host

This HTTP request header field is not needed for RTSP. It should be
silently ignored if sent.

12.22 If-Match

See [H14.25].

This field is especially useful for ensuring the integrity of the
presentation description, in both the case where it is fetched via
means external to RTSP (such as HTTP), or in the case where the
server implementation is guaranteeing the integrity of the
description between the time of the DESCRIBE message and the SETUP
message.

The identifier is an opaque identifier, and thus is not specific to
any particular session description language.

12.23 If-Modified-Since

The If-Modified-Since request-header field is used with the DESCRIBE
and SETUP methods to make them conditional. If the requested variant
has not been modified since the time specified in this field, a
description will not be returned from the server (DESCRIBE) or a
stream will not be set up (SETUP). Instead, a 304 (not modified)
response will be returned without any message-body.

    If-Modified-Since = "If-Modified-Since" ":" HTTP-date

An example of the field is:

    If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

12.24 Last-Modified

The Last-Modified entity-header field indicates the date and time at
which the origin server believes the presentation description or
media stream was last modified. See [H14.29]. For the methods
DESCRIBE or ANNOUNCE, the header field indicates the last
modification date and time of the description, for SETUP that of the
media stream.

12.25 Location

    See [H14.30].

12.26 Proxy-Authenticate

    See [H14.33].

12.27 Proxy-Require

The Proxy-Require header is used to indicate proxy-sensitive features
that MUST be supported by the proxy. Any Proxy-Require header
features that are not supported by the proxy MUST be negatively
acknowledged by the proxy to the client if not supported. Servers

should treat this field identically to the Require field.

See Section 12.32 for more details on the mechanics of this message
and a usage example.

12.28 Public

See [H14.35].

12.29 Range

This request and response header field specifies a range of time.
The range can be specified in a number of units. This specification
defines the smpte (Section 3.5), npt (Section 3.6), and clock
(Section 3.7) range units. Within RTSP, byte ranges [H14.36.1] are
not meaningful and MUST NOT be used. The header may also contain a
time parameter in UTC, specifying the time at which the operation is
to be made effective. Servers supporting the Range header MUST
understand the NPT range format and SHOULD understand the SMPTE range
format. The Range response header indicates what range of time is
actually being played or recorded. If the Range header is given in a
time format that is not understood, the recipient should return "501
Not Implemented".

Ranges are half-open intervals, including the lower point, but
excluding the upper point. In other words, a range of a-b starts
exactly at time a, but stops just before b. Only the start time of a
media unit such as a video or audio frame is relevant. As an example,
assume that video frames are generated every 40 ms. A range of 10.0-
10.1 would include a video frame starting at 10.0 or later time and
would include a video frame starting at 10.08, even though it lasted
beyond the interval. A range of 10.0-10.08, on the other hand, would
exclude the frame at 10.08.

```
Range             = "Range" ":" 1\#ranges-specifier
                       [ ";" "time" "=" utc-time ]
ranges-specifier = npt-range | utc-range | smpte-range
```

Example:
   Range: clock=19960213T143205Z-;time=19970123T143720Z

The notation is similar to that used for the HTTP/1.1 [2] byte-
range header. It allows clients to select an excerpt from the media
object, and to play from a given point to the end as well as from
the current location to a given point. The start of playback can be
scheduled for any time in the future, although a server may refuse
to keep server resources for extended idle periods.

12.30 Referer

   See [H14.37]. The URL refers to that of the presentation description,
   typically retrieved via HTTP.

12.31 Retry-After

   See [H14.38].

12.32 Require

   The Require header is used by clients to query the server about
   options that it may or may not support. The server MUST respond to
   this header by using the Unsupported header to negatively acknowledge
   those options which are NOT supported.

     This is to make sure that the client-server interaction will
     proceed without delay when all options are understood by both
     sides, and only slow down if options are not understood (as in the
     case above). For a well-matched client-server pair, the interaction
     proceeds quickly, saving a round-trip often required by negotiation
     mechanisms. In addition, it also removes state ambiguity when the
     client requires features that the server does not understand.

   Require =   "Require" ":" 1#option-tag

   Example:
     C->S: SETUP rtsp://server.com/foo/bar/baz.rm RTSP/1.0
             CSeq: 302
             Require: funky-feature
             Funky-Parameter: funkystuff

     S->C: RTSP/1.0 551 Option not supported
             CSeq: 302
             Unsupported: funky-feature

     C->S: SETUP rtsp://server.com/foo/bar/baz.rm RTSP/1.0
             CSeq: 303

     S->C: RTSP/1.0 200 OK
             CSeq: 303

   In this example, "funky-feature" is the feature tag which indicates
   to the client that the fictional Funky-Parameter field is required.
   The relationship between "funky-feature" and Funky-Parameter is not
   communicated via the RTSP exchange, since that relationship is an
   immutable property of "funky-feature" and thus should not be
   transmitted with every exchange.

Proxies and other intermediary devices SHOULD ignore features that are not understood in this field. If a particular extension requires that intermediate devices support it, the extension should be tagged in the Proxy-Require field instead (see Section 12.27).

12.33 RTP-Info

This field is used to set RTP-specific parameters in the PLAY response.

url:
      Indicates the stream URL which for which the following RTP parameters correspond.

seq:
      Indicates the sequence number of the first packet of the stream. This allows clients to gracefully deal with packets when seeking. The client uses this value to differentiate packets that originated before the seek from packets that originated after the seek.

rtptime:
      Indicates the RTP timestamp corresponding to the time value in the Range response header. (Note: For aggregate control, a particular stream may not actually generate a packet for the Range time value returned or implied. Thus, there is no guarantee that the packet with the sequence number indicated by seq actually has the timestamp indicated by rtptime.) The client uses this value to calculate the mapping of RTP time to NPT.

A mapping from RTP timestamps to NTP timestamps (wall clock) is available via RTCP. However, this information is not sufficient to generate a mapping from RTP timestamps to NPT. Furthermore, in order to ensure that this information is available at the necessary time (immediately at startup or after a seek), and that it is delivered reliably, this mapping is placed in the RTSP control channel.

In order to compensate for drift for long, uninterrupted presentations, RTSP clients should additionally map NPT to NTP, using initial RTCP sender reports to do the mapping, and later reports to check drift against the mapping.

   Syntax:

   RTP-Info   = "RTP-Info" ":" 1#stream-url 1*parameter
   stream-url   = "url" "=" url
   parameter   = ";" "seq" "=" 1*DIGIT
                     | ";" "rtptime" "=" 1*DIGIT

   Example:

     RTP-Info: url=rtsp://foo.com/bar.avi/streamid=0;seq=45102,
               url=rtsp://foo.com/bar.avi/streamid=1;seq=30211

12.34 Scale

   A scale value of 1 indicates normal play or record at the normal
   forward viewing rate. If not 1, the value corresponds to the rate
   with respect to normal viewing rate. For example, a ratio of 2
   indicates twice the normal viewing rate ("fast forward") and a ratio
   of 0.5 indicates half the normal viewing rate. In other words, a
   ratio of 2 has normal play time increase at twice the wallclock rate.
   For every second of elapsed (wallclock) time, 2 seconds of content
   will be delivered. A negative value indicates reverse direction.

   Unless requested otherwise by the Speed parameter, the data rate
   SHOULD not be changed. Implementation of scale changes depends on the
   server and media type. For video, a server may, for example, deliver
   only key frames or selected key frames. For audio, it may time-scale
   the audio while preserving pitch or, less desirably, deliver
   fragments of audio.

   The server should try to approximate the viewing rate, but may
   restrict the range of scale values that it supports. The response
   MUST contain the actual scale value chosen by the server.

   If the request contains a Range parameter, the new scale value will
   take effect at that time.

   Scale = "Scale" ":" ["-"] 1*DIGIT ["." *DIGIT]

   Example of playing in reverse at 3.5 times normal rate:

     Scale: -3.5

12.35 Speed

   This request header fields parameter requests the server to deliver
   data to the client at a particular speed, contingent on the server's
   ability and desire to serve the media stream at the given speed.
   Implementation by the server is OPTIONAL. The default is the bit rate
   of the stream.

   The parameter value is expressed as a decimal ratio, e.g., a value of
   2.0 indicates that data is to be delivered twice as fast as normal. A
   speed of zero is invalid. If the request contains a Range parameter,
   the new speed value will take effect at that time.

   Speed = "Speed" ":" 1*DIGIT ["." *DIGIT]

   Example:
     Speed: 2.5

   Use of this field changes the bandwidth used for data delivery. It is
   meant for use in specific circumstances where preview of the
   presentation at a higher or lower rate is necessary. Implementors
   should keep in mind that bandwidth for the session may be negotiated
   beforehand (by means other than RTSP), and therefore re-negotiation
   may be necessary. When data is delivered over UDP, it is highly
   recommended that means such as RTCP be used to track packet loss
   rates.

12.36 Server

   See [H14.39]

12.37 Session

   This request and response header field identifies an RTSP session
   started by the media server in a SETUP response and concluded by
   TEARDOWN on the presentation URL. The session identifier is chosen by
   the media server (see Section 3.4). Once a client receives a Session
   identifier, it MUST return it for any request related to that
   session. A server does not have to set up a session identifier if it
   has other means of identifying a session, such as dynamically
   generated URLs.

 Session = "Session" ":" session-id [";" "timeout" "=" delta-seconds]

   The timeout parameter is only allowed in a response header. The
   server uses it to indicate to the client how long the server is
   prepared to wait between RTSP commands before closing the session due
   to lack of activity (see Section A). The timeout is measured in

seconds, with a default of 60 seconds (1 minute).

Note that a session identifier identifies a RTSP session across
transport sessions or connections. Control messages for more than one
RTSP URL may be sent within a single RTSP session. Hence, it is
possible that clients use the same session for controlling many
streams constituting a presentation, as long as all the streams come
from the same server. (See example in Section 14). However, multiple
"user" sessions for the same URL from the same client MUST use
different session identifiers.

   The session identifier is needed to distinguish several delivery
   requests for the same URL coming from the same client.

The response 454 (Session Not Found) is returned if the session
identifier is invalid.

12.38 Timestamp

The timestamp general header describes when the client sent the
request to the server. The value of the timestamp is of significance
only to the client and may use any timescale. The server MUST echo
the exact same value and MAY, if it has accurate information about
this, add a floating point number indicating the number of seconds
that has elapsed since it has received the request. The timestamp is
used by the client to compute the round-trip time to the server so
that it can adjust the timeout value for retransmissions.

    Timestamp = "Timestamp" ":" *(DIGIT) ["." *(DIGIT)] [delay]
    delay   = *(DIGIT) ["." *(DIGIT)]

12.39 Transport

This request header indicates which transport protocol is to be used
and configures its parameters such as destination address,
compression, multicast time-to-live and destination port for a single
stream. It sets those values not already determined by a presentation
description.

Transports are comma separated, listed in order of preference.
Parameters may be added to each transport, separated by a semicolon.

The Transport header MAY also be used to change certain transport
parameters. A server MAY refuse to change parameters of an existing
stream.

The server MAY return a Transport response header in the response to
indicate the values actually chosen.

A Transport request header field may contain a list of transport
options acceptable to the client. In that case, the server MUST
return a single option which was actually chosen.

The syntax for the transport specifier is

    transport/profile/lower-transport.

The default value for the "lower-transport" parameters is specific to
the profile. For RTP/AVP, the default is UDP.

Below are the configuration parameters associated with transport:

General parameters:

unicast | multicast:
        mutually exclusive indication of whether unicast or multicast
        delivery will be attempted. Default value is multicast.
        Clients that are capable of handling both unicast and
        multicast transmission MUST indicate such capability by
        including two full transport-specs with separate parameters
        for each.

destination:
        The address to which a stream will be sent. The client may
        specify the multicast address with the destination parameter.
        To avoid becoming the unwitting perpetrator of a remote-
        controlled denial-of-service attack, a server SHOULD
        authenticate the client and SHOULD log such attempts before
        allowing the client to direct a media stream to an address not
        chosen by the server. This is particularly important if RTSP
        commands are issued via UDP, but implementations cannot rely
        on TCP as reliable means of client identification by itself. A
        server SHOULD not allow a client to direct media streams to an
        address that differs from the address commands are coming
        from.

source:
        If the source address for the stream is different than can be
        derived from the RTSP endpoint address (the server in playback
        or the client in recording), the source MAY be specified.

    This information may also be available through SDP. However, since
    this is more a feature of transport than media initialization, the
    authoritative source for this information should be in the SETUP
    response.

   layers:
        The number of multicast layers to be used for this media
        stream. The layers are sent to consecutive addresses starting
        at the destination address.

   mode:
        The mode parameter indicates the methods to be supported for
        this session. Valid values are PLAY and RECORD. If not
        provided, the default is PLAY.

   append:
        If the mode parameter includes RECORD, the append parameter
        indicates that the media data should append to the existing
        resource rather than overwrite it. If appending is requested
        and the server does not support this, it MUST refuse the
        request rather than overwrite the resource identified by the
        URI. The append parameter is ignored if the mode parameter
        does not contain RECORD.

   interleaved:
        The interleaved parameter implies mixing the media stream with
        the control stream in whatever protocol is being used by the
        control stream, using the mechanism defined in Section 10.12.
        The argument provides the channel number to be used in the $
        statement. This parameter may be specified as a range, e.g.,
        interleaved=4-5 in cases where the transport choice for the
        media stream requires it.

     This allows RTP/RTCP to be handled similarly to the way that it is
     done with UDP, i.e., one channel for RTP and the other for RTCP.

   Multicast specific:

   ttl:
        multicast time-to-live

   RTP Specific:

   port:
        This parameter provides the RTP/RTCP port pair for a multicast
        session. It is specified as a range, e.g., port=3456-3457.

   client_port:
        This parameter provides the unicast RTP/RTCP port pair on
        which the client has chosen to receive media data and control
        information. It is specified as a range, e.g.,
        client_port=3456-3457.

   server_port:
        This parameter provides the unicast RTP/RTCP port pair on
        which the server has chosen to receive media data and control
        information. It is specified as a range, e.g.,
        server_port=3456-3457.

   ssrc:
        The ssrc parameter indicates the RTP SSRC [24, Sec. 3] value
        that should be (request) or will be (response) used by the
        media server. This parameter is only valid for unicast
        transmission. It identifies the synchronization source to be
        associated with the media stream.

```
Transport             =     "Transport" ":"
                            1\#transport-spec
transport-spec        =     transport-protocol/profile[/lower-transport]
                            *parameter
transport-protocol    =     "RTP"
profile               =     "AVP"
lower-transport       =     "TCP" | "UDP"
parameter             =     ( "unicast" | "multicast" )
                      |     ";" "destination" [ "=" address ]
                      |     ";" "interleaved" "=" channel [ "-" channel ]
                      |     ";" "append"
                      |     ";" "ttl" "=" ttl
                      |     ";" "layers" "=" 1*DIGIT
                      |     ";" "port" "=" port [ "-" port ]
                      |     ";" "client_port" "=" port [ "-" port ]
                      |     ";" "server_port" "=" port [ "-" port ]
                      |     ";" "ssrc" "=" ssrc
                      |     ";" "mode" = <"> 1\#mode <">
ttl                   =     1*3(DIGIT)
port                  =     1*5(DIGIT)
ssrc                  =     8*8(HEX)
channel               =     1*3(DIGIT)
address               =     host
mode                  =     <"> *Method <"> | Method
```


   Example:
     Transport: RTP/AVP;multicast;ttl=127;mode="PLAY",
             RTP/AVP;unicast;client_port=3456-3457;mode="PLAY"

   The Transport header is restricted to describing a single RTP
   stream. (RTSP can also control multiple streams as a single
   entity.) Making it part of RTSP rather than relying on a multitude
   of session description formats greatly simplifies designs of
   firewalls.

12.40 Unsupported

   The Unsupported response header lists the features not supported by
   the server. In the case where the feature was specified via the
   Proxy-Require field (Section 12.32), if there is a proxy on the path
   between the client and the server, the proxy MUST insert a message
   reply with an error message "551 Option Not Supported".

   See Section 12.32 for a usage example.

12.41 User-Agent

   See [H14.42]

12.42 Vary

   See [H14.43]

12.43 Via

   See [H14.44].

12.44 WWW-Authentica

   See [H14.46].

13 Caching

   In HTTP, response-request pairs are cached. RTSP differs
   significantly in that respect. Responses are not cacheable, with the
   exception of the presentation description returned by DESCRIBE or
   included with ANNOUNCE. (Since the responses for anything but
   DESCRIBE and GET_PARAMETER do not return any data, caching is not
   really an issue for these requests.) However, it is desirable for the
   continuous media data, typically delivered out-of-band with respect
   to RTSP, to be cached, as well as the session description.

   On receiving a SETUP or PLAY request, a proxy ascertains whether it
   has an up-to-date copy of the continuous media content and its
   description. It can determine whether the copy is up-to-date by
   issuing a SETUP or DESCRIBE request, respectively, and comparing the
   Last-Modified header with that of the cached copy. If the copy is not
   up-to-date, it modifies the SETUP transport parameters as appropriate
   and forwards the request to the origin server. Subsequent control
   commands such as PLAY or PAUSE then pass the proxy unmodified. The
   proxy delivers the continuous media data to the client, while
   possibly making a local copy for later reuse. The exact behavior
   allowed to the cache is given by the cache-response directives

described in Section 12.8. A cache MUST answer any DESCRIBE requests
if it is currently serving the stream to the requestor, as it is
possible that low-level details of the stream description may have
changed on the origin-server.

Note that an RTSP cache, unlike the HTTP cache, is of the "cut-
through" variety. Rather than retrieving the whole resource from the
origin server, the cache simply copies the streaming data as it
passes by on its way to the client. Thus, it does not introduce
additional latency.

To the client, an RTSP proxy cache appears like a regular media
server, to the media origin server like a client. Just as an HTTP
cache has to store the content type, content language, and so on for
the objects it caches, a media cache has to store the presentation
description. Typically, a cache eliminates all transport-references
(that is, multicast information) from the presentation description,
since these are independent of the data delivery from the cache to
the client. Information on the encodings remains the same. If the
cache is able to translate the cached media data, it would create a
new presentation description with all the encoding possibilities it
can offer.

14 Examples

   The following examples refer to stream description formats that are
   not standards, such as RTSL. The following examples are not to be
   used as a reference for those formats.

14.1 Media on Demand (Unicast)

   Client C requests a movie from media servers A (audio.example.com)
   and V (video.example.com). The media description is stored on a web
   server W . The media description contains descriptions of the
   presentation and all its streams, including the codecs that are
   available, dynamic RTP payload types, the protocol stack, and content
   information such as language or copyright restrictions. It may also
   give an indication about the timeline of the movie.

   In this example, the client is only interested in the last part of
   the movie.

      C->W: GET /twister.sdp HTTP/1.1
            Host: www.example.com
            Accept: application/sdp

      W->C: HTTP/1.0 200 OK
            Content-Type: application/sdp

```
        v=0
        o=- 2890844526 2890842807 IN IP4 192.16.24.202
        s=RTSP Session
        m=audio 0 RTP/AVP 0
        a=control:rtsp://audio.example.com/twister/audio.en
        m=video 0 RTP/AVP 31
        a=control:rtsp://video.example.com/twister/video

   C->A: SETUP rtsp://audio.example.com/twister/audio.en RTSP/1.0
         CSeq: 1
         Transport: RTP/AVP/UDP;unicast;client_port=3056-3057

   A->C: RTSP/1.0 200 OK
         CSeq: 1
         Session: 12345678
         Transport: RTP/AVP/UDP;unicast;client_port=3056-3057;
                    server_port=5000-5001

   C->V: SETUP rtsp://video.example.com/twister/video RTSP/1.0
         CSeq: 1
         Transport: RTP/AVP/UDP;unicast;client_port=3058-3059

   V->C: RTSP/1.0 200 OK
         CSeq: 1
         Session: 23456789
         Transport: RTP/AVP/UDP;unicast;client_port=3058-3059;
                    server_port=5002-5003

   C->V: PLAY rtsp://video.example.com/twister/video RTSP/1.0
         CSeq: 2
         Session: 23456789
         Range: smpte=0:10:00-

   V->C: RTSP/1.0 200 OK
         CSeq: 2
         Session: 23456789
         Range: smpte=0:10:00-0:20:00
         RTP-Info: url=rtsp://video.example.com/twister/video;
           seq=12312232;rtptime=78712811

   C->A: PLAY rtsp://audio.example.com/twister/audio.en RTSP/1.0
         CSeq: 2
         Session: 12345678
         Range: smpte=0:10:00-

   A->C: RTSP/1.0 200 OK
         CSeq: 2
         Session: 12345678
```

```
          Range: smpte=0:10:00-0:20:00
          RTP-Info: url=rtsp://audio.example.com/twister/audio.en;
            seq=876655;rtptime=1032181
```

```
  C->A: TEARDOWN rtsp://audio.example.com/twister/audio.en RTSP/1.0
        CSeq: 3
        Session: 12345678
```

```
  A->C: RTSP/1.0 200 OK
        CSeq: 3
```

```
  C->V: TEARDOWN rtsp://video.example.com/twister/video RTSP/1.0
        CSeq: 3
        Session: 23456789
```

```
  V->C: RTSP/1.0 200 OK
        CSeq: 3
```

Even though the audio and video track are on two different servers,
and may start at slightly different times and may drift with respect
to each other, the client can synchronize the two using standard RTP
methods, in particular the time scale contained in the RTCP sender
reports.

14.2 Streaming of a Container file

For purposes of this example, a container file is a storage entity in
which multiple continuous media types pertaining to the same end-user
presentation are present. In effect, the container file represents an
RTSP presentation, with each of its components being RTSP streams.
Container files are a widely used means to store such presentations.
While the components are transported as independent streams, it is
desirable to maintain a common context for those streams at the
server end.

   This enables the server to keep a single storage handle open
   easily. It also allows treating all the streams equally in case of
   any prioritization of streams by the server.

It is also possible that the presentation author may wish to prevent
selective retrieval of the streams by the client in order to preserve
the artistic effect of the combined media presentation. Similarly, in
such a tightly bound presentation, it is desirable to be able to
control all the streams via a single control message using an
aggregate URL.

The following is an example of using a single RTSP session to control
multiple streams. It also illustrates the use of aggregate URLs.

   Client C requests a presentation from media server M . The movie is
   stored in a container file. The client has obtained an RTSP URL to
   the container file.

```
   C->M: DESCRIBE rtsp://foo/twister RTSP/1.0
         CSeq: 1

   M->C: RTSP/1.0 200 OK
         CSeq: 1
         Content-Type: application/sdp
         Content-Length: 164

         v=0
         o=- 2890844256 2890842807 IN IP4 172.16.2.93
         s=RTSP Session
         i=An Example of RTSP Session Usage
         a=control:rtsp://foo/twister
         t=0 0
         m=audio 0 RTP/AVP 0
         a=control:rtsp://foo/twister/audio
         m=video 0 RTP/AVP 26
         a=control:rtsp://foo/twister/video

   C->M: SETUP rtsp://foo/twister/audio RTSP/1.0
         CSeq: 2
         Transport: RTP/AVP;unicast;client_port=8000-8001

   M->C: RTSP/1.0 200 OK
         CSeq: 2
         Transport: RTP/AVP;unicast;client_port=8000-8001;
                    server_port=9000-9001
         Session: 12345678

   C->M: SETUP rtsp://foo/twister/video RTSP/1.0
         CSeq: 3
         Transport: RTP/AVP;unicast;client_port=8002-8003
         Session: 12345678

   M->C: RTSP/1.0 200 OK
         CSeq: 3
         Transport: RTP/AVP;unicast;client_port=8002-8003;
                    server_port=9004-9005
         Session: 12345678

   C->M: PLAY rtsp://foo/twister RTSP/1.0
         CSeq: 4
         Range: npt=0-
         Session: 12345678
```

```
      M->C: RTSP/1.0 200 OK
            CSeq: 4
            Session: 12345678
            RTP-Info: url=rtsp://foo/twister/video;
              seq=9810092;rtptime=3450012

      C->M: PAUSE rtsp://foo/twister/video RTSP/1.0
            CSeq: 5
            Session: 12345678

      M->C: RTSP/1.0 460 Only aggregate operation allowed
            CSeq: 5

      C->M: PAUSE rtsp://foo/twister RTSP/1.0
            CSeq: 6
            Session: 12345678

      M->C: RTSP/1.0 200 OK
            CSeq: 6
            Session: 12345678

      C->M: SETUP rtsp://foo/twister RTSP/1.0
            CSeq: 7
            Transport: RTP/AVP;unicast;client_port=10000

      M->C: RTSP/1.0 459 Aggregate operation not allowed
            CSeq: 7
```

   In the first instance of failure, the client tries to pause one
   stream (in this case video) of the presentation. This is disallowed
   for that presentation by the server. In the second instance, the
   aggregate URL may not be used for SETUP and one control message is
   required per stream to set up transport parameters.

      This keeps the syntax of the Transport header simple and allows
      easy parsing of transport information by firewalls.

14.3 Single Stream Container Files

   Some RTSP servers may treat all files as though they are "container
   files", yet other servers may not support such a concept. Because of
   this, clients SHOULD use the rules set forth in the session
   description for request URLs, rather than assuming that a consistent
   URL may always be used throughout. Here's an example of how a multi-
   stream server might expect a single-stream file to be served:

           Accept: application/x-rtsp-mh, application/sdp

```
        CSeq: 1

   S->C  RTSP/1.0 200 OK
        CSeq: 1
        Content-base: rtsp://foo.com/test.wav/
        Content-type: application/sdp
        Content-length: 48

        v=0
        o=- 872653257 872653257 IN IP4 172.16.2.187
        s=mu-law wave file
        i=audio test
        t=0 0
        m=audio 0 RTP/AVP 0
        a=control:streamid=0

   C->S  SETUP rtsp://foo.com/test.wav/streamid=0 RTSP/1.0
        Transport: RTP/AVP/UDP;unicast;
                   client_port=6970-6971;mode=play
        CSeq: 2

   S->C  RTSP/1.0 200 OK
        Transport: RTP/AVP/UDP;unicast;client_port=6970-6971;
                   server_port=6970-6971;mode=play
        CSeq: 2
        Session: 2034820394

   C->S  PLAY rtsp://foo.com/test.wav RTSP/1.0
        CSeq: 3
        Session: 2034820394

   S->C  RTSP/1.0 200 OK
        CSeq: 3
        Session: 2034820394
        RTP-Info: url=rtsp://foo.com/test.wav/streamid=0;
          seq=981888;rtptime=3781123
```

   Note the different URL in the SETUP command, and then the switch back
   to the aggregate URL in the PLAY command. This makes complete sense
   when there are multiple streams with aggregate control, but is less
   than intuitive in the special case where the number of streams is
   one.

   In this special case, it is recommended that servers be forgiving of
   implementations that send:

```
   C->S  PLAY rtsp://foo.com/test.wav/streamid=0 RTSP/1.0
        CSeq: 3
```

   In the worst case, servers should send back:

    S->C RTSP/1.0 460 Only aggregate operation allowed
          CSeq: 3

   One would also hope that server implementations are also forgiving of
   the following:

    C->S SETUP rtsp://foo.com/test.wav RTSP/1.0
          Transport: rtp/avp/udp;client_port=6970-6971;mode=play
          CSeq: 2

   Since there is only a single stream in this file, it's not ambiguous
   what this means.

14.4 Live Media Presentation Using Multicast

   The media server M chooses the multicast address and port. Here, we
   assume that the web server only contains a pointer to the full
   description, while the media server M maintains the full description.

      C->W: GET /concert.sdp HTTP/1.1
            Host: www.example.com

      W->C: HTTP/1.1 200 OK
            Content-Type: application/x-rtsl

            <session>
              <track src="rtsp://live.example.com/concert/audio">
            </session>

      C->M: DESCRIBE rtsp://live.example.com/concert/audio RTSP/1.0
            CSeq: 1

      M->C: RTSP/1.0 200 OK
            CSeq: 1
            Content-Type: application/sdp
            Content-Length: 44

            v=0
            o=- 2890844526 2890842807 IN IP4 192.16.24.202
            s=RTSP Session
            m=audio 3456 RTP/AVP 0
            a=control:rtsp://live.example.com/concert/audio
            c=IN IP4 224.2.0.1/16

      C->M: SETUP rtsp://live.example.com/concert/audio RTSP/1.0
            CSeq: 2

```
        Transport: RTP/AVP;multicast

  M->C: RTSP/1.0 200 OK
        CSeq: 2
        Transport: RTP/AVP;multicast;destination=224.2.0.1;
                   port=3456-3457;ttl=16
        Session: 0456804596

  C->M: PLAY rtsp://live.example.com/concert/audio RTSP/1.0
        CSeq: 3
        Session: 0456804596

  M->C: RTSP/1.0 200 OK
        CSeq: 3
        Session: 0456804596
```

14.5 Playing media into an existing session

   A conference participant C wants to have the media server M play back
   a demo tape into an existing conference. C indicates to the media
   server that the network addresses and encryption keys are already
   given by the conference, so they should not be chosen by the server.
   The example omits the simple ACK responses.

```
  C->M: DESCRIBE rtsp://server.example.com/demo/548/sound RTSP/1.0
        CSeq: 1
        Accept: application/sdp

  M->C: RTSP/1.0 200 1 OK
        Content-type: application/sdp
        Content-Length: 44

        v=0
        o=- 2890844526 2890842807 IN IP4 192.16.24.202
        s=RTSP Session
        i=See above
        t=0 0
        m=audio 0 RTP/AVP 0

  C->M: SETUP rtsp://server.example.com/demo/548/sound RTSP/1.0
        CSeq: 2
        Transport: RTP/AVP;multicast;destination=225.219.201.15;
                   port=7000-7001;ttl=127
        Conference: 199702170042.SAA08642@obiwan.arl.wustl.edu%20Starr

  M->C: RTSP/1.0 200 OK
        CSeq: 2
        Transport: RTP/AVP;multicast;destination=225.219.201.15;
```

```
                 port=7000-7001;ttl=127
        Session: 91389234234
        Conference: 199702170042.SAA08642@obiwan.arl.wustl.edu%20Starr

   C->M: PLAY rtsp://server.example.com/demo/548/sound RTSP/1.0
        CSeq: 3
        Session: 91389234234

   M->C: RTSP/1.0 200 OK
        CSeq: 3
```

14.6 Recording

   The conference participant client C asks the media server M to record
   the audio and video portions of a meeting. The client uses the
   ANNOUNCE method to provide meta-information about the recorded
   session to the server.

```
   C->M: ANNOUNCE rtsp://server.example.com/meeting RTSP/1.0
        CSeq: 90
        Content-Type: application/sdp
        Content-Length: 121

        v=0
        o=camera1 3080117314 3080118787 IN IP4 195.27.192.36
        s=IETF Meeting, Munich - 1
        i=The thirty-ninth IETF meeting will be held in Munich, Germany
        u=http://www.ietf.org/meetings/Munich.html
        e=IETF Channel 1 <ietf39-mbone@uni-koeln.de>
        p=IETF Channel 1 +49-172-2312 451
        c=IN IP4 224.0.1.11/127
        t=3080271600 3080703600
        a=tool:sdr v2.4a6
        a=type:test
        m=audio 21010 RTP/AVP 5
        c=IN IP4 224.0.1.11/127
        a=ptime:40
        m=video 61010 RTP/AVP 31
        c=IN IP4 224.0.1.12/127

   M->C: RTSP/1.0 200 OK
        CSeq: 90

   C->M: SETUP rtsp://server.example.com/meeting/audiotrack RTSP/1.0
        CSeq: 91
        Transport: RTP/AVP;multicast;destination=224.0.1.11;
                   port=21010-21011;mode=record;ttl=127
```

```
        M->C: RTSP/1.0 200 OK
              CSeq: 91
              Session: 50887676
              Transport: RTP/AVP;multicast;destination=224.0.1.11;
                         port=21010-21011;mode=record;ttl=127

        C->M: SETUP rtsp://server.example.com/meeting/videotrack RTSP/1.0
              CSeq: 92
              Session: 50887676
              Transport: RTP/AVP;multicast;destination=224.0.1.12;
                         port=61010-61011;mode=record;ttl=127

        M->C: RTSP/1.0 200 OK
              CSeq: 92
              Transport: RTP/AVP;multicast;destination=224.0.1.12;
                         port=61010-61011;mode=record;ttl=127

        C->M: RECORD rtsp://server.example.com/meeting RTSP/1.0
              CSeq: 93
              Session: 50887676
              Range: clock=19961110T1925-19961110T2015

        M->C: RTSP/1.0 200 OK
              CSeq: 93
```

15 Syntax

   The RTSP syntax is described in an augmented Backus-Naur form (BNF)
   as used in RFC 2068 [2].

15.1 Base Syntax

```
   OCTET             =       <any 8-bit sequence of data>
   CHAR              =       <any US-ASCII character (octets 0 - 127)>
   UPALPHA           =       <any US-ASCII uppercase letter "A".."Z">
   LOALPHA           =       <any US-ASCII lowercase letter "a".."z">
   ALPHA             =       UPALPHA | LOALPHA

   DIGIT             =       <any US-ASCII digit "0".."9">
   CTL               =       <any US-ASCII control character
                              (octets 0 - 31) and DEL (127)>
   CR                =       <US-ASCII CR, carriage return (13)>
   LF                =       <US-ASCII LF, linefeed (10)>

   SP                =       <US-ASCII SP, space (32)>
   HT                =       <US-ASCII HT, horizontal-tab (9)>
   <">               =       <US-ASCII double-quote mark (34)>
   CRLF              =       CR LF
```

```
   LWS              =       [CRLF] 1*( SP | HT )
   TEXT             =       <any OCTET except CTLs>
   tspecials        =       "(" | ")" | "<" | ">" | "@"
                        |    "," | ";" | ":" | "\" | <">
                        |    "/" | "[" | "]" | "?" | "="
                        |    "{" | "}" | SP | HT

   token            =       1*<any CHAR except CTLs or tspecials>
   quoted-string    =       ( <"> *(qdtext) <"> )
   qdtext           =       <any TEXT except <">>
   quoted-pair      =       "\" CHAR

   message-header   =       field-name ":" [ field-value ] CRLF
   field-name       =       token
   field-value      =       *( field-content | LWS )
   field-content    =       <the OCTETs making up the field-value and
                             consisting of either *TEXT or
                             combinations of token, tspecials, and
                             quoted-string>

   safe             =    "\$" | "-" | "_" | "." | "+"
   extra            =    "!" | "*" | "$'$" | "(" | ")" | ","

   hex              =    DIGIT | "A" | "B" | "C" | "D" | "E" | "F" |
                         "a" | "b" | "c" | "d" | "e" | "f"
   escape           =    "\%" hex hex
   reserved         =    ";" | "/" | "?" | ":" | "@" | "&" | "="

   unreserved       =    alpha | digit | safe | extra
   xchar            =    unreserved | reserved | escape
```

16 Security Considerations

   Because of the similarity in syntax and usage between RTSP servers
   and HTTP servers, the security considerations outlined in [H15]
   apply. Specifically, please note the following:

   Authentication Mechanisms:
         RTSP and HTTP share common authentication schemes, and thus
         should follow the same prescriptions with regards to
         authentication. See [H15.1] for client authentication issues,
         and [H15.2] for issues regarding support for multiple
         authentication mechanisms.

   Abuse of Server Log Information:
         RTSP and HTTP servers will presumably have similar logging
         mechanisms, and thus should be equally guarded in protecting
         the contents of those logs, thus protecting the privacy of the

users of the servers. See [H15.3] for HTTP server
recommendations regarding server logs.

Transfer of Sensitive Information:
       There is no reason to believe that information transferred via
       RTSP may be any less sensitive than that normally transmitted
       via HTTP. Therefore, all of the precautions regarding the
       protection of data privacy and user privacy apply to
       implementors of RTSP clients, servers, and proxies. See
       [H15.4] for further details.

Attacks Based On File and Path Names:
       Though RTSP URLs are opaque handles that do not necessarily
       have file system semantics, it is anticipated that many
       implementations will translate portions of the request URLs
       directly to file system calls. In such cases, file systems
       SHOULD follow the precautions outlined in [H15.5], such as
       checking for ".." in path components.

Personal Information:
       RTSP clients are often privy to the same information that HTTP
       clients are (user name, location, etc.) and thus should be
       equally. See [H15.6] for further recommendations.

Privacy Issues Connected to Accept Headers:
       Since may of the same "Accept" headers exist in RTSP as in
       HTTP, the same caveats outlined in [H15.7] with regards to
       their use should be followed.

DNS Spoofing:
       Presumably, given the longer connection times typically
       associated to RTSP sessions relative to HTTP sessions, RTSP
       client DNS optimizations should be less prevalent.
       Nonetheless, the recommendations provided in [H15.8] are still
       relevant to any implementation which attempts to rely on a
       DNS-to-IP mapping to hold beyond a single use of the mapping.

Location Headers and Spoofing:
       If a single server supports multiple organizations that do not
       trust one another, then it must check the values of Location
       and Content-Location headers in responses that are generated
       under control of said organizations to make sure that they do
       not attempt to invalidate resources over which they have no
       authority. ([H15.9])

In addition to the recommendations in the current HTTP specification
(RFC 2068 [2], as of this writing), future HTTP specifications may
provide additional guidance on security issues.

The following are added considerations for RTSP implementations.

Concentrated denial-of-service attack:
     The protocol offers the opportunity for a remote-controlled
     denial-of-service attack. The attacker may initiate traffic
     flows to one or more IP addresses by specifying them as the
     destination in SETUP requests. While the attacker's IP address
     may be known in this case, this is not always useful in
     prevention of more attacks or ascertaining the attackers
     identity. Thus, an RTSP server SHOULD only allow client-
     specified destinations for RTSP-initiated traffic flows if the
     server has verified the client's identity, either against a
     database of known users using RTSP authentication mechanisms
     (preferably digest authentication or stronger), or other
     secure means.

Session hijacking:
     Since there is no relation between a transport layer
     connection and an RTSP session, it is possible for a malicious
     client to issue requests with random session identifiers which
     would affect unsuspecting clients. The server SHOULD use a
     large, random and non-sequential session identifier to
     minimize the possibility of this kind of attack.

Authentication:
     Servers SHOULD implement both basic and digest [8]
     authentication. In environments requiring tighter security for
     the control messages, the RTSP control stream may be
     encrypted.

Stream issues:
     RTSP only provides for stream control. Stream delivery issues
     are not covered in this section, nor in the rest of this memo.
     RTSP implementations will most likely rely on other protocols
     such as RTP, IP multicast, RSVP and IGMP, and should address
     security considerations brought up in those and other
     applicable specifications.

Persistently suspicious behavior:
     RTSP servers SHOULD return error code 403 (Forbidden) upon
     receiving a single instance of behavior which is deemed a
     security risk. RTSP servers SHOULD also be aware of attempts
     to probe the server for weaknesses and entry points and MAY
     arbitrarily disconnect and ignore further requests clients
     which are deemed to be in violation of local security policy.

Appendix A: RTSP Protocol State Machines

   The RTSP client and server state machines describe the behavior of
   the protocol from RTSP session initialization through RTSP session
   termination.

   State is defined on a per object basis. An object is uniquely
   identified by the stream URL and the RTSP session identifier. Any
   request/reply using aggregate URLs denoting RTSP presentations
   composed of multiple streams will have an effect on the individual
   states of all the streams. For example, if the presentation /movie
   contains two streams, /movie/audio and /movie/video, then the
   following command:

     PLAY rtsp://foo.com/movie RTSP/1.0
     CSeq: 559
     Session: 12345678

   will have an effect on the states of movie/audio and movie/video.

     This example does not imply a standard way to represent streams in
     URLs or a relation to the filesystem. See Section 3.2.

   The requests OPTIONS, ANNOUNCE, DESCRIBE, GET_PARAMETER,
   SET_PARAMETER do not have any effect on client or server state and
   are therefore not listed in the state tables.

A.1 Client State Machine

   The client can assume the following states:

   Init:
         SETUP has been sent, waiting for reply.

   Ready:
         SETUP reply received or PAUSE reply received while in Playing
         state.

   Playing:
         PLAY reply received

   Recording:
         RECORD reply received

   In general, the client changes state on receipt of replies to
   requests. Note that some requests are effective at a future time or
   position (such as a PAUSE), and state also changes accordingly. If no
   explicit SETUP is required for the object (for example, it is

available via a multicast group), state begins at Ready. In this
case, there are only two states, Ready and Playing. The client also
changes state from Playing/Recording to Ready when the end of the
requested range is reached.

The "next state" column indicates the state assumed after receiving a
success response (2xx). If a request yields a status code of 3xx, the
state becomes Init, and a status code of 4xx yields no change in
state. Messages not listed for each state MUST NOT be issued by the
client in that state, with the exception of messages not affecting
state, as listed above. Receiving a REDIRECT from the server is
equivalent to receiving a 3xx redirect status from the server.

```
state          message sent      next state after response
Init           SETUP             Ready
               TEARDOWN          Init
Ready          PLAY              Playing
               RECORD            Recording
               TEARDOWN          Init
               SETUP             Ready
Playing        PAUSE             Ready
               TEARDOWN          Init
               PLAY              Playing
               SETUP             Playing (changed transport)
Recording      PAUSE             Ready
               TEARDOWN          Init
               RECORD            Recording
               SETUP             Recording (changed transport)
```

A.2 Server State Machine

   The server can assume the following states:

   Init:
         The initial state, no valid SETUP has been received yet.

   Ready:
         Last SETUP received was successful, reply sent or after
         playing, last PAUSE received was successful, reply sent.

   Playing:
         Last PLAY received was successful, reply sent. Data is being
         sent.

   Recording:
         The server is recording media data.

In general, the server changes state on receiving requests. If the
server is in state Playing or Recording and in unicast mode, it MAY
revert to Init and tear down the RTSP session if it has not received
"wellness" information, such as RTCP reports or RTSP commands, from
the client for a defined interval, with a default of one minute. The
server can declare another timeout value in the Session response
header (Section 12.37). If the server is in state Ready, it MAY
revert to Init if it does not receive an RTSP request for an interval
of more than one minute. Note that some requests (such as PAUSE) may
be effective at a future time or position, and server state changes
at the appropriate time. The server reverts from state Playing or
Recording to state Ready at the end of the range requested by the
client.

The REDIRECT message, when sent, is effective immediately unless it
has a Range header specifying when the redirect is effective. In such
a case, server state will also change at the appropriate time.

If no explicit SETUP is required for the object, the state starts at
Ready and there are only two states, Ready and Playing.

The "next state" column indicates the state assumed after sending a
success response (2xx). If a request results in a status code of 3xx,
the state becomes Init. A status code of 4xx results in no change.

```
   state            message received  next state
   Init             SETUP             Ready
                    TEARDOWN          Init
   Ready            PLAY              Playing
                    SETUP             Ready
                    TEARDOWN          Init
                    RECORD            Recording
   Playing          PLAY              Playing
                    PAUSE             Ready
                    TEARDOWN          Init
                    SETUP             Playing
   Recording        RECORD            Recording
                    PAUSE             Ready
                    TEARDOWN          Init
                    SETUP             Recording
```

Appendix B: Interaction with RTP

   RTSP allows media clients to control selected, non-contiguous
   sections of media presentations, rendering those streams with an RTP
   media layer[24]. The media layer rendering the RTP stream should not
   be affected by jumps in NPT. Thus, both RTP sequence numbers and RTP
   timestamps MUST be continuous and monotonic across jumps of NPT.

   As an example, assume a clock frequency of 8000 Hz, a packetization
   interval of 100 ms and an initial sequence number and timestamp of
   zero. First we play NPT 10 through 15, then skip ahead and play NPT
   18 through 20. The first segment is presented as RTP packets with
   sequence numbers 0 through 49 and timestamp 0 through 39,200. The
   second segment consists of RTP packets with sequence number 50
   through 69, with timestamps 40,000 through 55,200.

     We cannot assume that the RTSP client can communicate with the RTP
     media agent, as the two may be independent processes. If the RTP
     timestamp shows the same gap as the NPT, the media agent will
     assume that there is a pause in the presentation. If the jump in
     NPT is large enough, the RTP timestamp may roll over and the media
     agent may believe later packets to be duplicates of packets just
     played out.

     For certain datatypes, tight integration between the RTSP layer and
     the RTP layer will be necessary. This by no means precludes the
     above restriction. Combined RTSP/RTP media clients should use the
     RTP-Info field to determine whether incoming RTP packets were sent
     before or after a seek.

   For continuous audio, the server SHOULD set the RTP marker bit at the
   beginning of serving a new PLAY request. This allows the client to
   perform playout delay adaptation.

   For scaling (see Section 12.34), RTP timestamps should correspond to
   the playback timing. For example, when playing video recorded at 30
   frames/second at a scale of two and speed (Section 12.35) of one, the
   server would drop every second frame to maintain and deliver video
   packets with the normal timestamp spacing of 3,000 per frame, but NPT
   would increase by 1/15 second for each video frame.

   The client can maintain a correct display of NPT by noting the RTP
   timestamp value of the first packet arriving after repositioning. The
   sequence parameter of the RTP-Info (Section 12.33) header provides
   the first sequence number of the next segment.

Appendix C: Use of SDP for RTSP Session Descriptions

   The Session Description Protocol (SDP, RFC 2327 [6]) may be used to
   describe streams or presentations in RTSP. Such usage is limited to
   specifying means of access and encoding(s) for:

   aggregate control:
         A presentation composed of streams from one or more servers
         that are not available for aggregate control. Such a
         description is typically retrieved by HTTP or other non-RTSP
         means. However, they may be received with ANNOUNCE methods.

   non-aggregate control:
         A presentation composed of multiple streams from a single
         server that are available for aggregate control. Such a
         description is typically returned in reply to a DESCRIBE
         request on a URL, or received in an ANNOUNCE method.

   This appendix describes how an SDP file, retrieved, for example,
   through HTTP, determines the operation of an RTSP session. It also
   describes how a client should interpret SDP content returned in reply
   to a DESCRIBE request. SDP provides no mechanism by which a client
   can distinguish, without human guidance, between several media
   streams to be rendered simultaneously and a set of alternatives
   (e.g., two audio streams spoken in different languages).

C.1 Definitions

   The terms "session-level", "media-level" and other key/attribute
   names and values used in this appendix are to be used as defined in
   SDP (RFC 2327 [6]):

C.1.1 Control URL

   The "a=control:" attribute is used to convey the control URL. This
   attribute is used both for the session and media descriptions. If
   used for individual media, it indicates the URL to be used for
   controlling that particular media stream. If found at the session
   level, the attribute indicates the URL for aggregate control.

   Example:
     a=control:rtsp://example.com/foo

   This attribute may contain either relative and absolute URLs,
   following the rules and conventions set out in RFC 1808 [25].
   Implementations should look for a base URL in the following order:

        1.   The RTSP Content-Base field
        2.   The RTSP Content-Location field
        3.   The RTSP request URL

   If this attribute contains only an asterisk (*), then the URL is
   treated as if it were an empty embedded URL, and thus inherits the
   entire base URL.

C.1.2 Media streams

   The "m=" field is used to enumerate the streams. It is expected that
   all the specified streams will be rendered with appropriate
   synchronization. If the session is unicast, the port number serves as
   a recommendation from the server to the client; the client still has
   to include it in its SETUP request and may ignore this
   recommendation. If the server has no preference, it SHOULD set the
   port number value to zero.

      Example:
        m=audio 0 RTP/AVP 31

C.1.3 Payload type(s)

   The payload type(s) are specified in the "m=" field. In case the
   payload type is a static payload type from RFC 1890 [1], no other
   information is required. In case it is a dynamic payload type, the
   media attribute "rtpmap" is used to specify what the media is. The
   "encoding name" within the "rtpmap" attribute may be one of those
   specified in RFC 1890 (Sections 5 and 6), or an experimental encoding
   with a "X-" prefix as specified in SDP (RFC 2327 [6]). Codec-
   specific parameters are not specified in this field, but rather in
   the "fmtp" attribute described below. Implementors seeking to
   register new encodings should follow the procedure in RFC 1890 [1].
   If the media type is not suited to the RTP AV profile, then it is
   recommended that a new profile be created and the appropriate profile
   name be used in lieu of "RTP/AVP" in the "m=" field.

C.1.4 Format-specific parameters

   Format-specific parameters are conveyed using the "fmtp" media
   attribute. The syntax of the "fmtp" attribute is specific to the
   encoding(s) that the attribute refers to. Note that the packetization
   interval is conveyed using the "ptime" attribute.

C.1.5 Range of presentation

   The "a=range" attribute defines the total time range of the stored
   session. (The length of live sessions can be deduced from the "t" and
   "r" parameters.) Unless the presentation contains media streams of
   different durations, the range attribute is a session-level
   attribute. The unit is specified first, followed by the value range.
   The units and their values are as defined in Section 3.5, 3.6 and
   3.7.

   Examples:
     a=range:npt=0-34.4368
     a=range:clock=19971113T2115-19971113T2203

C.1.6 Time of availability

   The "t=" field MUST contain suitable values for the start and stop
   times for both aggregate and non-aggregate stream control. With
   aggregate control, the server SHOULD indicate a stop time value for
   which it guarantees the description to be valid, and a start time
   that is equal to or before the time at which the DESCRIBE request was
   received. It MAY also indicate start and stop times of 0, meaning
   that the session is always available. With non-aggregate control, the
   values should reflect the actual period for which the session is
   available in keeping with SDP semantics, and not depend on other
   means (such as the life of the web page containing the description)
   for this purpose.

C.1.7 Connection Information

   In SDP, the "c=" field contains the destination address for the media
   stream. However, for on-demand unicast streams and some multicast
   streams, the destination address is specified by the client via the
   SETUP request. Unless the media content has a fixed destination
   address, the "c=" field is to be set to a suitable null value. For
   addresses of type "IP4", this value is "0.0.0.0".

 C.1.8 Entity Tag

   The optional "a=etag" attribute identifies a version of the session
   description. It is opaque to the client. SETUP requests may include
   this identifier in the If-Match field (see section 12.22) to only
   allow session establishment if this attribute value still corresponds
   to that of the current description. The attribute value is opaque and
   may contain any character allowed within SDP attribute values.

   Example:
     a=etag:158bb3e7c7fd62ce67f12b533f06b83a

   One could argue that the "o=" field provides identical
   functionality. However, it does so in a manner that would put
   constraints on servers that need to support multiple session
   description types other than SDP for the same piece of media
   content.

C.2 Aggregate Control Not Available

   If a presentation does not support aggregate control and multiple
   media sections are specified, each section MUST have the control URL
   specified via the "a=control:" attribute.

   Example:
     v=0
     o=- 2890844256 2890842807 IN IP4 204.34.34.32
     s=I came from a web page
     t=0 0
     c=IN IP4 0.0.0.0
     m=video 8002 RTP/AVP 31
     a=control:rtsp://audio.com/movie.aud
     m=audio 8004 RTP/AVP 3
     a=control:rtsp://video.com/movie.vid

   Note that the position of the control URL in the description implies
   that the client establishes separate RTSP control sessions to the
   servers audio.com and video.com.

   It is recommended that an SDP file contains the complete media
   initialization information even if it is delivered to the media
   client through non-RTSP means. This is necessary as there is no
   mechanism to indicate that the client should request more detailed
   media stream information via DESCRIBE.

C.3 Aggregate Control Available

   In this scenario, the server has multiple streams that can be
   controlled as a whole. In this case, there are both media-level
   "a=control:" attributes, which are used to specify the stream URLs,
   and a session-level "a=control:" attribute which is used as the
   request URL for aggregate control. If the media-level URL is
   relative, it is resolved to absolute URLs according to Section C.1.1
   above.

   If the presentation comprises only a single stream, the media-level
   "a=control:" attribute may be omitted altogether. However, if the
   presentation contains more than one stream, each media stream section
   MUST contain its own "a=control" attribute.

Example:
```
v=0
o=- 2890844256 2890842807 IN IP4 204.34.34.32
s=I contain
i=<more info>
t=0 0
c=IN IP4 0.0.0.0
a=control:rtsp://example.com/movie/
m=video 8002 RTP/AVP 31
a=control:trackID=1
m=audio 8004 RTP/AVP 3
a=control:trackID=2
```

In this example, the client is required to establish a single RTSP
session to the server, and uses the URLs
rtsp://example.com/movie/trackID=1 and
rtsp://example.com/movie/trackID=2 to set up the video and audio
streams, respectively. The URL rtsp://example.com/movie/ controls the
whole movie.

Appendix D: Minimal RTSP implementation

D.1 Client

   A client implementation MUST be able to do the following :

     * Generate the following requests: SETUP, TEARDOWN, and one of PLAY
       (i.e., a minimal playback client) or RECORD (i.e., a minimal
       recording client). If RECORD is implemented, ANNOUNCE must be
       implemented as well.
     * Include the following headers in requests: CSeq, Connection,
       Session, Transport. If ANNOUNCE is implemented, the capability to
       include headers Content-Language, Content-Encoding, Content-
       Length, and Content-Type should be as well.
     * Parse and understand the following headers in responses: CSeq,
       Connection, Session, Transport, Content-Language, Content-
       Encoding, Content-Length, Content-Type. If RECORD is implemented,
       the Location header must be understood as well. RTP-compliant
       implementations should also implement RTP-Info.
     * Understand the class of each error code received and notify the
       end-user, if one is present, of error codes in classes 4xx and
       5xx. The notification requirement may be relaxed if the end-user
       explicitly does not want it for one or all status codes.
     * Expect and respond to asynchronous requests from the server, such
       as ANNOUNCE. This does not necessarily mean that it should
       implement the ANNOUNCE method, merely that it MUST respond
       positively or negatively to any request received from the server.

   Though not required, the following are highly recommended at the time
   of publication for practical interoperability with initial
   implementations and/or to be a "good citizen".

     * Implement RTP/AVP/UDP as a valid transport.
     * Inclusion of the User-Agent header.
     * Understand SDP session descriptions as defined in Appendix C
     * Accept media initialization formats (such as SDP) from standard
       input, command line, or other means appropriate to the operating
       environment to act as a "helper application" for other
       applications (such as web browsers).

   There may be RTSP applications different from those initially
   envisioned by the contributors to the RTSP specification for which
   the requirements above do not make sense. Therefore, the
   recommendations above serve only as guidelines instead of strict
   requirements.

D.1.1 Basic Playback

   To support on-demand playback of media streams, the client MUST
   additionally be able to do the following:
     * generate the PAUSE request;
     * implement the REDIRECT method, and the Location header.

D.1.2 Authentication-enabled

   In order to access media presentations from RTSP servers that require
   authentication, the client MUST additionally be able to do the
   following:
     * recognize the 401 status code;
     * parse and include the WWW-Authenticate header;
     * implement Basic Authentication and Digest Authentication.

D.2 Server

   A minimal server implementation MUST be able to do the following:

     * Implement the following methods: SETUP, TEARDOWN, OPTIONS and
       either PLAY (for a minimal playback server) or RECORD (for a
       minimal recording server). If RECORD is implemented, ANNOUNCE
       should be implemented as well.
     * Include the following headers in responses: Connection,
       Content-Length, Content-Type, Content-Language, Content-Encoding,
       Transport, Public. The capability to include the Location header
       should be implemented if the RECORD method is. RTP-compliant
       implementations should also implement the RTP-Info field.
     * Parse and respond appropriately to the following headers in
       requests: Connection, Session, Transport, Require.

   Though not required, the following are highly recommended at the time
   of publication for practical interoperability with initial
   implementations and/or to be a "good citizen".

     * Implement RTP/AVP/UDP as a valid transport.
     * Inclusion of the Server header.
     * Implement the DESCRIBE method.
     * Generate SDP session descriptions as defined in Appendix C

     There may be RTSP applications different from those initially
     envisioned by the contributors to the RTSP specification for which
     the requirements above do not make sense. Therefore, the
     recommendations above serve only as guidelines instead of strict
     requirements.

D.2.1 Basic Playback

   To support on-demand playback of media streams, the server MUST
   additionally be able to do the following:

     * Recognize the Range header, and return an error if seeking is not
       supported.
     * Implement the PAUSE method.

   In addition, in order to support commonly-accepted user interface
   features, the following are highly recommended for on-demand media
   servers:

     * Include and parse the Range header, with NPT units.
       Implementation of SMPTE units is recommended.
     * Include the length of the media presentation in the media
       initialization information.
     * Include mappings from data-specific timestamps to NPT. When RTP
       is used, the rtptime portion of the RTP-Info field may be used to
       map RTP timestamps to NPT.

   Client implementations may use the presence of length information
   to determine if the clip is seekable, and visibly disable seeking
   features for clips for which the length information is unavailable.
   A common use of the presentation length is to implement a "slider
   bar" which serves as both a progress indicator and a timeline
   positioning tool.

   Mappings from RTP timestamps to NPT are necessary to ensure correct
   positioning of the slider bar.

D.2.2 Authentication-enabled

   In order to correctly handle client authentication, the server MUST
   additionally be able to do the following:

     * Generate the 401 status code when authentication is required for
       the resource.
     * Parse and include the WWW-Authenticate header
     * Implement Basic Authentication and Digest Authentication

Appendix E: Authors' Addresses

    Henning Schulzrinne
    Dept. of Computer Science
    Columbia University
    1214 Amsterdam Avenue
    New York, NY 10027
    USA

    EMail: schulzrinne@cs.columbia.edu

    Anup Rao
    Netscape Communications Corp.
    501 E. Middlefield Road
    Mountain View, CA 94043
    USA

    EMail: anup@netscape.com

    Robert Lanphier
    RealNetworks
    1111 Third Avenue Suite 2900
    Seattle, WA 98101
    USA

    EMail: robla@real.com

Appendix F: Acknowledgements

   This memo is based on the functionality of the original RTSP document
   submitted in October 96. It also borrows format and descriptions from
   HTTP/1.1.

   This document has benefited greatly from the comments of all those
   participating in the MMUSIC-WG. In addition to those already
   mentioned, the following individuals have contributed to this
   specification:

   Rahul Agarwal, Torsten Braun, Brent Browning, Bruce Butterfield,
   Steve Casner, Francisco Cortes, Kelly Djahandari, Martin Dunsmuir,
   Eric Fleischman, Jay Geagan, Andy Grignon, V. Guruprasad, Peter
   Haight, Mark Handley, Brad Hefta-Gaub, John K. Ho, Philipp Hoschka,
   Anne Jones, Anders Klemets, Ruth Lang, Stephanie Leif, Jonathan
   Lennox, Eduardo F. Llach, Rob McCool, David Oran, Maria Papadopouli,
   Sujal Patel, Ema Patki, Alagu Periyannan, Igor Plotnikov, Pinaki
   Shah, David Singer, Jeff Smith, Alexander Sokolsky, Dale Stammen, and
   John Francis Stracke.

References

    1   Schulzrinne, H., "RTP profile for audio and video conferences
            with minimal control", RFC 1890, January 1996.

    2   Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T.
            Berners-Lee, "Hypertext transfer protocol - HTTP/1.1", RFC
            2068, January 1997.

    3   Yergeau, F., Nicol, G., Adams, G., and M. Duerst,
            "Internationalization of the hypertext markup language", RFC
            2070, January 1997.

    4   Bradner, S., "Key words for use in RFCs to indicate
            requirement levels", BCP 14, RFC 2119, March 1997.

    5   ISO/IEC, "Information technology - generic coding of moving
            pictures and associated audio information - part 6: extension
            for digital storage media and control," Draft International
            Standard ISO 13818-6, International Organization for
            Standardization ISO/IEC JTC1/SC29/WG11, Geneva, Switzerland,
            Nov. 1995.

    6   Handley, M., and V. Jacobson, "SDP: Session Description
            Protocol", RFC 2327, April 1998.

    7   Franks, J., Hallam-Baker, P., and J. Hostetler, "An extension to
            HTTP: digest access authentication", RFC 2069, January 1997.

    8   Postel, J., "User Datagram Protocol", STD 6, RFC 768, August
            1980.

    9   Hinden, B. and C. Partridge, "Version 2 of the reliable data
            protocol (RDP)", RFC 1151, April 1990.

    10   Postel, J., "Transmission control protocol", STD 7, RFC 793,
            September 1981.

    11   H. Schulzrinne, "A comprehensive multimedia control
            architecture for the Internet," in Proc. International
            Workshop on Network and Operating System Support for Digital
            Audio and Video (NOSSDAV), (St. Louis, Missouri), May 1997.

    12   International Telecommunication Union, "Visual telephone
            systems and equipment for local area networks which provide a
            non-guaranteed quality of service," Recommendation H.323,
            Telecommunication Standardization Sector of ITU, Geneva,
            Switzerland, May 1996.

   13   McMahon, P., "GSS-API authentication method for SOCKS version
            5", RFC 1961, June 1996.

   14   J. Miller, P. Resnick, and D. Singer, "Rating services and
            rating systems (and their machine readable descriptions),"
            Recommendation REC-PICS-services-961031, W3C (World Wide Web
            Consortium), Boston, Massachusetts, Oct. 1996.

   15   J. Miller, T. Krauskopf, P. Resnick, and W. Treese, "PICS
            label distribution label syntax and communication protocols,"
            Recommendation REC-PICS-labels-961031, W3C (World Wide Web
            Consortium), Boston, Massachusetts, Oct. 1996.

   16   Crocker, D. and P. Overell, "Augmented BNF for syntax
            specifications: ABNF", RFC 2234, November 1997.

   17   Braden, B., "Requirements for internet hosts - application and
            support", STD 3, RFC 1123, October 1989.

   18   Elz, R., "A compact representation of IPv6 addresses", RFC
            1924, April 1996.

   19   Berners-Lee, T., Masinter, L. and M. McCahill, "Uniform
            resource locators (URL)", RFC 1738, December 1994.

   20   Yergeau, F., "UTF-8, a transformation format of ISO 10646",
            RFC 2279, January 1998.

   22   Braden, B., "T/TCP - TCP extensions for transactions
            functional specification", RFC 1644, July 1994.

   22   W. R. Stevens, TCP/IP illustrated: the implementation, vol. 2.
            Reading, Massachusetts: Addison-Wesley, 1994.

   23   Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson,
            "RTP: a transport protocol for real-time applications", RFC
            1889, January 1996.

   24   Fielding, R., "Relative uniform resource locators", RFC 1808,
            June 1995.