Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-mmusic-rtsp-02.ps

MMUSIC WG
H. Schulzrinne, A. Rao, R. Lanphier
Columbia U./Netscape/Progressive Networks

March 27, 1997
Expires: September 26, 1997

Real Time Streaming Protocol (RTSP)

Status of this Memo

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 delivery mechanisms based upon RTP (RFC 1889).

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 9.11). 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, 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. (RTSP servers and clients MAY use the HTTP state maintenance mechanism [2].)
- 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 carries only the absolute path in the request

  This makes virtual hosting easier. However, this is incompatible with HTTP/1.1, which may be a bad idea.

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.

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 xxxx [4].

1.3 Terminology

Some of the terminology has been adopted from HTTP/1.1 [5]. Terms not listed here are defined as in HTTP/1.1.

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.

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

Participant: Participants are members of conferences. A participant may be a machine, e.g., a media record or playback server.

Media server: The network entity 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 parameter: Parameter specific to a media type that may be changed while the stream is being played or prior to it.

(Media) stream: A single media instance, e.g., an audio stream or a video stream as well as a single white-board 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.

Message: The basic unit of RTSP communication, consisting of a structured sequence of octets matching the syntax defined in Section 14 and transmitted via a connection or a connectionless protocol.

Presentation: A set of one or more streams which the server allows the client to manipulate together. A presentation has a single time axis for all streams belonging to it. Presentations are defined by presentation descriptions (see below). A presentation description contains RTSP URIs that define which streams can be controlled individually and an RTSP URI to control the whole presentation. A movie or live concert consisting of one or more audio and video streams is be an example of a presentation.

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 [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 consist 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.

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, either at the transport level (TLS [7]) or within the protocol itself. All HTTP authentication mechanisms such as basic [5, Section 11.1] and digest authentication [8] are directly applicable.

Transport-independent: RTSP may use either an unreliable datagram protocol (UDP) [9], a reliable datagram protocol (RDP, not widely used [10]) or a reliable stream protocol such as TCP [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 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 [13]) firewalls. A firewall may need to understand the SETUP method to open a "hole" for the UDP media stream.

HTTP-friendly: Where sensible, RTSP re-uses HTTP concepts, so that the existing infrastructure can be re-used. This infrastructure includes JEPI (the Joint Electronic Payment Initiative) for electronic payments and PICS (Platform for Internet Content Selection) 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, not recording and thus has no need to support the RECORD request.
- A server may not be capable of seeking (absolute positioning), say, 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 11.

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, where the methods described in [H19.6] are not likely to be supported across all servers.

RTSP can be extended in three ways, listed 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.)
- 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 can then attempt an earlier, less functional version.
- 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.

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. Having the presentation description be returned by the web server makes it possible to have the web server take care of authentication and billing, by handing out a presentation description whose media identifier includes an encrypted version of the requestor's IP address and a timestamp, with a shared secret between web and media server.

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, in that 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).

All the mechanisms specified in this document are described in both prose and an augmented Backus-Naur form (BNF) similar to that used in RFC 2068 [H2.1]. It is described in detail in [14].

In this draft, we use indented and smaller-type paragraphs to provide background and motivation. Some of these paragraphs are marked with HS, AR and RL, designating opinions and comments by the individual authors which may not be shared by the co-authors and require resolution.

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>
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 be 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 [15]).

A presentation or a stream is identified by an textual media identifier, using the character set and escape conventions [H3.2] of URLs [16]. Requests described in Section 9 can refer to either the whole presentation or an individual stream within the presentation. Note that some methods can only be applied to streams, not presentations and vice versa. A specific instance of a presentation or stream, e.g., one of several concurrent transmissions of the same content, an *RTSP session*, is indicated by the Session header field (Section 11.26) where needed.

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`.

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*OCTET  ; LWS must be URL-escaped
```

Conference identifiers are used to allow 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 [17] 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. If the conference participant inviting the media server would only supply a conference identifier which is unique for that inviting party, the media server could add an internal identifier for that party, e.g., its Internet address. However, this would prevent that the conference participant and the initiator of the RTSP commands are two different entities.

3.4 SMPTE Relative Timestamps

A SMPTE relative time-stamp 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,

with the origin at the start of the clip. For NTSC, the frame rate is 29.97 frames per second. This is handled by dropping the first frame index of every minute, except every tenth minute. If the frame value is zero, it may be omitted.

```
smpte-range = "smpte" "=" smpte-time "-" [ smpte-time ]
smpte-time = 1*2DIGIT ":" 1*2DIGIT ":" 1*2DIGIT [ "." 1*2DIGIT ]
```

Examples:

```
smpte=10:12:33.40-
smpte=10:7:33-
smpte=10:7:0-10:7:33
```

3.5 Normal Play Time

Normal play time (NPT) indicates the stream absolute position relative to the beginning of the presentation, measured in seconds and microseconds. The beginning of a presentation corresponds to 0 seconds and 0 microseconds. Negative values are not defined. The microsecond field is always less than 1,000,000. 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.” [18]

```
npt-range = "npt" "=" npt-time "-" [ npt-time ]
npt-time  = 1*DIGIT [ ":" *DIGIT ]
```

Examples:

```
npt=123:45-125
```

3.6 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
```

Example

4 RTSP Message

RTSP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2044). 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.

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 11.12) 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 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 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 CRLF
          *request-header
          CRLF
          [ message-body ]

Request-Line = Method SP Request-URI SP RTSP-Version SP seq-no CRLF

Method = "DESCRIBE"          ; Section
       | "GET_PARAMETER"     ; Section
       | "OPTIONS"           ; Section
       | "PAUSE"             ; Section
       | "PLAY"              ; Section
       | "RECORD"            ; Section
       | "REDIRECT"          ; Section
       | "SETUP"             ; Section
       | "SET_PARAMETER"     ; Section
       | "TEARDOWN"          ; Section
       | extension-method

extension-method = token

Request-URI = "*" | absolute_URI

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

seq-no = 1*DIGIT
```

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

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
```

6 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 the table 1.

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

```
Response = Status-Line              ; Section
```

```
*( general-header        ; Section
 | response-header       ; Section
 | entity-header )       ; Section
CRLF
[ message-body ]         ; Section
```

6.1 Status-Line

The first line of a Response message is the `Status-Line`, consisting of the protocol version followed by a numeric status code, the sequence number of the corresponding request 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. Note that the addition of a

```
Status-Line = RTSP-Version SP Status-Code SP seq-no SP Reason-Phrase CRLF
```

6.1.1 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 section10. 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 status codes and adds RTSP-specific status codes in the starting at 450 to avoid conflicts with newly defined HTTP status codes.

```
Status-Code     = "100"   ; Continue
                | "200"   ; OK
                | "201"   ; Created
                | "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
                    | "409"   ; Conflict
                    | "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
                    | "45x"   ; Session Not Found
                    | "45x"   ; Method Not Valid in This State
                    | "45x"   ; Header Field Not Valid for Resource
                    | "45x"   ; Invalid Range
                    | "45x"   ; Parameter Is Read-Only
                    | "500"   ; Internal Server Error
                    | "501"   ; Not Implemented
                    | "502"   ; Bad Gateway
                    | "503"   ; Service Unavailable
                    | "504"   ; Gateway Time-out
                    | "505"   ; HTTP Version 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.

6.1.2 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
                | Proxy-Authenticate   ; Section
                | Public               ; Section
                | Retry-After          ; Section
                | Server               ; Section
                | Vary                 ; Section
                | WWW-Authenticate     ; Section
```

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.

7 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.

7.1 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 14.7
               | Content-Encoding         ; Section 14.12
               | Content-Language         ; Section 14.13
               | Content-Length           ; Section 14.14
               | Content-Type             ; Section 14.18
               | Expires                  ; Section 14.21
               | Last-Modified            ; Section 14.29
               | extension-header
```

Code	reason	
100	Continue	all
200	OK	all
201	Created	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
409	Conflict	
410	Gone	all
411	Length Required	SETUP
412	Precondition Failed	
413	Request Entity Too Large	SETUP
414	Request-URI Too Long	all
415	Unsupported Media Type	SETUP
45x	Only Valid for Stream	SETUP
45x	Invalid parameter	SETUP
45x	Not Enough Bandwidth	SETUP
45x	Illegal Conference Identifier	SETUP
45x	Illegal Session Identifier	PLAY, RECORD, TEARDOWN
45x	Parameter Is Read-Only	SET_PARAMETER
45x	Header Field Not Valid	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

Table 1: Status codes and their usage with RTSP methods

```
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.

7.2 Entity Body

See [H7.2]

8 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 send 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.

8.1 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.

8.2 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 TBD), 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, the timeout value MAY be set to an arbitrarily large value.

> This can greatly increase responsiveness for proxies operating in local-area networks with small RTTs. The mechanism is defined such that the client implementation does not have be aware of whether a reliable or unreliable transport protocol is being used. It is probably a bad idea to have two reliability mechanisms on top of each other, although the RTSP RTT estimate is likely to be larger than the TCP estimate.

Each request carries a sequence number, which is incremented by one for each request transmitted. If a request is repeated because of lack of acknowledgement, the sequence number is incremented.

This avoids ambiguities when computing round-trip time estimates.

[TBD: An initial sequence number negotiation needs to be added for UDP; otherwise, a new stream connection may see a request be acknowledged by a delayed response from an earlier "connection". This handshake can be avoided with a sequence number containing a timestamp of sufficiently high resolution.]

The reliability mechanism described here does not protect against reordering. This may cause problems in some instances. For example, a TEARDOWN followed by a PLAY has quite a different effect than the reverse. Similarly, if a PLAY request arrives before all parameters are set due to reordering, the media server would have to issue an error indication. Since sequence numbers for retransmissions are incremented (to allow easy RTT estimation), the receiver cannot just ignore out-of-order packets. [TBD: This problem could be fixed by including both a sequence number that stays the same for retransmissions and a timestamp for RTT estimation.]

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 method header MUST contain a Content-Length whenever that method contains a payload. Otherwise, an RTSP packet is terminated with an empty line immediately following the method header.

9 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	$C \to S, S \to C$	P,S	recommended
GET_PARAMETER	$C \to S, S \to C$	P,S	optional
OPTIONS	$C \to S$	P,S	required
PAUSE	$C \to S$	P,S	recommended
PLAY	$C \to S$	P,S	required
RECORD	$C \to S$	P,S	optional
REDIRECT	$S \to C$	P,S	optional
SETUP	$C \to S$	S	required
SET_PARAMETER	$C \to S, S \to C$	P,S	optional
TEARDOWN	$C \to S$	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 recommend, 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.

9.1 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 non-standard request. It does not influence server state.

In addition, if the optional `Require` header is present, option tags within the header indicate features needed by the requestor that are not required at the version level of the protocol.

Example 1:

```
C->S:  OPTIONS * RTSP/1.0 1
       Require: implicit-play, record-feature
       Transport-Require: switch-to-udp-control, gzipped-messages
```

Note that these are fictional features (though we may want to make them real one day).

Example 2 (using RFC2069-style authentication only as an example):

```
S->C: OPTIONS * RTSP/1.0 1
      Authenticate: Digest realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"

S->C: RTSP/1.0 200 1 OK
      Date: 23 Jan 1997 15:35:06 GMT
      Nack-Transport-Require: switch-to-udp-control
```

Note that these are fictional features (though we may want to make them real one day).

Example 2 (using RFC2069-style authentication only as an example):

```
C->S: RTSP/1.0 401 1 Unauthorized
      Authorization: Digest username="Mufasa",
          realm="testrealm@host.com",
          nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
          uri="/dir/index.html",
          response="e966c932a9242554e42c8ee200cec7f6",
          opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

9.2 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. Alternatively, the server may “push” a new description to the client, for example, if a new stream has become available. 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: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0 312
```

```
        Accept: application/sdp, application/rtsl, application/mheg

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

or

```
  S->C: RTSP/1.0 200 312 OK
        Date: 23 Jan 1997 15:35:06 GMT
        Content-Type: application/rtsl
        Content-Length: 2782

        <2782 octets of data containing stream description>
```

Server to client example:

```
  S->C: DESCRIBE /twister RTSP/1.0 902
        Session: 1234
        Content-Type: application/rtsl

        new RTSL presentation description
```

9.3 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. 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.

This avoids having firewall to parse numerous different presentation description formats, for information which is irrelevant.

If the optional Require header is present, option tags within the header indicate features needed by the requestor that are not required at the version level of the protocol. The Transport-Require header is used to indicate proxy-sensitive features that MUST be stripped by the proxy to the server if not supported. Furthermore, any Transport-Require header features that are not supported by the proxy MUST be negatively acknowledged by the proxy to the client if not supported.

HS: In my opinion, the Require header should be replaced by PEP since PEP is standards-track, has more functionality and somebody already did the work.

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 foo/bar/baz.rm RTSP/1.0 302
      Transport: rtp/udp;port=458

S->C: RTSP/1.0 200 302 OK
      Date: 23 Jan 1997 15:35:06 GMT
      Transport: cush/udp;port=458
```

9.4 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 commands in the example below arrive, the server will play first second 10 through 15 and then, immediately following, seconds 20 to 25 and finally seconds 30 through the end.

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

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

C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0 837
      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 synchronisation of streams obtained from different sources.

For a on-demand stream, the server replies back 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/1.0 833
      Range: smpte=0:10:20-;time=19970123T153600Z

S->C: RTSP/1.0 200 833 OK
      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/1.0 835
      Range: clock=19961108T142300Z-19961108T143520Z

S->C: RTSP/1.0 200 833 OK
      Date: 23 Jan 1997 15:35:06 GMT
```

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

9.5 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.

The PAUSE request may contain a Range header specifying when the stream or presentation is to be halted. The header must contain exactly one value rather than a time range. The normal play time for the stream is set to that value. The pause request becomes effective the first time the server is encountering the time point specified. If this header is missing, stream delivery is interrupted immediately on receipt of the message.

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, it stops immediately. If the pause request is for NPT 16, it 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 playing 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.

Example:

```
C->S: PAUSE /fizzle/foo RTSP/1.0 834

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

9.6 TEARDOWN

Stop the stream delivery for the given URI, freeing the resources associated with it. If the URI is the root node 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 /fizzle/foo RTSP/1.0 892

S->C: RTSP/1.0 200 892 OK
```

9.7 GET_PARAMETER

The requests retrieves the value of a parameter of a presentation or stream specified in the URI. Multiple parameters can be requested in the message body using the content type `text/rtsp-parameters`. Note that parameters include server and client statistics. IANA registers parameter names for statistics and other purposes. GET_PARAMETER with no entity body may be used to test client or server liveness (“ping”).

Example:

```
S->C: GET_PARAMETER /fizzle/foo RTSP/1.0 431
      Content-Type: text/rtsp-parameters
```

```
        Session: 1234
        Content-Length: 15

        packets_received
        jitter

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

        packets_received: 10
        jitter: 0.3838
```

9.8 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. 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.

A SET_PARAMETER request without parameters can be used as a way to detect client or server liveness.

Example:

```
  C->S: SET_PARAMETER /fizzle/foo RTSP/1.0 421
        Content-type: text/rtsp-parameters

        fooparam: foostuff
        barparam: barstuff

  S->C: RTSP/1.0 450 421 Invalid Parameter
        Content-Length: 6

        barparam
```

9.9 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 a DESCRIBE for that URL. It may contain the parameter Range, which indicates when the redirection takes effect.

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

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

9.10 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 Conference header is mandatory.

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 /meeting/audio.en RTSP/1.0 954
       Session: 1234
       Conference: 128.16.64.19/32492374
```

9.11 Embedded Binary Data

Binary packets such as RTP data are encapsulated by an ASCII dollar sign (24 decimal), followed by a one-byte session identifier, followed by the length of the encapsulated binary data as a binary, two-byte integer in network byte order. The binary data follows immediately afterwards, without a CRLF.

10 Status Code Definitions

Where applicable, HTTP status [H10] codes are re-used. 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 request.

10.1 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.

10.2 Client Error 4xx

10.2.1 451 Parameter Not Understood

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

10.2.2 452 Conference Not Found

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

10.2.3 453 Not Enough Bandwidth

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

10.2.4 45x Session Not Found

The RTSP session identifier is invalid or has timed out.

10.2.5 45x Method Not Valid in This State

The client or server cannot process this request in its current state.

10.2.6 45x 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.

10.2.7 45x Invalid Range

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

10.2.8 45x Parameter Is Read-Only

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

11 Header Field Definitions

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

Tables 3 summarizes the header fields used by RTSP. Type “R” designates request headers, type “r” response headers. Fields marked with “r” in the column label “support” MUST be implemented by the recipient for a particular method, while fields marked “o” are optional. Note that not all fields marked 'r' will be send in every request of this type; merely, that client (for response headers) and server (for request headers) MUST implement them. 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.

If the field content does not apply to the particular resource, the server MUST return status 45x (Header Field Not Valid for Resource).

Header	type	support	methods
Accept	R	o	entity
Accept-Encoding	R	o	entity
Accept-Language	R	o	all
Authorization	R	o	all
Bandwidth	R	o	SETUP
Blocksize	R	o	all but OPTIONS, TEARDOWN
Cache-Control	Rr	o	SETUP
Conference	R	o	SETUP
Connection	Rr	r	all
Content-Encoding	R	r	SET_PARAMETER
Content-Encoding	r	r	DESCRIBE
Content-Length	R	r	SET_PARAMETER
Content-Length	r	r	entity
Content-Type	R	r	SET_PARAMETER
Content-Type	r	r	entity
Date	Rr	o	all
Expires	r	o	DESCRIBE
If-Modified-Since	R	o	DESCRIBE, SETUP
Last-Modified	r	o	entity
Public	r	o	all
Range	R	o	PLAY, PAUSE, RECORD
Range	r	o	PLAY, PAUSE, RECORD
Referer	R	o	all
Require	R	r	all
Retry-After	r	o	all
Scale	Rr	o	PLAY, RECORD
Session	Rr	r	all but SETUP, OPTIONS
Server	r	o	all
Speed	Rr	o	PLAY
Transport	Rr	r	SETUP
Transport-Require	R	x	all
User-Agent	R	o	all
Via	Rr	o	all
WWW-Authenticate	r	o	all

Table 3: Overview of RTSP header fields

11.1 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
```

11.2 Accept-Encoding

See [H14.3]

11.3 Accept-Language

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

11.4 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
```

11.5 Authorization

See [H14.8]

11.6 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.

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

Example:

```
Bandwidth: 4000
```

11.7 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 overrides it with the media specific size if necessary. The block size is a strictly positive decimal number and measured in octets. The server only returns an error (416) if the value is syntactically invalid.

11.8 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.

[HS: Should there be an exception for 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 cachable 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 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. (I.e., 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.)

11.9 Conference

This request header field establishes a logical connection between a conference, established using non-RTSP means, 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
```

11.10 Connection

See [H14.10].

11.11 Content-Encoding

See [H14.12]

11.12 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. It is interpreted according to [H14.14].

11.13 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.

11.14 Date

See [H14.19].

11.15 Expires

The Expires entity-header field gives the date/time after which the media-stream 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 13.2 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 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 cachable, unless indicated otherwise by a Cache-Control header field (Section 11.8.

11.16 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 setup (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
```

11.17 Last-modified

The Last-Modified entity-header field indicates the date and time at which the origin server believes the variant was last modified. See [H14.29]. If the request URI refers to an aggregate, the field indicates the last modification time across all leave nodes of that aggregate.

11.18 Location

See [H14.30].

11.19 Nack-Transport-Require

Negative acknowledgement of features not supported by the server. 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 506 (Feature not supported).

HS: Same caveat as for Require applies.

11.20 Range

This request header field specifies a range of time. The range can be specified in a number of units. This specification defines the smpte (see Section 3.4) and clock (see Section 3.6) 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.

```
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 header. It allows to select a clip from the media object, to play from a given point to the end and from the current location to a given point.

11.21 Require

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

HS: Naming of features – yet another name space. I believe this header field to be redundant. PEP should be used instead.

For example

```
C->S:   SETUP /foo/bar/baz.rm RTSP/1.0 302
        Require: funky-feature
        Funky-Parameter: funkystuff

S->C:   RTSP/1.0 200 506 Option not supported
        Unsupported: funky-feature

C->S:   SETUP /foo/bar/baz.rm RTSP/1.0 303

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

This is to make sure that the client-server interaction will proceed optimally when all options are understood by both sides, and only slow down if options aren't 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 doesn't understand.

11.22 Retry-After

See [H14.38].

11.23 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
```

11.24 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. A negative value indicates that the stream is to be played back in reverse direction.

> HS: With 'Scale', the negative value is redundant and should probably be removed since it only leads to possible conflicts when Scale is positive and Speed negative.

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
```

11.25 Server

See [H14.39]

11.26 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 and has the same syntax as a conference identifier. Once a client receives a Session identifier, it MUST return it for any request related to that session.

HS: This may be redundant with the standards-track HTTP state maintenance mechanism [2]. The equivalent way of doing this would be for the server to send `Set-Cookie: Session="123"; Version=1; Path = "/twister"` and for the client to return later `Cookie: Session = "123"; $Version=1; $Path = "/twister"`. In the response to the TEARDOWN message, the server would simply send `Set-Cookie: Session="123"; Version=1; Max-Age=0` to get rid of the cookie on the client side. Cookies also have a time-out, so that a server may limit the lifetime of a session at will. Unlike a web browser, a client would not store these states on disk. To avoid privacy issues, we should prohibit the Host parameter.

11.27 Transport

This request header indicates which transport protocol is to be used and configures its parameters such as multicast, compression, multicast time-to-live and destination port for a single stream. It sets those values not already determined by a presentation description. In some cases, the presentation description contains all necessary information. In those cases, a Transport header field (and the SETUP request containing it) are not needed.

'Interleaved' implies mixing the media stream with the control stream, in whatever protocol is being used by the control stream. Currently, the next-layer protocols RTP is defined. Parameters may be added to each protocol, separated by a semicolon. For RTP, the boolean parameter `compressed` is defined, indicating compressed RTP according to RFC XXXX. For multicast UDP, the integer parameter ttl defines the time-to-live value to be used. The client may specify the multicast address with the multicast parameter. A server SHOULD authenticate the client before allowing the client to direct a media stream to a multicast address not chosen by the server to avoid becoming the unwitting perpetrator of a denial-of-service attack. For UDP and TCP, the parameter port defines the port data is to be sent to.

The SSRC parameter indicates the RTP SSRC 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.

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 Transport header field makes sense only for an individual media stream, not a presentation.

```
Transport = "Transport" ":"
            1#transport-protocol/upper-layer *parameter
transport-protocol = "UDP" | "TCP"
upper-layer  = "RTP"
parameters = ";" "multicast" [ "=" mca ]
           | ";" "compressed"
           | ";" "interleaved"
           | ";" "ttl" "=" ttl
           | ";" "port" "=" port
           | ";" "ssrc" "=" ssrc
ttl        = 1*3(DIGIT)
```

```
port       = 1*5(DIGIT)
ssrc       = 8*8(HEX)
mca        = host
```

Example:

```
Transport: udp/rtp;compressed;ttl=127;port=3456
```

11.28 Transport-Require

The Transport-Require header is used to indicate proxy-sensitive features that MUST be stripped by the proxy to the server if not supported. Furthermore, any Transport-Require header features that are not supported by the proxy MUST be negatively acknowledged by the proxy to the client if not supported.

See Section 11.21 for more details on the mechanics of this message and a usage example.

> HS: Same caveat as for Require applies.

11.29 Unsupported

See Section 11.21 for a usage example.

> HS: same caveat as for Require applies.

11.30 User-Agent

See [H14.42]

11.31 Via

See [H14.44].

11.32 WWW-Authenticate

See [H14.46].

12 Caching

In HTTP, response-request pairs are cached. RTSP differs significantly in that respect. Responses are not cachable, with the exception of the stream description returned by DESCRIBE. (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.

On receiving a SETUP or PLAY request, the proxy would ascertain as to whether it has an up-to-date copy of the continuous media content. If not, it would modify the SETUP transport parameters as appropriate and forward the request to the origin server. Subsequent control commands such as PLAY or PAUSE would pass the proxy unmodified. The proxy would then pass the continuous media data to the client, while possibly making a local copy for later re-use. The exact behavior allowed to the cache is given by the cache-response directives described in Section 11.8. 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 would appear like a regular media server, to the media origin server like a client. Just like an HTTP cache has to store the content type, content language, etc. for the objects it caches, a media cache has to store the presentation description. Typically, a cache would eliminate 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.

13 Examples

The following examples reference stream description formats that are not finalized, such as RTSL and SDP. Please do not use these examples as a reference for those formats.

13.1 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 time line of the movie.

In our example, the client is only interested in the last part of the movie. The server requires authentication for this movie. The audio track can be dynamically switched between between two sets of encodings. The URL with scheme rtpsu indicates the media servers want to use UDP for exchanging RTSP messages.

```
C->W: DESCRIBE /twister HTTP/1.1
      Host: www.example.com
      Accept: application/rtsl; application/sdp

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

      <session>
        <group language=en lipsync>
          <switch>
            <track type=audio
              e="PCMU/8000/1"
              src="rtsp://audio.example.com/twister/audio.en/lofi">
            <track type=audio
```

```
              e="DVI4/16000/2" pt="90 DVI4/8000/1"
              src="rtsp://audio.example.com/twister/audio.en/hifi">
          </switch>
          <track type="video/jpeg"
            src="rtspu://video.example.com/twister/video">
        </group>
      </session>

C->A: SETUP rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 1
      Transport: rtp/udp;compression;port=3056

A->C: RTSP/1.0 200 1 OK
      Session: 1234

C->V: SETUP rtsp://video.example.com/twister/video RTSP/1.0 1
      Transport: rtp/udp;compression;port=3058

V->C: RTSP/1.0 200 1 OK
      Session: 1235

C->V: PLAY rtsp://video.example.com/twister/video RTSP/1.0 2
      Session: 1235
      Range: smpte=0:10:00-

V->C: RTSP/1.0 200 2 OK

C->A: PLAY rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 2
      Session: 1234
      Range: smpte=0:10:00-

A->C: 200 2 OK

C->A: TEARDOWN rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 3
      Session: 1234

A->C: 200 3 OK

C->V: TEARDOWN rtsp://video.example.com/twister/video RTSP/1.0 3
      Session: 1235

V->C: 200 3 OK
```

Even though the audio and video track are on two different servers, 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.

13.2 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. During the RTSP session, a new subtitling stream is added.

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

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

      <session>
        <track id=17 src="rtsp://live.example.com/concert/audio">
      </session>

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

M->C: RTSP/1.0 200 1 OK
      Content-Type: application/rtsl

      <track id=17 type=audio address=224.2.0.1 port=3456 ttl=16>

C->M: SETUP rtsp://live.example.com/concert/audio RTSP/1.0 2
      Transport: multicast=224.2.0.1; port=3456; ttl=16

C->M: PLAY rtsp://live.example.com/concert/audio RTSP/1.0 3
      Range: smpte 1:12:0

M->C: RTSP/1.0 405 3 No positioning possible

M->C: DESCRIBE concert RTSP/1.0
      Content-Type: application/rtsl

      <session>
        <track id=17
          media=audio/g.728 src="rtsp://live.example.com/concert/audio">
        <track id=18
          media=text/html src="rtsp://live.example.com/concert/lyrics">
      </session>

C->M: PLAY rtsp://live.example.com/concert/lyrics RTSP/1.0
```

The attempt to position the stream fails since this is a live presentation.

13.3 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. When retrieving the presentation description, 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: GET /demo HTTP/1.1
      Host: www.example.com
      Accept: application/rtsl, application/sdp

M->C: HTTP/1.1 200 1 OK
      Content-type: application/rtsl

      <session>
        <track type=audio/g.723.1
          src="rtsp://server.example.com/demo/548/sound">
      </session>

C->M: SETUP rtsp://server.example.com/demo/548/sound RTSP/1.0 2
      Conference: 218kadjk
```

13.4 Recording

The conference participant C asks the media server M to record a meeting. If the presentation description contains any alternatives, the server records them all.

```
C->M: DESCRIBE rtsp://server.example.com/meeting RTSP/1.0 89
      Content-Type: application/sdp

      v=0
      s=Mbone Audio
      i=Discussion of Mbone Engineering Issues

M->C: 415 89 Unsupported Media Type
      Accept: application/rtsl

C->M: DESCRIBE rtsp://server.example.com/meeting RTSP/1.0 90
      Content-Type: application/rtsl

M->C: 200 90 OK

C->M: RECORD rtsp://server.example.com/meeting RTSP/1.0 91
      Range: clock 19961110T1925-19961110T2015
```

14 Syntax

The RTSP syntax is described in an augmented Backus-Naur form (BNF) as used in RFC 2068 (HTTP/1.1).

14.1 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>
```

15 Security Considerations

The protocol offers the opportunity for a remote-control denial-of-service attack. The attacker, using a forged source IP address, can ask for a stream to be played back to that forged IP address.

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. This does not require spoofing of network packet addresses. The server SHOULD use a large random session identifier to make this attack more difficult.

Both problems can be be prevented by appropriate authentication.

In addition, the security considerations outlined in [H15] apply.

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.

[TBD: should we allow for the trivial case of a server that only implements the PLAY message, with no control.]

State is defined on a per object basis. An object is uniquely identified by the stream URL and the RTSP session identifier. (A server may choose to generate dynamic presentation descriptions where the URL is unique for a particular RTSP session and thus may not need an explicit RTSP session identifier in the request header.) Any request/reply using URLs denoting an RTSP session comprised of multiple streams will have an effect on the individual states of all the substreams. For example, if the stream /movie contains two substreams /movie/audio and /movie/video, then the following command:

```
PLAY /movie RTSP/1.0 559
Session: 12345
```

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

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

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

Client and servers MUST disregard messages with a sequence number less than the last one. If no message has been received, the first received message's sequence number will be the starting point.

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 after playing, PAUSE reply received.

Playing: PLAY reply received

Recording: RECORD reply received

The client changes state on receipt of replies to requests. 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 "next state" column indicates the state assumed after receiving a success response (2xx). If a request yields a status code greater or equal to 300, the client state becomes Init, with the exception of status codes

401 (Unauthorized) and 402 (Payment Required), where the state remains unchanged and the request should be re-issued with the appropriate authentication or payment information. 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.

HS: Depends on allowing PLAY without SETUP. After 4xx or 5xx error, do we go back to Init?

state	message	next state
Init	SETUP	Ready
	TEARDOWN	Init
Ready	PLAY	Playing
	RECORD	Recording
	TEARDOWN	Init
Playing	PAUSE	Ready
	TEARDOWN	Init
	PLAY	Playing
	RECORD	Recording
	SETUP	Playing (changed transport)
Recording	PAUSE	Init
	TEARDOWN	Init
	PLAY	Playing
	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 received.

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.

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, from the client for a defined interval, with a default of one minute. 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.

The REDIRECT message, when sent, is effective immediately. If a similar change of location occurs at a certain time in the future, this is assumed to be indicated by the presentation description.

SETUP is valid in states Init and Ready only. An error message should be returned in other cases. If no explicit `SETUP` is required for the object, state starts at READY, there are only two states READY and PLAYING.

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

B Open Issues

- Define text/rtsp-parameter MIME type.
- HS believes that RTSP should only control individual media objects rather than aggregates. This avoids disconnects between presentation descriptions and streams and avoids having to deal separately with single-host and multi-host case. Cost: several PLAY/PAUSE/RECORD in one packet, one for each stream.
- Allow changing of transport for a stream that's playing? May not be a great idea since the same can be accomplished by tear down and re-setup.
- Allow fragment (#) identifiers for controlling substreams in Quicktime, AVI and ASF files?
- How does the server get back to the client unless a persistent connection is used? Probably cannot, in general.
- Cache and proxy behavior?
- Session: or Set-Cookie: ?
- When do relative RTSP URLs make sense?
- Nack-require, etc. are dubious. This is getting awfully close to the HTTP extension mechanisms [19] in complexity, but is different.
- Use HTTP absolute path + Host field or do the right thing and carry full URL, including host in request?

C Changes

Since the February 1997 version, the following changes were made:

- Various editorial changes and clarifications.
- Removed references to SDF and replaced by RTSL.
- Added Scale general header.
- Clarify behavior of PLAY.
- Rename GET to DESCRIBE.
- Removed SESSION since it is just DESCRIBE in the other direction.
- Rename CLOSE to TEARDOWN, in symmetry with SETUP.
- Terminology adjusted to "presentation" and "stream".
- Redundant syntax BNF in appendix removed since it just duplicates HTTP spec.
- Beginnings of cache control.

Changes are marked by changebars in the margins of the PostScript version.

D Author Addresses

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

Anup Rao
Netscape Communications Corp.
USA
electronic mail: anup@netscape.com

Robert Lanphier
Progressive Networks
1111 Third Avenue Suite 2900
Seattle, WA 98101
USA
electronic mail: robla@prognet.com

E Acknowledgements

This draft is based on the functionality of the RTSP draft. 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	Eduardo F. Llach
Bruce Butterfield	Rob McCool
Martin Dunsmuir	Sujal Patel
Eric Fleischman	
Mark Handley	Igor Plotnikov
Peter Haight	Pinaki Shah
Brad Hefta-Gaub	Jeff Smith
John K. Ho	Alexander Sokolsky
Ruth Lang	Dale Stammen
Stephanie Leif	John Francis Stracke

References

[1] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, Internet Engineering Task Force, Jan. 1996.

[2] D. Kristol and L. Montulli, "HTTP state management mechanism," RFC 2109, Internet Engineering Task Force, Feb. 1997.

[3] F. Yergeau, G. Nicol, G. Adams, and M. Duerst, "Internationalization of the hypertext markup language," RFC 2070, Internet Engineering Task Force, Jan. 1997.

[4] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Internet Draft, Internet Engineering Task Force, Jan. 1997. Work in progress.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," RFC 2068, Internet Engineering Task Force, Jan. 1997.

[6] M. Handley, "SDP: Session description protocol," Internet Draft, Internet Engineering Task Force, Nov. 1996. Work in progress.

[7] A. Freier, P. Karlton, and P. Kocher, "The TLS protocol," Internet Draft, Internet Engineering Task Force, Dec. 1996. Work in progress.

[8] J. Franks, P. Hallam-Baker, J. Hostetler, P. A. Luotonen, and E. L. Stewart, "An extension to HTTP: digest access authentication," RFC 2069, Internet Engineering Task Force, Jan. 1997.

[9] J. Postel, "User datagram protocol," STD 6, RFC 768, Internet Engineering Task Force, Aug. 1980.

[10] R. Hinden and C. Partridge, "Version 2 of the reliable data protocol (RDP)," RFC 1151, Internet Engineering Task Force, Apr. 1990.

[11] J. Postel, "Transmission control protocol," STD 7, RFC 793, Internet Engineering Task Force, Sept. 1981.

[12] M. Handley, H. Schulzrinne, and E. Schooler, "SIP: Session initiation protocol," Internet Draft, Internet Engineering Task Force, Dec. 1996. Work in progress.

[13] P. McMahon, "GSS-API authentication method for SOCKS version 5," RFC 1961, Internet Engineering Task Force, June 1996.

[14] D. Crocker, "Augmented BNF for syntax specifications: ABNF," Internet Draft, Internet Engineering Task Force, Oct. 1996. Work in progress.

[15] R. Elz, "A compact representation of IPv6 addresses," RFC 1924, Internet Engineering Task Force, Apr. 1996.

[16] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (URL)," RFC 1738, Internet Engineering Task Force, Dec. 1994.

[17] 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.

[18] ISO/IEC, "Information technology – generic coding of moving pictures and associated audio informaiton – 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.

[19] D. Connolly, "PEP: an extension mechanism for http," Internet Draft, Internet Engineering Task Force, Jan. 1997. Work in progress.