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

February 21, 1997
Expires: August 20, 1997

Real Time Streaming Protocol (RTSP)

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited.

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

RTSP establishes and controls either single or several (time-synchronized) streams of continuous media. It does not typically deliver the continuous streams itself, although interleaving of the continuous media stream with the control stream is possible (Section 6.13).

There is no notion of an RTSP connection, but rather a session maintained by an identifier. An RTSP session is in no way tied to a transport-level session. 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 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 [1].)
- 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 [2].

  HS: Probably the right thing to do, but may lead to confusion with GET.

- 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. Makes definition of GET confusing, if it is included in RTSP.

The protocol supports the following operations:

Retrieval of media from media server: The client can request a session description via HTTP or some other method. If the session is being multicast, the session description contains the multicast addresses and ports to be used for the continuous media. If the session 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 session or to record all or a subset of the media in a session. 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 session: Particularly for live events, 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 [3].

1.3 Terminology

Some of the terminology has been adopted from HTTP/1.1 [4]. Terms not listed here are defined as in HTTP/1.1.

Conference: a multiparty, multimedia session, 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.

Entity: An entity is a participant in a conference. This participant may be non-human, 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 session may originate from different media servers. A media server may reside on the same or a different host as the web server the media session is invoked from.

Media session: A collection of media streams to be treated as an aggregate, with a single time axis. Typically, a client will synchronize in time all media streams within a media session. An example of a media session is a movie consisting of a video and audio track.

(Media) stream: A single media instance, e.g., an audio stream or a video stream as well as a single whiteboard or shared application group. When using RTP, a stream consists of all RTP and RTCP packets created by a source within an RTP session.

[TBD: terminology is confusing since there's an RTP session, which is used by a single RTSP stream.]

Message: The basic unit of RTSP communication, consisting of a structured sequence of octets matching the syntax defined in Section 11 and transmitted via a connection or a connectionless protocol.

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.

Session description: A session description contains information about one or more media within a session, such as the set of encodings, network addresses and information about the content. The session description may take several different formats, including SDP and SDF.

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

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 [5]) or within the protocol itself. All HTTP authentication mechanisms such as basic [4, Section 11.1] and digest authentication [6] are directly applicable.

Transport-independent: RTSP may use either an unreliable datagram protocol (UDP) [7], a reliable datagram protocol (RDP, not widely used [8]) or a reliable stream protocol such as TCP [9] as it implements application-level reliability.

Multi-server capable: Each media stream within a session 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 [10] 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.

Session description neutral: The protocol does not impose a particular session description or metafile format and can convey the type of format to be used. However, the session description must contain an RTSP URI.

Proxy and firewall friendly: The protocol should be readily handled by both application and transport-layer (SOCKS [11]) 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

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 media stream and session may be identified by an RTSP URL. The overall session and the properties of the media the session is made up of are defined by a session description file, the format of which is outside the scope of this specification. The session description file is retrieved using HTTP, either from the web server or the media server, typically using an URL with scheme HTTP.

The session description file contains a description of the media streams making up the media session, including their encodings, language, and other parameters that enable the client to choose the most appropriate combination of media. In this session description, each media stream 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 tracks can be split across servers for load sharing. The description also enumerates which transport methods the server is capable of. If desired, the session description can also contain only an RTSP URL, with the complete session description retrieved via RTSP.

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 picked 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.

> HS: This does not imply that the protocol has to be stateful in the way described here. If state were to be defined by identifier only, the first PLAY or RECORD request would initiate stream flow, with no need for SETUP. It has been argued that a separate setup simplifies the life of firewall writers.

Many methods in RTSP do not contribute to state. However, there are four that play a central role in defining the allocation and usage of stream resources on the server: SETUP, PLAY, PAUSE, and CLOSE. The roles they play are defined as follows.

Step	session allocation method	session control method
1	SETUP	
2		PLAY
3		PAUSE
4	CLOSE	

SETUP	Causes the server to allocate resources for a stream.
PLAY	Starts data transmission on a stream allocated via SETUP.
PAUSE	Temporarily halts a stream, without freeing server resources.
CLOSE	Frees resources associated with the stream. The session ceases to exist on the server.

A client must issue a SETUP request unless all necessary transport information is already available.

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 session description can be retrieved using HTTP or RTSP. Having the session 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 session 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 session description format that can express both static and temporal properties of a media session 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 [12].

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 [13]).

A media presentation is identified by an textual media identifier, using the character set and escape conventions [H3.2] of URLs [14]. Requests described in Section 6 can refer to either the whole presentation or an individual track within the presentation. Note that some methods can only be applied to tracks, not presentations. A specific instance of a session, e.g., one of several concurrent transmissions of the same content, is indicated by the Session header field (Section 8.24) where needed.

For example, the RTSP URL

```
rtsp://media.example.com:554/twister/audiotrack
```

identifies the audio track 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 tracks in URLs. The session description defines the hierarchical relationships in the presentation and the URLs for the individual tracks. A session description may name a track '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 session 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 [15] or SIP [10]. 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 Relative Timestamps

A 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:

```
10:12:33.40
10:7:33
10:7:0
```

3.5 Absolute Time

Absolute time is expressed as ISO 8601 timestamps. It is always expressed as UTC (GMT).

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

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

```
19961108T143720Z
```

4 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 = "GET"              ; Section
         | "SETUP"            ; Section
         | "PLAY"             ; Section
         | "PAUSE"            ; Section
         | "CLOSE"            ; Section
         | "RECORD"           ; Section
         | "REDIRECT"         ; Section
         | "HELLO"            ; Section
         | "SESSION"          ; Section
         | "BYE"              ; Section
         | "SET_PARAMETER"    ; Section
         | "GET_PARAMETER"    ; Section

  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.

5 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 and 2.

6 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`.

> HS: 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. Similarly, SETUP is not needed for a server that only handles multicast events with transport parameters set outside of RTSP. GET and BYE are controversial.

Code	reason	HELLO	GET	SETUP	PLAY	RECORD	PAUSE
100	Continue	x	x	x	x	x	x
200	OK	x	x	x	x	x	x
300	Multiple Choices	x	x	x	x	x	
301	Moved Permanently	x	x	x	x	x	
302	Moved Temporarily	x	x	x	x	x	
303	See Other	x	x	x	x	x	
304	Not Modified	x	x	x	x	x	
305	Use Proxy	x	x	x	x	x	
400	Bad Request	x	x	x	x	x	x
401	Unauthorized	x	x	x	x	x	x
402	Payment Required	x	x	x	x	x	x
403	Forbidden	x	x	x	x	x	
404	Not Found	x	x	x	x	x	
405	Method Not Allowed	x	x	x	x	x	x
406	Not Acceptable	x	x	x	x	x	
407	Proxy Authentication Required	x	x	x	x	x	x
408	Request Timeout	x	x	x	x	x	x
409	Conflict						
410	Gone	x	x	x	x	x	x
411	Length Required	x	x	x			
412	Precondition Failed	x	x				
413	Request Entity Too Large	x	x				
414	Request-URI Too Long	x	x	x	x	x	x
415	Unsupported Media Type	x	x				
45x	Only Valid for Stream	x	x	x			
45x	Invalid parameter						
45x	Not Enough Bandwidth			x			
45x	Illegal Conference Identifier						
45x	Illegal Session Identifier			x	x	x	x
45x	Parameter Is Read-Only						
45x	Header Field Not Valid						
500	Internal Server Error	x	x	x	x	x	x
501	Not Implemented	x	x	x	x	x	x
502	Bad Gateway	x	x	x	x	x	x
503	Service Unavailable	x	x	x	x	x	x
504	Gateway Timeout	x	x	x	x	x	x
505	RTSP Version Not Supported	x	x	x	x	x	x

Table 1: Status codes and their usage with RTSP methods

Code	reason	CLOSE	REDIRECT	GET_PARAMETER	SET_PARAMETER
100	Continue	x	x	x	x
200	OK	x	x	x	x
300	Multiple Choices		x	x	x
301	Moved Permanently	x	x	x	x
302	Moved Temporarily	x	x	x	x
303	See Other	x	x	x	x
304	Not Modified	x	x	x	x
305	Use Proxy	x	x	x	x
400	Bad Request	x	x	x	x
401	Unauthorized		x	x	x
402	Payment Required		x	x	x
403	Forbidden	x	x	x	x
404	Not Found	x	x	x	x
405	Method Not Allowed	x	x	x	x
406	Not Acceptable	x	x	x	x
407	Proxy Authentication Required	x	x	x	x
408	Request Timeout	x	x	x	x
409	Conflict			x	x
410	Gone			x	x
411	Length Required			x	x
412	Precondition Failed			x	x
413	Request Entity Too Large	x	x	x	x
414	Request-URI Too Long	x	x	x	x
415	Unsupported Media Type				
45x	Only Valid for Stream				
45x	Invalid parameter				
45x	Not Enough Bandwidth				
45x	Illegal Conference Identifier				
45x	Illegal Session Identifier	x	x		
45x	Parameter Is Read-Only				x
45x	Header Field Not Valid				
500	Internal Server Error	x	x	x	x
501	Not Implemented	x	x	x	x
502	Bad Gateway	x	x	x	x
503	Service Unavailable	x	x	x	x
504	Gateway Timeout	x	x	x	x
505	RTSP Version Not Supported	x	x	x	x

Table 2: Status codes and their usage with RTSP methods

method	direction	requirement
GET	$C \rightarrow S$	recommended
SETUP	$C \rightarrow S$	recommended
PLAY	$C \rightarrow S$	required
PAUSE	$C \rightarrow S$	recommended
CLOSE	$C \rightarrow S$	required
REDIRECT	$S \rightarrow C$	optional
SESSION	$S \rightarrow C$	optional
RECORD	$C \rightarrow S$	optional
BYE	$C \rightarrow S$	required?
SET_PARAMETER	$C \rightarrow S, S \rightarrow C$	optional
GET_PARAMETER	$C \rightarrow S, S \rightarrow C$	optional

Table 3: Overview of RTSP methods

6.1 HELLO

RTSP sessions MAY be initiated by a HELLO message. The request URI is "*" to indicate that the request pertains to the session itself. The primary use of the HELLO message is to verify the identity of the sender to the receiver; both sides must authorize one another to enable full access to server resources. Unauthorized clients may be disconnected or restricted to a subset of server resources.

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:  HELLO * 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: HELLO * RTSP/1.0 1
      Authenticate: Digest realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Response:

Possible errors: 401, Parameter Not Understood

Examples:

```
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"
```

HS: I consider HELLO superfluous, not fully specified and just complicating client and server. It is also not clear how this is supposed to work when a client connects to the server, since it can't know ahead of time whether the server will issus a HELLO request. So it may connect, issue a SETUP, be refused and then somehow guess that it's supposed to wait for HELLO. Authentication of the client can be readily achieved by standard HTTP-like methods. On either retrieving the session description or the first SETUP, the server would refuse with 401, supply the authentication methods (and nonces) it is willing to accept and wait for the request to be re-issued with proper authentication. As with standard web browser, a client can cache the authentication message for efficiency.

Similarly, the client can ask the server to authenticate itself with the first request.

Feature announcement can be done using standard HTTP mechanism, with a well-defined registration mechanism for feature names.

RL: HELLO offers the opportunity to negotiate server features prior to actually needing them. A client may wish to poll a server for its features without actually causing any action to occur, and HELLO offers that opportunity.

6.2 GET

The GET method retrieves a session description from a server. It may use the Accept header to specify the session description formats that the client understands.

If the media server has previously been invited to a conference by the client, the GET request SHOULD contain the Conference header field. If the GET request contains a conference identifier, the media server MAY locate the conference description and use the multicast addresses and port numbers supplied in that description. The media server SHOULD only offer media types corresponding to the media types currently active within the conference. If the media server has no local reference to this conference, it returns status code 452.

The conference invitation should also contain an indication whether the media server is expected to receive or generate media, or both. (A VCR-like device would support both directions.) If the invitation does not contain an indication of the operations to be performed, the media server should accept and then reject inappropriate operations.

The server responds with a *description* of the requested resource.

Example:

```
C->S: GET /fizzle/foo RTSP/1.0 312
      Accept: application/sdp, application/sdf, application/mheg
      Bandwidth: 4000

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/x-rtsp-mh
        Content-Length: 2782

        <2782 octets of data containing stream descriptions and
        headers for the requested presentation>
```

The authors disagree amongst themselves as to whether having an GET method within RTSP is appropriate. The alternative would be that the stream header would be done as an HTTP GET, and then RTSP would be used for SETUP, PLAY, etc. RL believes there are a number of reasons why a GET method is appropriate within RTSP:

- An RTSP GET is a request for header information, while an HTTP GET is a request for the entire file. For instance, an RTSP GET on a Quicktime file with the name foo.mov would mean ”please send me the header and packetization information for foo.mov”, whereas an HTTP GET for that file would mean ”please send me foo.mov”.
- Assuming the client only has an URL to a resource, it is highly desireable to get to the point where the client is actually receiving data all over one connection. Though this would be possible if one assumed that one can multiplex RTSP and HTTP on the same connection, there is a question of how much of a web server would have to be supported in order to fulfill this simpler requirement.
- Since the RTSP GET contains information such as codec, packetization, total size, and whether the clip is live or stored, it is important to insure integrity between the session description and the media it represents. This information may be cached by HTTP proxies, but it would be needed by caching RTSP proxies.

RL and AR feel that the scope and applicability of this message should be limited, and therefore, it may be appropriate to come up with another name for this message.

HS believes that this only works if GET is required. Otherwise, the client has no way of knowing whether to first send a GET or SETUP. The easy alternative is to have any further descriptive information that is necessary encoded in the session description. Thus, the naming does not matter; the resource description can either have a separate name or the same name if the server can distinguish variants based on the requested type, as modern web servers can. (A single URL can return different objects depending on the content of the Accept fields.) It appears likely that the RTSP GET will over time acquire all the functionality of the HTTP GET and thus lead to unnecessary duplication. If the description is lengthy, the ability to use HTTP caches is likely to compensate for any additional latency due to having to open two streams. Also, note that relying on HTTP get does not mean that this has to be a separate server.

6.3 SETUP

The SETUP request for a URI specifies the transport mechanism to be used for the streamed media. Note that SETUP makes sense only for an individual media stream, rather than an aggregate. A client can issue a SETUP request for a stream that is already playing to change transport parameters.

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 transports acceptable to the client for data transmission; the response will contain the transport 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
```

6.4 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 request have been acknowledged as successful.

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 following example plays the whole session starting at SMPTE time code 0:10:20 until the end of the clip.

```
C->S: PLAY rtsp://audio.example.com/twister.en RTSP/1.0 833
      Range: smpte=0:10:20-
```

For playing back a recording of a live event, 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.

> RL says: We had considered optionally overloading PLAY with SETUP information. This would have potentially allowed a case where one could implement a minimal RTSP server that only handles the PLAY command. However, we decided that supporting that minimal of a server was problematic for a couple of reasons:
>
> - We must be able to negotiate the transport (i.e. have server acknowledgment) prior to actually needing to deal with the data. We don't want to have a server start spewing packets at us before we are ready to deal with them. The server acknowledgment with setup information MUST arrive before the first packet.
> - We need make sure that we aren't dealing with an allocation method every time we are dealing with PLAY. We anticipate the potential of dealing with PLAY frequently when a client chooses to issue several seeks, and so simplifying this message is imperative.
>
> HS says: PLAY without SETUP is useful and possible, in particular if the session description contains all necessary information, without options. The client knows whether it needs to configure transport parameters or not. For multicast delivery, for example, it likely would not have to set additional parameters. I doubt that allowing one additional parameter is going to greatly complicate or slow down a server.

6.5 PAUSE

The PAUSE request causes the stream delivery to be interrupted (halted) temporarily. If the request URL names a track, only playback and recording of that track is halted. If the request URL names a presentation, delivery of all currently active tracks is halted. After resuming playback or recording, synchronization of the tracks MUST be maintained. Any server resources are kept.

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
```

6.6 CLOSE

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

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

6.7 BYE

Terminate the session.

Example:

```
C->S: BYE /movie RTSP/1.0 894

S->C: RTSP/1.0 200 894 OK
      Date: 23 Jan 1997 15:35:06 GMT
```

HS: I believe BYE to be unnecessary since CLOSE already frees resources and session descriptor.

6.8 GET_PARAMETER

The requests retrieves the value value of a parameter of a session component 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. [HS: registry of parameter names for statistics and other purposes, possibly using the HTTP feature registration mechanism.] A 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
```

6.9 SET_PARAMETER

This method requests to set the value of a parameter for a session component 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
```

6.10 REDIRECT

A redirect request informs the client that it must connect to another server location. It contains the mandatory header Location, which indicates that the client should issue a GET for that URL. It may contain the parameter Range, which indicates when the redirection takes effect.

Mandatory header: Location [XXX: add this to table if accepted]

Example: This 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-
```

6.11 SESSION

This request is used by a media server to send new media information to the client. If a new media type is added to a session (e.g., during a live event), the whole session description should be sent again, rather than just the additional components.

This allows the deletion of session components.

Example:

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

      Session Description
```

6.12 RECORD

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

A media server supporting recording of live events 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
```

6.13 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.

7 Status Codes Definitions

Where applicable, HTTP status [H10] codes are re-used. Status codes that have the same meaning are not repeated here. See Tables 1 and 2 for a listing of which status codes may be returned by which request.

7.1 Client Error 4xx

7.1.1 451 Parameter Not Understood

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

7.1.2 452 Conference Not Found

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

7.1.3 453 Not Enough Bandwidth

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

7.1.4 45x Session Not Found

7.1.5 45x Method Not Valid in This State

7.1.6 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.

7.1.7 45x Invalid Range

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

7.1.8 45x Session Id Not Valid

7.1.9 45x Parameter Is Read-Only

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

8 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 4 and 5 summarize the header fields used by RTSP. Type “R” designates requests, type “r” responses. Fields marked with “x” MUST be implemented by the recipient. If the field content does not apply to the particular resource, the server MUST return status 45x (Header Field Not Valid for Resource).

	type	HELLO	GET	SETUP	PLAY	RECORD	PAUSE
Accept	R		x				
Accept-Encoding	R		x				
Accept-Language	R		x	o	o		
Authorization	R	o	o	o	o	o	o
Bandwidth	R			o			
Blocksize	R			o			
Conference	R		o	o	?	?	?
Connection	Rr		x	x	x	x	x
Content-Encoding	Rr		x				
Content-Length	Rr		x				
Content-Type	Rr		x				
Date	Rr	o	o	o	o	o	o
If-Modified-Since	R		o				
Last-Modified	r		o				
Public	r	o	o	o	o	o	o
Range	R				x	x	
Referer	R	o	o	o	o	o	o
Require	R	x	o	x	o	o	o
Retry-After	r	o	o	o	o	o	o
Session	Rr			x	x	x	x
Server	r	o	o	o	o	o	o
Speed	Rr				o	o	
Transport	Rr			x			
Transport-Require	R	x	o	x	o	o	o
User-Agent	R	o	o	o	o	o	o
Via	Rr	o	o	o	o	o	o
WWW-Authenticate	r	o	o	o	o	o	o

Table 4: Overview of RTSP header fields for GET, SETUP, PLAY, RECORD and PAUSE

	CLOSE	REDIRECT	GET_PARAMETER	SET_PARAMETER
Accept				
Accept-Encoding				
Accept-Language				
Authorization	o	o	o	o
Bandwidth				
Blocksize				
Conference				
Connection	x	x	x	x
Content-Encoding	x	x	x	x
Content-Length	x	x	x	x
Content-Type	x	x	x	x
Date	o	o	o	o
If-Modified-Since				
Last-Modified				
Public	o	o	o	o
Range				
Referer	o	o	o	o
Retry-After	o	o	o	o
Session	x	x	x	x
Server	o	o	o	o
Speed				
Transport				
User-Agent	o	o	o	o
Via	o	o	o	o
WWW-Authenticate	o	o	o	o

Table 5: Overview of RTSP header fields for PAUSE, CLOSE, GET_PARAMETER and SET_PARAMETER

8.1 Accept

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

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

See [H14.1] for syntax.
Example of use:

```
Accept: application/sdf, application/sdp;level=2
```

8.2 Accept-Encoding

See [H14.3]

8.3 Accept-Language

See [H14.4]. Note that the language specified applies to the session description, not the media content.

8.4 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
```

8.5 Authorization

See [H14.8]

8.6 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
```

8.7 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.

8.8 Conference

This request header field establishes a logical connection between a conference, established using non-RTSP means, and an RTSP stream.

```
Conference = "Conference" ":" conference-id
```

Example:

```
Conference: 199702170042.SAA08642@obiwan.arl.wustl.edu%20Starr
```

8.9 Content-Encoding

See [H14.12]

8.10 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].

8.11 Content-Type

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

8.12 Date

See [H14.19].

8.13 If-Modified-Since

See [H14.24]. If the request URL refers to a presentation rather than a track, the server is to return the presentation if any of the track has been modified since the time stated in the header field.

8.14 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.

8.15 Location

See [H14.30].

8.16 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 3.5) range units. Within RTSP, byte ranges [H14.36.1] are not meaningful and MUST NOT be used.

```
Range = "Range" ":" 1#ranges-specifier

ranges-specifier = utc-range | smpte-range
```

Example:

```
Range: clock=19960213T143205Z-
```

8.17 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.

8.18 Unsupported

See Section 8.17 for a usage example.

> HS: same caveat as for Require applies.

8.19 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.

8.20 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 8.17 for more details on the mechanics of this message and a usage example.

> HS: Same caveat as for Require applies.

8.21 Retry-After

See [H14.38].

8.22 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.

```
speed = "Speed" ":" ["-"]1*DIGIT [ "." *DIGIT ]
```

Example:

```
Speed: 2.5
```

8.23 Server

See [H14.39]

8.24 Session

This request and response header field identifies a session, started by the media server in a SETUP or PLAY response and concluded by CLOSE on the session URL (presentation). 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 [1]. 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 CLOSE 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.

8.25 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 session description. In some cases, the session 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. 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)i 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 session.

```
Transport = "Transport" ":"
            1#transport-protocol/upper-layer *parameter
transport-protocol = "UDP" | "TCP"
upper-layer  = "RTP"
parameters = ";" "multicast" |
```

```
                   ";" "compressed" |
                   ";" "interleaved" |
                   ";" "ttl" "=" ttl |
                   ";" "port" "=" port |
                   ";" "ssrc" "=" ssrc
  ttl          = 1*3(DIGIT)
  port         = 1*5(DIGIT)
  ssrc         = 8*8(HEX)
```

Example:

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

8.26 User-Agent

See [H14.42]

8.27 Via

See [H14.44].

8.28 WWW-Authenticate

See [H14.46].

9 Caching

In HTTP, response-request pairs are cached. RTSP differs significantly in that respect. Typically, responses are not cachable (except maybe for the GET response), rather it is desirable for the media data (that is typically delivered outside of RTSP) to be cached. Since the responses for anything but GET and GET_PARAMETER do not return any data, caching is not an issue for these requests.

HS: A proxy cache for RTSP would look not much different from an HTTP cache. To the client, the proxy cache would appear like a regular media server, to the media 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 session description. Typically, a cache would eliminate all transport-references (that is, multicast information) from the session 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 session description with all the encoding possibilities it can offer.

10 Examples

To emphasize that RTSP is independent of the session description format, the following examples use a fictional session description language which is chosen to be sufficiently self-explanatory.

10.1 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. This, however, is transparent to the client. 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: GET /twister HTTP/1.1
      Host: www.example.com
      Accept: application/sdf; application/sdp

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

       (session
         (all
          (media (t audio) (oneof
             ((e PCMU/8000/1 89 DVI4/8000/1 90) (id lofi))
             ((e DVI4/16000/2 90 DVI4/16000/2 91) (id hifi))
            )
            (language en)
            (id rtspu://audio.example.com/twister/audio.en)
            )
            (media (t video) (e JPEG)
              (id rtspu://video.example.com/twister/video)
            )
          )
        )

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: CLOSE rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0 3
      Session: 1234

A->C: 200 3 OK

C->V: CLOSE 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.

10.2 Live Media Event 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 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/sdf

       (session
         (id rtsp://live.example.com/concert)
       )

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

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

       (session (all
```

```
    (media (t audio) (id music) (a IP4 224.2.0.1) (p 3456))
  ))

C->M: PLAY rtsp://live.example.com/concert/music RTSP/1.0 2
      Range: smpte 1:12:0

M->C: RTSP/1.0 405 2 No positioning possible

M->C: SESSION concert RTSP/1.0
      Content-Type: application/sdf

      (session (all
        (media (t audio) (id music))
        (media (t text) (id lyrics))
      ))

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

Since the session description already contains the necessary address information, the client does not set the transport address. The attempt to position the stream fails since this is a live event.

10.3 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 session 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/sdf, application/sdp

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

      (session
         (id 548)
         (media (t audio) (id sound)
      )

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

10.4 Recording

Conference participant C asks the media server M to record a session. If the session description contains any alternatives, the server records them all.

```
C->M: SESSION 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/sdf

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

M->C: 200 90 OK

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

11 Syntax

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

11.1 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>
```

11.2 Internet Media Type Syntax

```
media-type = type "/" subtype *( ";" parameter )
type = token
subtype = token
parameter = attribute "=" value
attribute = token
value = token | quoted-string
```

11.3 Universal Resource Identifier Syntax

```
uri = ( absoluteURI | relativeURI ) [ "#" fragment ]
absoluteURI = scheme ":" *( uchar | reserved )
relativeURI = net-path | abs-path | rel-path
net-path = "//" net-loc [ abs-path ]
abs-path = "/" rel-path
rel-path = [ path ] [ ";" params ] [ "?" query ]
path = fsegment *( "/" segment )
fsegment = 1*pchar
segment = *pchar
params = param *( ";" param )
param = *( pchar | "/" )
scheme = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc = *( pchar | ";" | "?" )
query = *( uchar | reserved )
fragment = *( uchar | reserved )
pchar = uchar | ":" | "@" | "&" | "=" | "+"
uchar = unreserved | escape
unreserved = ALPHA | DIGIT | safe | extra | national
escape = "%" HEX HEX
```

```
reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra = "!" | "*" | "'" | "(" | ")" | ","
safe = "$" | "-" | "_" | "."
unsafe = CTL | SP | <"> | "#" | "%" | "<" | ">"
national = <any OCTET excluding ALPHA, DIGIT, reserverd, extra,
    safe, and unsafe>
```

11.4 RTSP-specific syntax

```
response = response-line CRLF
   *response-header
   CRLF
   [ message-body ]

response-line = rtsp-version status-code
              sequence-number reason-phrase CRLF

setup-response = response-line
                 date-type-header
 *( nack-require-header
  | nack-require-transport-header )
  CRLF


get-response = response-line
               date-type-header
               *( authorize-header
        | content-length-header
| content-type-header )
CRLF
[ message-body ]

redirect-response = response-line
                    date-type-header

session-response = response-line
                   date-type-header

play-response = response-line
                date-type-header

pause-response = response-line
                 date-type-header

response-header = allow-header
```

```
                | address-header
                | authorize-header
| blocksize-header
                | content-length-header
| content-type-header
| location-header
| extension-header

message-body = *OCTET

accept-header = "Accept" ":" 1#media-type
allow-header = "Allow" ":" 1#method
blocksize-header = "Blocksize" ":" 1*DIGIT
content-length-header = "Content-Length" ":" 1*DIGIT
content-type-header = "Content-Type" ":" media-type
date-type-header = "Date" ":" rfc1123-date
location-header = "Location" ":" request-uri
require-header = "Require" ":" #parameters
transport-require-header = "Transport-Require" ":" #parameters
nack-require-header = "Nack-Require" ":" #parameters
nack-transport-require-header = "Nack-Transport-Require" ":" #parameters

auth-scheme = token
ip-address = <IP address in dotted-decimal form per RFC 1123>
port-number = 1*DIGIT
blocksize-value = 1*DIGIT
credentials = auth-scheme ":" #parameter
rfc1123-date = wkday "," SP date SP time SP "GMT"
date = 2DIGIT SP month SP 4DIGIT ; day month year (e.g., 12 Dec 1998)
time = 2DIGIT ":" 2DIGIT ":" 2DIGIT ; 00:00:00 - 23:59:59
wkday = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"
month = "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun"
      | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"

method = "SETUP"
       | "GET"
       | "REDIRECT"
       | "PLAY"
       | "PAUSE"
       | "SESSION"
       | "CLOSE"
       | "BYE"
       | extension-method
extension-method = token
```

```
status-code = "200"   ; OK
            | "301"   ; Moved Permanently
            | "302"   ; Moved Temporarily
            | "400"   ; Bad Request
            | "401"   ; Unauthorized
            | "402"   ; Payment Required
            | "404"   ; Not Found
            | "405"   ; Method Not Allowed
            | "406"   ; Not Acceptable
            | "408"   ; Request Time-out
            | "411"   ; Length Required
            | "414"   ; Request-URI Too Large
            | "415"   ; Unsupported Media Type
    | "450"   ; Invalid Parameter
    | "451"   ; Parameter Not Understood
    | "453"   ; Not Enough Bandwidth
    | "454"   ; Unsupported Transport
            | "500"   ; Internal Server Error
            | "501"   ; Not Implemented
            | "502"   ; Bad Gateway
            | "503"   ; Service Unavailable
            | "504"   ; Gateway Time-out
            | "505"   ; RTSP Version not supported
            | extension-code
extension-code = 3DIGIT

reason-phrase  = *<TEXT, excluding CR, LF>
```

12 Experimental

This section gathers parts of the protocol which are less well understood and require extensive further discussion.

12.1 Header Field Definitions

The following additional HTTP headers may be useful for RTSP:

- Accept-Language
- Cache-Control
- From
- Max-Forwards
- Proxy-Authenticate

- Proxy-Authorization
- Public
- Referer

12.1.1 Address

Designates address to send multimedia data to.

It appears that in almost all cases, the destination address is the same one where the RTSP command originates from. If TCP is used for control, this also eliminates the possibilities of pointing a data stream at an unsuspecting third party.

13 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 State Machines

The RTSP client and server state machines describe the behavior of the protocol from session initialization through 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 session identifier. (A server may choose to generate dynamic session descriptions where the URL is unique for a particular session and thus may not need an explicit session identifier in the request header.) Any request/reply using URLs denoting a session comprised of multiple streams will have an effect on the individual states of all the substreams. For example:

Assuming the stream /coolmovie contains two substreams /coolmovie/audio and /coolmovie/video, then the following command:

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

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

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

A.1 Client State Machine

A.1.1 Client States

These are defined as follows:

NULL: No state

INIT: GET or SETUP has been sent, waiting for reply.

READY: SETUP reply received OR after playing, PAUSE reply received.

PLAYING: PLAY reply received

A.1.2 Notes

In general, the client transitions state on receipt of specific replies. After a period of inactivity, state transitions back to NULL. ”Inactivity” is defined as one of the following:

- For state PLAYING, no data being received and/or lack of wellness information from the server.
- The client stays in any other state continuously for more than a specific interval. The choice of this interval is left to the implementation.

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, i.e READY and PLAYING.

A client 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.1.3 State Table

In the NEXT STATE column, + indicates that the message was successful, -indicates that it was unsuccessful.

```
STATE           MESSAGES                NEXT STATE(+)   NEXT STATE(-)


INIT            GET REPLY               INIT            NULL
                SETUP REPLY             READY           INIT
                REDIRECT                NULL            NULL
                BYE                     NULL            NULL
                OTHER                   INIT            INIT

READY           PLAY REPLY              PLAYING         READY
                SETUP REPLY             READY           INIT
                BYE                     NULL            NULL
                OTHER                   READY           READY

PLAYING         PAUSE REPLY             READY           PLAYING
```

```
                PLAY REPLY          PLAYING        CLOSED
                BYE                 NULL           NULL
                CLOSE REPLY         NULL           PLAYING
                OTHER               PLAYING        PLAYING
```

This assumes that a `PLAY` during state PLAYING is an implicit `PAUSE`, `PLAY`.

HS: BYE should be replaced by CLOSE.

A.2 Server State Machine

A.2.1 Server States

INIT: The initial state, no valid SETUP receieved.

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 actually being sent.

In general, server state transitions occur on receiving requests. On receiving a BYE, state transitions back to INIT. After inactivity for a period, state also transitions back to INIT. "Inactivity" is defined as:

- For states other than PLAYING, no messages for that object for a specific interval. The choice of interval is left to the implementation.
- In state PLAYING, lack of wellness information from the client.(This information could be either RTCP or be requested by the server by other means)

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 session description. For purposes of this table, a `REDIRECT` is considered an unsuccessful `GET`.

A server 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.

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, ie. there are only two states READY and PLAYING.

A.2.2 State Table

In the NEXT STATE column, + indicates that the message was successful, -indicates that it was unsuccessful.

```
STATE           MESSAGES            NEXT STATE(+)  NEXT STATE(-)

INIT            GET                 INIT           INIT
                SETUP               READY          INIT
```

	BYE	INIT	INIT
	OTHER	-	INIT
READY	PLAY	PLAYING	READY
	SETUP	READY	INIT
	CLOSE	INIT	-
	BYE	INIT	-
	OTHER	-	READY
PLAYING	PLAY	PLAYING	READY
	PAUSE	READY	PLAYING
	CLOSE	INIT	PLAYING
	BYE	INIT	-
	OTHER	-	PLAYING

B Open Issues

- Define text/rtsp-parameter MIME type.
- Lots of inconsistencies need to be fixed: naming of methods in state definition, syntax.
- 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.
- Cache and proxy behavior?
- Session: or Set-Cookie: ?
- Behavior of all methods in state diagram.
- Error message for method
- When do relative RTSP URLs make sense?
- Nack-require, etc. are dubious. This is getting awfully close to the HTTP extension mechanisms [16] in complexity, but is different.
- Suggestion (HS): shelve REDIRECT method for now, until necessity becomes clear.
- Use HTTP absolute path + Host field or do the right thing and carry full URL, including host in request?

C 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

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

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

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

[4] 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.

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

[6] 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.

[7] J. Postel, “User datagram protocol,” STD 6, RFC 768, Internet Engineering Task Force, Aug. 1980.

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

[9] J. Postel, “Transmission control protocol,” STD 7, RFC 793, Internet Engineering Task Force, Sept. 1981.

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

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

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

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

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

[15] 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.

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