Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-mmusic-stream-00.ps

MMUSIC WG
H. Schulzrinne
Columbia U.

November 26, 1996
Expires: 26/8/97

A real-time stream control protocol (RTSP')

Status of this Memo

Abstract

This strawman proposal presents a revised version of the RTSP proposal put forward to the MMUSIC group, borrowing liberally from the original.

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).

1 Introduction

1.1 Terminology

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

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

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) stream: A single media instance, e.g., an audio stream or a video stream as well as a whiteboard or shared application session. 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.]

media session: A collection of media streams to be treated. Typically, a client will synchronize all media streams within a media session.

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.

Both client and server can send commands.
The protocol supports the following operations:

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

Invitation of media server to 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".

Adding media to an existing session: Particularly for live events, it is useful if the server can tell the client about additional media becoming available.

1.2 Requirements

The protocol satisfies the following requirements

extendable: new commands and parameters can be easily added

easy to parse: standard HTTP or MIME parsers can (but do not have to be) used

secure: re-uses web security mechanisms, either at the transport level (SSL) or within the requests (basic and digest authentication)

transport-independent: may use either an unreliable datagram protocol (UDP), a reliable datagram protocol (RDP, not widely used) or a reliable stream protocol (TCP) by implementing 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.

multi-client capable: Stream identifiers can be used by several control streams, so that "passing the remote" is possible. The protocol does not address how several clients negotiate access; this is left to either a "social protocol" or some other floor control mechanism.

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, S*IP 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.

S*IP compatible: As much as possible, stream control should be aligned with the IETF conference initiation effort. However, for simple applications, a media server should not have to implement a conference initiation protocol.

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) firewalls. For proxies, re-use of existing proxies should be possible, but remains to be verified. [TBD: what exactly is needed to make a protocol firewall-friendly?] A firewall may need to understand the SET_PORT directive 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.

1.3 Extending the Protocol

The protocol described below can be extended in three ways, listed in order of the magnitude of changes supported:

- Existing commands 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.

1.4 Overall Operation

Each media stream and session is 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. 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.

1.5 Relationship with Other Protocols

RTSP' has some overlap in functionality with HTTP. It also needs to interact with the web 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. The current RTSP already has first hints on caches and proxies, but is nowhere near as complete as HTTP in that regard.

It is possible to very quickly build a simple RTSP' server by adding a PLAY and, optionally, a SET_PARAMETER method to an existing HTTP/1.1 web server. All of RTSP' can be implemented as part of an HTTP server as long as only the client issues requests.

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 Protocol Parameters

2.1 Message Format and Transmission

RTSP is a text-based protocol [TBD] and uses the ISO 10646 character set in UTF-8 encoding (RFC 2044) [TBD; this conflicts with]. 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 in scripting languages such as Tcl, VisualBasic 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.

Commands 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 STOP 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 [PORT] 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. An RTSP packet is terminated with an empty line. (TBD: doesn’t work well for including session descriptions. Maybe use Content-length for payloads - these are usually imported anyway? or new page? Wrapping a packet in some kind of braces or parenthesis is another possibility, but again puts restrictions on the SDF.)

> Unless all but the RTP data is textual, there is not much point in keeping the payload as textual data, since visual debugging is more difficult and "telnet protocol emulation" is no longer possible. Length fields don't make much sense for textual data, particularly because of the line termination ambiguities, i.e., CR, LF and CRLF. There does not seem to be a need for an explicit, connection-oriented framing layer as in the original RTSP proposal. However, if we allow interleaving with RTP, a textual format gets very awkward.

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.

A message has the following format:

```
Method Object Version Sequence-Number
*(Parameter-Value)
CRLF
```

A message with a message body has the following format:

```
Method Object Version Sequence-Number
Content-length:
*(Parameter-Value)
CRLF
message-body
```

After receiving and interpreting an RTSP' request, the server responds with an RTSP' response message. [TBD: proper BNF]

A typical response to a request with sequence number 17 might be:

```
RTSP/1.0 200 17 OK
```

> This format is HTTP-friendly; the sequence number is simply ignored by HTTP servers. The likelihood that a textual protocol will share the same port and not have that format seems fairly remote. RTP packets have the most-significant bit set and can thus be easily distinguished.

If a connectionless transport protocol is used, the media server considers all packets originating from a single port number and network address to be part of the same session. [TBD: is this necessary?]

2.2 Session and Media URI

The RTSP URL scheme is used to locate and control stream resources via the RTSP protocol.

A media stream is identified by an textual session and media identifier, using the character set and escape conventions of URLs. The media identifier is separated from the session by a slash. Commands below can refer to either the whole session or an individual stream. Stream identifiers can be passed between clients ("passing the remote control"). A specific instance of a session, e.g., one of several concurrent transmissions of the same content, is appended where needed. The instance identifies the whole session, so that all media streams within that session have the same instance identifier.

For example,

```
rtsp://media.content.com:5000/twister/audio.en/1234
```

identifies instance 1234 of the stream audio.en within the session "twister", which is located at port 5000 of host media.content.com.

The ordering and significance of the path components of the rtsp URL is only of significance to the media server.

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

2.3 Encoding Identifiers

RTP profile and/or MIME types. [TBD: should probably register all the RTP data types as MIME types.]

2.4 Conference Identifiers

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

> 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.

2.5 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.

Examples:

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

2.6 Absolute Time

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

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

```
19961108T143720Z
```

3 Header Field Definitions

3.1 Accept

The Accept request-header field can be used to specify certain session description types which are acceptable for the response. The only parameter allowed is that of `level`, which indicates the highest level or version accepted by the requestor.

Example of use:

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

3.2 Address

3.3 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: PLAY, RECORD, SET_PARAMETER
```

3.4 Authorization

3.5 Blocksize

3.6 Conference

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

[TBD: This parameter is for further study. May not be needed with the Given parameter.]

3.7 Content-Length

3.8 Content-Type

3.9 Given

3.10 Location

3.11 Port

3.12 Range

3.13 Speed

3.14 Transport

3.15 TTL

4 Methods

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`.

4.1 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.

```
GET twister RTSP/1.0 937
Accept: application/sdp, application/sdf, application/mheg
```

If the media server has previously been invited to a conference, the GET method also contains a conference identifier or a `Given` parameter.

```
GET twister RTSP/1.0 834
Conference: 128.16.64.19/32492374
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

If the GET request contains a conference identifier, the media server MUST 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.

A typical response might be:

```
200 18 OK
Content-Type:  application/sdf
```
session description

4.2 SESSION

This method is used by a media server to send 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:

```
SESSION twister/*/1234 Content-Type:  application/sdp
```
Session Description

Response: 200, 302, 303, 500, can't do this operation, busy,

4.3 PLAY

The PLAY method tells the server to start sending data via the previously set transport mechanism. The `Range` header specifies the range. The range can be specified in a number of units. This specification defines the `smpte` (see Section 2.5) and `clock` (see Section 2.6) range units.

`PLAY` *media-name*
`Range:  smpte=`*range-value*

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

```
PLAY twister/*/1234
Range: smpte=0:10:20-
```

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

```
PLAY meeting/*/1234
Range: clock=19961108T142300Z-19961108T143520Z
```

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

[TBD: It may be desirable to allow several ranges, so that remote digital editing can be done easily.]

Response: 200, 500, 501, clock format not supported.

4.4 RECORD

This method initiates eecording 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.

```
RECORD meeting/audio.en/1234
Conference: 128.16.64.19/32492374
```

4.5 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.

4.6 SET_PARAMETER

Both client and media server can issue this request.

The following parameters are defined:

Blocksize: This advisory parameter is sent from the client to the media server setting the transport packet size. The server truncates 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 bytes. The server only returns an error (416) if the value is syntactically invalid, but not if the server adjusts it according to the mechanism described above or decides to simply ignore the advice.

Port: UDP or TCP port to be used for this media.

SSRC: RTP SSRC value to be 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. This can be used for demultiplexing by the client of data received on the same port.

Address: Destination network address, consisting of the address class identifier and the address. Currently, the address classes IP4 and IP6 are defined.

Transport: Transport protocol stack to be used: UDP or TCP or interleaved, followed by the next-layer transport protocol. '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. Example: `UDP RTP;compressed`

TTL: Multicast time-to-live value. In some cases, it may make sense for a client to ask a media server sending on a given multicast address to expand its range.

Speed: This advisory parameter sets the speed at which the server delivers data to the client, 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., 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.

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.

The parameters are split in a fine-grained fashion so that, for example, the client can set just the unicast port, without having to modify the destination address. There is no substantial difference between the privileged parameters and the parameters identified by family and parameter id in the current RTSP spec. If desired, parameter names could easily take the form `family/parameter`, e.g., `Audio/Annotations`.

A SET_PARAMETER request without parameters can be used as a way to detect whether the other side is still responding.

Example:

```
SET_PARAMETER twister/1234/audio.en RTSP/1.0 68
Speed: 2.3
```

[TBD: Or should this be like SET_PARAMETER? Bit longer, but forces single parameter per request.]

4.7 GET_PARAMETER

Both client and media server can issue a GET_PARAMETER request to retrieve a specific parameter. All parameters described for the SET_PARAMETER request are valid. In the request, the message body contains the parameter value. Only one parameter can be requested in each GET_PARAMETER request.

Example:

```
C->S: GET_PARAMETER twister/1234/audio.en RTSP/1.0 6
      Content-length: 17

      Audio/Annotations

S->C: RTSP/1.0 200 6 OK
      Content-type: text/ascii
      Content-length: 2

      64
```

4.8 STOP

Stops delivery of stream immediately. Returns indication of current position to allow play instead of resume.

Thus, RESUME is not needed.

```
C->M: STOP movie RTSP/1.0 76

M->C: RTSP/1.0 200 76 OK
```

4.9 BYE

Sent by either client or server to terminate a connection and release resources.

4.10 Embedded Data Stream

The command DATA is used to indicate an embedded media data object, together with the content types. DATA requests are not acknowledged by RTSP'. The embedded object can have any type. For space-efficient encapsulation of binary data, the method in Section 4.11 should be used instead.

```
DATA twisters/audio.en/1234 RTSP/1.1
Content-Length: 500
Content-Type: message/rtp

(RTP data)
```

> This is workable, but not very space-efficient. However, the interesting case is that of a single TCP stream carrying both commands and media data. There is no particular reason to have small chunks in that case.

4.11 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.

> This makes the encapsulation overhead 4 bytes, less than the 8 bytes imposed by SCP.

5 Status Codes Definitions

Where applicable, HTTP status codes are re-used. [TBD: add those relevant here]

5.1 Successful 2xx

5.1.1 200 OK

The request has succeeded. The information returned with the response depends on the method used in the request, for example:

GET: the session description;

GET_PARAMETER: the value of the parameter.

5.2 Redirection 3xx

5.2.1 301 Moved Permanently

5.2.2 303 Moved Temporarily

5.3 Client Error 4xx

5.3.1 400 Bad Request

The request could not be understood by the recipient due to malformed syntax. The request SHOULD NOT be repeated without modification.

5.3.2 401 Unauthorized

The request requires user authentication.

5.3.3 402 Payment Required

This code is reserved for future use.

5.3.4 405 Method Not Allowed

5.3.5 406 Not Acceptable

5.3.6 408 Request Timeout

5.3.7 411 Length Required

5.3.8 414 Request URI Too Long

5.3.9 415 Unsupported Mediatype

The recipient of the request is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

5.3.10 450 Invalid Parameter

The parameter in the request is not valid, i.e., out of range or malformed.

5.3.11 451 Parameter Not Understood

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

5.3.12 452 Conference Not Found

The conference indicated by a `Conference:` identifier is unknown to the media server.

5.3.13 453 Not Enough Bandwidth

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

5.4 Server Error 5xx

5.4.1 500 Internal Server Error

5.4.2 501 Not Implemented

5.4.3 502 Bad Gateway

5.4.4 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated.

5.4.5 504 Gateway Timeout

5.4.6 505 RTSP Version Not Supported

6 Examples

6.1 Media on demand (unicast)

Client C requests a movie media servers A (audio.content.com) and V (video.content.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/1.0
      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.content.com/twister/audio.en/1234)
            )
            (media (t video) (e JPEG)
              (id rtspu://video.content.com/twister/video/1234)
            )
          )
        )
```

```
C->A: SET_PARAMETER twister/audio.en/1234/lofi RTSP/1.0 1
      Port: 3056
      Transport: RTP;compression

A->C: RTSP/1.0 200 1 OK

C->V: SET_PARAMETER twister/video/1234/hifi RTSP/1.0 2
      Port: 3058
      Transport: RTP;compression

V->C: RTSP/1.0 200 2 OK

C->V: PLAY twister/video/1234 RTSP/1.0 3
      Range: smpte 0:10:00-

V->C: RTSP/1.0 200 3 OK

C->A: PLAY twister/audio.en/1234/lofi RTSP/1.0 4
      Range: smpte 0:10:00-

S->C: 200 4 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.

6.2 Live Media Event Using Multicast

The media server 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/1.0

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

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

C->M: GET concert RTSP/1.0 1

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

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

C->M: PLAY concert/music RTSP/1.0
      Range: smpte 1:12:0

M->C: RTSP/1.0 405 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 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.

6.3 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 RTSP/1.0 1
      Accept: application/sdf, application/sdp
      Given: address, privacy

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

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

C->M: SET_PARAMETER demo/548/sound RTSP/1.0 2
      Address: IP4 224.2.0.1
      Port:    3456
      TTL:     127
```

6.4 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 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 meeting RTSP/1.0 90
      Content-type: application/sdf

M->C: 200 90 OK

C->M: RECORD meeting
      Range: clock 19961110T1925-19961110T2015
```

7 Access Authentication

Besides limiting access, access authentication is also needed to avoid denial-of-service attacks.

8 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. This can be prevented by a challenge-response authentication. If the goal is simply to prevent this denial-of-service attack, a default, widely known key can be used.

If the client retrieves a session description, the server hand out an encrypted version of the client's IP address to the client during the initial retrieval of the session description.

A Session Description

A session description must be able to identify sessions and individual media streams. The per-media identifier is created by the entity creating the session description and is opaque to anyone else. It may contain any 8-bit value except CR and LF.

B Notes on RTSP

- The STREAM_HEADER functionality has been subsumed by the session description.
- SEND_REPORT is not really needed. Should define an RTCP request with a random response interval.
- Error reports are sent automatically. If server wants to terminate connection, it sends a BYE.
- Resending (UDP_RESEND) should be handled by RTCP since it is always media-specific and RTCP can be readily flow-controlled to avoid congestion collapse.
- Is STOP really needed? What's the difference between STOP and PAUSE? Resources (which?) cannot be released since there may be a PLAY command immediately. Bearing on resource reservation?

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`

D Acknowledgements

This draft is based on the functionality of the RTSP draft. It also borrows format and descriptions from HTTP/1.1.