Internet Engineering Task Force MMUSIC WG
INTERNET-DRAFT H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, A. Narasimhan
draft-draft-ietf-mmusic-rfc2326bis-04.ps Columbia U./Cisco/RealNetworks/Ericsson/Sun
June 30, 2003
Expires: December, 2003

Real Time Streaming Protocol (RTSP)

Status of this Memo

Copyright Notice

Abstract

This memorandum is a revision of RFC 2326, which is currently a Proposed Standard.

The Real Time Streaming Protocol, or RTSP, is an application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions, provide a means for choosing delivery channels such as UDP, multicast UDP and TCP, and provide a means for choosing delivery mechanisms based upon RTP (RFC 1889).

Contents

1 Introduction

1.1 The Update of the RTSP Specification

This is the draft to an update of RTSP which is currently a proposed standard defined in RFC 2326 [21]. Many flaws have been found in RTSP since it was published. While this draft tries to address the flaws, not all known issues have been resolved.

The goal of the current work on RTSP is to progress it to draft standard status. Whether this is possible without first publishing RTSP as a proposed standard depends on the changes necessary to make the protocol work. The list of changes in chapter F indicates the issues that have already been addressed. The currently open issues are listed in chapter E.

There is also a list of reported bugs available at "http://rtspspec.sourceforge.net". These bugs should be taken into account when reading this specification. While a lot of these bugs are addressed, not all are yet accounted for in this specification. Input on the unresolved bugs and other issues can be sent via e-mail to the MMUSIC WG's mailing list mmusic@ietf.org and the authors.

Take special notice of the following:

- The example section 15 has not yet been revised since the changes to protocol have not been completed.
- The BNF chapter 16 has not been compiled completely.
- Not all of the contents of RFC 2326 are part of this draft. In an attempt to prevent the draft from exploding in size, the specification has been reduced and split. The content of this draft is the core specification of the protocol. It contains the general idea behind RTSP and the basic functionality necessary to establish an on-demand play-back session. It also contains the mechanisms for extending the protocol. Any other functionality will be published as extension documents. Two proposals exist at this time:
- NAT and FW traversal mechanisms for RTSP are described in a document called "How to make Real-Time Streaming Protocol (RTSP) traverse Network Address Translators (NAT) and interact with Firewalls." [33].
- The MUTE extension [34] contains a proposal on adding functionality to mute and unmute media streams in an aggregated media session without affecting the time-line of the playback.

There have also been discussions about the following extensions to RTSP:

- Transport security for RTSP messages (rtsps).
- Unreliable transport of RTSP messages (rtspu).
- The Record functionality.
- A text body type with suitable syntax for basic parameters to be used in SET_PARAMETER, and GET_PARAMETER. Including IANA registry within the defined name space.
- A RTSP MIB.

However, so far, they have not become concrete proposals.

1.2 Purpose

The Real-Time Streaming Protocol (RTSP) establishes and controls single or several time-synchronized streams of continuous media such as audio and video. Put simply, RTSP acts as a "network remote control" for multimedia servers.

There is no notion of a RTSP connection in the protocol. Instead, a RTSP server maintains a session labelled by an identifier to associate groups of media streams and their states. A RTSP session is normally not tied to a transport-level connection such as a TCP connection. During a session, a client may open and close many reliable transport connections to the server to issue RTSP requests for that session.

This memorandum describes the use of RTSP over a reliable connection based transport level protocol such as TCP. RTSP may be implemented over an unreliable connectionless transport protocol such as UDP. While nothing in RTSP precludes this, additional definition of this problem area must be handled as an extension to the core specification.

The mechanisms of RTSP's operation over UDP were left out of this spec. because they were poorly defined in RFC 2336 [21] and the tradeoff in size and complexity of this spec. for a small gain in a targeted problem space was not deemed justifiable.

The set of streams to be controlled is defined by a presentation description. This memorandum does not define a format for the presentation description. The streams controlled by RTSP may use RTP [1] for their data transport, but the operation of RTSP does not depend on the transport mechanism used to carry continuous media. The protocol is intentionally similar in syntax and operation to HTTP/1.1 [26] 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.
- RTSP has the notion of a session built into the protocol.
- A RTSP server needs to maintain state by default in almost all cases, as opposed to the stateless nature of HTTP.
- Both a RTSP server and client can issue requests.
- Data is usually carried out-of-band by a different protocol. Session descriptions returned in a DESCRIBE response (see Section 11.2) and interleaving of RTP with RTSP over TCP are exceptions to this rule (see Section 11.11).
- RTSP is defined to use ISO 10646 (UTF-8) rather than ISO 8859-1, consistent with current HTML internationalization efforts [3].
- The Request-URI always contains the absolute URI. Because of backward compatibility with a historical blunder, HTTP/1.1 [26] carries only the absolute path in the request and puts the host name in a separate header field.

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

The protocol supports the following operations:

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

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

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

RTSP requests may be handled by proxies, tunnels and caches as in HTTP/1.1 [26].

1.3 Requirements

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

1.4 Terminology

Some of the terminology has been adopted from HTTP/1.1 [26]. Terms not listed here are defined as in HTTP/1.1.

Aggregate control: The concept of controlling multiple streams using a single timeline, generally maintained by the server. A client, for example, uses aggregate control when it issues a single play or pause message to simultaneously control both the audio and video in a movie.

Aggregate control URI: The URI used in a RTSP request to refer to and control an aggregated session. It normally, but not always, corresponds to the presentation URI specified in the session description. See Section 11.3 for more information.

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

Client: The client requests media service from the media server.

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

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

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

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

Feature-tag: A tag representing a certain set of functionality, i.e. a feature.

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

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

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

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

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

Message: The basic unit of RTSP communication, consisting of a structured sequence of octets matching the syntax defined in Section 16 and transmitted via a connection or a connectionless protocol.

Non-Aggregated Control: Control of a single media stream. Only possible in RTSP sessions with a single media.

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

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

Presentation description: A presentation description contains information about one or more media streams within a presentation, such as the set of encodings, network addresses and information about the content. Other IETF protocols such as SDP (RFC 2327 [24]) use the term "session" for a live presentation. The presentation description may take several different formats, including but not limited to the session description format SDP.

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

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

RTSP session: A stateful abstraction upon which the main control methods of RTSP operate. A RTSP session is a server entity; it is created, maintained and destroyed by the server. It is established by a RTSP server upon the completion of a successful SETUP request (when 200 OK response is sent) and is labelled by a session identifier at that time. The session exists until timed out by the server or explicitly removed by a TEARDOWN request. A RTSP session is also a stateful entity; a RTSP server maintains an explicit session state machine (see Appendix A) where most state transitions are triggered by client requests. The existence of a session implies the existence of state about the session's media streams and their respective transport mechanisms. A given session can have zero or more media streams associated with it. A RTSP server uses the session to aggregate control over multiple media streams.

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

1.5 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, RFC 2246 [27]) or within the protocol itself. All HTTP authentication mechanisms such as basic (RFC 2616 [26, Section 11]) and digest authentication (RFC 2069 [6]) are directly applicable.

Transport-independent: RTSP does not preclude the use of an unreliable datagram protocol (UDP) (RFC 768 [7]), a reliable datagram protocol (RDP, RFC 1151, not widely used [8]) or a reliable stream protocol such as TCP (RFC 793 [9]) as it implements application-level reliability. The use of a connectionless datagram protocol such as UDP or RDP requires additional definition that may be provided as extensions to the core RTSP specification.

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

Separation of stream control and conference initiation: Stream control is divorced from inviting a media server to a conference. In particular, SIP [10] or H.323 [28] may be used to invite a server to a conference.

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

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

Proxy and firewall friendly: The protocol should be readily handled by both application and transport-layer (SOCKS [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 reuses HTTP concepts, so that the existing infrastructure can be reused. This infrastructure includes PICS (Platform for Internet Content Selection [12, 13]) 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.

1.6 Extending RTSP

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

- A server may not be capable of seeking (absolute positioning) if it is to support live events only.
- Some servers may not support setting stream parameters and thus not support GET_PARAMETER and SET_PARAMETER.

A server SHOULD implement all header fields described in Section 13.

It is up to the creators of presentation descriptions not to ask the impossible of a server. This situation is similar in HTTP/1.1 [26], where the methods described in [H19.5] are not likely to be supported across all servers.

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

- Existing methods can be extended with new parameters, as long as these parameters can be safely ignored by the recipient. (This is equivalent to adding new parameters to an HTML tag.) If the client needs negative acknowledgement when a method extension is not supported, a tag corresponding to the extension may be added in the Require: field (see Section 13.32).
- New methods can be added. If the recipient of the message does not understand the request, it responds with error code 501 (Not Implemented) and the sender should not attempt to use this method again. A client may also use the OPTIONS method to inquire about methods supported by the server. The server SHOULD list the methods it supports using the Public response header.
- A new version of the protocol can be defined, allowing almost all aspects (except the position of the protocol version number) to change.

The basic capability discovery mechanism can be used to both discover support for a certain feature and to ensure that a feature is available when performing a request. For detailed explanation of this see chapter 10.

1.7 Overall Operation

Each presentation and media stream may be identified by a RTSP URL. The overall presentation and the properties of the media the presentation is made up of are defined by a presentation description file, the format of which is outside the scope of this specification. The presentation description file may be obtained by the client using HTTP or other means such as email and may not necessarily be stored on the media server.

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

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

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

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

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

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

1.8 RTSP States

RTSP controls a stream which may be sent via a separate protocol, independent of the control channel. For example, RTSP control may occur on a TCP connection while the data flows via UDP. Thus, data delivery continues even if no RTSP requests are received by the media server. Also, during its lifetime, a single media stream may be controlled by RTSP requests issued sequentially on different TCP connections. Therefore, the server needs to maintain “session state” to be able to correlate RTSP requests with a stream. The state transitions are described in Appendix A.

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

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

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

PAUSE: Temporarily halts a stream without freeing server resources.

REDIRECT: Indicates that the session should be moved to new server / location

PING: Prevents the identified session from being timed out.

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

RTSP methods that contribute to state use the Session header field (Section 13.37) to identify the RTSP session whose state is being manipulated. The server generates session identifiers in response to SETUP requests (Section 11.3).

1.9 Relationship with Other Protocols

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

RTSP assumes the existence of a presentation description format that can express both static and temporal properties of a presentation containing several media streams. Session Description Protocol (SDP) [24] is generally the format of choice; however, RTSP is not bound to it. For data delivery, most real-time media will use RTP as a transport protocol. While RTSP works well with RTP, it is not tied to RTP.

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 2616 [26]).

All the mechanisms specified in this document are described in both prose and an augmented Backus-Naur form (BNF) similar to that used in [H2.1]. It is described in detail in RFC 2234 [14], with the difference that this RTSP specification maintains the “#” notation for comma-separated lists from [H2.1].

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

3 Protocol Parameters

3.1 RTSP Version

HTTP Specification Section [H3.1] applies, with HTTP replaced by RTSP. This specification defines version 1.0 of RTSP.

3.2 RTSP URL

The “rtsp”, “rtsps” 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. The RTSP URL is case sensitive.

```
rtsp_URL  =  ( "rtsp:" / "rtspu:" / "rtsps:" )
             "//" host [ ":" port ] [ abs_path [ "?" query ]]
host      =  As defined by RFC 2732 [30]
abs_path  =  As defined by RFC 2396 [22]
port      =  *DIGIT
query     =  As defined by RFC 2396 [22]
```

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). The scheme rtsps identifies a reliable transport using secure transport, perhaps TLS [27]. The rtspu and rtsps is not defined in this specification and if for future extensions of the protocol.

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

The use of IP addresses in URLs SHOULD be avoided whenever possible (see RFC 1924 [16]). Note: Using qualified domain names in any URL is one requirement for making it possible for RFC 2326 implementations of RTSP to use IPv6. This specification is updated to allow for literal IPv6 addresses in RTSP URLs using the host specification in RFC 2732 [30].

A presentation or a stream is identified by a textual media identifier, using the character set and escape conventions [H3.2] of URLs (RFC 2396 [22]). URLs may refer to a stream or an aggregate of streams, i.e., a presentation. Accordingly, requests described in Section 11 can apply to either the whole presentation or an individual stream within the presentation. Note that some request methods can only be applied to streams, not presentations and vice versa.

For example, the RTSP URL:

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

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

Also, the RTSP URL:

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

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

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

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

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

3.3 Session Identifiers

Session identifiers are strings of any arbitrary length. A session identifier MUST be chosen randomly and MUST be at least eight characters long to make guessing it more difficult. (See Section 17.)

```
session-id   =   8*( ALPHA / DIGIT / safe )
```

3.4 SMPTE Relative Timestamps

A SMPTE relative timestamp expresses time relative to the start of the clip. Relative timestamps are expressed as SMPTE time codes for frame-level access accuracy. The time code has the format

hours:minutes:seconds:frames.subframes,

with the origin at the start of the clip. The default smpte format is“SMPTE 30 drop” format, with frame rate is 29.97 frames per second. Other SMPTE codes MAY be supported (such as ”SMPTE 25”) through the use of alternative use of ”smpte time”. For the “frames” field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute. If the frame value is zero, it may be omitted. Subframes are measured in one-hundredth of a frame.

```
smpte-range        =   smpte-type ”=” smpte-range-spec
smpte-range-spec   =   ( smpte-time ”-” [ smpte-time ] )
                   /   ( ”-” smpte-time )
smpte-type         =   ”smpte” / ”smpte-30-drop” / ”smpte-25”
                       ; other timecodes may be added
smpte-time         =   1*2DIGIT ”:” 1*2DIGIT ”:” 1*2DIGIT
                       [ ”:” 1*2DIGIT [ ”.” 1*2DIGIT ] ]
```

Examples:

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

3.5 Normal Play Time

Normal play time (NPT) indicates the stream absolute position relative to the beginning of the presentation, not to be confused with the Network Time Protocol (NTP). The timestamp consists of a decimal fraction. The part left of the decimal may be expressed in either seconds or hours, minutes, and seconds. The part right of the decimal point measures fractions of a second.

The beginning of a presentation corresponds to 0.0 seconds. Negative values are not defined. The special constant now is defined as the current instant of a live event. It MAY only be used for live events, and SHALL NOT be used for on-demand content.

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

```
npt-range       =  ["npt" "="] npt-range-spec
                   ; implementations SHOULD use npt= prefix, but SHOULD
                   ; be prepared to interoperate with RFC 2326
                   ; implementations which don't use it
npt-range-spec  =  ( npt-time "-" [ npt-time ] ) / ( "-" npt-time )
npt-time        =  "now" / npt-sec / npt-hhmmss
npt-sec         =  1*DIGIT [ "." *DIGIT ]
npt-hhmmss      =  npt-hh ":" npt-mm ":" npt-ss [ "." *DIGIT ]
npt-hh          =  1*DIGIT ; any positive number
npt-mm          =  1*2DIGIT ; 0-59
npt-ss          =  1*2DIGIT ; 0-59
```

Examples:

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

The syntax conforms to ISO 8601. The npt-sec notation is optimized for automatic generation, the ntp-hhmmss notation for consumption by human readers. The “now” constant allows clients to request to receive the live feed rather than the stored or time-delayed version. This is needed since neither absolute time nor zero time are appropriate for this case.

3.6 Absolute Time

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

```
utc-range       =  "clock" "=" utc-range-spec
utc-range-spec  =  ( utc-time "-" [ utc-time ] ) / ( "-" utc-time )
utc-time        =  utc-date "T" utc-time "Z"
utc-date        =  8DIGIT ; < YYYYMMDD >
utc-time        =  6DIGIT [ "." fraction ] ; < HHMMSS.fraction >
fraction        =  1*DIGIT
```

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

```
19961108T143720.25Z
```

3.7 Feature-tags

Feature-tags are unique identifiers used to designate new features in RTSP. These tags are used in in Require (Section 13.32), Proxy-Require (Section 13.27), Unsupported (Section 13.41), and Supported (Section 13.38) header fields.
Syntax:

```
feature-tag   =   token
```

The creator of a new RTSP feature-tag should either prefix the feature-tag with a reverse domain name (e.g., “com.foo.mynewfeature” is an apt name for a feature whose inventor can be reached at “foo.com”), or register the new feature-tag with the Internet Assigned Numbers Authority (IANA), see IANA Section 18.

4 RTSP Message

RTSP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279 [18]). Lines are terminated by CRLF, but receivers should be prepared to also interpret CR and LF by themselves as line terminators.

> Text-based protocols make it easier to add optional parameters in a self-describing manner. Since the number of parameters and the frequency of commands is low, processing efficiency is not a concern. Text-based protocols, if done carefully, also allow easy implementation of research prototypes in scripting languages such as Tcl, Visual Basic and Perl.
>
> The 10646 character set avoids tricky character set switching, but is invisible to the application as long as US-ASCII is being used. This is also the encoding used for RTCP. ISO 8859-1 translates directly into Unicode with a high-order octet of zero. ISO 8859-1 characters with the most-significant bit set are represented as 1100001x 10xxxxxx. (See RFC 2279 [18])

RTSP messages can be carried over any lower-layer transport protocol that is 8-bit clean. RTSP messages are vulnerable to bit errors and SHOULD NOT be subjected to them.

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

4.1 Message Types

See [H4.1].

4.2 Message Headers

See [H4.2].

4.3 Message Body

See [H4.3]

4.4 Message Length

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

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

2. If a Content-Length header field (section 13.14) is present, its value in bytes represents the length of the message-body. If this header field is not present, a value of zero is assumed.

Note that RTSP does not (at present) support the HTTP/1.1 "chunked" transfer coding(see [H3.6.1]) and requires the presence of the Content-Length header field.

> Given the moderate length of presentation descriptions returned, the server should always be able to determine its length, even if it is generated dynamically, making the chunked transfer encoding unnecessary.

5 General Header Fields

See [H4.5], except that Pragma, Trailer, Transfer-Encoding, Upgrade, and Warning headers are not defined. RTSP further defines the CSeq, and Timestamp:

```
general-header  =  Cache-Control   ; Section 13.9
                /  Connection      ; Section 13.10
                /  CSeq            ; Section 13.17
                /  Date            ; Section 13.18
                /  Timestamp       ; Section 13.39
                /  Via             ; Section 13.44
```

6 Request

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

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

6.1 Request Line

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

```
Method  =  "DESCRIBE"          ; Section 11.2
        /  "GET_PARAMETER"     ; Section 11.7
        /  "OPTIONS"           ; Section 11.1
        /  "PAUSE"             ; Section 11.5
        /  "PLAY"              ; Section 11.4
        /  "PING"              ; Section 11.10
        /  "REDIRECT"          ; Section 11.9
        /  "SETUP"             ; Section 11.3
        /  "SET_PARAMETER"     ; Section 11.8
        /  "TEARDOWN"          ; Section 11.6
        /  extension-method

extension-method  =  token
Request-URI       =  "*" / absolute_URI
RTSP-Version      =  "RTSP" "/" 1*DIGIT "." 1*DIGIT
```

6.2 Request Header Fields

```
request-header  =  Accept               ; Section 13.1
                /  Accept-Encoding      ; Section 13.2
                /  Accept-Language      ; Section 13.3
                /  Authorization        ; Section 13.6
                /  Bandwidth            ; Section 13.7
                /  Blocksize            ; Section 13.8
                /  From                 ; Section 13.20
                /  If-Modified-Since    ; Section 13.23
                /  Proxy-Require        ; Section 13.27
                /  Range                ; Section 13.29
                /  Referer              ; Section 13.30
                /  Require              ; Section 13.32
                /  Scale                ; Section 13.34
                /  Session              ; Section 13.37
                /  Speed                ; Section 13.35
                /  Supported            ; Section 13.38
                /  Transport            ; Section 13.40
                /  User-Agent           ; Section 13.42
```

Note that in contrast to HTTP/1.1 [26], RTSP requests always contain the absolute URL (that is, including the scheme, host and port) rather than just the absolute path.

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

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

One example would be as follows:

```
OPTIONS * RTSP/1.0
```

An OPTIONS in this form will determine the capabilities of the server or the proxy that first receives the request. If one needs to address the server explicitly, then one should use an absolute URL with the server's address.

```
OPTIONS rtsp://example.com RTSP/1.0
```

7 Response

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

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

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

7.1 Status-Line

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

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

7.1.1 Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in Section 12. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

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

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3rr: Redirection - Further action must be taken in order to complete the request

- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

The individual values of the numeric status codes defined for RTSP/1.0, and an example set of corresponding Reason-Phrase's, are presented below. The reason phrases listed here are only recommended – they may be replaced by local equivalents without affecting the protocol. Note that RTSP adopts most HTTP/1.1 [26] status codes and adds RTSP-specific status codes starting at x50 to avoid conflicts with newly defined HTTP status codes.

```
Status-Code  =  "100"  ; Continue
             /  "200"  ; OK
             /  "201"  ; Created
             /  "250"  ; Low on Storage Space
             /  "300"  ; Multiple Choices
             /  "301"  ; Moved Permanently
             /  "302"  ; Moved Temporarily
             /  "303"  ; See Other
             /  "304"  ; Not Modified
             /  "305"  ; Use Proxy
             /  "350"  ; Going Away
             /  "351"  ; Load Balancing
             /  "400"  ; Bad Request
             /  "401"  ; Unauthorized
             /  "402"  ; Payment Required
             /  "403"  ; Forbidden
             /  "404"  ; Not Found
             /  "405"  ; Method Not Allowed
             /  "406"  ; Not Acceptable
             /  "407"  ; Proxy Authentication Required
             /  "408"  ; Request Time-out
             /  "410"  ; Gone
             /  "411"  ; Length Required
             /  "412"  ; Precondition Failed
             /  "413"  ; Request Entity Too Large
             /  "414"  ; Request-URI Too Large
             /  "415"  ; Unsupported Media Type
             /  "451"  ; Parameter Not Understood
             /  "452"  ; reserved
             /  "453"  ; Not Enough Bandwidth
             /  "454"  ; Session Not Found
             /  "455"  ; Method Not Valid in This State
             /  "456"  ; Header Field Not Valid for Resource
             /  "457"  ; Invalid Range
             /  "458"  ; Parameter Is Read-Only
             /  "459"  ; Aggregate operation not allowed
             /  "460"  ; Only aggregate operation allowed
             /  "461"  ; Unsupported transport
             /  "462"  ; Destination unreachable
```

```
                  / "500"              ; Internal Server Error
                  / "501"              ; Not Implemented
                  / "502"              ; Bad Gateway
                  / "503"              ; Service Unavailable
                  / "504"              ; Gateway Time-out
                  / "505"              ; RTSP Version not supported
                  / "551"              ; Option not supported
                  / extension-code

extension-code    = 3DIGIT

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

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

7.1.2 Response Header Fields

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

```
response-header  = Accept-Ranges        ; Section 13.4
                 / Location             ; Section 13.25
                 / Proxy-Authenticate   ; Section 13.26
                 / Public               ; Section 13.28
                 / Range                ; Section 13.29
                 / Retry-After          ; Section 13.31
                 / RTP-Info             ; Section 13.33
                 / Scale                ; Section 13.34
                 / Session              ; Section 13.37
                 / Server               ; Section 13.36
                 / Speed                ; Section 13.35
                 / Transport            ; Section 13.40
                 / Unsupported          ; Section 13.41
                 / Vary                 ; Section 13.43
                 / WWW-Authenticate     ; Section 13.45
```

Response-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of response-header fields

Code	Reason	Method
100	Continue	all
200	OK	all
201	Created	RECORD
250	Low on Storage Space	RECORD
300	Multiple Choices	all
301	Moved Permanently	all
302	Found	all
303	See Other	all
305	Use Proxy	all
350	Going Away	all
351	Load Balancing	all
400	Bad Request	all
401	Unauthorized	all
402	Payment Required	all
403	Forbidden	all
404	Not Found	all
405	Method Not Allowed	all
406	Not Acceptable	all
407	Proxy Authentication Required	all
408	Request Timeout	all
410	Gone	all
411	Length Required	all
412	Precondition Failed	DESCRIBE, SETUP
413	Request Entity Too Large	all
414	Request-URI Too Long	all
415	Unsupported Media Type	all
451	Parameter Not Understood	SET_PARAMETER
452	reserved	n/a
453	Not Enough Bandwidth	SETUP
454	Session Not Found	all
455	Method Not Valid In This State	all
456	Header Field Not Valid	all
457	Invalid Range	PLAY, PAUSE
458	Parameter Is Read-Only	SET_PARAMETER
459	Aggregate Operation Not Allowed	all
460	Only Aggregate Operation Allowed	all
461	Unsupported Transport	all
462	Destination Unreachable	all
500	Internal Server Error	all
501	Not Implemented	all
502	Bad Gateway	all
503	Service Unavailable	all
504	Gateway Timeout	all
505	RTSP Version Not Supported	all
551	Option not support	all

Table 1: Status codes and their usage with RTSP methods

if all parties in the communication recognize them to be response-header fields. Unrecognized header fields are treated as entity-header fields.

8 Entity

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

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

8.1 Entity Header Fields

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

```
entity-header     =  Allow               ; Section 13.5
                  /  Content-Base        ; Section 13.11
                  /  Content-Encoding    ; Section 13.12
                  /  Content-Language    ; Section 13.13
                  /  Content-Length      ; Section 13.14
                  /  Content-Location    ; Section 13.15
                  /  Content-Type        ; Section 13.16
                  /  Expires             ; Section 13.19
                  /  Last-Modified       ; Section 13.24
                  /  extension-header
extension-header  =  message-header
```

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

8.2 Entity Body

See [H7.2] with the addition that a RTSP message with an entity body MUST include a Content-Type header.

9 Connections

RTSP requests can be transmitted in several different ways:

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

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

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

9.1 Pipelining

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

9.2 Reliability and Acknowledgements

Requests are acknowledged by the receiver unless they are sent to a multicast group. If there is no acknowledgement, the sender may resend the same message after a timeout of one round-trip time (RTT). The round-trip time is estimated as in TCP (RFC 1123) [15], with an initial round-trip value of 500 ms. An implementation MAY cache the last RTT measurement as the initial value for future connections.

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

> If both the underlying reliable transport such as TCP and the RTSP application retransmit requests, it is possible that each packet loss results in two retransmissions. The receiver cannot typically take advantage of the application-layer retransmission since the transport stack will not deliver the application-layer retransmission before the first attempt has reached the receiver. If the packet loss is caused by congestion, multiple retransmissions at different layers will exacerbate the congestion.
>
> If RTSP is used over a small-RTT LAN, standard procedures for optimizing initial TCP round trip estimates, such as those used in T/TCP (RFC 1644) [19], can be beneficial.

The Timestamp header (Section 13.39) is used to avoid the retransmission ambiguity problem [20, p. 301] and obviates the need for Karn’s algorithm.

Each request carries a sequence number in the CSeq header (Section 13.17), which MUST be incremented by one for each distinct request transmitted. If a request is repeated because of lack of acknowledgement, the request MUST carry the original sequence number (i.e., the sequence number is *not* incremented).

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

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

9.3 The usage of connections

TCP can be used for both persistent connections and for one message exchange per connection, as presented above. This section gives further rules and recommendations on how to handle these connections so maximum interoperability and flexibility can be achieved.

A server SHALL handle both persistent connections and one request/response transaction per connection. A persistent connection MAY be used for all transactions between the server and client, including messages to multiple RTSP sessions. However the persistent connection MAY also be closed after a few message exchanges, e.g. the initial setup and play command in a session. Later when the client wishes to send a new request, e.g. pause, to the session a new connection is opened. This connection may either be for a single message exchange or can be kept open for several messages, i.e. persistent.

A major motivation for allowing non-persistent connections are that they ensure fault tolerance. A server and client supporting non-persistent connection can survive a loss of a TCP connection, e.g. due to a NAT timeout. When the it is discovered that the TCP connection has been lost one sets up a new one.

The client MAY close the connection at any time when no outstanding request/response transactions exist. The server SHOULD NOT close the connection unless at least one RTSP session timeout period has passed without data traffic. A server MUST NOT initiate a close of a connection directly after responding to a TEARDOWN request for the whole session. A server MUST NOT close the connection as a result of responding to a request with an error code. Doing this would prevent or result in extra overhead for the client when testing advanced or special types of requests.

The client SHOULD NOT have more than one connection to the server at any given point. If a client or proxy handles multiple RTSP sessions on the same server, it is RECOMMENDED to use only a single connection.

Older services which was implemented according to RFC 2326 sometimes requires the client to use persistent connection. The client closing the connection may result in that the server removes the session. To achieve interoperability with old servers any client is strongly RECOMMENDED to use persistent connections.

A Client is also strongly RECOMMENDED to use persistent connections as it allows the server to send request to the client. In cases where no connection exist between the server and the client, this may cause the server to be forced to drop the RTSP session without notifying the client why,due to the lack of signalling channel. An example of such a case is when the server desires to send a REDIRECT request for a RTSP session to the client.

If a service requires the use of persistent connection an feature-tag is specified for usage in the Require and Proxy-Require headers.

con.persistent

A server implemented according to this specification MUST respond that it supports the "play.basic" feature-tag above. A client MAY send a request including the Supported header in a request to determine support of non-persistent connections. A server supporting non-persistent connections will return the "play.basic" feature-tag in its response. If the client receives the feature-tag in the response, it can be certain that the server handles non-persistent connections.

9.4 Use of IPv6

This specification has been updated so that it supports IPv6. However this support was not present in RFC 2326 therefore some interoperability issues exist. A RFC 2326 implementation can support IPv6 as long as no explicit IPv6 addresses are used within RTSP messages. This require that any RTSP URL pointing at a IPv6 host must use fully qualified domain name and not a IPv6 address. Further the Transport header must not use the parameters source and destination.

Implementations according to this specification MUST understand IPv6 addresses in URLs, and headers. By this requirement the feature-tag "play.basic" can be used to determine that a server or client is capable of handling IPv6 within RTSP.

10 Capability Handling

This chapter describes the capability handling mechanism available in RTSP which allows RTSP to be extended. Extensions too this version of the protocol are basically done in two ways. First, new headers can be added. Secondly, new methods can be added. The capability handling mechanism is designed to handle these two cases.

When a method is added the involved parties can use the OPTIONS method to discover if it is supported. This is done by issuing a OPTIONS request to the other party. Depending on the URL it will either apply in regards to a certain media resource, the whole server in general, or simply the next hop. The OPTIONS response will contain a Public which declares all methods supported for the indicated resource.

It is not necessary to use OPTIONS to discover support of a method, it is possible to simple try it. If the receiver of the request does not support the method it will respond with an error code indicating the the method are either not implemented (501) or does not apply for the resource (405). The choice between the two discovery methods depends on the requirements of the service.

To handle functionality additions that are not new methods feature-tags are defined. Each feature-tag represents a certain block of functionality. The amount of functionality that a feature-tag represents can vary significant. A simple feature-tag can simple represent the functionality a single header gives. Another feature-tag is "play.basic" which represents the minimal playback implementation according to the updated specification.

The feature-tags are then used to determine if the client, server or proxy supports the functionality that is necessary to achieve the desired service. To determine support of a feature-tag several different headers can be used, each explained below:

Supported: The supported header are used to determine the complete set of functionality that both client and server has. The intended usage is to determine before one needs to use a functionality that it is supported. If can be used in any method however OPTIONS is the most suitable as one at the same time determines all methods that are implemented. When sending a request the requestor declares all its capabilities by including all supported feature-tags. The results in that the receiver learns the requestors feature support. The receiver then includes its set of features in the response.

Require: The Require header can be included in any request where the end point, i.e. the client or server, is required to understand the feature to correctly perform the request. This can for example be a SETUP request where the server must understand a certain parameter to be able to set up the media delivery correctly. Ignoring this parameter would not have the desired effect and is not acceptable. Therefore the end-point receiving a request containing a Require must negatively acknowledge any feature that it does not understand and not perform the request. The response in cases where features are not understood are 551 (Option Not Supported). Also the features that are not understood are given in the Unsupported header in the response.

Proxy-Require: This method has the same purpose and workings as Require except that it only applies to proxies and not the end point. Features that needs to be supported by both proxies and end-point needs to be included in both the Require and Proxy-Require header.

Unsupported: This header is used in 551 error response to tell which feature(s) that was not supported. Such a response is only the result of the usage of the Require and/or Proxy-Require header where one or more feature where not supported. This information allows the requestor to make the best of situations as it knows which features that was not supported.

11 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` as defined by the ABNF. Methods are summarized in Table 2.

method	direction	object	Server req.	Client req.
DESCRIBE	$C \rightarrow S$	P,S	recommended	recommended
GET_PARAMETER	$C \rightarrow S, S \rightarrow C$	P,S	optional	optional
OPTIONS	$C \rightarrow S, S \rightarrow C$	P,S	R=Req, Sd=Opt	Sd=Req, R=Opt
PAUSE	$C \rightarrow S$	P,S	recommended	recommended
PING	$C \rightarrow S, S \rightarrow C$	P,S	recommended	optional
PLAY	$C \rightarrow S$	P,S	required	required
REDIRECT	$S \rightarrow C$	P,S	optional	optional
SETUP	$C \rightarrow S$	S	required	required
SET_PARAMETER	$C \rightarrow S, S \rightarrow C$	P,S	optional	optional
TEARDOWN	$C \rightarrow S$	P,S	required	required

Table 2: Overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on. Legend: R=Responde to, Sd=Send, Opt: Optional, Req: Required, Rec: Recommended

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

11.1 OPTIONS

The behavior is equivalent to that described in [H9.2]. An OPTIONS request may be issued at any time, e.g., if the client is about to try a nonstandard request. It does not influence the session state. The Public header MUST be included in responses to indicate which methods that are supported by the server. To specify which methods that are possible to use for the specified resource, the Allow MAY be used. By including in the OPTIONS request a Supported header, the requester can determine which features the other part supports.

The request URI determines which scope the OPTIONS request has. By giving the URI of a certain media the capabilities regarding this media will be responded. By using the "*" URI the request regards the next hop only, while having a URL with only the host address regards the server without any media relevance.

Example:

```
C->S:  OPTIONS * RTSP/1.0
       CSeq: 1
```

```
         User-Agent: PhonyClient/1.2
         Require:
         Proxy-Require: gzipped-messages
         Supported: play-basic

  S->C:  RTSP/1.0 200 OK
         CSeq: 1
         Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
         Supported: play-basic, implicit-play, gzipped-messages
         Server: PhonyServer/1.0
```

Note that some of the feature-tags in Require and Proxy-Require are necessarily fictional features (one would hope that we would not purposefully overlook a truly useful feature just so that we could have a strong example in this section).

11.2 DESCRIBE

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

Example:

```
  C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
        CSeq: 312
        User-Agent: PhonyClient 1.2
        Accept: application/sdp, application/rtsl, application/mheg

  S->C: RTSP/1.0 200 OK
        CSeq: 312
        Date: 23 Jan 1997 15:35:06 GMT
        Server: PhonyServer 1.0
        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=application 32416 UDP WB
a=orient:portrait
```

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

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

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

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

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

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

11.3 SETUP

The SETUP request for a URI specifies the transport mechanism to be used for the streamed media. A client can issue a SETUP request for a stream that is already set up or playing in the session to change transport parameters, which a server MAY allow. If it does not allow this, it MUST respond with error 455 (Method Not Valid In This State).

A server MAY allow a client to do SETUP while in playing state to add additional media streams. If not supported, the server SHALL respond with error 455 (Method Not Allowed In This State). If supported, the added media SHALL then start to play in sync with the already playing media. To be able to sync the media with the already playing streams the SETUP response MUST include a RTP-Info header with the timestamp value, and a Range header with the corresponding normal play time. To indicate support for this optional feature the feature-tag: ”setup.playing” is defined.

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

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

S->C: RTSP/1.0 200 OK
      CSeq: 302
      Date: 23 Jan 1997 15:35:06 GMT
```

```
Server: PhonyServer 1.0
Session: 47112344
Transport: RTP/AVP;unicast;
  client_port=4588-4589;server_port=6256-6257
```

For the benefit of any intervening firewalls, a client must indicate the transport parameters even if it has no influence over these parameters, for example, where the server advertises a fixed multicast address.

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

The server generates a session identifier in response to a SETUP request. If a SETUP request to a server includes a session identifier, the server MUST bundle this setup request into the existing session (aggregated session) or return error 459 (Aggregate Operation Not Allowed) (see Section 12.4.11). An Aggregate control URI MUST be used to control an aggregated session. This URI MUST be different from the stream control URIs of the individual media streams included in the aggregate. The Aggregate control URI SHOULD be specified by the session description since there is no general rule for deriving it from the various stream control URIs in the session. If an Aggregate control URI is not specified in the session description, a client MUST create an URI for aggregate control of the session. This URI MUST contain the servers host address and MUST contain the port, if applicable. Once an URI is used to refer to an aggregation for a given session, that URI MUST be used to refer to that aggregation for the duration of the session. If the contents of the aggregation change, then a different aggregate control URI SHOULD be used.

> While the session ID has enough information for aggregate control of a session, the Aggregate control URI is still important for some methods such as SET_PARAMETER where the control URI enables the resource in question to be easily identified. The Aggregate control URI is also useful for proxies, enabling them to route the request to the appropriate server, and for logging, where it is useful to note the actual resource that a request was operating on. Finally, presence of the Aggregate control URI allows for backwards compatibility with RFC 2326 [21].

A session will exist until it is either removed by a TEARDOWN request or is timed-out by the server. The server MAY remove a session that has not demonstrated liveness signs from the client within a certain timeout period. The default timeout value is 60 seconds; the server MAY set this to a different value and indicate so in the timeout field of the Session header in the SETUP response. For further discussion see chapter 13.37. Signs of liveness for a RTSP session are:

- Any RTSP request from a client which includes a Session header with that session's ID.
- If RTP is used as a transport for the underlying media streams, an RTCP sender or receiver report from the client for any of the media streams in that RTSP session.

If a SETUP request on a session fails for any reason, the session state, as well as transport and other parameters for associated streams SHALL remain unchanged from their values as if the SETUP request had never been received by the server.

11.4 PLAY

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

In an aggregated session the PLAY request MUST contain an aggregated control URL. A server SHALL responde with error 460 (Only Aggregate Operation Allowed) if the client PLAY request URI is for one of the media. The media in an aggregate SHALL be played in sync. If a client want individual control of the media it must use separate RTSP sessions for each media.

The PLAY request positions the normal play time to the beginning of the range specified by the Range header and delivers stream data until the end of the range is reached. To allow for precise composition multiple ranges MAY be specified. The range values are valid if all given ranges are part of any media. If a given range value points outside of the media, the response SHALL be the 457 (Invalid Range) error code.

The below example will first play seconds 10 through 15, then, immediately following, seconds 20 to 25, and finally seconds 30 through the end.

```
C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
      CSeq: 835
      Session: 12345678
      Range: npt=10-15, npt=20-25, npt=30-
```

See the description of the PAUSE request for further examples.

A PLAY request without a Range header is legal. It starts playing a stream from the beginning unless the stream has been paused. If a stream has been paused via PAUSE, stream delivery resumes at the pause point.

The Range header MUST NOT contain a time parameter. The usage of time has been deprecated.

Server MUST include a "Range" header in any PLAY response. The response MUST use the same format as the request's range header contained. If no Range header was in the request, the NPT time format SHOULD be used unless the client showed support for other formats. For a session with live media streams the Range header MUST also be given, containing a valid time indication. It is RECOMMENDED that either "npt=now-" or a absolute time value (clock) for the corresponding time is given, i.e. "clock=20030213T143205Z-". The UTC clock format SHOULD only be used if client has shown support for it.

For a on-demand stream, the server MUST reply with the actual range that will be played back. This may differ from the requested range if alignment of the requested range to valid frame boundaries is required for the media source. If no range is specified in the request, the start position SHALL still be returned in the reply. The unit of the range in the reply is the same as that in the request. If the medias part of an aggregate has different lengths the PLAY request and any Range SHALL be performed as long it is valid for the longest media. Media will be sent whenever it is available for the given play-out point.

After playing the desired range, the presentation is NOT automatically paused, media deliver simply stops. A PAUSE request MUST be issued before another PLAY request can issued. Note: This is one change resulting in a non-operability with RFC 2326 implementations. A client not issuing a PAUSE request before a new PLAY will be stuck in PLAYING state. A client desiring to play the media from the beginning MUST send a PLAY request with a Range header pointing at the beginning, e.g. npt=0-.

The following example plays the whole presentation starting at SMPTE time code 0:10:20 until the end of the clip. The playback is to start at 15:36 on 23 Jan 1997. Note: The RTP-Info headers has been broken into several lines to fit the page.

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

S->C: RTSP/1.0 200 OK
      CSeq: 833
      Date: 23 Jan 1997 15:35:06 GMT
      Server: PhonyServer 1.0
      Range: smpte=0:10:22-;time=19970123T153600Z
      RTP-Info:url=rtsp://example.com/twister.en;
         seq=14783;rtptime=2345962545
```

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

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

S->C: RTSP/1.0 200 OK
      CSeq: 835
      Date: 23 Jan 1997 15:35:06 GMT
      Server:PhonyServer 1.0
      Range: clock=19961108T142300Z-19961108T143520Z
      RTP-Info:url=rtsp://example.com/meeting.en;
         seq=53745;rtptime=484589019
```

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

All range specifiers in this specification allow for ranges with unspecified begin times (e.g. “npt=-30”). When used in a PLAY request, the server treats this as a request to start/resume playback from the current pause point, ending at the end time specified in the Range header. If the pause point is located later than the given end value, a 457 (Invalid Range) response SHALL be given.

The queued play functionality described in RFC 2326 [21] is removed and multiple ranges can be used to achieve a similar performance. If a server receives a PLAY request while in the PLAY state, the server SHALL responde using the error code 455 (Method Not Valid In This State). This will signal the client that queued play are not supported.

The use of PLAY for keep-alive signaling, i.e. PLAY request without a range header, has also been depreciated. Instead a client can use, PING, SET_PARAMETER or OPTIONS for keep alive. A server receiving a PLAY keep alive SHALL respond with the 455 error code.

When playing live media, indicated by the Accept-Ranges header the session are in a live state. This live state will put some restrictions on the action available for a client. A PLAY request without a Range header will start media deliver at the current point in the live presentation, i.e. now. Any seeking in the media will be impossible. The only allowed usage of the Range header is npt=now-, and certain clock units. The usage of npt=now- is unnecessary as it has the exact same meaning as a request without Range header. The clock format can be used to specify start and stop times for media delivery in a live session.

11.5 PAUSE

The PAUSE request causes the stream delivery to be interrupted (halted) temporarily. A PAUSE request MUST be done with the aggregated control URI for aggregated sessions, resulting in all media being halted, or the media URI for non-aggregated sessions. Any attempt to do muting of a single media with an PAUSE request in an aggregated session SHALL be responded with error 460 (Only Aggregate Operation Allowed). After resuming playback, synchronization of the tracks MUST be maintained. Any server resources are kept, though servers MAY close the session and free resources after being paused for the duration specified with the timeout parameter of the Session header in the SETUP message.

Example:

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

S->C: RTSP/1.0 200 OK
      CSeq: 834
      Date: 23 Jan 1997 15:35:06 GMT
      Range: npt=45.76
```

The PAUSE request may contain a Range header specifying when the stream or presentation is to be halted. We refer to this point as the “pause point”. The time parameter in the Range MUST NOT be used. The header MUST contain a single value, expressed as the beginning value an open range. For example, the following clip will be played from 10 seconds through 21 seconds of the clip’s normal play time, under the assumption that the PAUSE request reaches the server within 11 seconds of the PLAY request. Note that some lines has been broken in an non-correct way to fit the page:

```
C->S: PLAY rtsp://example.com/fizzle/foo RTSP/1.0
      CSeq: 834
      Session: 12345678
      Range: npt=10-30

S->C: RTSP/1.0 200 OK
      CSeq: 834
      Date: 23 Jan 1997 15:35:06 GMT
      Server: PhonyServer 1.0
      Range: npt=10-30
```

```
        RTP-Info:url=rtsp://example.com/fizzle/audiotrack;
                seq=5712;rtptime=934207921,
                url=rtsp://example.com/fizzle/videotrack;
                seq=57654;rtptime=2792482193
        Session: 12345678

  C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
        CSeq: 835
        Session: 12345678
        Range: npt=21-

  S->C: RTSP/1.0 200 OK
        CSeq: 835
        Date: 23 Jan 1997 15:35:09 GMT
        Server: PhonyServer 1.0
        Range: npt=21-
        Session: 12345678
```

The pause request becomes effective the first time the server is encountering the time point specified in any of the multiple ranges. If the Range header specifies a time outside any range from the PLAY request, the error 457 (Invalid Range) SHALL be returned. If a media unit (such as an audio or video frame) starts presentation at exactly the pause point, it is not played. If the Range header is missing, stream delivery is interrupted immediately on receipt of the message and the pause point is set to the current normal play time. However, the pause point in the media stream MUST be maintained. A subsequent PLAY request without Range header resumes from the pause point and play until media end.

The actual pause point after any PAUSE request SHALL be returned to the client by adding a Range header with what remains unplayed of the PLAY request's ranges, i.e. including all the remaining ranges part of multiple range specification. If one desires to resume playing a ranged request, one simple included the Range header from the PAUSE response.

For example, if the server have a play request for ranges 10 to 15 and 20 to 29 pending and then receives a pause request for NPT 21, it would start playing the second range and stop at NPT 21. If the pause request is for NPT 12 and the server is playing at NPT 13 serving the first play request, the server stops immediately. If the pause request is for NPT 16, the server returns a 457 error message. To prevent that the second range is played and the server stops after completing the first range, a PAUSE request for 20 must be issued.

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

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

If a client issues a PAUSE request and the server acknowledges and enters the ready state, the proper server response, if the player issues another PAUSE, is 200 OK. The 200 OK response MUST include the

Range header with the current pause point, even if the PAUSE request is asking for some other pause point. See examples below:

Examples:

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

S->C: RTSP/1.0 200 OK
      CSeq: 834
      Session: 12345678
      Date: 23 Jan 1997 15:35:06 GMT
      Range: npt=45.76-

C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
      CSeq: 835
      Session: 12345678
      Range:  86-

S->C: RTSP/1.0 200 OK
      CSeq: 835
      Session: 12345678
      Date: 23 Jan 1997 15:35:07 GMT
      Range: npt=45.76-
```

11.6 TEARDOWN

The TEARDOWN request stops the stream delivery for the given URI, freeing the resources associated with it. If the URI is the aggregated control URI for this presentation, any RTSP session identifier associated with the session is no longer valid. The use of ”*” as URI in TEARDOWN will also result in that the session is removed independent of the number of medias that was part of it. If the URI in the request was for a media within an aggregated session that media is removed from the aggregate. However the session and any other media stream yet not torn down remains, and any valid request, e.g. PLAY or SETUP, can be issued. As an optional feature a server MAY keep the session in case the last remaining media is torn down with a TEARDOWN request with an URI equal to the media URI. To Indicate what has been performed, a server that after any TEARDOWN request, still has a valid session MUST in the response return a session header.

A server MAY choose to allow TEARDOWN of individual media while in PLAY state. When this is not allowed the response SHALL be 455 (Method Not Valid In This State). If a server implements TEARDOWN and SETUP in PLAY state it MUST signal this using the ”setup.playing” feature-tag.

Example:

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

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

11.7 GET_PARAMETER

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

Example:

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

      packets_received
      jitter

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

      packets_received: 10
      jitter: 0.3838
```

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

11.8 SET_PARAMETER

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

A request is RECOMMENDED to only contain a single parameter to allow the client to determine why a particular request failed. If the request contains several parameters, the server MUST only act on the request if all of the parameters can be set successfully. A server MUST allow a parameter to be set repeatedly to the same value, but it MAY disallow changing parameter values. If the receiver of the request does not understand or can locate a parameter error 451 (Parameter Not Understood) SHALL be used. In the case a parameter is not allowed to change the error code 458 (Parameter Is Read-Only). The response body SHOULD contain only the parameters that has errors. Otherwise no body SHALL be returned.

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

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

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

Example:

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

      barparam: barstuff

S->C: RTSP/1.0 451 Parameter Not Understood
      CSeq: 421
      Content-length: 10
      Content-type: text/parameters

      barparam
```

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

11.9 REDIRECT

A redirect request informs the client that it MUST connect to another server location. The REDIRECT request MAY contain the header Location, which indicates that the client should issue requests for that URL. If the Location URL only contains a host address the client shall connect to the given host, while using the path from the URL on the current server.

If a REDIRECT request contains a Session header, it is end-to-end and applies only to the given session. If there are proxies in the request chain, they SHOULD NOT disconnect the control channel unless there are no remaining sessions.

If a REDIRECT request does not contain a Session header, it is next-hop and applies to the control connection. The Location header SHOULD only contain a host address. If there are proxies in the request chain, they SHOULD do all of the following: (1) respond to the REDIRECT request, (2) disconnect the control channel from the requestor, (3) reconnect to the given host address, and (4) pass the request to each applicable client (typically those clients with an active session or unanswered request from the requestor). Note that the proxy is responsible for accepting the REDIRECT response from its clients and these responses MUST NOT be passed on to either the requesting or the destination server.

The redirect request MAY contain the header Range, which indicates when the redirection takes effect. If the Range contains a "time=" value that is the wall clock time that the redirection MUST at the latest take place. When the "time=" parameter is present the range value MUST be ignored. However the range entered MUST be syntactical correct and SHALL point at the beginning of any on-demand content. If no time parameter is part of the Range header then redirection SHALL take place when the media playout from the server reaches the given time. The range value MUST be a single value in the open ended form, e.g. npt=59-.

If the client wants to continue to send or receive media for this resource, the client MUST issue a TEARDOWN request for the current session. A new session must be established with the designated host. A client SHOULD issue a new DESCRIBE request with the URL given in the Location header, unless the URL only contains a host address. In the cases the Location only contains a host address the client MAY assume that the media on the server it is redirected to is identical. Identical media means that all media configuration information from the old session still is valid except for the host address. In the case of absolute URLs in the location header the media redirected to can be either identical, slightly different or totally different. This is the reason why a new DESCRIBE request SHOULD be issued.

This example request redirects traffic for this session to the new server at the given absolute time:

```
S->C: REDIRECT rtsp://example.com/fizzle/foo RTSP/1.0
      CSeq: 732
      Location: rtsp://bigserver.com:8001
      Range: npt=0- ;time=19960213T143205Z
      Session: uZ3ci0K+Ld-M
```

11.10 PING

This method is a bi-directional mechanism for server or client liveness checking. It has no side effects. The issuer of the request MUST include a session header with the session ID of the session that is being checked for liveness.

Prior to using this method, an OPTIONS method is RECOMMENDED to be issued in the direction which the PING method would be used. This method MUST NOT be used if support is not indicated by the Public header. Note: That an 501 (Not Implemented) response means that the keep-alive timer has not been updated.

When a proxy is in use, PING with a * indicates a single-hop liveness check, whereas PING with a URL including an host address indicates an end-to-end liveness check.

Example:

```
C->S: PING * RTSP/1.0
      CSeq: 123
      Session:12345678

S->C: RTSP/1.0 200 OK
      CSeq: 123
      Session:12345678
```

11.11 Embedded (Interleaved) Binary Data

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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| "$" = 24      | Channel ID    | Length in bytes               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
: Length number of bytes of binary data                         :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The channel identifier is defined in the Transport header with the interleaved parameter(Section 13.40).

When the transport choice is RTP, RTCP messages are also interleaved by the server over the TCP connection. The usage of RTCP messages is indicated by including a range containing a second channel in the interleaved parameter of the Transport header, see section 13.40. If RTCP is used, packets SHALL be sent on the first available channel higher than the RTP channel. The channels are bi-directional and therefore RTCP traffic are sent on the second channel in both directions.

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

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

S->C: RTSP/1.0 200 OK
      CSeq: 2
      Date: 05 Jun 1997 18:57:18 GMT
      Transport: RTP/AVP/TCP;unicast;interleaved=5-6
      Session: 12345678

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

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

```
S->C: $\005{2 byte length}{"length" bytes data, w/RTP header}
S->C: $\005{2 byte length}{"length" bytes data, w/RTP header}
S->C: $\006{2 byte length}{"length" bytes  RTCP packet}
```

12 Status Code Definitions

Where applicable, HTTP status [H10] codes are reused. Status codes that have the same meaning are not repeated here. See Table 1 for a listing of which status codes may be returned by which requests. All error messages, 4xx and 5xx MAY return a body containing further information about the error.

12.1 Success 1xx

12.1.1 100 Continue

See, [H10.1.1].

12.2 Success 2xx

12.2.1 250 Low on Storage Space

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

12.3 Redirection 3xx

The notation "3rr" indicates response codes from 300 to 399 inclusive which are meant for redirection. The response code 304 is excluded from this set, as it is not used for redirection.

See [H10.3] for definition of status code 300 to 305. However comments are given for some to how they apply to RTSP.

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

12.3.1 300 Multiple Choices

[TBW]

12.3.2 301 Moved Permanently

The request resource are moved permanently and resides now at the URI given by the location header. The user client SHOULD redirect automatically to the given URI. This response MUST NOT contain a message-body.

12.3.3 302 Found

The requested resource reside temporarily at the URI given by the Location header. The Location header MUST be included in the response. Is intended to be used for many types of temporary redirects, e.g. load balancing. It is RECOMMENDED that one set the reason phrase to something more meaningful than "Found" in these cases. The user client SHOULD redirect automatically to the given URI. This response MUST NOT contain a message-body.

12.3.4 303 See Other

This status code SHALL NOT be used in RTSP. However as it was allowed to use in RFC 2326 it is possible that such response may be received.

12.3.5 304 Not Modified

If the client has performed a conditional DESCRIBE or SETUP (see 12.23) and the requested resource has not been modified, the server SHOULD send a 304 response. This response MUST NOT contain a message-body.

The response MUST include the following header fields:

- Date
- ETag and/or Content-Location, if the header would have been sent in a 200 response to the same request.
- Expires, Cache-Control, and/or Vary, if the field-value might differ from that sent in any previous response for the same variant.

This response is independent for the DESCRIBE and SETUP requests. That is, a 304 response to DESCRIBE does NOT imply that the resource content is unchanged and a 304 response to SETUP does NOT imply that the resource description is unchanged. The ETag and If-Match headers may be used to link the DESCRIBE and SETUP in this manner.

12.3.6 305 Use Proxy

See [H10.3.6].

12.4 Client Error 4xx

12.4.1 400 Bad Request

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications [H10.4.1]. If the request does not have a CSeq header, the server MUST NOT include a CSeq in the response.

12.4.2 405 Method Not Allowed

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

12.4.3 451 Parameter Not Understood

The recipient of the request does not support one or more parameters contained in the request.When returning this error message the sender SHOULD return a entity body containing the offending parameter(s).

12.4.4 452 reserved

This error code was removed from RFC 2326 [21] and is obsolete.

12.4.5 453 Not Enough Bandwidth

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

12.4.6 454 Session Not Found

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

12.4.7 455 Method Not Valid in This State

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

12.4.8 456 Header Field Not Valid for Resource

The server could not act on a required request header. For example, if PLAY contains the Range header field but the stream does not allow seeking. This error message may also be used for specifying when the time format in Range is impossible for the resource. In that case the Accept-Ranges header SHOULD be returned to inform the client of which format(s) that are allowed.

12.4.9 457 Invalid Range

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

12.4.10 458 Parameter Is Read-Only

The parameter to be set by SET_PARAMETER can be read but not modified. When returning this error message the sender SHOULD return a entity body containing the offending parameter(s).

12.4.11 459 Aggregate Operation Not Allowed

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

12.4.12 460 Only Aggregate Operation Allowed

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

12.4.13 461 Unsupported Transport

The Transport field did not contain a supported transport specification.

12.4.14 462 Destination Unreachable

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

12.5 Server Error 5xx

12.5.1 551 Option not supported

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

13 Header Field Definitions

method	direction	object	acronym	Body
DESCRIBE	$C \rightarrow S$	P,S	DES	r
GET_PARAMETER	$C \rightarrow S, S \rightarrow C$	P,S	GPR	R,r
OPTIONS	$C \rightarrow S$ $S \rightarrow C$	P,S	OPT	
PAUSE	$C \rightarrow S$	P,S	PSE	
PING	$C \rightarrow S, S \rightarrow C$	P,S	PNG	
PLAY	$C \rightarrow S$	P,S	PLY	
REDIRECT	$S \rightarrow C$	P,S	RDR	
SETUP	$C \rightarrow S$	S	STP	
SET_PARAMETER	$C \rightarrow S, S \rightarrow C$	P,S	SPR	R,r
TEARDOWN	$C \rightarrow S$	P,S	TRD	

Table 3: Overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on. Body notes if a method is allowed to carry body and in which direction, R = Request, r=response. Note: It is allowed for all error messages 4xx and 5xx to have a body

The general syntax for header fields is covered in Section 4.2 This section lists the full set of header fields along with notes on syntax, meaning, and usage. Throughout this section, we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification RFC 2616 [26]. Examples of each header field are given.

Information about header fields in relation to methods and proxy processing is summarized in Table 4 and Table 5.

The ”where” column describes the request and response types in which the header field can be used. Values in this column are:

R: header field may only appear in requests;

r: header field may only appear in responses;

2xx, 4xx, etc.: A numerical value or range indicates response codes with which the header field can be used;

c: header field is copied from the request to the response.

An empty entry in the ”where” column indicates that the header field may be present in all requests and responses.

The ”proxy” column describes the operations a proxy may perform on a header field:

a: A proxy can add or concatenate the header field if not present.

m: A proxy can modify an existing header field value.

d: A proxy can delete a header field value.

r: A proxy must be able to read the header field, and thus this header field cannot be encrypted.

The rest of the columns relate to the presence of a header field in a method. The method names when abbreviated, are according to table 3:

c: Conditional; requirements on the header field depend on the context of the message.

m: The header field is mandatory.

m*: The header field SHOULD be sent, but clients/servers need to be prepared to receive messages without that header field.

o: The header field is optional.

***:** The header field is required if the message body is not empty. See sections 13.14, 13.16 and 4.3 for details.

-: The header field is not applicable.

”Optional” means that a Client/Server MAY include the header field in a request or response, and a Client/Server MAY ignore the header field if present in the request or response (The exception to this rule is the Require header field discussed in 13.32). A ”mandatory” header field MUST be present in a request, and MUST be understood by the Client/Server receiving the request. A mandatory response header field MUST be present in the response, and the header field MUST be understood by the Client/Server processing the response. "Not applicable" means that the header field MUST NOT be present in a request. If one is placed in a request by mistake, it MUST be ignored by the Client/Server receiving the request. Similarly, a header field labeled "not applicable" for a response means that the Client/Server MUST NOT place the header field in the response, and the Client/Server MUST ignore the header field in the response.

A Client/Server SHOULD ignore extension header parameters that are not understood.

The From, Location, and RTP-Info header fields contain a URI. If the URI contains a comma, or semicolon, the URI MUST be enclosed in double quotas ("). Any URI parameters are contained within these quotas. If the URI is not enclosed in double quotas, any semicolon- delimited parameters are header-parameters, not URI parameters.

Header	Where	Proxy	DES	OPT	SETUP	PLAY	PAUSE	TRD
Accept	R		o	-	-	-	-	-
Accept-Encoding	R	r	o	-	-	-	-	-
Accept-Language	R	r	o	-	-	-	-	-
Accept-Ranges	r	r	-	-	o	-	-	-
Accept-Ranges	456	r	-	-	-	o	o	-
Allow	r		-	o	-	-	-	-
Allow	405		-	-	-	m	m	-
Authorization	R		o	o	o	o	o	o
Bandwidth	R		o	o	o	o	-	-
Blocksize	R		o	-	o	o	-	-
Cache-Control		r	-	-	o	-	-	-
Connection			o	o	o	o	o	o
Content-Base	r		o	-	-	-	-	-
Content-Base	4xx		o	o	o	o	o	o
Content-Encoding	R	r	-	-	-	-	-	-
Content-Encoding	r	r	o	-	-	-	-	-
Content-Encoding	4xx	r	o	o	o	o	o	o
Content-Language	R	r	-	-	-	-	-	-
Content-Language	r	r	o	-	-	-	-	-
Content-Language	4xx	r	o	o	o	o	o	o
Content-Length	r	r	*	-	-	-	-	-
Content-Length	4xx	r	*	*	*	*	*	*
Content-Location	r		o	-	-	-	-	-
Content-Location	4xx		o	o	o	o	o	o
Content-Type	r		*	-	-	-	-	-
Content-Type	4xx		*	*	*	*	*	*
CSeq	Rc		m	m	m	m	m	m
Date		am	o	o	o	o	o	o
Expires	r	r	o	-	-	-	-	-
From	R	r	o	o	o	o	o	o
Host			o	o	o	o	o	o
If-Match	R	r	-	-	o	-	-	-
If-Modified-Since	R	r	o	-	o	-	-	-
Last-Modified	r	r	o	-	-	-	-	-
Location	3rr		o	o	o	o	o	o
Proxy-Authenticate	407	amr	m	m	m	m	m	m
Proxy-Require	R	ar	o	o	o	o	o	o
Public	r	admr	-	m*	-	-	-	-
Public	501	admr	m*	m*	m*	m*	m*	m*
Range	R		-	-	-	o	o	-
Range	r		-	-	c	m*	m*	-
Referer	R		o	o	o	o	o	o
Require	R		o	o	o	o	o	o
Retry-After	3rr,503		o	o	o	-	-	-
RTP-Info	r		-	-	o	m	-	-

Header	Where	Proxy	DES	OPT	SETUP	PLAY	PAUSE	TRD
Scale			-	-	-	o	-	-
Session	R		-	o	o	m	m	m
Session	r		-	c	m	m	m	o
Server	R		-	o	-	-	-	-
Server	r		o	o	o	o	o	o
Speed			-	-	-	o	-	-
Supported	R		o	o	o	o	o	o
Supported	r		c	c	c	c	c	c
Timestamp	R		o	o	o	o	o	o
Timestamp	c		m	m	m	m	m	m
Transport			-	-	m	-	-	-
Unsupported	r		c	c	c	c	c	c
User-Agent	R		m*	m*	m*	m*	m*	m*
Vary	r		c	c	c	c	c	c
Via	R	amr	o	o	o	o	o	o
Via	c	dr	m	m	m	m	m	m
WWW-Authenticate	401		m	m	m	m	m	m
Header	Where	Proxy	DES	OPT	SETUP	PLAY	PAUSE	TRD

Table 4: Overview of RTSP header fields related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.

13.1 Accept

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

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

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

```
Accept: application/rtsl q=1.0, application/sdp;level=2
```

13.2 Accept-Encoding

See [H14.3]

13.3 Accept-Language

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

13.4 Accept-Ranges

The Accept-Ranges response-header field allows the server to indicate its acceptance of range requests and possible formats for a resource:

Header	Where	Proxy	GPR	SPR	RDR	PNG
Allow	405		-	-	-	-
Authorization	R		o	o	o	o
Bandwidth	R		-	o	-	-
Blocksize	R		-	o	-	-
Connection			o	o	o	-
Content-Base	R		o	o	-	-
Content-Base	r		o	o	-	-
Content-Base	4xx		o	o	o	-
Content-Encoding	R	r	o	o	-	-
Content-Encoding	r	r	o	o	-	-
Content-Encoding	4xx	r	o	o	o	-
Content-Language	R	r	o	o	-	-
Content-Language	r	r	o	o	-	-
Content-Language	4xx	r	o	o	o	-
Content-Length	R	r	*	*	-	-
Content-Length	r	r	*	*	-	-
Content-Length	4xx	r	*	*	*	-
Content-Location	R		o	o	-	-
Content-Location	r		o	o	-	-
Content-Location	4xx		o	o	o	-
Content-Type	R		*	*	-	-
Content-Type	r		*	*	-	-
Content-Type	4xx		*	*	*	-
CSeq	Rc		m	m	m	m
Date		am	o	o	o	o
From	R	r	o	o	o	o
Host			o	o	o	o
Last-Modified	R	r	-	-	-	-
Last-Modified	r	r	o	-	-	-
Location	3rr		o	o	o	o
Location	R		-	-	m	-
Proxy-Authenticate	407	amr	m	m	m	m
Proxy-Require	R	ar	o	o	o	o
Public	501	admr	m*	m*	m*	m*
Range	R		-	-	o	-
Referer	R		o	o	o	-
Require	R		o	o	o	o
Retry-After	3rr,503		o	o	-	-
Scale			-	-	-	-
Session	R		o	o	o	m
Session	r		c	c	o	m
Server	R		o	o	o	o
Server	r		o	o	-	o
Supported	R		o	o	o	o
Supported	r		c	c	c	c
Timestamp	R		o	o	o	o
Timestamp	c		m	m	m	m
Unsupported	r		c	c	c	c
User-Agent	R		m*	m*	-	m*
User-Agent	r		-	-	m*	-
Vary	r		c	c	-	-
Via	R	amr	o	o	o	o
Via	c	dr	m	m	m	m
WWW-Authenticate	401		m	m	m	m
Header	Where	Proxy	GPR	SPR	RDR	PNG

Table 5: Overview of RTSP header fields related to methods GET_PARAMETER, SET_PARAMETER,REDIRECT, and PING.

```
Accept-Ranges       =  "Accept-Ranges" ":" acceptable-ranges
acceptable-ranges   =  1#range-unit / "none"
range-unit          =  NPT / SMPTE / UTC / LIVE / extension-format
extension-format    =  token
```

This header has the same syntax as [H14.5]. However new range-units are defined and byte-ranges SHALL NOT be used. Inclusion of any of the three time formats indicates acceptance by the server for PLAY and PAUSE requests with this format. Inclusion of the "LIVE" tag indicates that the resource has LIVE properties. The headers value is valid for the resource specified by the URI in the request, this response corresponds to.

A server is RECOMMENDED to use this header in SETUP responses to indicate to the client which range time formats the media supports. The header SHOULD also be included in "456" responses which is a result of use of unsupported range formats.

13.5 Allow

The Allow entity-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. See [H14.7] for syntax definition.

Example of use:

```
Allow: SETUP, PLAY, SET_PARAMETER
```

13.6 Authorization

See [H14.8]

13.7 Bandwidth

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

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

Example:

```
Bandwidth: 4000
```

13.8 Blocksize

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

(400) if the value is syntactically invalid.

```
Blocksize   =   "Blocksize" ":" 1*DIGIT
```

13.9 Cache-Control

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

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

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

```
Cache-Control              =  "Cache-Control" ":" 1#cache-directive
cache-directive            =  cache-request-directive
                           /  cache-response-directive
cache-request-directive    =  "no-cache"
                           /  "max-stale" ["=" delta-seconds]
                           /  "min-fresh" "=" delta-seconds
                           /  "only-if-cached"
                           /  cache-extension
cache-response-directive   =  "public"
                           /  "private"
                           /  "no-cache"
                           /  "no-transform"
                           /  "must-revalidate"
                           /  "proxy-revalidate"
                           /  "max-age" "=" delta-seconds
                           /  cache-extension

cache-extension            =  token [ "=" ( token / quoted-string ) ]
delta-seconds              =  1*DIGIT
```

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

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

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

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

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

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

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

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

proxy-revalidate: The proxy-revalidate directive has the same meaning as the must-revalidate directive, except that it does not apply to non-shared user agent caches. It can be used on a response to an authenticated request to permit the user's cache to store and later return the response without needing to revalidate it (since it has already been authenticated once by that user), while still requiring proxies that service many users to revalidate each time (in order to make sure that each user has been authenticated). Note that such authenticated responses also need the public cache control directive in order to allow them to be cached at all.

max-age: When an intermediate cache is forced, by means of a max-age=0 directive, to revalidate its own cache entry, and the client has supplied its own validator in the request, the supplied validator might differ from the validator currently stored with the cache entry. In this case, the cache MAY use either validator in making its own request without affecting semantic transparency.

However, the choice of validator might affect performance. The best approach is for the intermediate cache to use its own validator when making its request. If the server replies with 304 (Not Modified), then the cache can return its now validated copy to the client with a 200 (OK) response. If the server replies with a new entity and cache validator, however, the intermediate cache can compare the returned validator with the one provided in the client's request, using the strong comparison function. If the client's validator is equal to the origin server's, then the intermediate cache simply returns 304 (Not Modified). Otherwise, it returns the new entity with a 200 (OK) response.

13.10 Connection

See [H14.10]. The use of the connection option "close" in RTSP messages SHOULD be limited to error messages when the server is unable to recover and therefore see it necessary to close the connection. The reason is that the client shall have the choice of continue using a connection indefinitely as long as it sends valid messages.

13.11 Content-Base

The Content-Base entity-header field may be used to specify the base URI for resolving relative URLs within the entity.

```
Content-Base = "Content-Base" ":" absoluteURI
```

If no Content-Base field is present, the base URI of an entity is defined either by its Content-Location (if that Content-Location URI is an absolute URI) or the URI used to initiate the request, in that order of precedence. Note, however, that the base URI of the contents within the entity-body may be redefined within that entity-body.

13.12 Content-Encoding

See [H14.11]

13.13 Content-Language

See [H14.12]

13.14 Content-Length

The Content-Length general-header field contains the length of the content of the method (i.e. after the double CRLF following the last header). Unlike HTTP, it MUST be included in all messages that carry content beyond the header portion of the message. If it is missing, a default value of zero is assumed. It is interpreted according to [H14.13].

13.15 Content-Location

See [H14.14]

13.16 Content-Type

See [H14.17]. Note that the content types suitable for RTSP are likely to be restricted in practice to presentation descriptions and parameter-value types.

13.17 CSeq

The CSeq general-header field specifies the sequence number for an RTSP request-response pair. This field MUST be present in all requests and responses. For every RTSP request containing the given sequence number, the corresponding response will have the same number. Any retransmitted request must contain the same sequence number as the original (i.e. the sequence number is *not* incremented for retransmissions of the same request). For each new RTSP request the CSeq value SHALL be incremented by one. The initial sequence number MAY be any number. Each sequence number series is unique between each requester and responder, i.e. the client has one series for its request to a server and the server has another when sending request to the client. Each requester and responder is identified with its network address.

CSeq = "Cseq" ":" 1*DIGIT

13.18 Date

See [H14.18]. An RTSP message containing a body MUST include a Date header if the sending host has a clock. Servers SHOULD include a Date header in all other RTSP messages.

13.19 Expires

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

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

A stale cache entry may not normally be returned by a cache (either a proxy cache or an user agent cache) unless it is first validated with the origin server (or with an intermediate cache that has a fresh copy of the entity). See section 14 for further discussion of the expiration model.

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

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

Expires = "Expires" ":" HTTP-date

An example of its use is

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

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

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

The presence of an Expires header field with a date value of some time in the future on a media stream that otherwise would by default be non-cacheable indicates that the media stream is cacheable, unless indicated otherwise by a Cache-Control header field (Section 13.9).

13.20 From

See [H14.22].

13.21 Host

The Host HTTP request header field [H14.23] is not needed for RTSP. It should be silently ignored if sent.

13.22 If-Match

See [H14.24].

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

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

13.23 If-Modified-Since

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

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

An example of the field is:

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

13.24 Last-Modified

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

13.25 Location

See [H14.30].

13.26 Proxy-Authenticate

See [H14.33].

13.27 Proxy-Require

The Proxy-Require request-header field is used to indicate proxy-sensitive features that MUST be supported by the proxy. Any Proxy-Require header features that are not supported by the proxy MUST be negatively acknowledged by the proxy to the client using the Unsupported header. Servers should treat this field identically to the Require field, i.e. the Proxy-Require requirements does also apply to the server.

See Section 13.32 for more details on the mechanics of this message and a usage example.

Proxy-Require = "Proxy-Require" ":" 1#feature-tag

Example of use:

```
Proxy-Require: play.basic, con.persistent
```

13.28 Public

The Public response-header field lists the set of methods supported by the server. The purpose of this field is strictly to inform the recipient of the capabilities of the server regarding unusual methods. The methods listed may or may not be applicable to the Request-URI; the Allow header field (section 14.7) MAY be used to indicate methods allowed for a particular URI.

Public = "Public" ":" 1#method

Example of use:

```
Public: OPTIONS, SETUP, PLAY, PAUSE, TEARDOWN
```

This header field applies only to the server directly connected to the client (i.e., the nearest neighbor in a chain of connections). If the response passes through a proxy, the proxy MUST either remove the Public header field or replace it with one applicable to its own capabilities.

13.29 Range

The Range request and response header field specifies a range of time. The range can be specified in a number of units. This specification defines the smpte (Section 3.4), npt (Section 3.5), and clock (Section 3.6) range units. Within RTSP, byte ranges [H14.35.1] are normally not meaningful. The header MAY contain a time parameter in UTC, specifying the time at which the operation is to be made effective. This functionality SHALL only be used with the REDIRECT method. Servers supporting the Range header MUST understand the NPT range format and SHOULD understand the SMPTE range format. The Range response header indicates what range of time is actually being played. If the Range header is given in a time format that is not understood, the recipient should return 501 (Not Implemented).

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

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

Example:

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

The notation is similar to that used for the HTTP/1.1 [26] byte-range header. It allows clients to select an excerpt from the media object, and to play from a given point to the end as well as from the current location to a given point. The start of playback can be scheduled for any time in the future, although a server may refuse to keep server resources for extended idle periods.

By default, range intervals increase, where the second point is larger than the first point.
Example:

```
Range: npt=10-15
```

However, range intervals can also decrease if the Scale header (see section 13.34) indicates a negative scale value. For example, this would be the case when a playback in reverse is desired.
Example:

```
Scale: -1
Range: npt=15-10
```

Decreasing ranges are still half open intervals as described above. Thus, For range $A - B$, A is closed and B is open. In the above example, 15 is closed and 10 is open. An exception to this rule is the case when $B = 0$ in a decreasing range. In this case, the range is closed on both ends, as otherwise there would be no way to reach 0 on a reverse playback.
Example:

```
Scale: -1
Range: npt=15-0
```

In this range both 15 and 0 are closed.
A decreasing range interval without a corresponding negative Scale header is not valid.

13.30 Referer

See [H14.36]. The URL refers to that of the presentation description, typically retrieved via HTTP.

13.31 Retry-After

See [H14.37].

13.32 Require

The Require request-header field is used by clients or servers to ensure that the other end-point supports features that are required in respect to this request. It can also be used to query if the other end-point supports certain features, however the use of the Supported (Section 13.38) is much more effective in this purpose. The server MUST respond to this header by using the Unsupported header to negatively acknowledge those feature-tags which are NOT supported. The response SHALL use the error code 551 (Option Not Supported). This header does not apply to proxies, for the same functionality in respect to proxies see, header Proxy-Require (Section 13.27).

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

Require = "Require" ":" feature-tag *("," feature-tag)

Example:

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

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

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

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

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

Proxies and other intermediary devices SHOULD ignore features that are not understood in this field. If a particular extension requires that intermediate devices support it, the extension should be tagged in the Proxy-Require field instead (see Section 13.27).

13.33 RTP-Info

The RTP-Info response-header field is used to set RTP-specific parameters in the PLAY response. For streams using RTP as transport protocol the RTP-Info header SHALL be part of a 200 response to PLAY.

The RTP-Info response-header field is used to set RTP-specific parameters in the PLAY response. These parameters correspond to the synchronization source identified by the ssrc parameter of the Transport response header in the SETUP reponse. For streams using RTP as transport protocol the RTP-Info header SHALL be part of a 200 response to PLAY.

url: Indicates the stream URL which for which the following RTP parameters correspond, this URL MUST be the same used in the SETUP request for this media stream. Any relative URL SHALL use the request URL as base URL.

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

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

> A mapping from RTP timestamps to NTP timestamps (wall clock) is available via RTCP. However, this information is not sufficient to generate a mapping from RTP timestamps to NPT. Furthermore, in order to ensure that this information is available at the necessary time (immediately at startup or after a seek), and that it is delivered reliably, this mapping is placed in the RTSP control channel.
>
> In order to compensate for drift for long, uninterrupted presentations, RTSP clients should additionally map NPT to NTP, using initial RTCP sender reports to do the mapping, and later reports to check drift against the mapping.

Additionally, the RTP-Info header parameter fields only apply to a single SSRC within a stream (the SSRC reported in the transport response header; see section 13.40). If there are multiple synchronization sources present within a RTP session, RTCP must be used to map RTP and NTP timestamps for those sources, for both synchronization and drift-checking.

Syntax:

```
RTP-Info        =  ”RTP-Info” ”:” 1#rtsp-info-spec
rtsp-info-spec  =  stream-url 1*parameter
stream-url      =  quoted-url / unquoted-url
unquoted-url    =  ”url” ”=” safe-url
quoted-url      =  ”url” ”=” <”> needquote-url <”>
safe-url        =  url
needquote-url   =  url //That contains ; or ,
url             =  ( absoluteURI / relativeURI )
parameter       =  ”;” ”seq” ”=” 1*DIGIT
                /  ”;” ”rtptime” ”=” 1*DIGIT
```

Additional constraint: safe-url MUST NOT contain the semicolon (”;”) or comma (”,”) characters. The quoted-url form SHOULD only be used when a URL does not meet the safe-url constraint, in order to ensure compatibility with implementations conformant to RFC 2326 [21].

absoluteURI and relativeURI are defined in RFC 2396 [22] with RFC 2732 [30] applied.

Example:

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

13.34 Scale

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

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

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

If the server does not implement the possibility to scale, it will not return a Scale header. A server supporting Scale operations for PLAY SHALL indicate this with the use of the "play.scale" feature-tags.

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

When indicating a negative scale for a reverse playback, the Range header must indicate a decreasing range as described in section 13.29.

Example of playing in reverse at 3.5 times normal rate:

```
Scale: -3.5
Range: npt=15-10
```

13.35 Speed

The Speed request-header field 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. All speeds may not be possible to support. Therefore the actual used speed MUST be included in the response. The lack of a response header is indication of lack of support from the server of this functionality. Support of the speed functionality are indicated by the "play.speed" feature-tag.

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

Example:

```
Speed: 2.5
```

Use of this field changes the bandwidth used for data delivery. It is meant for use in specific circumstances where preview of the presentation at a higher or lower rate is necessary. Implementors should keep in mind that bandwidth for the session may be negotiated beforehand (by means other than RTSP), and therefore re-negotiation may be necessary. When data is delivered over UDP, it is highly recommended that means such as RTCP be used to track packet loss rates. If the data transport is performed over public best-effort networks the sender is responsible for performing congestion control of the stream. This MAY result in that the communicated speed is impossible to maintain.

13.36 Server

See [H14.38], however the header syntax is here corrected.

Server = "Server" ":" (product / comment) *(SP (product / comment))

13.37 Session

The Session request-header and response-header field identifies an RTSP session started by the media server in a SETUP response and concluded by TEARDOWN on the presentation URL. The session identifier is chosen by the media server (see Section 3.3) and MUST be returned in the SETUP response. Once a client receives a Session identifier, it MUST return it for any request related to that session.

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

The timeout parameter is only allowed in a response header. The server uses it to indicate to the client how long the server is prepared to wait between RTSP commands or other signs of life before closing the session due to lack of activity (see Section A). The timeout is measured in seconds, with a default of 60 seconds (1 minute).

The mechanisms for showing liveness of the client is, any RTSP message with a Session header, or a RTCP message. It is RECOMMENDED that a client does not wait to the last second of the timeout before trying to send a liveness message. Even for RTSP messages using reliable protocols, such as TCP, the message may take some time to arrive safely at the receiver. To show liveness between RTSP request with other effects, the following mechanisms can be used, in descending order of preference:

RTCP: Is used to report transport statistics and SHALL also work as keep alive. The server can determine the client by used network address and port together with the fact that the client is reporting on the servers SSRC(s). A downside of using RTCP is that it gives lower statistical guarantees to reach the server. However that probability is so little that it can be ignored in most cases. For example, a session with 60 seconds timeout and enough bitrate assigned to the RTCP messages, so the client sends a message on average every 5 seconds. That session have for a network with 5 % packet loss the probability to not get a liveness sign over to the server in the timeout interval is 2.4*E-16. In sessions with shorter timeout times, or much higher packet loss, or small RTCP bandwidths SHOULD use any of the mechanisms below.

PING: The use of the PING method is the best of the RTSP based methods. It has no other effects than updating the timeout timer. In that way it will be a minimal message, that also does not cause any extra processing for the server. The downside is that it may not be implemented. A client SHOULD use a OPTIONS request to verify support of the PING at the server. It is possible to detect support by sending a PING to the server. If a 200 (OK) message is received the server supports it. In case a 501 (Not Implemented) is received it does not support PING and there is no meaning in continue trying. Also the reception of a error message will also mean that the liveness timer is not updated.

SET_PARAMETER: When using SET_PARAMETER for keep alive, no body SHOULD be included. This method is basically as good as PING, however the implementation support of the method is today limited. The same considerations as for PING apply regarding checking of support in server and proxies.

OPTIONS: This method does also work. However it causes the server to perform unnecessary processing and result in bigger responses than necessary for the task. The reason for this is that the Public is always included creating overhead.

Note that a session identifier identifies an RTSP session across transport sessions or connections. Control messages for more than one RTSP URL may be sent within a single RTSP session. Hence, it is possible that clients use the same session for controlling many streams constituting a presentation, as long as all the streams come from the same server. (See example in Section 15). However, multiple "user" sessions for the same URL from the same client MUST use different session identifiers.

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

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

13.38 Supported

The Supported header field enumerates all the extensions supported by the client or server. When offered in a request, the receiver MUST respond with its corresponding Supported header.

The Supported header field contains a list of feature-tags, described in Section 3.7, that are understood by the client or server.

```
Supported = "Supported" ":" [feature-tag *("," feature-tag)]
```

Example:

```
C->S:  OPTIONS rtsp://example.com/ RTSP/1.0\\
       Supported: foo, bar, blech\\

S->C:  RTSP/1.0 200 OK \\
       Supported: bar, blech, baz \\
```

13.39 Timestamp

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

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

13.40 Transport

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

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

The Transport header field 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 field 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 the form of multiple transportspec entries. In that case, the server MUST return the single option (transport-spec) which was actually chosen.

A transport-spec transport option may only contain one of any given parameter within it. Parameters may be given in any order. Additionally, it may only contain the unicast or multicast transport parameter.

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

The syntax for the transport specifier is

transport/profile/lower-transport.

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

Below are the configuration parameters associated with transport:

General parameters:

unicast / multicast: This parameter is a mutually exclusive indication of whether unicast or multicast delivery will be attempted. One of the two values MUST be specified. Clients that are capable of handling both unicast and multicast transmission MUST indicate such capability by including two full transport-specs with separate parameters for each.

destination: The address of the stream recipient to which a stream will be sent. The client originating the RTSP request may specify the destination address of the stream recipient with the destination parameter. When the destination field is specified, the recipient may be a different party than the originator of the request. To avoid becoming the unwitting perpetrator of a remote-controlled denial-of-service attack, a server SHOULD authenticate the client originating the request and SHOULD log such attempts before allowing the client to direct a media stream to a recipient address not chosen by the server. While, this is particularly important if RTSP commands are issued via UDP, implementations cannot rely on TCP as reliable means of client identification by itself either.

The server SHOULD NOT allow the destination field to be set unless a mechanism exists in the system to authorize the request originator to direct streams to the recipient. It is preferred that this authorization be performed by the recipient itself and the credentials passed along to the server. However, in certain cases, such as when recipient address is a multicast group, or when the recipient is unable to communicate with the server in an out-of-band manner, this may not be possible. In these cases server may chose another method such as a server-resident authorization list to ensure that the request originator has the proper credentials to request stream delivery to the recipient.

This parameter SHALL NOT be used when src_addr and dst_addr is used in a transport declaration. IPv6 addresses are RECOMMENDED to be given as fully qualified domain to make it backwards compatible with RFC 2326 implementations.

source: If the source address for the stream is different than can be derived from the RTSP endpoint address (the server in playback), the source address SHOULD be specified. To maintain backwards compatibility with RFC 2326, any IPv6 host's address must be given as a fully qualified domain name. This parameter SHALL NOT be used when src_addr and dst_addr is used in a transport declaration.

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

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

dest_addr: A general destination address parameter that can contain one or more address and port pair. For each combination of Protocol/Profile/Lower Transport the interpretation of the address or addresses needs to be defined. The client or server SHALL NOT use this parameter unless both client and server has shown support. This parameter MUST be supported by client and servers that implements this specification. Support is indicated by the use of the feature-tag "play.basic". This parameter SHALL NOT be used in the same transport specification as any of the parameters "destination", "source", "port", "client_port", and "server_port".

The same security consideration that are given for the "Destination" parameter does also applies to this parameter. This parameter can be used for redirecting traffic to recipient not desiring the media traffic.

src_addr: A General source address parameter that can contain one or more address and port pair. For each combination of Protocol/Profile/Lower Transport the interpretation of the address or addresses needs to be defined. The client or server SHALL NOT use this parameter unless both client and server has shown support. This parameter MUST be supported by client and servers that implements this specification. Support is indicated by the use the feature-tag "play.basic". This parameter SHALL NOT be used in the same transport specification as any of the parameters "destination", "source", "port", "client_port", and "server_port".

The address or addresses indicated in the src_addr parameter SHOULD be used both for sending and receiving of the media streams data packet. The main reasons are two: First by sending from the indicated ports the source address will be known by the receiver of the packet. Secondly, in the presence of NATs some traversal mechanism requires either knowledge from which address and port a packet flow is coming, or having the possibility to send data to the sender port.

mode: The mode parameter indicates the methods to be supported for this session. Valid values are PLAY and RECORD. If not provided, the default is PLAY. The RECORD value was defined in RFC 2326 and is deprecated in this specification.

append: The append parameter was used together with RECORD and is now deprecated.

interleaved: The interleaved parameter implies mixing the media stream with the control stream in whatever protocol is being used by the control stream, using the mechanism defined in Section 11.11. The argument provides the channel number to be used in the $ statement and MUST be present. This parameter MAY be specified as a range, e.g., `interleaved=4-5` in cases where the transport choice for the media stream requires it, e.g. for RTP with RTCP. The channel number given in the request are only a guidance from the client to the server on what channel number(s) to use. The server MAY set any valid channel number in the response. The declared channel(s) are bi-directional, so both end-parties MAY send data on the given channel. One example of such usage is the second channel used for RTCP, where both server and client sends RTCP packets on the same channel.

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

Multicast-specific:

ttl: multicast time-to-live.

RTP-specific:
These parameters are MAY only be used if the media transport protocol is RTP.

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

client_port: This parameter provides the unicast RTP/RTCP port pair on the client where media data and control information is to be sent. It is specified as a range, e.g., `port=3456-3457`. This parameter SHALL NOT be used when src_addr and dst_addr is used in a transport declaration.

server_port: This parameter provides the unicast RTP/RTCP port pair on the server where media data and control information is to be sent. It is specified as a range, e.g., `port=3456-3457`. This parameter SHALL NOT be used when src_addr and dst_addr is used in a transport declaration.

ssrc: The ssrc parameter, if included in a SETUP response, indicates the RTP SSRC [23, Sec. 3] value that will be used by the media server for RTP packets within the stream. It is expressed as an eight digit hexadecimal value. If the server does not act as a synchronization source for stream data (for instance, server is a translator, reflector, etc.) the value will be the "packet sender's SSRC" that would have been used in the RTCP Receiver Reports generated by the server, regardless of whether the server actually generates RTCP RRs. If there are multiple sources within the stream, the ssrc parameter only indicates the value for a single synchronization source. Other sources must be deduced from the actual RTP/RTCP stream.

The functionality of specifying the ssrc parameter in a SETUP request is deprecated as it is incompatible with the specification of RTP in RFC 1889. If the parameter is included in the transport header of a SETUP request, the server MAY ignore the it, and choose an appropriate SSRC for the stream. It MAY set the ssrc parameter in the transport header of the response.

```
Transport                  =   ”Transport” ”:” 1#transport-spec
transport-spec             =   transport-id *parameter
transport-id               =   transport-protocol ”/” profile [”/” lower-transport]
                               ; no LWS is allowed inside transport-id
transport-protocol         =   ”RTP” / token
profile                    =   ”AVP” / token
lower-transport            =   ”TCP” / ”UDP” / token
parameter                  =   ”;” ( ”unicast” / ”multicast” )
                           /   ”;” ”source” ”=” host
                           /   ”;” ”destination” [ ”=” host ]
                           /   ”;” ”interleaved” ”=” channel [ ”-” channel ]
                           /   ”;” ”append”
                           /   ”;” ”ttl” ”=” ttl
                           /   ”;” ”layers” ”=” 1*DIGIT
                           /   ”;” ”port” ”=” port-spec
                           /   ”;” ”client_port” ”=” port-spec
                           /   ”;” ”server_port” ”=” port-spec
                           /   ”;” ”ssrc” ”=” ssrc
                           /   ”;” ”mode” ”=” mode-spec
                           /   ”;” ”dest_addr” ”=” addr-list
                           /   ”;” ”src_addr” ”=” addr-list
                           /   ”;” trn-parameter-extension
port-spec                  =   port [ ”-” port ]
trn-parameter-extension    =   par-name ”=” trn-par-value
par-name                   =   token
trn-par-value              =   *unreserved
ttl                        =   1*3(DIGIT)
ssrc                       =   8*8(HEX)
channel                    =   1*3(DIGIT)
mode-spec                  =   <”> 1#mode <”> / mode
mode                       =   ”PLAY” / ”RECORD” / token
addr-list                  =   host-port *(”/” host-port)
host-port                  =   host [”:” port]
host                       =   see chapter 16
port                       =   see chapter 16
```

The combination of transport protocol, profile and lower transport needs to be defined. A number of combinations are defined in the appendix B.

Below is a usage example, showing a client advertising the capability to handle multicast or unicast, preferring multicast. Since this is a unicast-only stream, the server responds with the proper transport parameters for unicast.

```
C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
      CSeq: 302
      Transport: RTP/AVP;multicast;mode="PLAY",
          RTP/AVP;unicast;client_port=3456-3457;mode="PLAY"
```

```
S->C: RTSP/1.0 200 OK
      CSeq: 302
      Date: 23 Jan 1997 15:35:06 GMT
      Session: 47112344
      Transport: RTP/AVP;unicast;client_port=3456-3457;
          server_port=6256-6257;mode="PLAY"
```

13.41 Unsupported

The Unsupported response-header field lists the features not supported by the server. In the case where the feature was specified via the Proxy-Require field (Section 13.27), if there is a proxy on the path between the client and the server, the proxy MUST send a response message with a status code of 551 (Option Not Supported). The request SHALL NOT be forwarded.

See Section 13.32 for a usage example.

Unsupported = "Unsupported" ":" feature-tag *("," feature-tag)

13.42 User-Agent

See [H14.43] for explanation, however the syntax is clarified due to an error in RFC 2616. A Client SHOULD include this header in all RTSP messages it sends.

User-Agent = "User-Agent" ":" (product / comment) 0*(SP (product / comment)

13.43 Vary

See [H14.44]

13.44 Via

See [H14.45].

13.45 WWW-Authenticate

See [H14.47].

14 Caching

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

On receiving a SETUP or PLAY request, a proxy ascertains whether it has an up-to-date copy of the continuous media content and its description. It can determine whether the copy is up-to-date by issuing a SETUP or DESCRIBE request, respectively, and comparing the Last-Modified header with that of the cached copy. If the copy is not up-to-date, it modifies the SETUP transport parameters as appropriate and forwards the request to the origin server. Subsequent control commands such as PLAY or PAUSE then pass the proxy unmodified. The proxy delivers the continuous media data to the client, while possibly making a local copy for later reuse. The exact behavior allowed to the cache is given by the cache-response directives described in Section 13.9. A cache MUST answer any DESCRIBE requests if it is currently serving the stream to the requestor, as it is possible that low-level details of the stream description may have changed on the origin-server.

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

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

15 Examples

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

15.1 Media on Demand (Unicast)

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

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

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

W->C: HTTP/1.0 200 OK
      Date: 23 Jan 1997 15:35:06 GMT
      Content-Type: application/sdp

      v=0
      o=- 2890844526 2890842807 IN IP4 192.16.24.202
      s=RTSP Session
      e=adm@example.com
      m=audio 0 RTP/AVP 0
```

```
      a=control:rtsp://audio.example.com/twister/audio.en
      m=video 0 RTP/AVP 31
      a=control:rtsp://video.example.com/twister/video

C->A: SETUP rtsp://audio.example.com/twister/audio.en RTSP/1.0
      CSeq: 1
      User-Agent: PhonyClient/1.2
      Transport: RTP/AVP/UDP;unicast;client_port=3056-3057

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

C->V: SETUP rtsp://video.example.com/twister/video RTSP/1.0
      CSeq: 1
      User-Agent: PhonyClient/1.2
      Transport: RTP/AVP/UDP;unicast;client_port=3058-3059

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

C->V: PLAY rtsp://video.example.com/twister/video RTSP/1.0
      CSeq: 2
      User-Agent: PhonyClient/1.2
      Session: 23456789
      Range: smpte=0:10:00-

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

C->A: PLAY rtsp://audio.example.com/twister/audio.en RTSP/1.0
      CSeq: 2
      User-Agent: PhonyClient/1.2
      Session: 12345678
      Range: smpte=0:10:00-
```

```
A->C: RTSP/1.0 200 OK
      CSeq: 2
      User-Agent: PhonyClient/1.2
      Session: 12345678
      Range: smpte=0:10:00-0:20:00
      RTP-Info: url=rtsp://audio.example.com/twister/audio.en;
    seq=876655;rtptime=1032181

C->A: TEARDOWN rtsp://audio.example.com/twister/audio.en RTSP/1.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Session: 12345678

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

C->V: TEARDOWN rtsp://video.example.com/twister/video RTSP/1.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Session: 23456789

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

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

15.2 Streaming of a Container file

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

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

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

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

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

```
C->M: DESCRIBE rtsp://example.com/twister RTSP/1.0
      CSeq: 1

M->C: RTSP/1.0 200 OK
      CSeq: 1
      Date: 23 Jan 1997 15:35:06 GMT
      Content-Type: application/sdp
      Content-Length: 164

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

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

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

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

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

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

M->C: RTSP/1.0 200 OK
      CSeq: 4
      Session: 12345678
      Range: npt=0-
      RTP-Info: url=rtsp://example.com/twister/video;
    seq=12345;rtptime=3450012,
    url=rtsp://example.com/twister/audio;
    seq=54321;rtptime=2876889

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

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

C->M: PAUSE rtsp://example.com/twister RTSP/1.0
      CSeq: 6
      Session: 12345678

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

C->M: SETUP rtsp://example.com/twister RTSP/1.0
      CSeq: 7
      Transport: RTP/AVP;unicast;client_port=10000
      Session: 12345678

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

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

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

15.3 Single Stream Container Files

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

```
C->S  DESCRIBE rtsp://foo.com/test.wav RTSP/1.0
      Accept: application/x-rtsp-mh, application/sdp
      CSeq: 1

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

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

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

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

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

S->C  RTSP/1.0 200 OK
      CSeq: 3
      Session: 2034820394
      Range: npt=0-600
      RTP-Info: url=rtsp://foo.com/test.wav/streamid=0;
```

```
          seq=981888;rtptime=3781123
```

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

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

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

In the worst case, servers should send back:

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

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

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

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

15.4 Live Media Presentation Using Multicast

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

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

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

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

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

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

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

C->M: SETUP rtsp://live.example.com/concert/audio RTSP/1.0
      CSeq: 2
      Transport: RTP/AVP;multicast

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

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

M->C: RTSP/1.0 200 OK
      CSeq: 3
      Session: 0456804596
      Range:npt=now-
```

16 Syntax

The RTSP syntax is described in an augmented Backus-Naur form (BNF) as defined in RFC 2234 [14]. Also the "#" rule from RFC 2616 [26] is also defined and used in this syntax description.

16.1 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)>
BACKSLASH       =  <US-ASCII backslash (92)>
CRLF            =  CR LF
LWS             =  [CRLF] 1*( SP / HT )
TEXT            =  <any OCTET except CTLs>
tspecials       =  ”(” / ”)” / ”<” / ”>” / ”@”
                /  ”,” / ”;” / ”:” / BACKSLASH / <”>
                /  ”/” / ”[” / ”]” / ”?” / ”=”
                /  ”{” / ”}” / SP / HT
token           =  1*<any CHAR except CTLs or tspecials>
quoted-string   =  ( <”> *(qdtext) <”> )
qdtext          =  <any TEXT except <”>>
quoted-pair     =  BACKSLASH CHAR

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

safe            =  ”$” / ”-” / ”_” / ”.” / ”+”
extra           =  ”!” / ”*” / ”’” / ”(” / ”)” / ”,”

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

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

16.2 RTSP Protocol Definition

16.2.1 Message Syntax

```
RTSP-message     =  Request / Response ; RTSP/1.0 messages
generic-message  =  start-line
                    *(message-header CRLF)
                    CRLF
                    [ message-body ]
start-line       =  Request-Line / Status-Line

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

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

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

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

Method  =  "DESCRIBE"          ; Section 11.2
        /  "GET_PARAMETER"     ; Section 11.7
        /  "OPTIONS"           ; Section 11.1
        /  "PAUSE"             ; Section 11.5
        /  "PLAY"              ; Section 11.4
        /  "PING"              ; Section 11.10
        /  "REDIRECT"          ; Section 11.9
        /  "SETUP"             ; Section 11.3
        /  "SET_PARAMETER"     ; Section 11.8
        /  "TEARDOWN"          ; Section 11.6
        /  extension-method

extension-method  =  token
Request-URI       =  "*" / absolute_URI
RTSP-Version      =  "RTSP" "/" 1*DIGIT "." 1*DIGIT
```

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

```
extension-code  =  3DIGIT

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

general-header  =   Cache-Control        ; Section 13.9
                /   Connection           ; Section 13.10
                /   CSeq                 ; Section 13.17
                /   Date                 ; Section 13.18
                /   Timestamp            ; Section 13.39
                /   Via                  ; Section 13.44

request-header  =   Accept               ; Section 13.1
                /   Accept-Encoding      ; Section 13.2
                /   Accept-Language      ; Section 13.3
                /   Authorization        ; Section 13.6
                /   Bandwidth            ; Section 13.7
                /   Blocksize            ; Section 13.8
                /   From                 ; Section 13.20
                /   If-Modified-Since    ; Section 13.23
                /   Proxy-Require        ; Section 13.27
                /   Range                ; Section 13.29
                /   Referer              ; Section 13.30
                /   Require              ; Section 13.32
                /   Scale                ; Section 13.34
                /   Session              ; Section 13.37
                /   Speed                ; Section 13.35
                /   Supported            ; Section 13.38
                /   Transport            ; Section 13.40
                /   User-Agent           ; Section 13.42

response-header =   Accept-Ranges        ; Section 13.4
                /   Location             ; Section 13.25
                /   Proxy-Authenticate   ; Section 13.26
                /   Public               ; Section 13.28
                /   Range                ; Section 13.29
                /   Retry-After          ; Section 13.31
                /   RTP-Info             ; Section 13.33
                /   Scale                ; Section 13.34
                /   Session              ; Section 13.37
                /   Server               ; Section 13.36
                /   Speed                ; Section 13.35
                /   Transport            ; Section 13.40
                /   Unsupported          ; Section 13.41
                /   Vary                 ; Section 13.43
                /   WWW-Authenticate     ; Section 13.45
```

```
rtsp_URL          =  ( "rtsp:" / "rtspu:" / "rtsps" )
                     "//" host [ ":" port ] [ abs_path ] [ "#" fragment ]
host              =  As defined by RFC 2732 [30]
abs_path          =  As defined by RFC 2396 [22]
port              =  *DIGIT

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

npt-range        =  ["npt" "="] npt-range-spec
                    ; implementations SHOULD use npt= prefix, but SHOULD
                    ; be prepared to interoperate with RFC 2326
                    ; implementations which don't use it
npt-range-spec   =  ( npt-time "-" [ npt-time ] ) / ( "-" npt-time )
npt-time         =  "now" / npt-sec / npt-hhmmss
npt-sec          =  1*DIGIT [ "." *DIGIT ]
npt-hhmmss       =  npt-hh ":" npt-mm ":" npt-ss [ "." *DIGIT ]
npt-hh           =  1*DIGIT ; any positive number
npt-mm           =  1*2DIGIT ; 0-59
npt-ss           =  1*2DIGIT ; 0-59

utc-range        =  "clock" "=" utc-range-spec
utc-range-spec   =  ( utc-time "-" [ utc-time ] ) / ( "-" utc-time )
utc-time         =  utc-date "T" utc-time "Z"
utc-date         =  8DIGIT ; < YYYYMMDD >
utc-time         =  6DIGIT [ "." fraction ]; < HHMMSS.fraction >
fraction         =  1*DIGIT

feature-tag   =   token
```

16.2.2 Header Syntax

```
Transport                 =  "Transport" ":" 1#transport-spec
transport-spec            =  transport-id *parameter
transport-id              =  transport-protocol "/" profile ["/" lower-transport]
                             ; no LWS is allowed inside transport-id
transport-protocol        =  "RTP" / token
profile                   =  "AVP" / token
lower-transport           =  "TCP" / "UDP" / token
parameter                 =  ";" ( "unicast" / "multicast" )
                          /  ";" "source" "=" host
                          /  ";" "destination" [ "=" host ]
                          /  ";" "interleaved" "=" channel [ "-" channel ]
                          /  ";" "append"
                          /  ";" "ttl" "=" ttl
                          /  ";" "layers" "=" 1*DIGIT
                          /  ";" "port" "=" port-spec
                          /  ";" "client_port" "=" port-spec
                          /  ";" "server_port" "=" port-spec
                          /  ";" "ssrc" "=" ssrc
                          /  ";" "client_ssrc" "=" ssrc
                          /  ";" "mode" "=" mode-spec
                          /  ";" "dest_addresses" "=" addr-list
                          /  ";" "src_addresses" "=" addr-list
                          /  ";" trn-parameter-extension
port-spec                 =  port [ "-" port ]
trn-parameter-extension   =  par-name "=" trn-par-value
par-name                  =  token
trn-par-value             =  *unreserved
ttl                       =  1*3(DIGIT)
ssrc                      =  8*8(HEX)
channel                   =  1*3(DIGIT)
mode-spec                 =  <"> 1#mode <"> / mode
mode                      =  "PLAY" / "RECORD" / token
addr-list                 =  host-port *("/" host-port)
host-port                 =  host [":" port]
host                      =  see chapter 16
port                      =  see chapter 16
```

17 Security Considerations

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

Authentication Mechanisms: RTSP and HTTP share common authentication schemes, and thus should follow the same prescriptions with regards to authentication . See chapter 15.1 of [2] for client authentication issues, and chapter 15.2 of [2] for issues regarding support for multiple authentication mechanisms. Also see [H15.6].

Abuse of Server Log Information: RTSP and HTTP servers will presumably have similar logging mechanisms, and thus should be equally guarded in protecting the contents of those logs, thus protecting the privacy of the users of the servers. See [H15.1.1] for HTTP server recommendations regarding server logs.

Transfer of Sensitive Information: There is no reason to believe that information transferred via RTSP may be any less sensitive than that normally transmitted via HTTP. Therefore, all of the precautions regarding the protection of data privacy and user privacy apply to implementors of RTSP clients, servers, and proxies. See [H15.1.2] for further details.

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

Personal Information: RTSP clients are often privy to the same information that HTTP clients are (user name, location, etc.) and thus should be equally. See [H15.1] for further recommendations.

Privacy Issues Connected to Accept Headers: Since may of the same "Accept" headers exist in RTSP as in HTTP, the same caveats outlined in [H15.1.4] with regards to their use should be followed.

DNS Spoofing: Presumably, given the longer connection times typically associated to RTSP sessions relative to HTTP sessions, RTSP client DNS optimizations should be less prevalent. Nonetheless, the recommendations provided in [H15.3] are still relevant to any implementation which attempts to rely on a DNS-to-IP mapping to hold beyond a single use of the mapping.

Location Headers and Spoofing: If a single server supports multiple organizations that do not trust one another, then it must check the values of Location and Content-Location header fields in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority. ([H15.4])

In addition to the recommendations in the current HTTP specification (RFC 2616 [26], as of this writing) and also of the previous RFC2068 [2], future HTTP specifications may provide additional guidance on security issues.

The following are added considerations for RTSP implementations.

Concentrated denial-of-service attack: The protocol offers the opportunity for a remote-controlled denial-of-service attack.

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

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

Authentication: Servers SHOULD implement both basic and digest [6] authentication. In environments requiring tighter security for the control messages, transport layer mechanisms such as TLS (RFC 2246 [27]) SHOULD be used.

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

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

18 IANA Considerations

This section set up a number of registers for RTSP that should be maintained by IANA. For each registry there is a description on what it shall contain, what specification is needed when adding a entry with IANA, and finally the entries that this document needs to register. See also the section 1.6 ”Extending RTSP”. There is also a IANA registration of two SDP attributes.

The sections describing how to register an item uses some of the requirements level described in RFC 2434 [29], namely ” First Come, First Served”, ”Specification Required”, and ”Standards Action”.

A registration request to IANA MUST contain the following information:

- A name of the item to register according to the rules specified by the intended registry.
- Indication of who has change control over the feature (for example, IETF, ISO, ITU-T, other international standardization bodies, a consortium or a particular company or group of companies);
- A reference to a further description, if available, for example (in order of preference) an RFC, a published standard, a published paper, a patent filing, a technical report, documented source code or a computer manual;
- For proprietary features, contact information (postal and email address);

18.1 Feature-tags

18.1.1 Description

When a client and server try to determine what part and functionality of the RTSP specification and any future extensions that its counter part implements there is need for a namespace. This registry contains named entries representing certain functionality.

The usage of feature-tags is explained in section 10 and 11.1.

18.1.2 Registering New Feature-tags with IANA

The registering of feature-tags is done on a first come, first served basis.

The name of the feature MUST follow these rules: The name may be of any length, but SHOULD be no more than twenty characters long. The name MUST not contain any spaces, or control characters. Any proprietary feature SHALL have as the first part of the name a vendor tag, which identifies the organization.

18.1.3 Registered entries

The following feature-tags are in this specification defined and hereby registered. The change control belongs to the Authors and the IETF MMUSIC WG.

play.basic: The minimal implementation for playback operations according to section D.

play.scale: Support of scale operations for media playback.

play.speed: Support of the speed functionality for playback.

setup.playing: The use of SETUP and TEARDOWN in play state.

con.persistent: Support and use of persistent connections, see chapter 9.3.

18.2 RTSP Methods

18.2.1 Description

What a method is, is described in section 11. Extending the protocol with new methods allow for totally new functionality.

18.2.2 Registering New Methods with IANA

A new method MUST be registered through an IETF standard track document. The reason is that new methods may radically change the protocols behavior and purpose.

A specification for a new RTSP method MUST consist of the following items:

- A method name which follows the BNF rules for methods.
- A clear specification on what action and response a request with the method will result in. Which directions the method is used, $C \rightarrow S$ or $S \rightarrow C$ or both. How the use of headers, if any, modifies the behavior and effect of the method.
- A list or table specifying which of the registered headers that are allowed to use with the method in request or/and response.
- Describe how the method relates to network proxies.

18.2.3 Registered Entries

This specification, RFCXXXX, registers 10 methods: DESCRIBE, GET_PARAMETER, OPTIONS, PAUSE, PING, PLAY, REDIRECT, SETUP, SET_PARAMETER, and TEARDOWN.

18.3 RTSP Status Codes

18.3.1 Description

A status code is the three digit numbers used to convey information in RTSP response messages, see 7. The number space is limited and care should be taken not to fill the space.

18.3.2 Registering New Status Codes with IANA

A new status code can only be registered by an IETF standards track document. A specification for a new status code MUST specify the following:

- The requested number.
- A description what the status code means and the expected behavior of the sender and receiver of the code.

18.3.3 Registered Entries

RFCXXX, registers the numbered status code defined in the BNF entry "Status-Code" except "extension-code" in section 7.1.1.

18.4 RTSP Headers

18.4.1 Description

By specifying new headers a method(s) can be enhanced in many different ways. An unknown header will be ignored by the receiving entity. If the new header is vital for a certain functionality, a feature-tag for the functionality can be created and demanded to be used by the counter-part with the inclusion of a Require header carrying the feature-tag.

18.4.2 Registering New Headers with IANA

A public available specification is required to register a header. The specification SHOULD be a standards document, preferable an IETF RFC.

The specification MUST contain the following information:

- The name of the header.
- A BNF specification of the header syntax.
- A list or table specifying when the header may be used, encompassing all methods, their request or response, the direction ($C \rightarrow S$ or $S \rightarrow C$).
- How the header shall be handled by proxies.
- A description of the purpose of the header.

18.4.3 Registered entries

All headers specified in section 13 in RFCXXXX are to be registered.

Furthermore the following RTSP headers defined in other specifications are registered:

- x-wap-profile defined in [35].
- x-wap-profile-diff defined in [35].
- x-wap-profile-warning defined in [35].
- x-predecbufsize defined in [35].
- x-initpredecbufperiod defined in [35].
- x-initpostdecbufperiod defined in [35].

  Note: The use of "X-" is NOT RECOMMENDED but the above headers in the register list was defined prior to the clarification.

18.5 Transport Header registries

The transport header contains a number of parameters which have possibilities for future extensions. Therefore registries for these must be defined.

18.5.1 Transport Protocols

A registry for the parameter transport-protocol shall be defined with the following rules:

- Registering requires public available standards specification.
- A contact person or organization with address and email.
- A value definition that are following the BNF token definition.
- A describing text that explains how the registered value are used in RTSP.

This specification register 1 value:

RTP: Use of the RTP [23] protocol for media transport. The usage is explained in RFC XXXX, appendix B.1.

18.5.2 Profile

A registry for the parameter profile shall be defined with the following rules:

- Registering requires public available standards specification.
- A contact person or organization with address and email.
- A value definition that are following the BNF token definition.

- A definition of which Transport protocol(s) that this profile is valid for.
- A describing text that explains how the registered value are used in RTSP.

AVP: The "RTP profile for audio and video conferences with minimal control" [1] MUST only be used when the transport headers transport-protocol is "RTP".

18.5.3 Lower Transport

A registry for the parameter lower-transport shall be defined with the following rules:

- Registering requires public available standards specification.
- A contact person or organization with address and email.
- A value definition that are following the BNF token definition.
- A describing text that explains how the registered value are used in RTSP. This includes

UDP: Indicates the use of the "User datagram protocol" [7] for media transport.

TCP: Indicates the use Transmission control protocol [9] for media transport.

18.5.4 Transport modes

A registry for the transport parameter mode shall be defined with the following rules:

- Registering requires a IETF standard tracks document.
- A contact person or organization with address and email.
- A value definition that are following the BNF token definition.
- A describing text that explains how the registered value are used in RTSP.

PLAY: See RFC XXXX.

RECORD: See RFC XXXX.

18.6 Cache Directive Extensions

There exist a number of cache directives which can be sent in the Cache-Control header. A registry for this cache directives shall be defined with the following rules:

- Registering requires a IETF standard tracks document.
- A registration shall name a contact person.
- Name of the directive and a definition of the value, if any.
- A describing text that explains how the cache directive is used for RTSP controlled media streams.

18.7 SDP attributes

This specification defines two SDP [24] attributes that it is request that IANA register.

```
SDP Attribute ("att-field"):

     Attribute name:     range
     Long form:          Media Range Attribute
     Type of name:       att-field
     Type of attribute:  Media and session level
     Subject to charset: No
     Purpose:            RFC XXXX
     Reference:          RFC XXXX
     Values:             See ABNF definition.

     Attribute name:     control
     Long form:          RTSP control URL
     Type of name:       att-field
     Type of attribute:  Media and session level
     Subject to charset: No
     Purpose:            RFC XXXX
     Reference:          RFC XXXX
     Values:             Absolute or Relative URLs.
```

A RTSP Protocol State Machine

The RTSP session state machine describe the behavior of the protocol from RTSP session initialization through RTSP session termination.

State machine is defined on a per session basis which is uniquely identified by the RTSP session identifier. The session may contain zero or more media streams depending on state. If a single media stream is part of the session it is in non-aggregated control. If two or more is part of the session it is in aggregated control.

This state machine is one possible representation that helps explain how the protocol works and when different requests are allowed. We find it a reasonable representation but does not mandate it, and other representations can be created.

A.1 States

The state machine contains five states, described below. For each state there exist a table which shows which requests and events that is allowed and if they will result in a state change.

Init: Initial state no session exist.

Ready-nm: Ready state without any medias.

Ready: Session is ready to start playing.

Play: Session is playing, i.e. sending media stream data in the direction $S \to C$.

A.2 State variables

This representation of the state machine needs more than its state to work. A small number of variables are also needed and is explained below.

NRM: The number of media streams part of this session.

RP: Resume point, the point in the presentation time line at which a request to continue will resume from. A time format for the variable is not mandated.

A.3 Abbreviations

To make the state tables more compact a number of abbreviations are used, which are explained below.

IFI: IF Implemented.

md: Media

PP: Pause Point, the point in the presentation time line at which the presentation was paused.

Prs: Presentation, the complete multimedia presentation.

RedP: Redirect Point, the point in the presentation time line at which a REDIRECT was specified to occur.

SES: Session.

A.4 State Tables

This section contains a table for each state. The table contains all the requests and events that this state is allowed to act on. The events which is method names are, unless noted, requests with the given method in the direction client to server ($C \to S$). In some cases there exist one or more requisite. The response column tells what type of response actions should be performed. Possible actions that is requested for an event includes: response codes, e.g. 200, headers that MUST be included in the response, setting of state variables, or setting of other session related parameters. The new state column tells which state the state machine shall change to.

The response to valid request meeting the requisites is normally a 2xx (SUCCESS) unless other noted in the response column. The exceptions shall be given a response according to the response column. If the request does not meet the requisite, is erroneous or some other type of error occur the appropriate response code MUST be sent. If the response code is a 4xx the session state is unchanged. A response code of 3rr will result in that the session is ended and its state is changed to Init. A response code of 304 results in no state change. However there exist restrictions to when a 3xx — response may be used. A 5xx response SHALL not result in any change of the session state, except if the error is not possible to recover from. A unrecoverable error SHALL result the ending of the session. As it in the general case can't be determined if it was a unrecoverable error or not the client will be required to test. In the case that the next request after a 5xx is responded with 454 (Session Not Found) the client SHALL assume that the session has been ended.

The server will timeout the session after the period of time specified in the SETUP response, if no activity from the client is detected. Therefore there exist a timeout event for all states except Init.

In the case that $NRM = 1$ the presentation URL is equal to the media URL. For $NRM > 1$ the presentation URL MUST be other than any of the medias that are part of the session. This applies to all states.

Event	Prerequisite	Response
DESCRIBE	Needs REDIRECT	3rr Redirect
DESCRIBE		200, Session description
OPTIONS	Session ID	200, Reset session timeout timer
OPTIONS		200
SET_PARAMETER	Valid parameter	200, change value of parameter
GET_PARAMETER	Valid parameter	200, return value of parameter

Table 6: None state-machine changing events

The methods in Table 6 do not have any effect on the state machine or the state variables. However some methods do change other session related parameters, for example SET_PARAMETER which will set the parameter(s) specified in its body.

Action	Requisite	New State	Response
SETUP		Ready	$NRM = 1$, $RP = 0.0$
SETUP	Needs Redirect	Init	3rr Redirect

Table 7: State: Init

The initial state of the state machine, see Table 7 can only be left by processing a correct SETUP request. As seen in the table the two state variables are also set by a correct request. This table also shows that a correct SETUP can in some cases be redirected to another URL and/or server by a 3rr response.

Action	Requisite	New State	Response
SETUP		Ready	$NRM = 1$,$RP = 0.0$
SETUP	Needs Redirect	Init	3rr
TEARDOWN	URL=*	Init	No session hdr.
Timeout		Init	
$S \rightarrow C$:REDIRECT	Range hdr	Ready-nm	Set RedP
$S \rightarrow C$:REDIRECT	no range hdr	Ready-nm	TEARDOWN of session
RedP reached		Ready-nm	TEARDOWN of session

Table 8: State: Ready-nm

The optional Ready-nm state has no media streams and therefore can't play. This state exist so that all session related parameters and resources can be kept while changing media stream(s). As seen in Table 8 the operations are limited to setting up a new media or tearing down the session. The established session can also be redirected with the REDIRECT method.

Action	Requisite	New State	Response
SETUP	New URL	Ready	$NRM+ = 1$
SETUP	Setten up URL	Ready	Change transport param.
TEARDOWN	URL=*	Init	No session hdr
TEARDOWN	Prs URL,$NRM > 1$	Init	No session hdr
TEARDOWN	md URL,$NRM = 1$IFI	Ready-nm	Session hdr, $NRM = 0$
TEARDOWN	md URL,$NRM = 1$	Init	No Session hdr, $NRM = 0$
TEARDOWN	md URL,$NRM > 1$	Ready	Session hdr, $NRM- = 1$
PLAY	Prs URL, No range	Play	Play from RP
PLAY	Prs URL, Range	Play	according to range
$S \to C$:REDIRECT	Range hdr	Ready	Set RedP
$S \to C$:REDIRECT	no range hdr	Ready	TEARDOWN of session
Timeout		Init	
RedP reached		Ready	TEARDOWN of session

Table 9: State: Ready

In the Ready state, see Table 9, some of the actions are depending on the number of media streams (NRM) in the session, i.e. aggregated or non-aggregated control. A setup request in the ready state can either add one more media stream to the session or if the media stream (same URL) already is part of the session change the transport parameters. TEARDOWN is depending on both the request URI and the number of media stream within the session. If the request URI is either * or the presentations URI the whole session is torn down. If a media URL is used in the TEARDOWN request and more than one media exist in the session, the session will remain and a session header MUST be returned in the response. If only a single media stream remains in the session when performing a TEARDOWN with a media URL , it is optional to keep the session. If the session still exist after the request a Session MUST be returned in the response. The number of media streams remaining after tearing down a media stream determines the new state.

The Play state table, see Table 10, is the largest. The table contains an number of request that has presentation URL as a prerequisite on the request URL, this is due to the exclusion of non-aggregated stream control in sessions with more than one media stream.

To avoid inconsistencies between the client and server, automatic state transitions are avoided. This can be seen at for example "End of media" event when all media has finished playing, the session still remain in Play state. An explicit PAUSE request must be sent to change the state to Ready. It may appear that there exist two automatic transitions in "RedP reached" and "PP reached", however they are requested and acknowledge before they take place. The time at which the transition will happen is known by looking at the range header. If the client sends request close in time to these transitions it must be prepared for getting error message as the state may or may not have changed.

SETUP and TEARDOWN requests with media URLs in aggregated sessions may not be handled by the server as it is optional functionality. Use the service discovery mechanism with OPTIONS to find out in beforehand if the server implements it. If the functionality is not implemented but still tried by the client a "501 Not Implemented" response SHALL be received.

Action	Requisite	New State	Response
PAUSE	PrsURL,No range	Ready	Set RP to present point
PAUSE	PrsURL,$Range > now$	Play	Set RP & PP to given point
PAUSE	PrsURL,$Range <= now$	Ready	Set RP to Range Hdr.
PP reached		Ready	RP = PP
End of media	All media	Play	No action, RP = Invalid
End of media	$>= 1$ Media plays	Play	No action
End of range		Play	Set RP = End of range
SETUP	New URL,IFI	Play	$NRM+ = 1$, 200, *A
SETUP	New URL	Play	455
SETUP	Setuped URL	Play	455
SETUP	Setuped URL, IFI	Play	Change transport param.
TEARDOWN	$URL = *$	Init	No session hdr
TEARDOWN	Prs URL,$NRM > 1$	Init	No session hdr
TEARDOWN	md URL,$NRM = 1$,IFI	Ready-nm	Session hdr
TEARDOWN	md URL,$NRM > 1$,IFI	Play	Session hdr
TEARDOWN	md URL	Play	455
$S \to C$:REDIRECT	Range hdr	Play	Set RedP
$S \to C$:REDIRECT	no range hdr	Play	TEARDOWN of session
RedP reached		Play	TEARDOWN of session
Timeout		Init	Stop Media playout

Table 10: State: Play, *A: RTP-Info and Range header

B Media Transport Alternatives

This chapter defines how certain combinations of protocols, profiles and lower transports are used. This includes the usage of the Transport header's general source and destination parameters "src_addresses" and "dst_addresses".

B.1 RTP

This section defines the interaction and needed media transport signalling in regards to the RTP protocol [23].

RTSP allows media clients to control selected, non-contiguous sections of media presentations, rendering those streams with an RTP media layer[23]. The media layer rendering the RTP stream should not be affected by jumps in NPT. Thus, both RTP sequence numbers and RTP timestamps MUST be continuous and monotonic across jumps of NPT.

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

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

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

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

For scaling (see Section 13.34), RTP timestamps should correspond to the playback timing. For example, when playing video recorded at 30 frames/second at a scale of two and speed (Section 13.35) of one, the server would drop every second frame to maintain and deliver video packets with the normal timestamp spacing of 3,000 per frame, but NPT would increase by 1/15 second for each video frame.

The client can maintain a correct display of NPT by noting the RTP timestamp value of the first packet arriving after repositioning. The sequence parameter of the RTP-Info (Section 13.33) header provides the first sequence number of the next segment.

Below the available RTP profiles and lower layer transports are given together with the necessary rules on how to signal that combination.

B.1.1 AVP

The usage of the "RTP Profile for Audio and Video Conferences with Minimal Control" [1] when using RTP for media transport over different lower layer transport protocols are defined below in regards to RTSP.

On such case is defined within this document, the use of embedded (interleaved) binary data as defined in section 11.11. The usage of this method is indicated by include the "interleaved" parameter.

When using embedded binary data the "src_addresses" and "dst_addresses" SHALL NOT be used. This addressing and multiplexing is used as defined with use of channel numbers and the interleaved parameter.

B.1.2 AVP/UDP

This part descibes sending of RTP [23] over lower transport layer UDP [7] according to the profile "RTP Profile for Audio and Video Conferences with Minimal Control" defined in RFC 1890 [1].

This profiles requires that one or two uni- or bi-directional UDP flows per media stream. The first UDP flow is for RTP and the second is for RTCP. Embedded (interleaved) data when RTSP messages is transported over UDP SHOULD NOT be performed.

The RTP/UDP and RTCP/UDP flows can be established in two ways using the Transport header's parameters. The way provided in RFC 2326 was to use the necessary parameters from the set of "source", "destination", "client_port", and "server_port". This has the advantage of being compatible with all RTP capable RTSP servers and clients. However this method does not provide a possibility to specify non-continues port ranges for RTP and RTCP. The other way is to use the parameters "src_addresses", and "dst_addresses". This method provides total flexibility in specifying address and port number for each transport flow. However the disadvantage is that it is not supported by non-updated clients, i.e. clients not supporting the "play.basic" feature-tag.

When using the "source", "destination", "client_port", and "server_port" the packets are be addressed in the following way for media playback:

- RTP/UDP packet from the server to the client SHALL be sent to the address specified in the "destination" parameter and first even port number given in client_port range. If there is only a single port number given that MUST be given.
- The server SHOULD send its RTP/UDP packets from the address specified in "source" parameter and from the first even port number specified in "server_port" parameter.
- If there is specified a range in "client_port" parameter that contains at least two port numbers, the RTCP/UDP packets from server to client SHALL be sent to address specified in the "destination" parameter and first odd port number part of the range specified in the client_port parameter.
- The Server SHOULD send its RTCP/UDP packets from the address specified in "source" parameter and from the first odd port number specified in "server_port" parameter.
- RTCP/UDP packets from the client to the server SHALL be sent to the address specified in the "source" parameter and first odd port number given in client_port range.
- The client SHOULD send its RTCP/UDP packets from the address specified in "destination" parameter and from the first odd port number specified in "server_port" parameter.

The usage of "src_addresses" and "dst_addresses" parameters to specify the address and port numbers are done in the following way for media playback, i.e. Mode=PLAY:

- The "src_addresses" and "dst_addresses" parameters MUST contain either 1 or 2 address and port pairs.
- Each address and port pair MUST contain both and address and a port number.
- The first address and port pair given in either of the parameters applies to the RTP stream. The second address and port pair if present applies to the RTCP stream.
- The RTP/UDP packets from the server to the client SHALL be sent to the address and port given by first address and port pair of the "dst_addresses" parameter.
- The RTCP/UDP packets from the server to the client SHALL be sent to the address and port given by the second address and port pair of the "dst_addresses" parameter. If no second pair is given RTCP SHALL NOT be sent.
- The RTCP/UDP packets from the client to the server SHALL be sent to the address and port given by the second address and port pair of the "dst_addresses" parameter. If no second pair is given RTCP SHALL NOT be sent.
- RTP and RTCP Packets SHOULD be sent from the corresponding receiver port, i.e. RTCP packets from server should be sent from the "src_addresses" parameters second address port pair.

B.1.3 AVP/TCP

Note that this combination is not yet defined using sperate TCP connections. However the use of embedded (interleaved) binary data transported on the RTSP connection is possible as specified in section 11.11. When using this declared combination of interleaved binary data the RTSP messages MUST be transported over TCP.

A possible future for this profile would be to define the use of a combination of the two drafts ”Connection-Oriented Media Transport in SDP” [36] and ”Framing RTP and RTCP Packets over Connection-Oriented Transport” [37].

B.2 Future Additions

It is the intention that any future protocol or profile regarding both for media delivery and lower transport should be easy to add to RTSP. This chapter provides the necessary steps that needs to be meet.

The following things needs to be considered when adding a new protocol of profile for use with RTSP:

- The protocol or profile needs to define a name tag representing it. This tag is required to be a ABNF ”token” to be possible to use in the Transport header specification.
- The useful combinations of protocol/profile/lower-layer needs to be defined and for each combination declare the necessary parameters to use in the Transport header.
- For new media protocols the interaction with RTSP needs to be addressed. One important factor will be the media synchronization.

See the IANA section (18) on how to register the necessary attributes.

C Use of SDP for RTSP Session Descriptions

The Session Description Protocol (SDP, RFC 2327 [24]) may be used to describe streams or presentations in RTSP. This description is typically returned in reply to a DESCRIBE request on a URL from a server to a client, received via HTTP from a server to a client.

This appendix describes how an SDP file determines the operation of an RTSP session. SDP provides no mechanism by which a client can distinguish, without human guidance, between several media streams to be rendered simultaneously and a set of alternatives (e.g., two audio streams spoken in different languages).

C.1 Definitions

The terms “session-level”, “media-level” and other key/attribute names and values used in this appendix are to be used as defined in SDP (RFC 2327 [24]):

C.1.1 Control URL

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

```
control-attribute = "a=" "control" ":" url
```

Example:

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

This attribute MAY contain either relative and absolute URLs, following the rules and conventions set out in RFC 2396 [22]. Implementations SHALL look for a base URL in the following order:

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

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

For SDP retrieved from a container file, there are certain things to consider. Lets say that the container file has the following URL: "rtsp://example.com/container.mp4". A media level relative URL needs to contain the file name container.mp4 in the beginning to be resolved correctly relative to the before given URL. An alternative if one does not desire to enter the container files name is to ensure that the base URL for the SDP document becomes: "rtsp://example.com/container.mp4/", i.e. an extra trailing slash. When using the URL resolution rules in RFC 2396 that will resolve correctly. However as a warning if the session level control URL is a * that control URL will be equal to "rtsp://example.com/container.mp4/" and include the slash.

C.1.2 Media Streams

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

Example:

```
m=audio 0 RTP/AVP 31
```

C.1.3 Payload Type(s)

The payload type(s) are specified in the "m=" field. In case the payload type is a static payload type from RFC 1890 [1], no other information is required. In case it is a dynamic payload type, the media attribute "rtpmap" is used to specify what the media is. The "encoding name" within the "rtpmap" attribute may be one of those specified in RFC 1890 (Sections 5 and 6), or an MIME type registered with IANA, or an experimental encoding with a "X-" prefix as specified in SDP (RFC 2327 [24]). Codec-specific parameters are not specified in this field, but rather in the "fmtp" attribute described below. Implementors seeking to register new encodings should follow the procedure in RFC 1890 [1]. If the media type is not suited to the RTP AV profile, then it is recommended that a new profile be created and the appropriate profile name be used in lieu of "RTP/AVP" in the "m=" field.

C.1.4 Format-Specific Parameters

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

C.1.5 Range of Presentation

The "a=range" attribute defines the total time range of the stored session. (The length of live sessions can be deduced from the "t" and "r" parameters.) The attribute is a session and media level attribute. For presentations that contains media streams of the same durations, the range attribute SHOULD only be used at session-level. In case of different length the range attribute MUST be given at media level for all media. The unit is specified first, followed by the value range. The units and their values are as defined in Section 3.4, 3.5 and 3.6. The range attribute SHOULD NOT be present for live media streams.

This attribute is defined in ABNF [14] as:

a-range-def = "a" "=" "range" ":" ranges-specifier CRLF

Examples:

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

C.1.6 Time of Availability

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

C.1.7 Connection Information

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

C.1.8 Entity Tag

The optional "a=etag" attribute identifies a version of the session description. It is opaque to the client. SETUP requests may include this identifier in the If-Match field (see section 13.22) to only allow session establishment if this attribute value still corresponds to that of the current description. The attribute value is opaque and may contain any character allowed within SDP attribute values.

Example:

```
a=etag:158bb3e7c7fd62ce67f12b533f06b83a
```

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

C.2 Aggregate Control Not Available

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

Example:

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

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

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

C.3 Aggregate Control Available

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

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

Example:

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

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

A client is **not** required to issues SETUP requests for all streams within an aggregate object. Servers SHOULD allow the client to ask for only a subset of the streams.

D Minimal RTSP implementation

D.1 Client

A client implementation MUST be able to do the following :

- Generate the following requests: SETUP, TEARDOWN, PLAY.
- Include the following headers in requests: CSeq, Connection, Session, Transport.
- Parse and understand the following headers in responses: CSeq, Connection, Session, Transport, Content-Language, Content-Encoding, Content-Length, Content-Type.
- Understand the class of each error code received and notify the end-user, if one is present, of error codes in classes 4xx and 5xx. The notification requirement may be relaxed if the end-user explicitly does not want it for one or all status codes.
- Expect and respond to asynchronous requests from the server, such as REDIRECT. This does not necessarily mean that it should implement the REDIRECT method, merely that it MUST respond positively or negatively to any request received from the server.

Though not required, the following are RECOMMENDED.

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

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

D.1.1 Basic Playback

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

- generate the PAUSE request;
- implement the REDIRECT method, and the Location header.

D.1.2 Authentication-enabled

In order to access media presentations from RTSP servers that require authentication, the client MUST additionally be able to do the following:

- recognize the 401 (Unauthorized) status code;
- parse and include the WWW-Authenticate header;
- implement Basic Authentication and Digest Authentication.

D.2 Server

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

- Implement the following methods: SETUP, TEARDOWN, OPTIONS and PLAY.
- Include the following headers in responses: Connection, Content-Length, Content-Type, Content-Language, Content-Encoding, Timestamp, Transport, Public, and Via, and Unsupported. RTP-compliant implementations MUST also implement the RTP-Info field.
- Parse and respond appropriately to the following headers in requests: Connection, Proxy-Require, Session, Transport, and Require.

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

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

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

D.2.1 Basic Playback

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

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

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

- Include and parse the Range header, with NPT units. Implementation of SMPTE units is recommended.

- Include the length of the media presentation in the media initialization information.
- Include mappings from data-specific timestamps to NPT. When RTP is used, the rtptime portion of the RTP-Info field may be used to map RTP timestamps to NPT.

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

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

D.2.2 Authentication-enabled

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

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

E Open Issues

1. Should we add the header Accept-Ranges as proposed in this specification?
2. Upon receiving a response on a REDIRECT request can the server close the session or should it wait for a TEARDOWN request from the client?
3. The proxy indications in the two header tables in chapter 13 needs review.
4. Should the Allow header be possible to use optional in request or responses besides the now specified 405 error code?
5. What text should be written on use of authorization in this spec?
6. How does entity tags relate to the If-Match header? The usage in SDP must also be clarified related to syntax, etc.
7. Should the Last-Modified header be required on other level than optional?
8. How to handle range headers for negative scale playback.
9. The minimal implementation must be looked over to see if it complies with the specification. All must and should shall be included in the minimal. Feature-tags for these needs to be defined. Further feature-tags needs to be discussed.
10. The list specifying which status codes are allowed on which request methods seem to be in error and need review.

F Changes

Compared to RFC 2326, the following issues are addressed:

- http://rtsp.org/bug448521 - ”URLs in Rtp-Info need to be quoted”. URLs in RTP-info header now MAY be quoted if needed.
- http://rtsp.org/bug448525 - Syntax for SSRC should be clarified. Require 8*8 HEX and corresponding text added.
- http://rtsp.org/bug461083 - ”Body w/o Content-Length clarification”. This is clarified and any message with a message body is required to have a Content-Length header.
- http://rtsp.org/bug477407 - Transport BNF doesn’t properly deal with semicolon and comma
- http://rtsp.org/bug477413 - Transport BNF: mode parameter issues
- http://rtsp.org/bug477416 - ”BNF error section 3.6 NPT”, Added an optional [NPT] definition. Fixed so that the same possibilities exist for all time formats.
- http://rtsp.org/bug477421 - ”When to send response”. A clarifying note in the status code chapter that when sending 400 responses, the server MUST NOT add cseq if missing.
- http://rtsp.org/bug507347 - Removal of destination redirection in the transport header.
- http://rtsp.org/bug477404 - ”Errors in table in chapter 12”. The table has been updated using the SIP structure. However the table become to big to fit in a single page and has been split.
- http://rtsp.org/bug477419 - Updating HTTP references to rfc2616 by adding public, and content-base header. Section references in header chapter updated. Known effects on RTSP due to HTTP clarifications:
  - Content-Encoding header can include encoding of type ”identity”.
- http://rtsp.org/bug500803 - Rewritten the complete chapter on the state machine.
- http://rtsp.org/bug513753 - Created a IANA section defining four registries.
- http://rtsp.org/bug477427 - A new subsection in the connections chapter clarifying how the server and client may handle transport connections. Includes defining a feature-tag.
- \- Accept-Ranges response header is added. This header clarifies which range formats that can be used for a resource.
- \- Added Headers Timestamp, Via, Unsupported as required for a minimal server implementation.
- http://rtsp.org/bug477425 - ”Inconsistency between timeformats”. Fixed so that all formats has the same capabilities as NPT.
- http://rtsp.org/bug499573 - ”Incorrect grammar on Server header”. Added corrected BNF for User-Agent and Server header as a complement to the reference.

- The definition in the introduction of the RTSP session has been changed.
- Updated RTSP URL's and source and destination parameters in the transport header to handle IPv6 addresses.
- All BNF definitions are updated according to the rules defined in RFC 2234 [14].
- The use of status code 303 "See Other" has been decapitated as it does not make sense to use in RTSP.
- Added status code 350, 351 and updated usage of the other redirect status codes, see chapter 12.3.
- Removed Queued play (http://rtsp.org/bug508211) and decapitated use of PLAY for keep-alive while in playing state.
- Explicitly wrote out the possibilities to use multiple ranges to allow for editing.
- Text specifying the special behavior of PLAY for live content.
- When sending response 451 and 458 the response body should contain the offending parameters.
- Fixed the missing definitions for the Cache-Control header. Also added to the syntax definition the missing delta-seconds for max-stale and min-fresh parameters.
- Added wording on the usage of Connection:Close for RTSP.
- Put requirement on CSeq header that the value is increased by one for each new RTSP request.
- Added requirement that the Date header must be used for all messages with entity. Also the Server should always include it.
- Removed possibility to use Range header combined with Scale header to indicate when it shall be activated, due to that it can't work as defined. Also added rule that lack of scale header in response indicate lack of support. Feature-tags for scaled playback defined.
- The Speed header must now be responded to indicate support and the actual speed going to be used. A feature-tag is defined. Notes on congestion control was also added.
- The Supported header was borrowed from SIP to help with the feature negotiation in RTSP.
- Clarified that the timestamp header can be used to resolve retransmission ambiguities.
- Added two transport header parameters to be used to signal RTCP port for server and client when not assigned in pairs. Shall be used for NAT traversal with mechanisms like STUN. The interoperability issue is solved by requiring a client to know that a server supports this specification.
- Defined a IANA registries for the transport headers parameters, transport-protocol, profile, lower-transport, and mode.
- The OPTIONS method has been clarified on how to use the Public and Allow headers.
- The Session header text has been expanded with a explanation on keep alive and which methods to use.

- http://rtsp.org/bug503949 - Range header format for PAUSE is unclear. This has been resolved by requiring a ranged pause to only contain a single value as a beginning of an open range.
- Servers may optional implement SETUP and TEARDOWN of a single media while in PLAY state. This is signaled using an feature-tag (play.setup).
- The transport headers interleave parameter's text was made more strict and use formal requirements levels. However no change on how it is used was made.
- Added a fragment part to the RTSP URL. This seem to be indicated by the note below the definition however it was not part of the BNF.
- The RECORD and ANNOUNCE methods are removed as they are lacking implementation and not considered necessary in the core specification. Any work on these methods should be done as a extension document to RTSP.
- The description on how rtspu and rtsps is not part of the core specification and will require external description.
- The Transport headers RTP port parameters has been updated to support non-continuous port numbers. Also a possibility for the client to specify SSRC has been added.
- Clarified that RTP-Info URLs that are relative uses the request URL as base URL. Also clarified that the URL that must be used is the SETUP.
- Included two new general address parameters "src_addresses" and "dst_addresses" to be used to give address source and destination of media traffic.
- Updated the text on the transport headers "destination" parameter regarding what security precautions the server shall perform.
- Wrote a new chapter about how to setup different media transport alternatives and their profiles, and lower layer protocols. This resulted that the appendix on RTP interaction was moved there instead in the part describing RTP. The chapter also includes guidelines what to think of when writing usage guidelines for new protocols and profiles.
- The embedded (interleaved) binary data and its transport parameter was clarified to being symmetric and that it is the server that sets the channel numbers.
- Added a new chapter describing the available mechanisms to determine if functionality is supported, called "Capability Handling". Renamed option-tags to feature-tags.
- Added a contributors chapter with people who has contribute actual text to the specification.
- Added text that requires the Range to always be present in PLAY responses. Clarified what should be sent in case of live streams.

Note that this list does not reflect minor changes in wording or correction of typographical errors.
A word-by-word diff from RFC 2326 can be found at http://rtsp.org/2002/drafts

G 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
Cisco
USA
electronic mail: anrao@cisco.com

Robert Lanphier
RealNetworks
P.O. Box 91123
Seattle, WA 98111-9223
USA
electronic mail: robla@real.com

Magnus Westerlund
Ericsson AB, ERA/TVA/A
Torshamsgatan 23
SE-164 80 STOCKHOLM
SWEDEN
electronic mail: magnus.westerlund@ericsson.com

Aravind Narasimhan
Sun Microsystems, Inc.
101 Park Avenue, 3rd & 4th Floor
New York, NY
USA
electronic mail: aravind.narasimhan@sun.com

H Contributors

The following people has made written contribution included in the specification:

- Tom Marshall has contributed with text about the usage of 3rr status codes.
- Thomas Zheng has contributed with text regarding the usage of the Range in PLAY responses.
- Aravind Narasimhan has contributed with updated text regarding the allowed usage of destination.

I Acknowledgements

This draft is based on the functionality of the original RTSP draft submitted in October 1996. 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, Jeff Ayars, Milko Boic, Torsten Braun, Brent Browning, Bruce Butterfield, Steve Casner, Francisco Cortes, Kelly Djahandari, Martin Dunsmuir, Eric Fleischman, Jay Geagan, Andy Grignon, V. Guruprasad, Peter Haight, Mark Handley, Brad Hefta-Gaub, Volker Hilt, John K. Ho, Go Hori, Philipp Hoschka, Anne Jones, Anders Klemets, Ruth Lang, Stephanie Leif, Jonathan Lennox, Eduardo F. Llach, Thomas Marshall, Rob McCool, David Oran, Joerg Ott, Maria Papadopouli, Sujal Patel, Ema Patki, Alagu Periyannan, Colin Perkins, Igor Plotnikov, Jonathan Sergent, Pinaki Shah, David Singer, Lior Sion, Jeff Smith, Alexander Sokolsky, Dale Stammen, John Francis Stracke, and David Walker.

References

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

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

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

[4] S. Bradner, “Key words for use in RFCs to indicate requirement levels,” RFC 2119, Internet Engineering Task Force, Mar. 1997.

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

[6] J. Franks, P. Hallam-Baker, and J. Hostetler, “An extension to HTTP: digest access authentication,” RFC 2069, Internet Engineering Task Force, Jan. 1997.

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

[8] B. 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,” RFC STD 7, 793, Internet Engineering Task Force, Sept. 1981.

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

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

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

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

[14] D. Crocker and P. Overell, "Augmented BNF for syntax specifications: ABNF," RFC 2234, Internet Engineering Task Force, Nov. 1997.

[15] B. Braden, "Requirements for internet hosts - application and support," RFC STD 3, 1123, Internet Engineering Task Force, Oct. 1989.

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

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

[18] F. Yergeau, "UTF-8, a transformation format of ISO 10646," RFC 2279, Internet Engineering Task Force, Jan. 1998.

[19] B. Braden, "T/TCP – TCP extensions for transactions functional specification," RFC 1644, Internet Engineering Task Force, July 1994.

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

[21] H. Schulzrinne, R. Lanphier, and A. Rao, "Real time streaming protocol (RTSP)," RFC 2326, Internet Engineering Task Force, Apr. 1998.

[22] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax," RFC 2396, Internet Engineering Task Force, Aug. 1998.

[23] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," RFC 1889, Internet Engineering Task Force, Jan. 1996.

[24] M. Handley and V. Jacobson, "SDP: session description protocol," RFC 2327, Internet Engineering Task Force, Apr. 1998.

[25] R. Fielding, "Relative uniform resource locators," RFC 1808, Internet Engineering Task Force, June 1995.

[26] R. Fielding, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, Internet Engineering Task Force, June 1999.

[27] T. Dierks, C. Allen, "The TLS Protocol, Version 1.0," RFC 2246, Internet Engineering Task Force, Januari 1999.

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

[29] T. Narten, H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," RFC2434, Internet Engineering Task Force, October 1998.

[30] R. Hinden, B. Carpenter, L. Masinter, "Format for Literal IPv6 Addresses in URL's," RFC 2732, Internet Engineering Task Force, December 1999.

[31] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, "STUN - Simple Traversal of UDP Through Network Address Translators," Internet Engineering Task Force, Work in Progress, October 2002.

[32] P. Srisuresh, K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Internet Engineering Task Force, January 2001.

[33] M. Westerlund, "How to make Real-Time Streaming Protocol (RTSP) traverse Network Address Translators (NAT) and interact with Firewalls.", Internet Engineering Task Force Draft, draft-ietf-mmusic-rtsp-nat-00.txt, Work in Progress, Feb 2003.

[34] A. Narasimhan, A. Narasimhan, "MUTE and UNMUTE extension to RTSP", Internet Engineering Task Force Draft, draft-sergent-rtsp-mute-00.txt, Work in Progress, Feb 2002.

[35] Third Generation Partnership Project (3GPP), "Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs" 3GPP Technical Specification 26.234, Release 5.

[36] D. Yon, "Connection-Oriented Media Transport in SDP", Internet Engineering Task Force Draft, draft-ietf-mmusic-sdp-comedia-04.txt, July 2002.

[37] John Lazzaro, "Framing RTP and RTCP Packets over Connection-Oriented Transport", Internet Engineering Task Force Draft , draft-lazzaro-avt-rtp-framing-contrans-00.txt, January 2003.

IPR Notice

Full Copyright Statement