Real Time Streaming Protocol 2.0 (RTSP)
draft-ietf-mmusic-rfc2326bis-16.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 22, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).
Abstract

This memorandum defines RTSP version 2.0 which is a revision of the Proposed Standard RTSP version 1.0 which is defined in RFC 2326.

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 3550).

Table of Contents

1. Introduction ..................................................... 8
   1.1. Scope and Background ....................................... 8
   1.2. RTSP Specification Update ................................. 9
   1.3. Notational Conventions .................................... 10
   1.4. Terminology ................................................. 10
2. RTSP Introduction ............................................... 15
   2.1. Protocol Properties ........................................ 15
   2.2. Extending RTSP .............................................. 16
   2.3. Overall Operation .......................................... 17
   2.4. RTSP States .................................................. 18
   2.5. Relationship with Other Protocols .......................... 19
3. RTSP Use Cases ................................................... 20
   3.1. On-demand Playback of Stored Content ...................... 20
   3.2. Unicast distribution of Live Content ....................... 21
   3.3. On-demand Playback using Multicast ......................... 22
   3.4. Inviting an RTSP server into a conference ................. 22
   3.5. Live Content using Multicast ............................... 23
4. Protocol Parameters ............................................... 25
   4.1. RTSP Version ................................................ 25
   4.2. RTSP IRI and URI ........................................... 25
   4.3. Session Identifiers ......................................... 27
   4.4. SMPTE Relative Timestamps .................................. 27
   4.5. Normal Play Time ........................................... 27
   4.6. Absolute Time .............................................. 28
   4.7. Feature-tags ................................................ 28
   4.8. Entity Tags ................................................ 29
5. RTSP Message ..................................................... 30
   5.1. Message Types ............................................... 30
   5.2. Message Headers ............................................ 30
   5.3. Message Body ................................................. 30
5.4. Message Length ............................................. 30
6. General Header Fields ........................................ 32
7. Request ...................................................... 33
7.1. Request Line ............................................... 33
7.2. Request Header Fields ..................................... 35
8. Response ...................................................... 37
8.1. Status-Line ................................................ 37
8.1.1. Status Code and Reason Phrase .......................... 37
8.2. Response Header Fields ................................. 40
9. Entity ........................................................ 43
9.1. Entity Header Fields ....................................... 43
9.2. Entity Body ................................................ 44
10. Connections .................................................. 45
10.1. Reliability and Acknowledgements ....................... 45
10.2. Using Connections ......................................... 46
10.3. Closing Connections ...................................... 47
10.4. Timing Out Connections and RTSP Messages ............. 48
10.5. Use of IPv6 ................................................. 48
11. Capability Handling ......................................... 49
12. Pipelining Support .......................................... 51
13. Method Definitions .......................................... 52
13.1. OPTIONS .................................................. 53
13.2. DESCRIBE .................................................. 54
13.3. SETUP .................................................... 56
13.3.1. Changing Transport Parameters ...................... 58
13.4. PLAY ...................................................... 59
13.5. PAUSE ..................................................... 64
13.6. TEARDOWN ................................................ 66
13.7. GET_PARAMETER .......................................... 67
13.8. SET_PARAMETER .......................................... 68
13.9. REDIRECT ................................................ 69
14. Embedded (Interleaved) Binary Data ....................... 73
15. Status Code Definitions .................................... 75
15.1. Success 1xx ............................................... 75
15.1.1. 100 Continue .......................................... 75
15.2. Success 2xx ............................................... 75
15.3. Redirection 3xx ........................................... 75
15.3.1. 300 Multiple Choices .................................. 76
15.3.2. 301 Moved Permanently ............................... 76
15.3.3. 302 Found .............................................. 76
15.3.4. 303 See Other ......................................... 76
15.3.5. 304 Not Modified .................................... 76
15.3.6. 305 Use Proxy ......................................... 77
15.4. Client Error 4xx .......................................... 77
15.4.1. 400 Bad Request ....................................... 77
15.4.2. 405 Method Not Allowed ................................ 77
15.4.3. 451 Parameter Not Understood ....................... 77
15.4.4. 452 reserved ........................................... 77
<table>
<thead>
<tr>
<th>15.4.5.</th>
<th>453 Not Enough Bandwidth</th>
<th>78</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.4.6.</td>
<td>454 Session Not Found</td>
<td>78</td>
</tr>
<tr>
<td>15.4.7.</td>
<td>455 Method Not Valid in This State</td>
<td>78</td>
</tr>
<tr>
<td>15.4.8.</td>
<td>456 Header Field Not Valid for Resource</td>
<td>78</td>
</tr>
<tr>
<td>15.4.9.</td>
<td>457 Invalid Range</td>
<td>78</td>
</tr>
<tr>
<td>15.4.10.</td>
<td>458 Parameter Is Read-Only</td>
<td>78</td>
</tr>
<tr>
<td>15.4.11.</td>
<td>459 Aggregate Operation Not Allowed</td>
<td>78</td>
</tr>
<tr>
<td>15.4.12.</td>
<td>460 Only Aggregate Operation Allowed</td>
<td>78</td>
</tr>
<tr>
<td>15.4.13.</td>
<td>461 Unsupported Transport</td>
<td>79</td>
</tr>
<tr>
<td>15.4.14.</td>
<td>462 Destination Unreachable</td>
<td>79</td>
</tr>
<tr>
<td>15.4.15.</td>
<td>463 Destination Prohibited</td>
<td>79</td>
</tr>
<tr>
<td>15.4.16.</td>
<td>464 Data Transport Not Ready Yet</td>
<td>79</td>
</tr>
<tr>
<td>15.4.17.</td>
<td>470 Connection Authorization Required</td>
<td>79</td>
</tr>
<tr>
<td>15.4.18.</td>
<td>471 Connection Credentials not accepted</td>
<td>79</td>
</tr>
<tr>
<td>15.4.19.</td>
<td>472 Failure to establish secure connection</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15.5.</th>
<th>Server Error 5xx</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.5.1.</td>
<td>551 Option not supported</td>
<td>80</td>
</tr>
</tbody>
</table>

16. Header Field Definitions

<table>
<thead>
<tr>
<th>16.1.</th>
<th>Accept</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.2.</td>
<td>Accept-Credentials</td>
<td>90</td>
</tr>
<tr>
<td>16.3.</td>
<td>Accept-Encoding</td>
<td>91</td>
</tr>
<tr>
<td>16.4.</td>
<td>Accept-Language</td>
<td>91</td>
</tr>
<tr>
<td>16.5.</td>
<td>Accept-Ranges</td>
<td>91</td>
</tr>
<tr>
<td>16.6.</td>
<td>Allow</td>
<td>91</td>
</tr>
<tr>
<td>16.7.</td>
<td>Authorization</td>
<td>92</td>
</tr>
<tr>
<td>16.8.</td>
<td>Bandwidth</td>
<td>92</td>
</tr>
<tr>
<td>16.9.</td>
<td>Blocksize</td>
<td>92</td>
</tr>
<tr>
<td>16.10.</td>
<td>Cache-Control</td>
<td>92</td>
</tr>
<tr>
<td>16.11.</td>
<td>Connection</td>
<td>95</td>
</tr>
<tr>
<td>16.12.</td>
<td>Connection-Credentials</td>
<td>95</td>
</tr>
<tr>
<td>16.13.</td>
<td>Content-Base</td>
<td>96</td>
</tr>
<tr>
<td>16.15.</td>
<td>Content-Language</td>
<td>96</td>
</tr>
<tr>
<td>16.16.</td>
<td>Content-Length</td>
<td>96</td>
</tr>
<tr>
<td>16.17.</td>
<td>Content-Location</td>
<td>97</td>
</tr>
<tr>
<td>16.18.</td>
<td>Content-Type</td>
<td>97</td>
</tr>
<tr>
<td>16.19.</td>
<td>CSeq</td>
<td>97</td>
</tr>
<tr>
<td>16.20.</td>
<td>Date</td>
<td>97</td>
</tr>
<tr>
<td>16.21.</td>
<td>ETag</td>
<td>97</td>
</tr>
<tr>
<td>16.22.</td>
<td>Expires</td>
<td>98</td>
</tr>
<tr>
<td>16.23.</td>
<td>From</td>
<td>99</td>
</tr>
<tr>
<td>16.24.</td>
<td>If-Match</td>
<td>99</td>
</tr>
<tr>
<td>16.25.</td>
<td>If-Modified-Since</td>
<td>99</td>
</tr>
<tr>
<td>16.26.</td>
<td>If-None-Match</td>
<td>100</td>
</tr>
<tr>
<td>16.27.</td>
<td>Last-Modified</td>
<td>100</td>
</tr>
<tr>
<td>16.28.</td>
<td>Location</td>
<td>100</td>
</tr>
<tr>
<td>16.29.</td>
<td>Pipelined-Requests</td>
<td>100</td>
</tr>
<tr>
<td>16.30.</td>
<td>Proxy-Authenticate</td>
<td>101</td>
</tr>
</tbody>
</table>
### 23.1.2. Registering New Feature-tags with IANA

- Registered entries

### 23.2. RTSP Methods

- Description
- Registering New Methods with IANA
- Registered Entries

### 23.3. RTSP Status Codes

- Description
- Registering New Status Codes with IANA
- Registered Entries

### 23.4. RTSP Headers

- Description
- Registering New Headers with IANA
- Registered entries

### 23.5. Transport Header Registries

- Transport Protocol Specification
- Transport modes
- Transport Parameters

### 23.6. Cache Directive Extensions

### 23.7. Accept-Credentials

- Accept-Credentials policies
- Accept-Credentials hash algorithms

### 23.8. Range header formats

### 23.9. URI Schemes

- The rtsp URI Scheme
- The rtspu URI Scheme
- The rtsps URI Scheme

### 23.10. SDP attributes

### 24. References

- Normative References
- Informative References

### Appendix A. RTSP Protocol State Machine

- States
- State variables
- Abbreviations
- State Tables

### Appendix B. Media Transport Alternatives

- RTP
- AVP
- AVP/UDP
- AVPF/UDP
- SAVP/UDP
- SAVPF/UDP
- RTCP usage with RTSP
- RTP over TCP
- Interleaved RTP over TCP
- RTP over independent TCP
- Handling NPT Jumps in the RTP Media Layer
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2.4</td>
<td>Handling RTP Timestamps after PAUSE</td>
</tr>
<tr>
<td>B.2.5</td>
<td>RTSP / RTP Integration</td>
</tr>
<tr>
<td>B.2.6</td>
<td>Scaling with RTP</td>
</tr>
<tr>
<td>B.2.7</td>
<td>Maintaining NPT synchronization with RTP timestamps</td>
</tr>
<tr>
<td>B.2.8</td>
<td>Continuous Audio</td>
</tr>
<tr>
<td>B.2.9</td>
<td>Multiple Sources in an RTP Session</td>
</tr>
<tr>
<td>B.2.10</td>
<td>Usage of SSRCs and the RTCP BYE Message During an RTSP Session</td>
</tr>
<tr>
<td>B.3</td>
<td>Future Additions</td>
</tr>
<tr>
<td>C.1</td>
<td>Definitions</td>
</tr>
<tr>
<td>C.1.1</td>
<td>Control URI</td>
</tr>
<tr>
<td>C.1.2</td>
<td>Media Streams</td>
</tr>
<tr>
<td>C.1.3</td>
<td>Payload Type(s)</td>
</tr>
<tr>
<td>C.1.4</td>
<td>Format-Specific Parameters</td>
</tr>
<tr>
<td>C.1.5</td>
<td>Directionality of media stream</td>
</tr>
<tr>
<td>C.1.6</td>
<td>Range of Presentation</td>
</tr>
<tr>
<td>C.1.7</td>
<td>Time of Availability</td>
</tr>
<tr>
<td>C.1.8</td>
<td>Connection Information</td>
</tr>
<tr>
<td>C.1.9</td>
<td>Entity Tag</td>
</tr>
<tr>
<td>C.2</td>
<td>Aggregate Control Not Available</td>
</tr>
<tr>
<td>C.3</td>
<td>Aggregate Control Available</td>
</tr>
<tr>
<td>C.4</td>
<td>RTSP external SDP delivery</td>
</tr>
<tr>
<td>D.1</td>
<td>Minimal Core Implementation</td>
</tr>
<tr>
<td>D.2</td>
<td>Recommended Core Implementation</td>
</tr>
<tr>
<td>D.3</td>
<td>The Basic Playback Feature Support</td>
</tr>
<tr>
<td>D.3.1</td>
<td>Client</td>
</tr>
<tr>
<td>D.3.2</td>
<td>Server</td>
</tr>
<tr>
<td>D.3.3</td>
<td>Proxy</td>
</tr>
<tr>
<td>D.4</td>
<td>Secure Transport</td>
</tr>
<tr>
<td>D.5</td>
<td>Minimal RTSP Implementation</td>
</tr>
<tr>
<td>D.6</td>
<td>RTSP external SDP delivery</td>
</tr>
<tr>
<td>E.1</td>
<td>Requirements for Unreliable Transport of RTSP</td>
</tr>
<tr>
<td>F.1</td>
<td>Backwards Compatibility Considerations</td>
</tr>
<tr>
<td>F.2</td>
<td>Using Persistent Connections</td>
</tr>
<tr>
<td>G.1</td>
<td>Open Issues</td>
</tr>
<tr>
<td>H.1</td>
<td>Changes needing to be updated</td>
</tr>
<tr>
<td>I.1</td>
<td>Contributors</td>
</tr>
<tr>
<td>J.1</td>
<td>RFC Editor Consideration</td>
</tr>
<tr>
<td>Authors’ Addresses</td>
<td></td>
</tr>
<tr>
<td>Intellectual Property and Copyright Statements</td>
<td></td>
</tr>
</tbody>
</table>
1. Introduction

1.1. Scope and Background

This memo defines version 2.0 of the Real Time Streaming Protocol (RTSP 2.0) which is an application-level protocol for control over the delivery of data with real-time properties, typically streaming media. Streaming media is, for instance, video on demand or audio life streaming. Put simply, RTSP acts as a "network remote control" for multimedia servers, as you know it from your TV set.

The protocol operates between RTSP 2.0 clients and servers, but also supports the usage of RTSP 2.0 proxies between clients and servers. Basically, clients can request information about streaming media from servers, by asking for a description of the media or use media description provided externally. Based on the media description clients can request to play out the media, pause it, or stop it completely, as known from a regular TV remote control. The requested media can consist of multiple audio and video streams that are delivered as a time-synchronized stream from servers to clients.

This memorandum describes the use of RTSP over a reliable connection based transport level protocol, such as TCP. For security, TLS over a connection oriented transport is supported.

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

The set of streams to be controlled in an RTSP session is defined by a presentation description. This memorandum does not define a format for the presentation description. However, Appendix C describes how SDP [RFC4566] is used for this purpose. The streams controlled by RTSP may use RTP [RFC3550] for their data transport, but the operation of RTSP does not depend on the transport mechanism used to carry continuous media. RTSP is intentionally similar in syntax and operation to HTTP/1.1 [RFC2616] so that extension mechanisms to HTTP can in most cases also be applied 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.

* An RTSP server needs to maintain state in almost all cases, as opposed to the stateless nature of HTTP.

* Both an 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 13.2) and interleaving of RTP with RTSP over TCP are exceptions to this rule (see Section 14).

* RTSP is defined to use ISO 10646 (UTF-8) rather than ISO 8859-1, consistent with HTML internationalization efforts [RFC2070].

* The Request-URI always contains the absolute URI. Because of backward compatibility with a historical blunder, HTTP/1.1 [RFC2616] 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 either request a presentation description via RTSP DESCRIBE, 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.

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 example, in distributed teaching applications. Several parties in the conference may take turns "pushing the remote control buttons". Note: This functionality will require RTSP external application level functionality.

RTSP requests may be handled by proxies, tunnels and caches as in HTTP/1.1 [RFC2616].

1.2. RTSP Specification Update

This memorandum specifies RTSP 2.0 which is an update of RTSP 1.0, a proposed standard defined in [RFC2326]. The goal of this version is...
to correct the many flaws that have been identified in RTSP 1.0 since its publication. The corrections are such that backwards compatibility was impossible. Thus a new version was deemed the most appropriate solution to get a more functional protocol. There are no plans to revise RTSP 1.0. Appendix H catalogs the changes of this version in relation to RTSP 1.0.

RTSP 2.0 has reduced functionality compared to RTSP 1.0 and aims at specifying the RTSP core, functionality and rules for extensions, and basic interaction with the media delivery protocol RTP [RFC3550].

Any other functionality would need to be published as extension documents. This specification provides rules for such extensions and defines registries to avoid naming collisions.

1.3. 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 ([RFC2616]).

All the mechanisms specified in this document are described in both prose and the Augmented Backus-Naur form (ABNF) described in detail in [RFC4234].

Indented and smaller-type paragraphs are used to provide informative 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 they are in RTSP.

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 [RFC2119].

The word, "unspecified" is used to indicate functionality or features that are not defined in this specification. Such functionality cannot be used in a standardized manner without further definition in an extension specification to RTSP.

1.4. Terminology

Some of the terminology has been adopted from HTTP/1.1 [RFC2616]. 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. A session which is under aggregate control is referred to as an aggregated session.

Aggregate control URI: The URI used in an 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 13.3 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 constitutes a presentation when played together. The concept of a container file is not embedded in the protocol. However, RTSP servers may offer aggregate control on the media streams within these files.

Continuous media: Data where there is a timing relationship between source and sink; that is, the sink needs to 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 or conversational), 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 meta-information in the form of entity-header fields and content in the form of an entity-body, as described in Section 9.

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

IRI: Internationalized Resource Identifier, is the same as an URI, with the exception that it allows characters from the whole Universal Character Set (Unicode/ISO 10646), rather than the US-ASCII only. See [RFC3987] for more information.
Live: Normally used to describe a presentation or session with media coming from an ongoing event. This generally results in the session having an unbound or only loosely defined duration, and sometimes no seek operations are possible.

Media initialization: Datatype/codec specific initialization. This includes such things as clock rates, 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 host or on a different host from which the presentation is invoked.

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.

Message: The basic unit of RTSP communication, consisting of a structured sequence of octets matching the syntax defined in Section 21 and transmitted over a connection or a connectionless transport.

Non-Aggregated Control: Control of a single media stream. This is 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 and described by a presentation description as defined below. Presentations with more than one media stream are often handled in RTSP under aggregate control.

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 (RFC4566)
use the term "session" for a presentation. The presentation
description may take several different formats, including but not
limited to the session description protocol format, SDP.

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

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

Request-URI: The URI used in a request to indicate the resource on
which the request is to be performed.

RTSP agent: Refers to either an RTSP client, an RTSP server, or an
RTSP Proxy. In this specification, there are many capabilities
that are common to these three entities such as the capability to
send requests or receive responses. This term will be used when
describing functionality that is applicable to all three of these
entities.

RTSP session: A stateful abstraction upon which the main control
methods of RTSP operate. An RTSP session is a server entity; it
is created, maintained and destroyed by the server. It is
established by an RTSP server upon the completion of a successful
SETUP request (when a 200 OK response is sent) and is labelled
with a session identifier at that time. The session exists until
timed out by the server or explicitly removed by a TEARDOWN
request. An RTSP session is a stateful entity; an 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 one or more media streams associated with
it. An 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.

URI: Universal Resource Identifier, see [RFC3986]. The URIs used in
RTSP are generally URLs as they give a location for the resource.
As URLs are a subset of URIs, they will be referred to as URIs to
cover also the cases when an RTSP URI would not be an URL.
URL: Universal Resource Locator, is an URI which identifies the resource through its primary access mechanism, rather than identifying the resource by name or by some other attribute(s) of that resource.
2. RTSP Introduction

2.1. 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, [RFC4346]) or within the protocol itself. All HTTP authentication mechanisms such as basic ([RFC2616]) and digest authentication ([RFC2617]) are directly applicable.

Transport-independent: RTSP does not preclude the use of unreliable datagram protocol (UDP) ([RFC0768]) as it would be possible to implement application-level reliability. The use of a connectionless datagram protocol such as UDP requires additional definition that may be provided as extensions to the core RTSP specification. The reliable stream protocol TCP ([RFC0793]) and the secured reliable stream protocol TLS over TCP [RFC4346] are the currently defined transport protocols for RTSP messages.

Media-delivery protocol independent: The operation of RTSP does not depend on the transport mechanism used to carry continuous media. While most real-time media will use RTP as a transport protocol, RTSP does not preclude the use of other protocols such as MPEG-2 [ISO.13818-1.2000]. The use of other protocols 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 in those cases 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 [RFC3261] or H.323 [ITU.H323.1996] may be used to invite a server to a conference; however, the exact procedures are unspecified.

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 is required to contain at least one RTSP URI.

Proxy and firewall friendly: The protocol should be readily handled by both application and transport-layer (SOCKS [RFC1961]) firewalls. A firewall may need to understand the SETUP method to open a "hole" for the 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 [W3C.REC-PICS-services] [W3C.REC-PICS-labels]) for associating labels with content. However, RTSP does not just add methods to HTTP since controlling continuous media requires server state in most cases.

Appropriate server control: If a client can start a stream, it needs to 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.

2.2. 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.
- Some server may support an RTSP extension.

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 [RFC2616], 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:
o Existing methods can be extended with new parameters, e.g. headers, as long as these parameters can be safely ignored by the recipient. If the client needs negative acknowledgement when a method extension is not supported, a tag corresponding to the extension may be added in the field of the Require or Proxy-Require headers (see Section 16.32).

o New methods can be added. If the recipient of the message does not understand the request, it MUST respond with error code 501 (Not Implemented) so that the sender can avoid using this method again. A client may also use the OPTIONS method to inquire about methods supported by the server. The server MUST list the methods it supports using the Public response header.

o A new version of the protocol can be defined, allowing almost all aspects (except the position of the protocol version number) to change. A new version of the protocol MUST be registered through an IETF standard track document.

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 Section 11.

2.3. Overall Operation

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

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

The presentation description file contains a description of the media streams making up the presentation, including their encodings, language, and other parameters that enable the client to choose the most appropriate combination of media. In this presentation description, each media stream that is individually controllable by RTSP is identified by an RTSP URI, 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 or the requested destination, 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, for example by a SIP created conference.

2.4. RTSP States

RTSP controls a stream which may be sent via a separate protocol, independent of the control channel. For example, RTSP control may be transported on a TCP connection while the media data is conveyed 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, and TEARDOWN.

SETUP: Causes the server to allocate resources for a stream and create an 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 a new server or location

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 16.44) to identify the RTSP session whose state is being manipulated. The server generates session identifiers in response to SETUP requests (Section 13.3).

2.5. 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 will often 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 round trips in web-browser-based scenarios, yet also allows for stand alone 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) [RFC4566] 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.
3. RTSP Use Cases

This section describes the most important and considered use cases for RTSP. They are listed in descending order of importance in regards to ensuring that all necessary functionality is present. This specification only fully supports usage of the two first. Also in these first two cases, there are special cases or exceptions that are not supported without extensions, e.g. the redirection of media to another address than the controlling entity.

3.1. On-demand Playback of Stored Content

An RTSP capable server stores content suitable for being streamed to a client. A client desiring playback of any of the stored content uses RTSP to set up the media transport required to deliver the desired content. RTSP is then used to initiate, halt and manipulate the actual transmission (playout) of the content. RTSP is also required to provide necessary description and synchronization information for the content.

The above high level description can be broken down into a number of functions that RTSP needs to be capable of.

Presentation Description: Provide initialization information about the presentation (content); for example, which media codecs are needed for the content. Other information that is important includes the number of media stream the presentation contains, the transport protocols used for the media streams, and identifiers for these media streams. This information is required before setup of the content is possible and to determine if the client is even capable of using the content.

This information need not be sent using RTSP; other external protocols can be used to transmit the transport presentation descriptions. Two good examples are the use of HTTP [RFC2616] or email to fetch or receive presentation descriptions like SDP [RFC4566]

Setup: Set up some or all of the media streams in a presentation. The setup itself consist of selecting the protocol for media transport and the necessary parameters for the protocol, like addresses and ports.

Control of Transmission: After the necessary media streams have been established the client can request the server to start transmitting the content. The client must be allowed to start or stop the transmission of the content at arbitrary times. The client must also be able to start the transmission at any
point in the timeline of the presentation.

Synchronization: For media transport protocols like RTP [RFC3550] it might be beneficial to carry synchronization information within RTSP. This may be due to either the lack of inter-media synchronization within the protocol itself, or the potential delay before the synchronization is established (which is the case for RTP when using RTCP).

Termination: Terminate the established contexts.

For this use case there are a number of assumptions about how it works. These are:

On-Demand content: The content is stored at the server and can be accessed at any time during a time period when it is intended to be available.

Independent sessions: A server is capable of serving a number of clients simultaneously, including from the same piece of content at different points in that presentation's timeline.

Unicast Transport: Content for each individual client is transmitted to them using unicast traffic.

It is also possible to redirect the media traffic to a different destination than that of the entity controlling the traffic. However, allowing this without appropriate mechanisms for checking that the destination approves of this allows for distributed denial of service attacks (DDoS).

### 3.2. Unicast distribution of Live Content

This use case is similar to the above on-demand content case (see Section 3.1) the difference is the nature of the content itself. Live content is continuously distributed as it becomes available from a source; i.e., the main difference from on-demand is that one starts distributing content before the end of it has become available to the server.

In many cases the consumer of live content is only interested in consuming what is actually happens "now"; i.e., very similar to broadcast TV. However in this case it is assumed that there exist no broadcast or multicast channel to the users, and instead the server functions as a distribution node, sending the same content to multiple receivers, using unicast traffic between server and client. This unicast traffic and the transport parameters are individually negotiated for each receiving client.
Another aspect of live content is that it often has a very limited
time of availability, as it is only is available for the duration of
the event the content covers. An example of such a live content
could be a music concert which lasts 2 hour and starts at a
predetermined time. Thus there is need to announce when and for how
long the live content is available.

In some cases, the server providing live content may be saving some
or all of the content to allow clients to pause the stream and resume
it from the paused point, or to "rewind" and play continuously from a
point earlier than the live point. Hence, this use case does not
necessarily exclude playing from other than the live point of the
stream, playing with scales other than 1.0, etc.

3.3. On-demand Playback using Multicast

It is possible to use RTSP to request that media be delivered to a
multicast group. The entity setting up the session (the controller)
will then control when and what media is delivered to the group.
This use case has some potential for denial of service attacks by
flooding a multicast group. Therefore, a mechanism is needed to
indicate that the group actually accepts the traffic from the RTSP
server.

An open issue in this use case is how one ensures that all receivers
listening to the multicast or broadcast receives the session
presentation configuring the receivers.

3.4. Inviting an RTSP server into a conference

If one has an established conference or group session, it is possible
to have an RTSP server distribute media to the whole group.
Transmission to the group is simplest when controlled by a single
participant or leader of the conference. Shared control might be
possible, but would require further investigation and possibly
extensions.

This use case assumes that there exists either multicast or a
conference focus that redistribute media to all participants.

This use case is intended to be able to handle the following
scenario: A conference leader or participant (hereafter called the
controller) has some pre-stored content on an RTSP server that he
wants to share with the group. The controller sets up an RTSP
session at the streaming server for this content and retrieves the
session description for the content. The destination for the media
content is set to the shared multicast group or conference focus.
When desired by the controller, he/she can start and stop the
transmission of the media to the conference group.

There are several issues with this use case that are not solved by this core specification for RTSP:

Denial of service: To avoid an RTSP server from being an unknowing participant in a denial of service attack the server needs to be able to verify the destination’s acceptance of the media. Such a mechanism to verify the approval of received media does not yet exist; instead, only policies can be used, which can be made to work in controlled environments.

Distributing the presentation description to all participants in the group: To enable a media receiver to correctly decode the content the media configuration information needs to be distributed reliably to all participants. This will most likely require support from an external protocol.

Passing control of the session: If it is desired to pass control of the RTSP session between the participants, some support will be required by an external protocol to exchange state information and possibly floor control of who is controlling the RTSP session.

If there is interest in this use case, further work is required on the necessary extensions.

3.5. Live Content using Multicast

This use case in its simplest form does not require any use of RTSP at all; this is what multicast conferences being announced with SAP and SDP are intended to handle. However in use cases where more advanced features like access control to the multicast session are desired, RTSP could be used for session establishment.

A client desiring to join a live multicasted media session with cryptographic (encryption) access control could use RTSP in the following way. The source of the session announces the session and gives all interested an RTSP URI. The client connects to the server and requests the presentation description, allowing configuration for reception of the media. In this step it is possible for the client to use secured transport and any desired level of authentication; for example, for billing or access control. An RTSP link also allows for load balancing between multiple servers.

If these were the only goals, they could be achieved by simply using HTTP. However, for cases where the sender likes to keep track of each individual receiver of a session, and possibly use the session
as a side channel for distributing key-updates or other information on a per-receiver basis, and the full set of receivers is not know prior to the session start, the state establishment that RTSP provides can be beneficial. In this case a client would establish an RTSP session for this multicast group with the RTSP server. The RTSP server will not transmit any media, but instead will point to the multicast group. The client and server will be able to keep the session alive for as long as the receiver participates in the session thus enabling, for example, the server to push updates to the client.

This use case will most likely not be able to be implemented without some extensions to the server-to-client push mechanism. Here a method like ANNOUNCE (see [RFC2326]) might be suitable; however, it will require a RTSP extension to revive the method.
4. Protocol Parameters

4.1. RTSP Version

HTTP specification section [H3.1] applies, with "HTTP" replaced by "RTSP". This specification defines version 2.0 of RTSP.

4.2. RTSP IRI and URI

RTSP 2.0 defines and registers three URI schemas "rtsp", "rtsps" and "rtspu". The usage of the last, "rtspu", is unspecified in RTSP 2.0, and is defined here to register and reserve the URI scheme that is defined in RTSP 1.0. The "rtspu" scheme indicates undefined transport of the RTSP messages over unreliable transport (UDP). The syntax of "rtsp" and "rtsps" URIs has been changed from RTSP 1.0.

This specification also defines the format of the RTSP IRI [RFC3987] that can be used as RTSP resource identifiers and locators, in web pages, user interfaces, on paper, etc. However, the RTSP request message format only allows usage of the absolute URI format. The RTSP IRI format SHALL use the rules and transformation for IRIs defined in [RFC3987]. This way RTSP 2.0 URIs for request can be produced from an RTSP IRI.

The RTSP IRI and URI are both syntax restricted compared to the generic syntax defined in [RFC3986] and RFC [RFC3987]:

- An absolute URI requires the authority part; i.e., a host identity must be provided.
- Parameters in the path element are prefixed with the reserved separator ";".

The RTSP URI and IRI is case sensitive, with the exception of those parts that [RFC3986] and [RFC3987] defines as case-insensitive; for example, the scheme and host part.

The fragment identifier is used as defined in sections 3.5 and 4.3 of [RFC3986], i.e. the fragment is to be stripped from the URI by the requestor and not included in the request. The user agent also needs to interpret the value of the fragment based on the media type the request relates to; i.e., the media type indicated in Content-Type header in the response to DESCRIBE.

The syntax of any URI query string is unspecified and responder (usually the server) specific. The query is, from the requestor’s perspective, an opaque string and needs to be handled as such.
The URI scheme "rtsp" requires that commands are issued via a
reliable protocol (within the Internet, TCP), while the scheme
"rtsps" identifies a reliable transport using secure transport (TLS
[RFC4346]), see (Section 20).

For the scheme "rtsp", if no port number is provided in the authority
part of the URI port number 554 SHALL be used. For the scheme
"rtsps", the TCP port 322 is registered and SHALL be assumed.

A presentation or a stream is identified by a textual media
identifier, using the character set and escape conventions of URIs
(RFC 3986 [RFC3986]). URIs may refer to a stream or an aggregate of
streams; i.e., a presentation. Accordingly, requests described in
(Section 13) 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 URI:

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

may identify 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 URI:

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

identifies the presentation "twister", which may be composed of audio
and video streams, but could also be something else like a random
media redirector.

This does not imply a standard way to reference streams in URIs.
The presentation description defines the hierarchical
relationships in the presentation and the URIs 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 URI 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 URI.
4.3. Session Identifiers

Session identifiers are strings of any arbitrary length. A session identifier MUST be chosen cryptographically random (See [RFC4086]) and MUST be at least eight characters (can contain a maximum of 48 bits of entropy) long to make guessing it more difficult. It is RECOMMENDED that it contains 128 bits of entropy, i.e. approximately 22 characters from a high quality generator. (See Section 22.) However, it needs to be noted that the session identifier does not provide any security against session hijacking unless it is kept confidential between client, server and trusted proxies.

4.4. 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

\[
\text{hours:minutes:seconds:frames.subframes},
\]

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-type". For SMPTE 30, 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 and the subframe values are zero, they may be omitted. Subframes are measured in one-hundredth of a frame.

Examples:

\[
\begin{align*}
\text{smpte=10:12:33:20-} \\
\text{smpte=10:07:33-} \\
\text{smpte=10:07:00-20:07:33:05.01} \\
\text{smpte-25=10:07:00-10:07:33:05.01}
\end{align*}
\]

4.5. 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) [RFC1305]. 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 (i.e., non-live) content.

NPT is defined as in DSM-CC [ISO.13818-6.1995]: "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."

Examples:

- npt=123.45-125
- npt=12:05:35:3-
- npt=now-

The syntax conforms to ISO 8601 [ISO.8601.2000]. The npt-sec notation is optimized for automatic generation, the npt-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.

4.6. Absolute Time

Absolute time is expressed as ISO 8601 [ISO.8601.2000] timestamps, using UTC (GMT). Fractions of a second may be indicated.

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

19961108T143720.25Z

4.7. Feature-tags

Feature-tags are unique identifiers used to designate features in RTSP. These tags are used in Require (Section 16.38), Proxy-Require (Section 16.32), Proxy-Supported (Section 16.33), Unsupported (Section 16.47), and header fields.

A feature-tag definition MUST indicate which combination of clients, servers or proxies they applies to.

The creator of a new RTSP feature-tag should either prefix the feature-tag with a reverse domain name (e.g., "com.example.mynewfeature" is an apt name for a feature whose
inventor can be reached at "example.com"), or register the new feature-tag with the Internet Assigned Numbers Authority (IANA) (see IANA Section 23).

The usage of feature-tags is further described in Section 11 that deals with capability handling.

4.8. Entity Tags

Entity tags are opaque strings that are used to compare two entities from the same resource, for example in caches or to optimize setup after a redirect. Further explanation is present in [H3.11]. For an explanation of how to compare entity tags see [H13.3]. Entity tags can be carried in the ETag header (see Section 16.21) or in SDP (see Appendix C.1.9).

Entity tags are used in RTSP to make some methods conditional. The methods are made conditional through the inclusion of headers, see Section 16.24 and Section 16.26. Note that RTSP entity tags apply to the complete presentation; i.e., both the session description and the individual media streams. Thus entity tags can be used to verify at setup time after a redirect that the same session description applies to the media at the new location using the If-Match header.
5. RTSP Message

RTSP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 3629 [RFC3629]). Lines SHALL be terminated by CRLF.

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 ISO 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 [RFC3550]. 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 3629 [RFC3629])

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.

5.1. Message Types

See [H4.1].

5.2. Message Headers

See [H4.2].

5.3. Message Body

See [H4.3].

Unlike HTTP, the presence of a message-body in either a request or a response MUST be signaled by the inclusion of a Content-Length header field (see Section 16.16).

5.4. 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 is a line with nothing preceding the CRLF.)

2. If a Content-Length header field (Section 16.16) 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.

Unlike an HTTP message, an RTSP message MUST contain a Content-Length header field whenever it contains a message body. Note that RTSP does not support the HTTP/1.1 "chunked" transfer coding (see [H3.6.1]).

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.
6. General Header Fields

See [H4.5], except that the Pragma, Trailer, Transfer-Encoding, Upgrade, and Warning headers are not defined. RTSP further defines the CSeq, Pipelined-Requests, Proxy-Supported and Timestamp headers. The general headers are listed in Table 1:

<table>
<thead>
<tr>
<th>Header Name</th>
<th>Defined in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-Control</td>
<td>Section 16.10</td>
</tr>
<tr>
<td>Connection</td>
<td>Section 16.11</td>
</tr>
<tr>
<td>CSeq</td>
<td>Section 16.19</td>
</tr>
<tr>
<td>Date</td>
<td>Section 16.20</td>
</tr>
<tr>
<td>Pipelined-Requests</td>
<td>Section 16.29</td>
</tr>
<tr>
<td>Proxy-Supported</td>
<td>Section 16.33</td>
</tr>
<tr>
<td>Supported</td>
<td>Section 16.44</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Section 16.45</td>
</tr>
<tr>
<td>Via</td>
<td>Section 16.50</td>
</tr>
</tbody>
</table>

Table 1: The general headers used in RTSP
7. Request

A request message uses the format outlined below regardless of the
direction of a request, client to server or server to client:

- Request line, containing the method to be applied to the resource,
  the identifier of the resource, and the protocol version in use;

- Zero or more Header lines, that can be of the following types:
  general (Section 6), request (Section 7.2), or entity
  (Section 9.1);

- One empty line (CRLF) to indicate the end of the header section;

- Optionally a message body (entity), consisting of one or more
  lines. The length of the message body in bytes is indicated by
  the Content-Length entity header.

7.1. Request Line

The request line provides the key information about the request: what
method, on what resources and using which RTSP version. The methods
that are defined by this specification are listed in Table 2.
The syntax of the RTSP request line is the following:

\[
\text{<Method> <Request-URI> <RTSP-Version> CRLF}
\]

Note: This syntax cannot be freely changed in future versions of RTSP. This line needs to remain parsable by older RTSP implementations since it indicates the RTSP version of the message.

In contrast to HTTP/1.1 [RFC2616], RTSP requests identify the resource through an absolute RTSP URI (scheme, host, and port) (see Section 4.2) rather than just the absolute path.

HTTP/1.1 requires servers to understand the absolute URI, 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.

An asterisk "*" can be used instead of an absolute URI in the Request-URI part to indicate that the request does not apply to a particular resource, but to the server or proxy itself, and is only allowed when the request method does not necessarily apply to a resource.

For example:
OPTIONS * RTSP/2.0

An OPTIONS in this form will determine the capabilities of the server or the proxy that first receives the request. If the capability of the specific server needs to be determined, without regard to the capability of an intervening proxy, the server should be addressed explicitly with an absolute URI that contains the server’s address.

For example:

OPTIONS rtsp://example.com RTSP/2.0

7.2. Request Header Fields

The RTSP headers in Table 3 can be included in a request, as request headers, to modify the specifics of the request. Some of these headers may also be used in the response to a request, as response headers, to modify the specifics of a response (Section 8.2).

<table>
<thead>
<tr>
<th>Header</th>
<th>Defined in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Section 16.1</td>
</tr>
<tr>
<td>Accept-Credentials</td>
<td>Section 16.2</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>Section 16.3</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>Section 16.4</td>
</tr>
<tr>
<td>Authorization</td>
<td>Section 16.7</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Section 16.8</td>
</tr>
<tr>
<td>Blocksize</td>
<td>Section 16.9</td>
</tr>
<tr>
<td>From</td>
<td>Section 16.23</td>
</tr>
<tr>
<td>If-Match</td>
<td>Section 16.24</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>Section 16.25</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>Section 16.26</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>Section 16.32</td>
</tr>
<tr>
<td>Range</td>
<td>Section 16.35</td>
</tr>
</tbody>
</table>
Table 3: The RTSP request headers

Detailed headers definition are provided in Section 16.

New request headers may be defined. If the receiver of the request is required to understand the request header, the request MUST include a corresponding feature tag in a Require or Proxy-Require header to ensure the correct processing of the header.
8. Response

[H6] applies except that HTTP-Version is replaced by RTSP-Version. Also, RTSP defines additional status codes and does not define some of the HTTP codes. The valid response codes and the methods they can be used with are listed in Table 4.

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

8.1. 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.

<RTSP-Version> SP <Status-Code> SP <Reason-Phrase> CRLF

8.1.1. 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 15. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

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

1xx: Informational - Request received, continuing process

2xx: Success - The action was successfully received, understood, and accepted

3xx: Redirection - Further action needs to 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/2.0, and an example set of corresponding Reason-Phrases, are presented in Table 4. 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 [RFC2616] status codes and adds RTSP-specific status codes starting at x50 to avoid conflicts with newly defined HTTP status codes.

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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Reason</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Continue</td>
<td>all</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
<td>all</td>
</tr>
<tr>
<td>300</td>
<td>Multiple Choices</td>
<td>all</td>
</tr>
<tr>
<td>301</td>
<td>Multiple Choices</td>
<td>all</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
<td>all</td>
</tr>
<tr>
<td>302</td>
<td>Found</td>
<td>all</td>
</tr>
<tr>
<td>303</td>
<td>See Other</td>
<td>all</td>
</tr>
<tr>
<td>305</td>
<td>Use Proxy</td>
<td>all</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>all</td>
</tr>
<tr>
<td>Status Code</td>
<td>Description</td>
<td>Reservation</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>all</td>
</tr>
<tr>
<td>402</td>
<td>Payment Required</td>
<td>all</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>all</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>all</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
<td>all</td>
</tr>
<tr>
<td>406</td>
<td>Not Acceptable</td>
<td>all</td>
</tr>
<tr>
<td>407</td>
<td>Proxy Authentication Required</td>
<td>all</td>
</tr>
<tr>
<td>408</td>
<td>Request Timeout</td>
<td>all</td>
</tr>
<tr>
<td>409</td>
<td>Gone</td>
<td>all</td>
</tr>
<tr>
<td>410</td>
<td>Length Required</td>
<td>all</td>
</tr>
<tr>
<td>411</td>
<td>Precondition Failed</td>
<td>DESCRIBE, SETUP</td>
</tr>
<tr>
<td>412</td>
<td>Request Entity Too Large</td>
<td>all</td>
</tr>
<tr>
<td>413</td>
<td>Request-URI Too Long</td>
<td>all</td>
</tr>
<tr>
<td>414</td>
<td>Unsupported Media Type</td>
<td>all</td>
</tr>
<tr>
<td>415</td>
<td>Parameter Not Understood</td>
<td>SET_PARAMETER</td>
</tr>
<tr>
<td>416</td>
<td>reserved</td>
<td>n/a</td>
</tr>
<tr>
<td>417</td>
<td>Not Enough Bandwidth</td>
<td>SETUP</td>
</tr>
<tr>
<td>418</td>
<td>Session Not Found</td>
<td>all</td>
</tr>
<tr>
<td>419</td>
<td>Method Not Valid In This State</td>
<td>all</td>
</tr>
<tr>
<td>420</td>
<td>Header Field Not Valid</td>
<td>all</td>
</tr>
<tr>
<td>421</td>
<td>Invalid Range</td>
<td>PLAY, PAUSE</td>
</tr>
<tr>
<td>422</td>
<td>Parameter Is Read-Only</td>
<td>SET_PARAMETER</td>
</tr>
<tr>
<td>423</td>
<td>Aggregate Operation Not Allowed</td>
<td>all</td>
</tr>
<tr>
<td>424</td>
<td>Only Aggregate Operation Allowed</td>
<td>all</td>
</tr>
<tr>
<td>425</td>
<td></td>
<td></td>
</tr>
<tr>
<td>426</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Method</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>461</td>
<td>Unsupported Transport</td>
<td>all</td>
</tr>
<tr>
<td>462</td>
<td>Destination Unreachable</td>
<td>all</td>
</tr>
<tr>
<td>463</td>
<td>Destination Prohibited</td>
<td>SETUP</td>
</tr>
<tr>
<td>464</td>
<td>Data Transport Not Ready Yet</td>
<td>PLAY</td>
</tr>
<tr>
<td>470</td>
<td>Connection Authorization Required</td>
<td>all</td>
</tr>
<tr>
<td>471</td>
<td>Connection Credentials not accepted</td>
<td>all</td>
</tr>
<tr>
<td>472</td>
<td>Failure to establish secure connection</td>
<td>all</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
<td>all</td>
</tr>
<tr>
<td>501</td>
<td>Not Implemented</td>
<td>all</td>
</tr>
<tr>
<td>502</td>
<td>Bad Gateway</td>
<td>all</td>
</tr>
<tr>
<td>503</td>
<td>Service Unavailable</td>
<td>all</td>
</tr>
<tr>
<td>504</td>
<td>Gateway Timeout</td>
<td>all</td>
</tr>
<tr>
<td>505</td>
<td>RTSP Version Not Supported</td>
<td>all</td>
</tr>
<tr>
<td>551</td>
<td>Option not supported</td>
<td>all</td>
</tr>
</tbody>
</table>

Table 4: Status codes and their usage with RTSP methods

8.2. 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. All headers currently classified as response headers are listed in Table 5.
<table>
<thead>
<tr>
<th>Header</th>
<th>Defined in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept-Credentials</td>
<td>Section 16.2</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>Section 16.5</td>
</tr>
<tr>
<td>Connection-Credentials</td>
<td>Section 16.12</td>
</tr>
<tr>
<td>ETag</td>
<td>Section 16.21</td>
</tr>
<tr>
<td>Location</td>
<td>Section 16.28</td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>Section 16.30</td>
</tr>
<tr>
<td>Public</td>
<td>Section 16.34</td>
</tr>
<tr>
<td>Range</td>
<td>Section 16.35</td>
</tr>
<tr>
<td>Retry-After</td>
<td>Section 16.37</td>
</tr>
<tr>
<td>RTP-Info</td>
<td>Section 16.39</td>
</tr>
<tr>
<td>Scale</td>
<td>Section 16.40</td>
</tr>
<tr>
<td>Session</td>
<td>Section 16.43</td>
</tr>
<tr>
<td>Server</td>
<td>Section 16.42</td>
</tr>
<tr>
<td>Speed</td>
<td>Section 16.41</td>
</tr>
<tr>
<td>Transport</td>
<td>Section 16.46</td>
</tr>
<tr>
<td>Unsupported</td>
<td>Section 16.47</td>
</tr>
<tr>
<td>Vary</td>
<td>Section 16.49</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Section 16.51</td>
</tr>
</tbody>
</table>

Table 5: The RTSP response headers

Response-header field names can be extended reliably only in combination with a change in the protocol version. However, the usage of feature-tags in the request allows the responding party to learn the capability of the receiver of the response. New or experimental header fields MAY be given the semantics of response-header fields if all parties in the communication recognize them to be response-header...
fields. Unrecognized header fields in responses are treated as entity-header fields.
9. 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.

The SETPARAMETER and GET_PARAMETER request and response, and DESCRIBE response MAY have an entity. All 4xx and 5xx responses MAY also have an entity.

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

9.1. Entity Header Fields

Entity-header fields define meta-information about the entity-body or, if no body is present, about the resource identified by the request. The entity header fields are listed in Table 6.

+-----------------+--------------------+
| Header          | Defined in Section |
+-----------------+--------------------+
| Allow           | Section 16.6       |
| Content-Base    | Section 16.13      |
| Content-Encoding| Section 16.14      |
| Content-Language| Section 16.15      |
| Content-Length  | Section 16.16      |
| Content-Location| Section 16.17      |
| Content-Type    | Section 16.18      |
| Expires         | Section 16.22      |
| Last-Modified   | Section 16.27      |
+-----------------+--------------------+

Table 6: The RTSP entity headers

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.
9.2. Entity Body

See [H7.2] with the addition that an RTSP message with an entity body MUST include the Content-Type and Content-Length headers.
10. Connections

RTSP requests can be transmitted using the two different connection scenarios listed below:

- **persistent** - a transport connection is used for several request/response transactions;
- **transient** - a transport connection is used for a single request/response transaction.

RFC 2326 attempted to specify an optional mechanism for transmitting RTSP messages in connectionless mode over a transport protocol such as UDP. However, it was not specified in sufficient detail to allow for interoperable implementations. In an attempt to reduce complexity and scope, and due to lack of interest, RTSP 2.0 does not attempt to define a mechanism for supporting RTSP over UDP or other connectionless transport protocols. A side-effect of this is that RTSP requests SHALL NOT be sent to multicast groups since no connection can be established with a specific receiver in multicast environments.

Certain RTSP headers, such as the CSeq header Section 16.19), which may appear to be relevant only to connectionless transport scenarios are still retained and must be implemented according to the specification. In the case of CSeq, it is quite useful for matching responses to requests if the requests are pipelined (see Section 12). It is also useful in proxies for keeping track of the different requests when aggregating several client requests on a single TCP connection.

10.1. Reliability and Acknowledgements

When RTSP messages are transmitted using reliable transport protocols, they MUST NOT be retransmitted at the RTSP protocol level. Instead, the implementation must rely on the underlying transport to provide reliability. The RTSP implementation may use any indication of reception acknowledgement of the message from the underlying transport protocols to optimize the RTSP behavior.

If both the underlying reliable transport such as TCP and the RTSP application retransmit requests, each packet loss or message loss may result in two retransmissions. The receiver typically cannot 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.

Lack of acknowledgement of an RTSP request should be handled within the constraints of the connection timeout considerations described below (Section 10.4).

10.2. Using Connections

A TCP transport can be used for both persistent connections (for several message exchanges) and transient connections (for a single message exchange). Implementations of this specification MUST support RTSP over TCP. The scheme of the RTSP URI (Section 4.2) indicates the default port that the server will listen on.

A server MUST handle both persistent and transient connections.

Transient connections facilitate mechanisms for fault tolerance. They also allow for application layer mobility. A server and client pair that support transient connections can survive the loss of a TCP connection; e.g., due to a NAT timeout. When the client has discovered that the TCP connection has been lost, it can set up a new one when there is need to communicate again.

A persistent connection MAY be used for all transactions between the server and client, including messages for multiple RTSP sessions. However a persistent connection MAY also be closed after a few message exchanges. For example, a client may use a persistent connection for the initial SETUP and PLAY message exchanges in a session and then close the connection. Later, when the client wishes to send a new request, such as a PAUSE for the session, a new connection would be opened. This connection may either be transient or persistent.

An RTSP agent 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 SHOULD use only one connection for managing those sessions.

This saves connection resources on the server. It also reduces complexity by and enabling the server to maintain less state about its sessions and connections.

Unlike HTTP, RTSP allows a server to send requests to a client. However, this can be supported only if a client establishes a persistent connection with the server. In cases where a persistent connection does not exist between a server and its client, due to the lack of a signalling channel the server may be forced to drop an RTSP session without notifying the client. An example of such a case is
when the server desires to send a REDIRECT request for an RTSP session to the client but is not able to do so because it cannot reach the client.

Without a persistent connection between the client and the server, the media server has no reliable way of reaching the client. Also, this is the only way that requests from a server to its client are likely to traverse firewalls.

In light of the above, it is RECOMMENDED that clients use persistent connections whenever possible. A client that supports persistent connections MAY "pipeline" its requests (see Section 12).

10.3. Closing Connections

The client MAY close a connection at any point when no outstanding request/response transactions exist for any RTSP session being managed through the connection. The server, however, SHOULD NOT close a connection until all RTSP sessions being managed through the connection have been timed out (Section 16.43). A server SHOULD NOT close a connection immediately after responding to a session-level TEARDOWN request for the last RTSP session being controlled through the connection. Instead, it should wait for a reasonable amount of time for the client to receive the TEARDOWN response, take appropriate action, and initiate the connection closing. The server SHOULD wait at least 10 seconds after sending the TEARDOWN response before closing the connection.

This is to ensure that the client has time to issue a SETUP for a new session on the existing connection after having torn the last one down. 10 seconds should give the client ample opportunity to get its message to the server.

A server SHOULD NOT close the connection directly as a result of responding to a request with an error code.

Certain error responses such as "460 Only Aggregate Operation Allowed" (Section 15.4.12) are used for negotiating capabilities of a server with respect to content or other factors. In such cases, it is inefficient for the server to close a connection on an error response. Also, such behavior would prevent implementation of advanced/special types of requests or result in extra overhead for the client when testing for new features. On the flip side, keeping connections open after sending an error response poses a Denial of Service security risk (Section 22).

If a server initiates a connection close while the client is attempting to send a new request, the client will have to close its
current connection, establish a new connection and send its request over the new connection.

An RTSP message should not be terminated by closing the connection. Such a message MAY be considered to be incomplete by the receiver and discarded. An RTSP message is properly terminated as defined in Section 5.

10.4. Timing Out Connections and RTSP Messages

Receivers of a request (responder) SHOULD respond to requests in a timely manner even when a reliable transport such as TCP is used. Similarly, the sender of a request (requestor) SHOULD wait for a sufficient time for a response before concluding that the responder will not be acting upon its request.

A responder SHOULD respond to all requests within 5 seconds. If the responder recognizes that processing of a request will take longer than 5 seconds, it SHOULD send a 100 (Continue) response as soon as possible. It SHOULD continue sending a 100 response every 5 seconds thereafter until it is ready to send the final response to the requestor. After sending a 100 response, the receiver MUST send a final response indicating the success or failure of the request.

A requestor SHOULD wait at least 10 seconds for a response before concluding that the responder will not be responding to its request. After receiving a 100 response, the requestor SHOULD continue waiting for further responses. If more than 10 seconds elapses without receiving any response, the requestor MAY assume that the responder is unresponsive and abort the connection.

A requestor SHOULD wait longer than 10 seconds for a response if it is experiencing significant transport delays on its connection to the responder. The requestor is capable of determining the RTT of the request/response cycle using the Timestamp header (Section 16.45) in any RTSP request.

10.5. Use of IPv6

Explicit IPv6 support was not present in RTSP 1.0 (RFC 2326). RTSP 2.0 has been updated for explicit IPv6 support. Implementations of RTSP 2.0 MUST understand literal IPv6 addresses in URIs and headers.
11. Capability Handling

This section describes the capability handling mechanism available in RTSP which allows RTSP to be extended. Extensions to 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 both cases.

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

It is not necessary to use OPTIONS to discover support of a method, the client could simply try the method. If the receiver of the request does not support the method it will respond with an error code indicating the the method is 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.

Feature-Tags are defined to handle functionality additions that are not new methods. Each feature-tag represents a certain block of functionality. The amount of functionality that a feature-tag represents can vary significantly. A feature-tag can for example represent the functionality a single RTSP header provides. Another feature-tag can represent much more functionality, such as the "play.basic" feature-tag which represents the minimal playback implementation.

Feature-tags are used to determine whether 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 is used to determine the complete set of functionality that both client and server have. The intended usage is to determine before one needs to use a functionality that it is supported. It can be used in any method, however OPTIONS is the most suitable one as it 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. This results in that the receiver learns the requestor’s feature support. The receiver then includes its set of features in the response.
Proxy-Supported: The Proxy-Supported header is used similar to the Supported header, but instead of giving the supported functionality of the client or server it provides both the requestor and the responder a view of what functionality the proxy chain between the two supports. Proxies are required to add this header whenever the Supported header is present, but proxies may independently of the requestor add it.

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 is required to 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 supported are 551 (Option Not Supported). Also the features that are not supported 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 a 551 error response, to indicate 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 are not supported.
12. Pipelining Support

Pipelining is a general method to improve performance of request response protocols by allowing the requesting entity to have more than one request outstanding and send them over the same persistent connection. For RTSP where the relative order of requests will matter it is important to maintain the order of the requests. Because of this the responding entity SHALL process the incoming requests in their sending order. The sending order can be determined by the CSeq header and its sequence number. For TCP the delivery order will be the same as the sending order. The processing of the request SHALL also have been finished before processing the next request from the same entity. The responses MUST be sent in the order the requests was processed.

RTSP 2.0 has extended support for pipelining compared to RTSP 1.0. The major improvement is to allow all requests to setup and initiate media playback to be pipelined after each other. This is accomplished by the utilization of the Pipelined-Requests header (See Section 16.29). This header allows a client to request that two or more requests is to be processed in the same RTSP session context which the first request creates. In other words a client can request that two or more media streams are set-up and then played without needing to wait for a single response. This speeds up the initial startup time for an RTSP session with at least one RTT.

When pipelining requests care must be taken. If a pipelined request builds on the successful completion of one or more prior requests the requestor must verify that all requests was executed as expected. A common example will be two SETUP requests and a PLAY request. In case one of the SETUP fails unexpectedly, the PLAY request can still be successfully executed. However, not as expected by the requesting client as only a single media instead of two will be played. In this case the client can send a PAUSE request, correct the failing SETUP request and then request it to be played.
13. Method Definitions

The method indicates what is to be performed on the resource identified by the Request-URI. The method name is case-sensitive. New methods may be defined in the future. Method names SHALL NOT start with a $ character (decimal 24) and MUST be a token as defined by the ABNF [RFC4234] in the syntax chapter Section 21. The methods are summarized in Table 7.

<table>
<thead>
<tr>
<th>method</th>
<th>direction</th>
<th>object</th>
<th>Server req.</th>
<th>Client req.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>recommended</td>
<td>recommended</td>
</tr>
<tr>
<td>GET_PARAMETER</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td></td>
<td>S -&gt; C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTIONS</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>R=Req,</td>
<td>Sd=Req, R=Opt</td>
</tr>
<tr>
<td></td>
<td>S -&gt; C</td>
<td></td>
<td>Sd=Opt</td>
<td></td>
</tr>
<tr>
<td>PAUSE</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>PLAY</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>REDIRECT</td>
<td>S -&gt; C</td>
<td>P,S</td>
<td>optional</td>
<td>required</td>
</tr>
<tr>
<td>SETUP</td>
<td>C -&gt; S</td>
<td>S</td>
<td>required</td>
<td>required</td>
</tr>
<tr>
<td>SETPARAMETER</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>required</td>
<td>optional</td>
</tr>
<tr>
<td></td>
<td>S -&gt; C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>required</td>
<td>required</td>
</tr>
</tbody>
</table>

Table 7: Overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on. Legend: R=Respond, Sd=Send, Opt: Optional, Req: Required

Note on Table 7: GET_PARAMETER is recommended, but not required. For example, a fully functional server can be built to deliver media without any parameters. SETPARAMETER is required however due to its usage for keep-alive. PAUSE is now required due to that it is the only way of getting out of the state machines play state without terminating the whole session.
If an RTSP agent does not support a particular method, it MUST return 501 (Not Implemented) and the requesting RTSP agent, in turn, SHOULD NOT try this method again for the given agent / resource combination.

13.1. OPTIONS

The semantics of the RTSP OPTIONS method is equivalent to that of the HTTP OPTIONS method described in [H9.2]. In RTSP however, OPTIONS is bi-directional, in that a client can request it to a server and vice versa. A client MUST implement the capability to send an OPTIONS request and a server or a proxy MUST implement the capability to respond to an OPTIONS request. The client, server or proxy MAY also implement the converse of their required capability.

An OPTIONS request may be issued at any time. Such a request does not modify the session state. However, it may prolong the session lifespan (see below). The URI in an OPTIONS request determines the scope of the request and the corresponding response. If the Request-URI refers to a specific media resource on a given host, the scope is limited to the set of methods supported for that media resource by the indicated RTSP agent. A Request-URI with only the host address limits the scope to the specified RTSP agent’s general capabilities without regard to any specific media. If the Request-URI is an asterisk (“*”), the scope is limited to the general capabilities of the next hop (i.e. the RTSP agent in direct communication with the request sender).

Regardless of scope of the request, the Public header MUST always be included in the OPTIONS response listing the methods that are supported by the responding RTSP agent. In addition, if the scope of the request is limited to a media resource, the Allow header MUST be included in the response to enumerate the set of methods that are allowed for that resource unless the set of methods completely matches the set in the Public header. If the given resource is not available, the RTSP agent SHOULD return an appropriate response code such as 3rr or 4xx. The Supported header MAY be included in the request to query the set of features that are supported by the responding RTSP agent.

The OPTIONS method can be used to keep an RTSP session alive. However, it is not the preferred means of session keep-alive signalling, see Section 16.43. An OPTIONS request intended for keeping alive an RTSP session MUST include the Session header with the associated session ID. Such a request SHOULD also use the media or the aggregated control URI as the Request-URI.

Example:
13.2. DESCRIBE

The DESCRIBE method is used to retrieve the description of a presentation or media object from a server. The Request-URI of the DESCRIBE request identifies the media resource of interest. The client MAY include the Accept header in the request to list the description formats that it understands. The server SHALL respond with a description of the requested resource and return the description in the entity of the response. The DESCRIBE reply-response pair constitutes the media initialization phase of RTSP.

Example:
The DESCRIBE response SHOULD contain all media initialization information for the resource(s) that it describes. Servers SHOULD NOT use the DESCRIBE response as a means of media indirection by having the description point at another server, instead usage of 3rr responses are recommended.

By forcing a DESCRIBE response to contain all media initialization for the set of streams that it describes, and discouraging the use of DESCRIBE for media indirection, any looping problems can be avoided that might have resulted from other approaches.

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

- via an RTSP DESCRIBE request
- via some other protocol (HTTP, email attachment, etc.)
- via some form of a user interface
If a client obtains a valid description from an alternate source, the client MAY use this description for initialization purposes without issuing a DESCRIBE request for the same media.

It is RECOMMENDED that minimal servers support the DESCRIBE method, and highly recommended that minimal clients support the ability to act as "helper applications" that accept a media initialization file from a user interface, and/or other means that are appropriate to the operating environment of the clients.

13.3. SETUP

The SETUP request for an URI specifies the transport mechanism to be used for the streamed media. The SETUP method may be used in three different cases; Create an RTSP session, add a media to a session, and change the transport parameters of already set up media stream. When in PLAY state, using SETUP to create or add media to a session when in PLAY state is unspecified. Otherwise SETUP can be used in all three states; INIT, and READY, for both purposes and in PLAY to change the transport parameters.

The Transport header, see Section 16.46, specifies the transport parameters acceptable to the client for data transmission; the response will contain the transport parameters selected by the server. This allows the client to enumerate in descending order of preference the transport mechanisms and parameters acceptable to it, while the server can select the most appropriate. It is expected that the session description format used will enable the client to select a limited number possible configurations that are offered to the server to choose from. All transport related parameters shall be included in the Transport header, the use of other headers for this purpose is discouraged due to middle boxes such as firewalls, or NATs.

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

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 client SHALL include the Accept-Ranges header in the request indicating all supported unit formats in the Range header. This allows the server to know which format it may use in future session
related responses, such as PLAY response without any range in the request. If the client does not support a time format necessary for the presentation the server SHALL respond using 456 (Header Field Not Valid for Resource) and include the Accept-Ranges header with the range unit formats supported for the resource.

In a SETUP response the server SHALL include the Accept-Ranges header (see Section 16.5) to indicate which time formats that are acceptable to use for this media resource.

C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/2.0
   CSeq: 302
   Transport: RTP/AVP;unicast;dest_addr=":4588"/":4589",
   RTP/AVP/TCP;unicast;interleaved=0-1
   Accept-Ranges: NPT, UTC
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 302
   Date: 23 Jan 1997 15:35:06 GMT
   Server: PhonyServer 1.1
   Session: 47112344;timeout=60
   Transport: RTP/AVP;unicast;dest_addr=":4588"/":4589",
   src_addr="192.0.2.241:6256"/"192.0.2.241:6257"
   ssrc=2A3F93ED
   Accept-Ranges: NPT

In the above example the client wants to create an RTSP session containing the media resource "rtsp://example.com/foo/bar/baz.rm". The transport parameters acceptable to the client is either RTP/AVP/UDP (UDP per default) to be received on client port 4588 and 4589 or RTP/AVP interleaved on the RTSP control channel. The server selects the RTP/AVP/UDP transport and adds the ports it will send and received RTP and RTCP from, and the RTP SSRC that will be used by the server.

The server MUST generate a session identifier in response to a successful SETUP request, unless a SETUP request to a server includes a session identifier, in which case the server MUST bundle this setup request into the existing session (aggregated session) or return error 459 (Aggregate Operation Not Allowed) (see Section 15.4.11). 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 is to be specified by the session description if the server supports aggregated control and aggregated control is desired for the session. However even if aggregated control is offered the client MAY chose to not set up the session in aggregated
control. If an Aggregate control URI is not specified in the session description, it is normally an indication that non-aggregated control should be used. The SETUP of media streams in an aggregate which has not been given an aggregated control URI is unspecified.

While the session ID sometimes has enough information for aggregate control of a session, the Aggregate control URI is still important for some methods such as SETPARAMETER 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.

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(s) 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 Section 16.43. Signs of liveness for an RTSP session are:

- Any RTSP request from a client(s) 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(s) for any of the media streams in that RTSP session. RTCP Sender Reports may for example be received in sessions where the server is invited into a conference session and is as valid for keep-alive.

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.

13.3.1. Changing Transport Parameters

A client MAY 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 changing of parameters, it MUST respond with error 455 (Method Not Valid In This State). Reasons to support changing transport parameters, is to allow for application layer mobility and flexibility to utilize the best available transport as it becomes available. If a client receives a 455 when trying to change transport parameters while the server is in play state, it MAY try to put the server in ready state using PAUSE,
before issuing the SETUP request again. If also that fails the changing of transport parameters will require that the client performs a TEARDOWN of the affected media and then setting it up again. In aggregated session avoiding tearing down all the media at the same time will avoid the creation of a new session.

All transport parameters MAY be changed. However the primary usage expected is to either change transport protocol completely, like switching from Interleaved TCP mode to UDP or vice versa or change delivery address.

In a SETUP response for a request to change the transport parameters while in PLAY state, the server SHALL include the Range to indicate from what point the new transport parameters are used. Further, if RTP is used for delivery, the server SHALL also include the RTP-Info header to indicate from what timestamp and RTP sequence number the change has taken place. If both RTP-Info and Range is included in the response the "rtp_time" parameter and range MUST be for the corresponding time, i.e. be used in the same way as for PLAY to ensure the correct synchronization information is available.

If the transport parameters change while in PLAY state results in a change of synchronization related information, for example changing RTP SSRC, the server MUST provide in the SETUP response the necessary synchronization information. However the server is RECOMMENDED to avoid changing the synchronization information if possible.

13.4. PLAY

The PLAY method tells the server to start sending data via the mechanism specified in SETUP. PLAY requests are valid when the session is in READY or PLAY states. A PLAY request MUST include a Session header to indicate which session the request applies to.

For aggregated sessions where the initial SETUP request (creating a session) is followed by one or more additional SETUP request, a PLAY request MAY be pipelined after those additional SETUP requests without awaiting their responses. This can procedure can reduce the delay from start of session establishment until media play-out has started with one round trip time. However an client needs to be aware that using this procedure will result in the playout of the server state established at the time of processing the PLAY, i.e. after the processing of all the requests prior to the PLAY request in the pipeline. This may not be the intended one due to failure of any of the prior requests. However a client easily determine this based on the responses from those requests. In case of failure the client can halt the media playout using PAUSE and try to establish the intended state again before issuing another PLAY request.
In an aggregated session the PLAY request MUST contain an aggregated control URI. A server SHALL respond 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 needs to use separate RTSP sessions for each media.

Upon receipt of the PLAY request, the server SHALL position the normal play time to the beginning of the range specified in the received Range header and delivers stream data until the end of the range if given, or until a new PLAY request is received, else to the end of the media is reached. To allow for precise composition multiple ranges MAY be specified in one PLAY Request. The range values are valid if all given ranges are part of any media within the aggregate. 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/2.0
CSeq: 835
Session: 12345678
Range: npt=10-15, npt=20-25, npt=30-
User-Agent: PhonyClient/1.2

See the description of the PAUSE request for further examples.

A PLAY request without a Range header is legal. It SHALL start playing a stream from the beginning (npt=0-) unless the stream has been paused or is currently playing. If a stream has been paused via PAUSE, stream delivery resumes at the pause point. If a stream is currently playing, the new PLAY begins at the current stream position. The stream SHALL play until the end of the media.

The Range header MUST NOT contain a time parameter. The usage of time in PLAY method has been deprecated. If a request with time parameter is received the server SHOULD respond with a 457 (Invalid Range) to indicate that the time parameter is not supported.

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 an other format more appropriate. Also for a session with live media streams the Range header MUST indicate a valid time. It is RECOMMENDED that normal play time is used, either the "now" indicator, for example
"npt=now-", or the time since session start as an open interval, e.g. 
"npt=96.23-". An absolute time value (clock) for the corresponding 
time MAY be given, i.e. "clock=20030213T143205Z-". The UTC clock 
format SHOULD only be used if client has shown support for it.

For an on-demand stream, the server MUST reply with the actual range 
that will be played back, i.e. for which duration any media (having 
content at this time) is delivered. This may differ from the 
requested range if alignment of the requested range to valid frame 
boundaries is required for the media source. Note that some media 
streams in an aggregate may need to be delivered from even earlier 
points. Also, some media format have a very long duration per 
individual data unit, therefore it might be necessary for the client 
to parse the data unit, and select where to start.

Example: Single audio stream (MIDI)

C->S: PLAY rtsp://example.com/audio RTSP/2.0
CSeq: 836
Session: 12345678
Range: npt=7.05-
User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
CSeq: 836
Date: 23 Jan 1997 15:35:06 GMT
Server: PhonyServer 1.0
Range: npt=3.52-
RTP-Info:url="rtsp://example.com/audio"
ssrc=0D12F123:seq=14783;rtptime=2345962545

S->C: RTP Packet TS=2345962545 => NPT=3.52
Duration=4.15 seconds

In this example the client receives the first media packet that 
stretches all the way up and past the requested playtime. Thus, it 
is the client’s decision if to render to the user the time between 
3.52 and 7.05, or to skip it. In most cases it is probably most 
suitable to not render that time period.

For live media sources it might be impossible to specify from which 
point in time all media streams carrying active content can actually 
be delivered. Therefore a server MAY specify a start time (or now-) 
in the range header, for which not all media will be available from.

If no range is specified in the request, the start position SHALL 
still be returned in the reply. If the medias that are part of an 
aggregate has different lengths, the PLAY request SHALL be performed
as long as the given range is valid for any media, for example the
longest media. Media will be sent whenever it is available for the
given play-out point.

A PLAY response MAY include a header(s) carrying synchronization
information. As the information necessary is dependent on the media
transport format, further rules specifying the header and its usage
is needed. For RTP the RTP-Info header is specified, see
Section 16.39.

After playing the desired range, the presentation does NOT transition
to the READY state, media delivery simply stops. A PAUSE request
MUST be issued before the stream enters the READY state. A PLAY
request while the stream is still in the PLAYING state is legal, and
can be issued without an intervening PAUSE request. Such a request
SHALL replace the current PLAY action with the new one requested,
i.e. being handle the same as the request was received in ready
state. In the case the first time range in Range header has a open
start time (-endtime), the server SHALL continue to play from where
it currently was until the specified end point. This is useful to
change ongoing playback to play another sequence, or end at another
point than in the previous request.

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.
\text{npt=0-}. If a PLAY request is received without a Range header when
media delivery has stopped at the end, the server SHOULD respond with
a 457 "Invalid Range" error response. In that response the current
pause point in a Range header SHALL be included.

The following example plays the whole presentation starting at SMPTE
time code 0:10:20 until the end of the clip. 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/2.0
    CSeq: 833
    Session: 12345678
    Range: smpte=0:10:20-
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 833
    Date: 23 Jan 1997 15:35:06 GMT
    Server: PhonyServer 1.0
    Range: smpte=0:10:22-0:15:45
    RTP-Info:url="rtsp://example.com/twister.en"
    ssrc=0D12F123: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/2.0
    CSeq: 835
    Session: 12345678
    Range: clock=19961108T142300Z-19961108T143520Z
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 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"
      ssrc=0D12F123:seq=53745;rtptime=484589019

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 possibility to replace a current PLAY request with a new one replaces two RTSP 1.0 functions:

- The queued play functionality described in RFC 2326 [RFC2326] is removed and multiple ranges can be used to achieve a similar functionality.
- The use of PLAY for keep-alive signaling, i.e. PLAY request without a range header in PLAY state, has also been deprecated. Instead a client can use, SETPARAMETER (recommended) or OPTIONS (allowed) for keep alive.

An example of using PLAY request to change the behavior, if a server has received requests to play ranges 10 to 15 and then 13 to 20 (that is, overlapping ranges), a PLAY request 4 seconds after the first would take effect while the server plays the first range. Thus changing the behavior to continue to play to 25 seconds, i.e. the played range equal play with range: npt=10-25. If the second PLAY request would arrive as the second range in the first request was playing, then the equivalent request would be play with range:npt=10-15,npt=13-25.
13.5. PAUSE

The PAUSE request causes the stream delivery to immediately be interrupted (halted). A PAUSE request MUST be done either 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/2.0
   CSeq: 834
   Session: 12345678
   User-Agent: PhonyClient/1.2

S→C: RTSP/2.0 200 OK
   CSeq: 834
   Date: 23 Jan 1997 15:35:06 GMT
   Range: npt=45.76-

The PAUSE request causes stream delivery to be interrupted immediately on receipt of the message and the pause point is set to the current point in the presentation. That pause point in the media stream needs to be maintained. A subsequent PLAY request without Range header SHALL resume from the pause point and play until media end.

The 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 simply includes the Range header from the PAUSE response.

C→S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
   CSeq: 834
   Session: 12345678
   Range: npt=10-30
   User-Agent: PhonyClient/1.2

S→C: RTSP/2.0 200 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"
   ssrc=0D12F123:seq=5712;rtptime=934207921,
             url="rtsp://example.com/fizzle/videotrack"
   ssrc=4FAD8726:seq=57654;rtptime=2792482193
   Session: 12345678

after 11 seconds, i.e. at 21 seconds into the presentation:

C→S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
   CSeq: 835
   Session: 12345678
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 still 200 OK. The 200 OK response MUST include the Range header with the current pause point. See examples below:

C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
    CSeq: 834
    Session: 12345678
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 834
    Session: 12345678
    Date: 23 Jan 1997 15:35:06 GMT
    Range: npt=45.76-98.36

C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
    CSeq: 835
    Session: 12345678
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 835
    Session: 12345678
    Date: 23 Jan 1997 15:35:07 GMT
    Range: npt=45.76-98.36

13.6. TEARDOWN

The TEARDOWN client to server request stops the stream delivery for the given URI, freeing the resources associated with it. A TEARDOWN request MAY be performed on either an aggregated or a media control URI. However some restrictions apply depending on the current state. The TEARDOWN request SHALL contain a Session header indicating what session the request applies to.

A TEARDOWN using the aggregated control URI or the media URI in a session under non-aggregated control (single media session) MAY be
done in any state (Ready, and Play). A successful request SHALL result in that media delivery is immediately halted and the session state is destroyed. This SHALL be indicated through the lack of a Session header in the response.

A TEARDOWN using a media URI in an aggregated session MAY only be done in Ready state. Such a request only removes the indicated media stream and associated resources from the session. This may result in that a session returns to non-aggregated control, due to that it only contains a single media after the requests completion. A session that will exist after the processing of the TEARDOWN request SHALL in the response to that TEARDOWN request contain a Session header. Thus the presence of the Session header indicates to the receiver of the response if the session is still existing or has been removed.

Example:

C->S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/2.0
    CSeq: 892
    Session: 12345678
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 892
    Server: PhonyServer 1.0

13.7. GET_PARAMETER

The GET_PARAMETER request retrieves the value of a parameter or parameters for 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 specified session context. The content of the reply and response is left to the implementation.

The method MAY also be used without a body (entity). If the this request is successful, i.e. a 200 OK response is received, then the keep-alive timer has been updated. Any non-required header present in such a request may or may not been processed. To allow a client to determine if any such header has been processed, it is necessary to use a feature-tag and the Require header. Due to this reason it is RECOMMENDED that any parameters to be retrieved are sent in the body, rather than using any header.

Example:
S->C: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/2.0
    CSeq: 431
    Content-Type: text/parameters
    Session: 12345678
    Content-Length: 26
    User-Agent: PhonyClient/1.2

        packets_received
        jitter

C->S: RTSP/2.0 200 OK
    CSeq: 431
    Content-Length: 38
    Content-Type: text/parameters

        packets_received: 10
        jitter: 0.3838

The "text/parameters" section is only an example type for a body carrying parameters.

13.8. SET_PARAMETER

This method requests to set the value of a parameter or a set of parameters for a presentation or stream specified by the URI. The method MAY also be used without a body (entity). It is the RECOMMENDED method to use in request sent for the sole purpose of updating the keep-alive timer. If this request is successful, i.e. a 200 OK response is received, then the keep-alive timer has been updated. Any non-required header present in such a request may or may not be processed. To allow a client to determine if any such header has been processed, it is necessary to use a feature tag and the Require header. Due to this reason it is RECOMMENDED that any parameters are sent in the body, rather than using any header.

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 cannot locate a parameter, error 451 (Parameter Not Understood) SHALL be used. In the case a parameter is not allowed to change, the error code is 458 (Parameter Is Read-Only). The response body SHOULD contain only the parameters that have errors. Otherwise no body SHALL be returned.

Note: transport parameters for the media stream MUST only be set with
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/2.0
      CSeq: 421
      User-Agent: PhonyClient/1.2
      Content-length: 20
      Content-type: text/parameters

      barparam: barstuff

S->C: RTSP/2.0 451 Parameter Not Understood
      CSeq: 421
      Content-length: 10
      Content-type: text/parameters

      barparam: barstuff

The "text/parameters" section is only an example type for parameters. This method is intentionally loosely defined with the intention that the reply content and response content will be defined by the one desiring to use this mechanism.

13.9. REDIRECT

The REDIRECT method is issued by a server to inform a client that it required to connect to another server location to access the resource indicated by the Request-URI. The presence of the Session header in a REDIRECT request indicates the scope of the request, and determines the specific semantics of the request.

A REDIRECT request with a Session header has end-to-end (i.e. server to client) scope and applies only to the given session. Any intervening proxies SHOULD NOT disconnect the control channel while there are other remaining end-to-end sessions. The OPTIONAL Location header, if included in such a request, SHALL contain a complete absolute URI pointing to the resource to which the client SHOULD
reconnect. Specifically, the Location SHALL NOT contain just
the host and port. A client may receive a REDIRECT request with a
Session header, if and only if, an end-to-end session has been
established.

A client may receive a REDIRECT request without a Session header at
any time when it has communication or a connection established with a
server. The scope of such a request is limited to the next-hop (i.e.
the RTSP agent in direct communication with the server) and applies,
as well, to the control connection between the next-hop RTSP agent
and the server. A REDIRECT request without a Session header
indicates that all sessions and pending requests being managed via
the control connection MUST be redirected. The OPTIONAL Location
header, if included in such a request, SHOULD contain an absolute URI
with only the host address and the OPTIONAL port number of the server
to which the RTSP agent SHOULD reconnect. Any intervening proxies
SHOULD do all of the following in the order listed:

1. respond to the REDIRECT request

2. disconnect the control channel from the requesting server

3. connect to the server at the given host address

4. pass the REDIRECT request to each applicable client (typically
those clients with an active session or an unanswered request)

Note: The proxy is responsible for accepting REDIRECT responses
from its clients; these responses MUST NOT be passed on to either
the original server or the redirected server.

The lack of a Location header in any REDIRECT request is indicative
of the server no longer being able to fulfill the current request and
having no alternatives for the client to continue with its normal
operation. It is akin to a server initiated TEARDOWN that applies
both to sessions as well as the general connection associated with
that client.

When the Range header is not included in a REDIRECT request, the
client SHOULD perform the redirection immediately and return a
response to the server. The server can consider the session as
terminated and can free any associated state after it receives the
successful (2xx) response. The server MAY close the signalling
connection upon receiving the response and the client SHOULD close
the signalling connection after sending the 2xx response. The
exception to this is when the client has several sessions on the
server being managed by the given signalling connection. In this
case, the client SHOULD close the connection when it has received and
responded to REDIRECT requests for all the sessions managed by the signalling connection.

If the OPTIONAL Range header is included in a REDIRECT request, it indicates when the redirection takes effect. The range value MUST be an open ended single value, e.g. npt=59-, indicating the play out time when redirection SHALL occur. Alternatively, a range with a time= parameter indicates the wall clock time by when the redirection MUST take place. When the time= parameter is present in the range, any range value MUST be ignored even though it MUST be syntactically correct. To allow a client to determine that redirect time without being time synchronized with the server, the server SHALL include a Date header in the request. When the indicated redirect point is reached, a client MUST issue a TEARDOWN request and SHOULD close the signalling connection after receiving a 2xx response. The normal connection considerations apply for the server.

The differentiation of REDIRECT requests with and without range headers is to allow for clear and explicit state handling. As the state in the server needs to be kept until the point of redirection, the handling becomes more clear if the client is required to TEARDOWN the session at the redirect point.

If the REDIRECT request times out following the rules in Section 10.4 the server MAY terminate the session or transport connection that would be redirected by the request. This is a safeguard against misbehaving clients that refuses to respond to a REDIRECT request. That should not provide any benefit.

After a REDIRECT request has been processed, a client that wants to continue to send or receive media for the resource identified by the Request-URI will have to establish a new session with the designated host. If the URI given in the Location header is a valid resource URI, a client SHOULD issue a DESCRIBE request for the URI.

Note: The media resource indicated by the \header {Location header} can be identical, slightly different or totally different. This is the reason why a new DESCRIBE request SHOULD be issued.

If the Location header contains only a host address, the client MAY assume that the media on the new server is identical to the media on the old server, i.e. all media configuration information from the old session is still valid except for the host address. However the usage of conditional SETUP using ETag identifiers are RECOMMENDED to verify the assumption.

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/2.0
    CSeq: 732
    Location: rtsp://s2.example.com:8001
    Range: npt=0- ;time=19960213T143205Z
    Session: uZ3ci0K+Ld-M

C->S: RTSP/2.0 200 OK
    CSeq: 732
    User-Agent: PhonyClient/1.2
14. Embedded (Interleaved) Binary Data

In order to fulfill certain requirements on the network side, e.g. in conjunction with network address translators that block RTP traffic over UDP, it may be necessary 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 SHALL contain exactly one upper-layer protocol data unit, e.g., one RTP packet.

```
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 16.46).

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 16.46. 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 sometime 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://example.com/bar.file RTSP/2.0
    CSeq: 2
```
Transport: RTP/AVP/TCP;unicast;interleaved=0-1
Accept-Ranges: NPT, SMPTE, UTC
User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
CSeq: 2
Date: 05 Jun 1997 18:57:18 GMT
Transport: RTP/AVP/TCP;unicast;interleaved=5-6
Session: 12345678
Accept-Ranges: NPT

C->S: PLAY rtsp://example.com/bar.file RTSP/2.0
CSeq: 3
Session: 12345678
User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
CSeq: 3
Session: 12345678
Date: 05 Jun 1997 18:59:15 GMT
RTP-Info: url="rtsp://example.com/bar.file"
ssrc=0D12F123:seq=232433;rtptime=972948234
Range: npt=0-56.8

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
15. Status Code Definitions

Where applicable, HTTP status [H10] codes are reused. Status codes that have the same meaning are not repeated here. See Table 4 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.

15.1. Success 1xx

15.1.1. 100 Continue

See, [H10.1.1].

15.2. Success 2xx

15.3. 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.

A 3rr code MAY be used to respond to any request. It is RECOMMENDED that they are used if necessary before a session is established, i.e. in response to DESCRIBE or SETUP. However in cases where a server is not able to send a REDIRECT request to the client, the server MAY need to resort to using 3rr responses to inform a client with an established session about the need for redirecting the session. If an 3rr response is received for an request in relation to a established session, the client SHOULD send a TEARDOWN request for the session, and MAY reestablish the session using the resource indicated by the Location.

If the the Location header is used in a response it SHALL contain an absolute URI pointing out the media resource the client is redirected to, the URI SHALL NOT only contain the host name.
15.3.1.  300 Multiple Choices

See [H10.3.1].

15.3.2.  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. The Location header MUST be included in the response.

15.3.3.  302 Found

The requested resource resides temporarily at the URI given by the
Location header. The Location header MUST be included in the
response. This response is intended to be used for many types of
temporary redirects; e.g., load balancing. It is RECOMMENDED that
the server 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.

This example shows a client being redirected to a different server:

    C->S: SETUP rtsp://example.com/fizzle/foo RTSP/2.0
          CSeq: 2
          Transport: RTP/AVP/TCP;unicast;interleaved=0-1
          Accept-Ranges: NPT, SMPTE, UTC
          User-Agent: PhonyClient/1.2

    S->C: RTSP/2.0 302 Try Other Server
          CSeq: 2
          Location: rtsp://s2.example.com:8001/fizzle/foo

15.3.4.  303 See Other

This status code SHALL NOT be used in RTSP. However as it was
allowed to use in RTSP 1.0 (RFC 2326).

15.3.5.  304 Not Modified

If the client has performed a conditional DESCRIBE or SETUP (see
Section 16.25) 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:
o Date

o ETag and/or Content-Location, if the header(s) would have been sent in a 200 response to the same request.

o 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 (only the session description) 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.

15.3.6. 305 Use Proxy

See [H10.3.6].

15.4. Client Error 4xx

15.4.1. 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.

15.4.2. 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.

15.4.3. 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).

15.4.4. 452 reserved

This error code was removed from RFC 2326 [RFC2326] and is obsolete.
15.4.5.  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.

15.4.6.  454 Session Not Found

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

15.4.7.  455 Method Not Valid in This State

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

15.4.8.  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 SHALL be returned to inform the client of which format(s) that are allowed.

15.4.9.  457 Invalid Range

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

15.4.10.  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).

15.4.11.  459 Aggregate Operation Not Allowed

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

15.4.12.  460 Only Aggregate Operation Allowed

The requested method may not be applied on the URI in question since it is not an aggregate control (presentation) URI. The method may be applied on the aggregate control URI.
15.4.13. 461 Unsupported Transport

The Transport field did not contain a supported transport specification.

15.4.14. 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 dest_addr parameter in the Transport field.

15.4.15. 463 Destination Prohibited

The data transmission channel was not established because the server prohibited access to the client address. This error is most likely the result of a client attempt to redirect media traffic to another destination with a dest_addr parameter in the Transport header.

15.4.16. 464 Data Transport Not Ready Yet

The data transmission channel to the media destination is not yet ready for carrying data. However the responding entity still expects that the data transmission channel will be established at this point in time. Note however that this may result in a permanent failure like 462 "Destination Unreachable".

An example when this error may occur is in the case a client sends a PLAY request to a server prior to ensuring that the TCP connections negotiated for carrying media data was successful established (In violation of this specification). The server would use this error code to indicate that the requested action could not be performed due to the failure of completing the connection establishment.

15.4.17. 470 Connection Authorization Required

The secured connection attempt need user or client authorization before proceeding. The next hops certificate is included in this response in the Accept-Credentials header.

15.4.18. 471 Connection Credentials not accepted

When performing a secure connection over multiple connections, a intermediary has refused to connect to the next hop and carry out the request due to unacceptable credentials for the used policy.
15.4.19.  472 Failure to establish secure connection

A proxy fails to establish a secure connection to the next hop RTSP agent. This is primarily caused by a fatal failure at the TLS handshake, for example due to server not accepting any cipher suits.

15.5.  Server Error 5xx

15.5.1.  551 Option not supported

A feature-tag given in the Require or the Proxy-Require fields was not supported. The Unsupported header SHALL be returned stating the feature for which there is no support.
16. Header Field Definitions

+---------------+--------+--------+---------+------+
| method        | direction | object | acronym | Body |
+---------------+--------+--------+---------+------+
| DESCRIBE      | C -> S | P,S    | DES     | r    |
|               |        |        |         |      |
| GET_PARAMETER | C -> S, S -> C | P,S    | GPR     | R,r  |
| OPTIONS       | C -> S | P,S    | OPT     |      |
|               |        |        |         |      |
|               | S -> C |        |         |      |
| PAUSE         | C -> S | P,S    | PSE     |      |
| PLAY          | C -> S | P,S    | PLY     |      |
| REDIRECT      | S -> C | P,S    | RDR     |      |
| SETUP         | C -> S | S      | STP     |      |
| SETPARAMETER  | C -> S, S -> C | P,S    | SPR     | R,r  |
| TEARDOWN      | C -> S | P,S    | TRD     |      |
+---------------+--------+--------+---------+------+

Table 8: 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 5.2. This section lists the full set of header fields along with notes on meaning, and usage. The syntax definition for header fields are present in Section 21.2.3. Throughout this section, we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification RFC 2616 [RFC2616]. Examples of each header field are given.

Information about header fields in relation to methods and proxy processing is summarized in Table 9, Table 10, Table 11, and Table 12.

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 both requests and responses.

The "proxy" column describes the operations a proxy may perform on a header field. An empty proxy column indicates that the proxy SHALL NOT do any changes to that header, all allowed operations are explicitly stated:

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 needs to 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 XXX {tab:methods2:}

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 SHALL be present if the message body is not empty. See Section 16.16, Section 16.18 and Section 5.3 for details.

-: The header field is not applicable.

"Optional" means that a Client/Server MAY include the header field in a request or response. The Client/Server behavior when receiving
such headers varies, for some it may ignore the header field, in other case it is request to process the header. This is regulated by the method and header descriptions. Example of such headers that require processing are the Require and Proxy-Require header fields discussed in Section 16.38 and Section 16.32. 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.

An RTSP agent SHALL ignore extension headers that are not understood.

The From and Location header fields contain an URI. If the URI contains a comma, or semicolon, the URI MUST be enclosed in double quotes ("). 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.

<table>
<thead>
<tr>
<th>Header</th>
<th>Where</th>
<th>Proxy</th>
<th>DES</th>
<th>OPT</th>
<th>SETU</th>
<th>PLA</th>
<th>PAUS</th>
<th>TRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Credentials</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Accept-Encodings</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>R</td>
<td>r</td>
<td>m</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>456</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Allow</td>
<td>r</td>
<td>am</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Allow</td>
<td>405</td>
<td>am</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Bandwidth</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Blocksize</td>
<td>R</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cache-Control</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Connection</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Connection-Credentials</td>
<td>470, ar</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td>407</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Base</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>R</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>4xx, 5xx</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Language</td>
<td>R</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Language</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Language</td>
<td>4xx, 5xx</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Length</td>
<td>r</td>
<td>r</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Length</td>
<td>4xx, 5xx</td>
<td>r</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Location</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Location</td>
<td>4xx, 5xx</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Type</td>
<td>r</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Header</td>
<td>Where</td>
<td>Prox</td>
<td>DES</td>
<td>OPT</td>
<td>SETUP</td>
<td>PLAY</td>
<td>PAUSE</td>
<td>TRD</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>Pipelined-Requests</td>
<td>amdr</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>407</td>
<td>amr</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Proxy-Authorization</td>
<td>R</td>
<td>rd</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>R</td>
<td>ar</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>r</td>
<td>r</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

Table 9: Overview of RTSP header fields (A-L) related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.
| Proxy-Supported | R | amr | c | c | c | c | c | c |
| Proxy-Supported | r | c | c | c | c | c | c | c |
| Public | r | admr | - | m | - | - | - | - |
| Public | 501 | admr | m | m | m | m | m | m |
| Range | R | - | - | - | o | - | - | - |
| Range | r | - | - | c | m | m | - | - |
| Referer | R | o | o | o | o | o | o | o |
| Require | R | o | o | o | o | o | o | o |
| Retry-After | 3rr,503 | o | o | o | - | - | - | - |
| RTP-Info | r | - | - | c | c | - | - | - |
| Scale | - | - | - | o | - | - | - | - |
| Session | R | r | - | o | o | m | m | m |
| Session | r | r | - | c | m | m | m | o |
| Server | R | r | - | o | - | - | - | - |
| Server | r | r | o | o | o | o | o | o |
| Speed | - | - | - | o | - | - | - | - |
| Supported | R | amr | o | o | o | o | o | o |
| Supported | r | amr | c | c | c | c | c | c |
| Timestamp | R | admr | o | o | o | o | o | o |
| Timestamp | c | admr | m | m | m | m | m | m |
| Transport | amr | - | - | m | - | - | - | - |
| Unsupported | r | c | c | c | c | c | c | c |
### Table 10: Overview of RTSP header fields (P-W) related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.

<table>
<thead>
<tr>
<th>Header</th>
<th>Where</th>
<th>Proxy</th>
<th>GPR</th>
<th>SPR</th>
<th>RDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept-Credentials</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Allow</td>
<td>405</td>
<td>amr</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Authorization</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>R</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Blocksize</td>
<td>R</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Connection</td>
<td></td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Connection-Credentials</td>
<td>470,407</td>
<td>ar</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Content-Base</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Base</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Base</td>
<td>4xx,5xx</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>4xx,5xx</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Content-Language</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Content-Language</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th></th>
<th>4xx,5xx</th>
<th>r</th>
<th>o</th>
<th>o</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Length</td>
<td>R</td>
<td>r</td>
<td>*</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>Content-Length</td>
<td>r</td>
<td>r</td>
<td>*</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>Content-Length</td>
<td>4xx,5xx</td>
<td>r</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Content-Location</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Content-Location</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Content-Location</td>
<td>4xx,5xx</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Content-Type</td>
<td>R</td>
<td>*</td>
<td>*</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Type</td>
<td>r</td>
<td>*</td>
<td>*</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Content-Type</td>
<td>4xx</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>CSeq</td>
<td>R,c</td>
<td>mr</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Date</td>
<td>R</td>
<td>a</td>
<td>o</td>
<td>o</td>
<td>m</td>
</tr>
<tr>
<td>Date</td>
<td>r</td>
<td>am</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>From</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>R</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Location</td>
<td>3rr</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Pipelined-Requests</td>
<td>amdr</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>407</td>
<td>amr</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Proxy-Authorization</td>
<td>R</td>
<td>rd</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>R</td>
<td>ar</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>r</td>
<td>r</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>Proxy-Supported</td>
<td>R</td>
<td>amr</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>Header</td>
<td>Where</td>
<td>Proxy</td>
<td>GPR</td>
<td>SPR</td>
<td>RDR</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Range</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Referer</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Require</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Retry-After</td>
<td>3rr,503</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Session</td>
<td>r</td>
<td>r</td>
<td>c</td>
<td>c</td>
<td>o</td>
</tr>
<tr>
<td>Server</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Server</td>
<td>r</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Supported</td>
<td>R</td>
<td>adrm</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Supported</td>
<td>r</td>
<td>adrm</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>Timestamp</td>
<td>R</td>
<td>adrm</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Timestamp</td>
<td>c</td>
<td>adrm</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Unsupported</td>
<td>r</td>
<td>arm</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>User-Agent</td>
<td>R</td>
<td>r</td>
<td>m*</td>
<td>m*</td>
<td>-</td>
</tr>
<tr>
<td>User-Agent</td>
<td>r</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>m*</td>
</tr>
<tr>
<td>Vary</td>
<td>r</td>
<td>c</td>
<td>c</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Via</td>
<td>R</td>
<td>amr</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Via</td>
<td>c</td>
<td>dr</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
</tbody>
</table>

Table 11: Overview of RTSP header fields (A-P) related to methods GETPARAMETER, SETPARAMETER, and REDIRECT.
Table 12: Overview of RTSP header fields (R-W) related to methods
GETPARAMETER, SETPARAMETER, and REDIRECT.

16.1. Accept

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

See [H14.1] for syntax.

Example of use:

Accept: application/example q=1.0, application/sdp

16.2. Accept-Credentials

The Accept-Credentials header is a request header used to indicate to
any trusted intermediary how to handle further secured connections to
proxies or servers. See Section 20 for the usage of this header. It
SHALL NOT be included in server to client requests.

In a request the header SHALL contain the method (User, Proxy, or
Any) for approving credentials selected by the requestor. The method
SHALL NOT be changed by any proxy, unless it is "proxy" when a proxy
MAY change it to "user" to take the role of user approving each
further hop. If the method is "User" the header contains zero or
more of credentials that the client accept. The header may contain
zero credentials in the first RTSP request to a RTSP server when
using the "User" method. This as the client has not yet received any
credentials to accept. Each credential SHALL consist of one URI
identifying the proxy or server, the hash algorithm identifier, and
the hash over that entity’s DER encoded certificate [RFC3280] in
Base64 [RFC4648]. All RTSP clients and proxies SHALL implement the
SHA-256 [FIPS-pub-180-2] algorithm for computation of the hash of the
DER encoded certificate. The SHA-256 algorithm is identified by the
token "sha-256".

The intention with allowing for other hash algorithms is to enable
the future retirement of algorithms that are not implemented
somewhere else than here. Thus the definition of future algorithms
for this purpose is intended to be extremely limited. A feature tag
can be used to ensure that support for the replacement algorithm
exist.
16.3. Accept-Encoding

See [H14.3].

16.4. Accept-Language

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

16.5. Accept-Ranges

The Accept-Ranges request and response-header field allows indication of the format supported in the Range header. The client SHALL include the header in SETUP requests to indicate which formats it support to receive in PLAY and PAUSE responses, and REDIRECT requests. The server SHALL include the header in SETUP and 456 error responses to indicate the formats supported for the resource indicated by the request URI.

Accept-Ranges: NPT, SMPTE

This header has the same syntax as [H14.5] and the syntax is defined in Section 21.2.3. However, new range-units are defined.

16.6. 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. The Allow header MUST also be present in all OPTIONS responses where the content of the header will not include exactly the same methods as listed in the Public header.

The Allow SHALL also be included in SETUP and DESCRIBE responses, if the methods allowed for the resource is different than the minimal implementation set.

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

16.7. Authorization

See [H14.8].

16.8. 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 mobility, congestion, etc.

Example:

Bandwidth: 62360

16.9. 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 (4xx) if the value is syntactically invalid.

16.10. 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, instead it applies to the media stream identified by the SETUP request. The RTSP requests are generally not cacheable, for further information see Section 18. Below is the description of the cache directives that can be included in the
Cache-Control header.

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. Note, there are no security function enforcing that the content can’t be cached.

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 streams.

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 media stream. 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 is required to 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.
16.11. 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 has the choice of continuing using a connection indefinitely, as long as it sends valid messages.

16.12. Connection-Credentials

The Connection-Credentials response header is used to carry the chain of credentials of any next hop that need to be approved by the requestor. It SHALL only be used in server to client responses.

The Connection-Credentials header in an RTSP response SHALL, if included, contain the credential information (in form of a list of certificates providing the chain of certification) of the next hop that an intermediary needs to securely connect to. The header MUST include the URI of the next hop (proxy or server) and a base64 encoded binary structure containing a sequence of DER encoded X.509v3 certificates [RFC3280].

The binary structure starts with the number of certificates (NR_CERTS) included as a 16 bit unsigned integer. This is followed by NR_CERTS number of 16 bit unsigned integers providing the size in octets of each DER encoded certificate. This is followed by NR_CERTS number of DER encoded X.509v3 certificates in a sequence (chain). The proxy or server’s certificate must come first in the structure. Each following certificate must directly certify the one preceding it. Because certificate validation requires that root keys be distributed independently, the self-signed certificate which specifies the root certificate authority may optionally be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case.

Example:

Connection-Credentials: "rtsps://proxy2.example.com/"; MiIDNTCC...

Where MiIDNTCC... is a BASE64 encoding of the following structure:
16.13. Content-Base

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

Content-Base: rtsp://media.example.com/movie/twister

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.


See [H14.11].

16.15. Content-Language

See [H14.12].

16.16. Content-Length

The Content-Length general-header field contains the length of the body (entity) of the message (i.e. after the double CRLF following the last header). Unlike HTTP, it MUST be included in all messages that carry body 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].
16.17.  Content-Location

See [H14.14].

16.18.  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.

16.19.  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, however it is RECOMMENDED to start at 0. 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.

Proxies that aggregate several sessions on the same transport will regularly need to renumber the CSeq header field in requests and responses to fulfill the rules for the header.

Example:

CSeq: 239

16.20.  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.

16.21.  ETag

The ETag response header MAY be included in DESCRIBE or SETUP responses. The entity tags (Section 4.8) returned in a DESCRIBE response, and the one in SETUP refers to the presentation, i.e. both the returned session description and the media stream. This allows for verification that one has the right session description to a media resource at the time of the SETUP request. However it has the
disadvantage that a change in any of the parts results in invalidation of all the parts.

If the ETag is provided both inside the entity, e.g. within the "a=etag" attribute in SDP, and in the response message, then both tags SHALL be identical. It is RECOMMENDED that the ETag is primarily given in the RTSP response message, to ensure that caches can use the ETag without requiring content inspection. However for session descriptions that are distributed outside of RTSP, for example using HTTP, etc. it will be necessary to include the entity tag in the session description as specified in Appendix C.1.9.

SETUP and DESCRIBE requests can be made conditional upon the ETag using the headers If-Match (Section 16.24) and If-None-Match (Section 16.26).

16.22. 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 presentation description (body) SHOULD be considered stale.

SETUP response: The Expires header indicate a date and time after which the media stream SHOULD be considered stale.

A stale cache entry may not normally be returned by a cache (either a proxy cache or an user agent cache) unless it is first validated with the origin server (or with an intermediate cache that has a fresh copy of the entity). See Section 18 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:

An example of its use is

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

RTSP/2.0 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/2.0 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 16.10).

16.23. From

See [H14.22].

16.24. 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. By including the ETag given in or with the session description in a SETUP request, the client ensures that resources set up are matching the description. A SETUP request for which the ETag validation check fails, SHALL respond using 412 (Precondition Failed).

This validation check is also very useful if a session has been redirected from one server to another.

16.25. 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 SHALL be returned without any message-body.

An example of the field is:

16.26. If-None-Match

See [H14.26].

This request header can be used with one or several entity tags to make DESCRIBE requests conditional. A new session description is retrieved only if another entity than the ones already available would be included. If the entity available for delivery is matching the one the client already has, then a 304 (Not Modified) response is given.

16.27. 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.

16.28. Location

See [H14.30].

16.29. Pipelined-Requests

The Pipelined-Requests general header is used to indicate that a request is to be executed in the context created by previous requests. The primary usage of this header is to allow pipelining of SETUP requests so that any additional SETUP request after the first one doesn’t need to wait for the session ID to be sent back to the requesting entity. The header contains a unique identifier that is scoped by the persistent connection used to send the requests.

Upon receiving a request with the Pipelined-Requests the responding entity SHALL look up if there exist a binding between this Pipelined-Requests identifier for the current persistent connection and an RTSP session ID. If that exist then the received request is processed the same way as if it did contain the Session header with the looked up session ID. If there doesn’t exist a mapping and no Session header is included in the request, the responding entity SHALL create a binding upon the successful completion of a session creating request, i.e. SETUP. If the request failed to create an RTSP session no binding SHALL be created. In case the request contains both a Session header and the Pipelined-Requests header the Pipelined-Requests SHALL be ignored.

Note: Based on the above definition at least the first request containing a new unique Pipelined-Requests will be required to be a
SETUP request (unless the protocol is extended with new methods of creating a session). After that first one, additional SETUP requests or request of any type using the RTSP session context may include the Pipelined-Requests header.

For all responses to request that contained the Pipelined-Requests, the Session header and the Pipelined-Requests SHALL both be included, assuming that it is allowed for that response and that the binding between the header values exist. Pipelined-Requests SHOULD NOT be used in requests after that the client has received the RTSP Session ID. This as using the real session ID allows the request to be used also in cases the persistent connection has been terminated and a new connection is needed.

It is the sender of the request that is responsible for using a previously unused identifier within this transport connection scope when a new RTSP session is to be created with this method. A server side binding SHALL be deleted upon the termination of the related RTSP session. Note: Although this definition would allow for reusing previously used pipelining identifiers, this is NOT RECOMMENDED to allow for better error handling and logging.

RTSP Proxies may need to translate Pipelined-Requests identifier values from incoming request to outgoing to allow for aggregation of requests onto a persistent connection.

16.30. Proxy-Authenticate

See [H14.33].

16.31. Proxy-Authorization

See [H14.34].

16.32. 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. The proxy SHALL use the 551 (Option Not Supported) status code in the response. Any feature-tag included in the Proxy-Require does not apply to the end-point (server or client). To ensure that a feature is supported by both proxies and servers the tag needs to be included in also a Require header.

See Section 16.38 for more details on the mechanics of this message and a usage example.
Example of use:

   Proxy-Require: play.basic

16.33.  Proxy-Supported

The Proxy-Supported header field enumerates all the extensions supported by the proxy using feature-tags. The header carries the intersection of extensions supported by the forwarding proxies. The Proxy-Supported header MAY be included in any request by a proxy. It SHALL be added by any proxy if the Supported header is present in a request. When present in a request, the receiver MUST in the response copy the received Proxy-Supported header.

The Proxy-Supported header field contains a list of feature-tags applicable to proxies, as described in Section 4.7. The list are the intersection of all feature-tags understood by the proxies. To achieve an intersection, the proxy adding the Proxy-Supported header includes all proxy feature-tags it understands. Any proxy receiving a request with the header, checks the list and removes any feature-tag it do not support. A Proxy-Supported header present in the response SHALL NOT be touched by the proxies.

Example:

   C->P1: OPTIONS rtsp://example.com/ RTSP/2.0
       Supported: foo, bar, blech
       User-Agent: PhonyClient/1.2

   P1->P2: OPTIONS rtsp://example.com/ RTSP/2.0
       Supported: foo, bar, blech
       Proxy-Supported: proxy-foo, proxy-bar, proxy-blech
       Via: 2.0 prox1.example.com

   P2->S: OPTIONS rtsp://example.com/ RTSP/2.0
       Supported: foo, bar, blech
       Proxy-Supported: proxy-foo, proxy-blech
       Via: 2.0 prox1.example.com, 2.0 prox2.example.com

   S->C: RTSP/2.0 200 OK
       Supported: foo, bar, baz
       Proxy-Supported: proxy-foo, proxy-blech
       Public: OPTIONS, SETUP, PLAY, PAUSE, TEARDOWN
       Via: 2.0 prox1.example.com, 2.0 prox2.example.com
16.34. Public

The Public response header field lists the set of methods supported by the response sender. This header applies to the general capabilities of the sender and its only purpose is to indicate the sender’s capabilities to the recipient. 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.

Example of use:

   Public: OPTIONS, SETUP, PLAY, PAUSE, TEARDOWN

In the event that there are proxies between the sender and the recipient of a response, each intervening proxy MUST modify the Public header field to remove any methods that are not supported via that proxy. The resulting Public header field will contain an intersection of the sender’s methods and the methods allowed through by the intervening proxies.

In general proxies should allow all methods to transparently pass through from the sending RTSP agent to the receiving RTSP agent, but there may be cases where this is not desirable for a given proxy. Modification of the Public response header field by the intervening proxies ensures that the request sender gets an accurate response indicating the methods that can be used on the target agent via the proxy chain.

16.35. Range

The Range header specifies a time range in PLAY (Section 13.4), PAUSE (Section 13.5), SETUP (Section 13.3), and REDIRECT (Section 13.9) requests and responses.

The range can be specified in a number of units. This specification defines smpte (Section 4.4), npt (Section 4.5), and clock (Section 4.6) range units. While byte ranges [H14.35.1] and other extended units MAY be used, their behavior is unspecified since they are not normally meaningful in RTSP. Servers supporting the Range header MUST understand the NPT range format and SHOULD understand the SMPTE range format. If the Range header is sent in a time format that is not understood, the recipient SHOULD return 456 (Header Field Not Valid for Resource) and include an Accept-Ranges header indicating the supported time formats for the given resource.

The Range header MAY contain a time parameter in UTC, specifying the time at which the operation is to be made effective. This
functionality SHALL be used only with the REDIRECT method.

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. For 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.

Example:

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

The notation is similar to that used for the HTTP/1.1 [RFC2616] 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.

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 16.40) 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 for formats that have such a notion, like NPT and SMPTE.

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.

16.36. Referer

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

16.37. Retry-After

See [H14.37].

16.38. 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 16.44) 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 16.32).

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.

Example (Not complete):

C->S:   SETUP rtsp://server.com/foo/bar/baz.rm RTSP/2.0
        CSeq: 302
        Require: funky-feature
        Funky-Parameter: funkystuff

S->C:   RTSP/2.0 551 Option not supported
        CSeq: 302
Unsupported: funky-feature

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 SHALL ignore this header. If a particular extension requires that intermediate devices support it, the extension should be tagged in the Proxy-Require field instead (see Section 16.32).

16.39. 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 SHOULD be part of a 200 response to PLAY.

The exclusion of the RTP-Info in a PLAY response for RTP transported media will result in that a client needs to synchronize the media streams using RTCP. This may have negative impact as the RTCP can be lost, and does not need to be particularly timely in their arrival. Also functionality as informing the client from which packet a seek has occurred is affected.

The RTP-Info MAY also be included in SETUP responses to provide synchronization information when changing transport parameters, see Section 13.3.

The header can carry the following parameters:

- **url**: Indicates the stream URI which for which the following RTP parameters correspond, this URI MUST be the same used in the SETUP request for this media stream. Any relative URI SHALL use the Request-URI as base URI. This parameter SHALL be present.

- **ssrc**: The Synchronization source (SSRC) that the RTP timestamp and sequence number provide applies to. This parameter SHALL be present.
seq: Indicates the sequence number of the first packet of the stream that is direct result of the request. 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. Note that a client may not receive the packet with the expressed sequence number, and instead packets with a higher sequence number, due to packet loss or reordering. This parameter is RECOMMENDED to be present.

rtptime: SHALL indicate the RTP timestamp value corresponding to the start time value in the Range response header, or if not explicitly given the implied start point. The client uses this value to calculate the mapping of RTP time to NPT or other media timescale. This parameter SHOULD be present to ensure inter-media synchronization is achieved. There exist no requirement that any received RTP packet will have the same RTP timestamp value as the one in the parameter used to establish synchronization.

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 media clock time (NPT, etc.). 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.

Example:
Range:npt=3.25-15
RTP-Info:url="rtsp://example.com/foo/audio" ssrc=0A13C760:seq=45102;
           rtptime=12345678,url="rtsp://example.com/foo/video"
           ssrc=9A9DE123:seq=30211;rtptime=29567112

Let's assume that audio uses a 16kHz RTP timestamp clock and video a 90kHz RTP timestamp clock. Then the media synchronization is depicted in the following way.

NPT  3.0---3.1---3.2-X-3.3---3.4---3.5---3.6
Audio  PA  A
Video  V   PV

X: NPT time value = 3.25, from Range header.
A: RTP timestamp value for Audio from RTP-Info header (12345678).
V: RTP timestamp value for Video from RTP-Info header (29567112).
PA: RTP audio packet carrying an RTP timestamp of 12344878. Which corresponds to NPT = (12344878 - A) / 16000 + 3.25 = 3.2
PV: RTP video packet carrying an RTP timestamp of 29573412. Which corresponds to NPT = (29573412 - V) / 90000 + 3.25 = 3.32

16.40. 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. For certain media transports this may require certain considerations to work consistent, see Appendix B.1 for description on how RTP handles this.

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.

When indicating a negative scale for a reverse playback, the Range header MUST indicate a decreasing range as described in
Section 16.35.

Example of playing in reverse at 3.5 times normal rate:

Scale: -3.5
Range: npt=15-10

16.41. 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.

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 non-dedicated best-
effort networks the sender is required to perform congestion control
of the stream(s). This can result in that the communicated speed is
impossible to maintain.
16.42. Server

See [H14.38], however the header syntax is corrected in Section 21.2.3.

16.43. Session

The Session request-header and response-header field identifies an RTSP session. An RTSP session is created by the server as a result of a successful SETUP request and in the response the session identifier is given to the client. The RTSP session exist until destroyed by a TEARDOWN or timed out by the server.

The session identifier is chosen by the server (see Section 4.3) and MUST be returned in the SETUP response. Once a client receives a session identifier, it SHALL be included in any request related to that session. This means that the Session header MUST be included in a request using the following methods: PLAY, PAUSE, and TEARDOWN, and MAY be included in SETUP, OPTIONS, SETPARAMETER, GET_PARAMETER, and REDIRECT, and SHALL NOT be included in DESCRIBE. In an RTSP response the session header SHALL be included in methods, SETUP, PLAY, and PAUSE, and MAY be included in methods, TEARDOWN, and REDIRECT, and if included in the request of the following methods it SHALL also be included in the response, OPTIONS, GET_PARAMETER, and SETPARAMETER, and SHALL NOT be included in DESCRIBE.

The timeout parameter MAY be included in a SETUP response, and SHALL NOT be included in requests. 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 below and Appendix A). The timeout is measured in seconds, with a default of 60 seconds (1 minute). The length of the session timeout SHALL NOT be changed in a established session.

The mechanisms for showing liveness of the client is, any RTSP request with a Session header, if RTP & RTCP is used an RTCP message, or through any other used media protocol capable of indicating liveness of the RTSP client. It is RECOMMENDED that a client does not wait to the last second of the timeout before trying to send a liveness message. The RTSP message may be lost or when using reliable protocols, such as TCP, the message may take some time to arrive safely at the receiver. To show liveness between RTSP request issued to accomplish other things, the following mechanisms can be used, in descending order of preference:
RTCP: If RTP is used for media transport RTCP SHOULD be used. If RTCP is used to report transport statistics, it 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 only gives statistical guarantees to reach the server. However that probability is so low that it can be ignored in most cases. For example, a session with 60 seconds timeout and enough bitrate assigned to RTCP messages to send a message from client to server on average every 5 seconds. That client have for a network with 5 % packet loss, the probability to fail showing liveness sign in that session within the timeout interval of 2.4*E-16. In sessions with shorter timeout times, or much higher packet loss, or small RTCP bandwidths SHOULD also use any of the mechanisms below.

SETPARAMETER: When using SETPARAMETER for keep alive, no body SHOULD be included. This method is the RECOMMENDED RTSP method to use in request only intended to perform keep-alive.

OPTIONS: This method does also work. However it causes the server to perform more unnecessary processing and result in bigger responses than necessary for the task. The reason for this is that the server needs to determine what capabilities that are associated with the media resource to correctly populate the Public and Allow headers.

Note that a session identifier identifies an RTSP session across transport sessions or connections. RTSP requests for a given session can use different URIs (Presentation and media URIs). Note, that there are restrictions depending on the session which URIs that are acceptable for a given method. However, multiple "user" sessions for the same URI from the same client will require use of different session identifiers.

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

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

16.44. Supported

The Supported header field enumerates all the extensions supported by the client or server using feature tags. The header carries the extensions supported by the message sending entity. The Supported header MAY be included in any request. When present in a request, the receiver MUST respond with its corresponding Supported header.
Note, also in 4xx and 5xx responses is the supported header included.

The Supported header field contains a list of feature-tags, described in Section 4.7, that are understood by the client or server.

Example:

C->S: OPTIONS rtsp://example.com/ RTSP/2.0
  Supported: foo, bar, blech
  User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
  Supported: bar, blech, baz

16.45. Timestamp

The Timestamp general-header field describes when the agent sent the request. The value of the timestamp is of significance only to the agent and may use any timescale. The responding agent 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 agent to compute the round-trip time to the responding agent so that it can adjust the timeout value for retransmissions. It also resolves retransmission ambiguities for unreliable transport of RTSP.

16.46. 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 server SHOULD return a Transport response-header field in the response to indicate the values actually chosen. The Transport header field MAY also be used to change certain transport parameters. A server MAY refuse to change parameters of an existing stream.

A Transport request header field MAY contain a list of transport options acceptable to the client, in the form of multiple transport-spec entries. In that case, the server MUST return the single (transport-spec) which was actually chosen. The number of transport-spec entries is expected to be limited as the client will get guidance on what configurations that are possible from the presentation description.
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 the multicast transport type parameter. Unknown parameters SHALL be ignored. The requester need to ensure that the responder understands the parameters through the use of feature tags and the Require header.

Any parameters part of future extensions requires clarification if they are safe to ignore in accordance to this specification, or are required to be understood. If a parameter is required to be understood, then a feature-tag MUST be defined for the functionality and used in the Require or Proxy-Require headers.

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 general syntax for the transport specifier is a list of slash separated tokens:

Value1/Value2/Value3...

Which for RTP transports take the form:

RTP/profile/lower-transport.

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

There are two different methods for how to specify where the media should be delivered:

dest_addr: The presence of this parameter and its values indicates the destination address or addresses (host address and port pairs for IP flows) necessary for the media transport.

No dest_addr: The lack of the dest_addr parameter indicates that the server SHALL send media to same address for which the RTSP messages originates. Does not work for transports requiring explicitly given destination ports.

The choice of method for indicating where the media is to be delivered depends on the use case. In many case the only allowed method will be to use no explicit address indication and have the server deliver media to the source of the RTSP messages.
An RTSP proxy will need to take care. If the media is not desired to be routed through the proxy, the proxy will need to introduce the destination indication.

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 needs to indicate such capability by including two full transport-specs with separate parameters for each.

- **layers:** The number of multicast layers to be used for this media stream. The layers are sent to consecutive addresses starting at the dest_addr address. If the parameter is not included, it defaults to a single layer.

- **dest_addr:** A general destination address parameter that can contain one or more address specifications. Each combination of Protocol/Profile/Lower Transport needs to have the format and interpretation of its address specification defined. For RTP/AVP/UDP and RTP/AVP/TCP, the address specification is a tuple containing a host address and port. Note, only a single destination entity per transport spec is intended. The usage of multiple destination to distribute a single media to multiple entities is unspecified.

The client originating the RTSP request MAY specify the destination address of the stream recipient with the host address part of the tuple. When the destination address 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 MUST perform security checks (see Section 22.1) and SHOULD log such attempts before allowing the client to direct a media stream to a recipient address not chosen by the server. Implementations cannot rely on TCP as reliable means of client identification. If the server does not allow the host address part of the tuple to be set, it SHALL return 463 (Destination Prohibited).

The host address part of the tuple MAY be empty, for example ":58044", in cases when only destination port is desired to be specified.
src_addr: A general source address parameter that can contain one or
more address specifications. Each combination of Protocol/
Profile/Lower Transport needs to have the format and
interpretation of its address specification defined. For RTP/
AVP/UDP and RTP/AVP/TCP, the address specification is a tuple
containing a host address and port.

This parameter MUST be specified by the server if it transmits
media packets from another address than the one RTSP messages
are sent to. This will allow the client to verify source
address and give it a destination address for its RTCP feedback
packets if RTP is used. The address or addresses indicated in
the src_addr parameter SHOULD be used both for sending and
receiving of the media streams data packets. The main reasons
are threefold: First, indicating the port and source address(s)
lets the receiver know where from the packets is expected to
originate. Secondly, traversal of NATs are greatly simplified
when traffic is flowing symmetrically over a NAT binding.
Thirdly, certain NAT traversal mechanisms, needs to know to
which address and port to send so called "binding packets" from
the receiver to the sender, thus creating a address binding in
the NAT that the sender to receiver packet flow can use.

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.

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 in this specification unspecified but reserved.

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 14. 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., tt 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. When included in requests the value indicate the TTL value that the client desires to use. In response the value actually being used is returned. A server will need to consider what values that are reasonable and also the authority of the user to set this value.

RTP-specific:

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

ssrc: The ssrc parameter, if included in a SETUP response, indicates the RTP SSRC [RFC3550] value(s) that will be used by the media server for RTP packets within the stream. It is expressed as an eight digit hexadecimal value.

The ssrc parameter SHALL NOT be specified in requests. The functionality of specifying the ssrc parameter in a SETUP request is deprecated as it is incompatible with the specification of RTP in RFC 3550[RFC3550]. If the parameter is included in the Transport header of a SETUP request, the server MAY ignore it, and choose appropriate SSRCs for the stream. The server MAY set the ssrc parameter in the Transport header of the response.

The parameters defined below MAY only be used if the media transport protocol if the lower-level transport is connection-oriented (such as TCP). However, these parameters MUST NOT be used when interleaving data over the RTSP control connection.

setup: Clients use the setup parameter on the Transport line in a SETUP request, to indicate the roles it wishes to play in a TCP connection. This parameter is adapted from [RFC4145]. We discuss the use of this parameter in RTP/AVP/TCP non-interleaved transport in Appendix B.2.2; the discussion below is limited to syntactic issues. Clients may specify the following values for the setup parameter: ["active":] The client will initiate an outgoing connection. ["passive":] The client will accept an incoming connection. ["actpass":] The
client is willing to accept an incoming connection or to initiate an outgoing connection.

If a client does not specify a setup value, the "active" value is assumed.

In response to a client SETUP request where the setup parameter is set to "active", a server’s 2xx reply MUST assign the setup parameter to "passive" on the Transport header line.

In response to a client SETUP request where the setup parameter is set to "passive", a server’s 2xx reply MUST assign the setup parameter to "active" on the Transport header line.

In response to a client SETUP request where the setup parameter is set to "actpass", a server’s 2xx reply MUST assign the setup parameter to "active" or "passive" on the Transport header line.

Note that the "holdconn" value for setup is not defined for RTSP use, and MUST NOT appear on a Transport line.

connection: Clients use the setup parameter on the Transport line in a SETUP request, to indicate the SETUP request prefers the reuse of an existing connection between client and server (in which case the client sets the "connection" parameter to "existing"), or that the client requires the creation of a new connection between client and server (in which case the client sets the "connection" parameter to "new"). Typically, clients use the "new" value for the first SETUP request for a URL, and "existing" for subsequent SETUP requests for a URL.

If a client SETUP request assigns the "new" value to "connection", the server response MUST also assign the "new" value to "connection" on the Transport line.

If a client SETUP request assigns the "existing" value to "connection", the server response MUST assign a value of "existing" or "new" to "connection" on the Transport line, at its discretion.

The default value of "connection" is "existing", for all SETUP requests (initial and subsequent).

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/2.0
    CSeq: 302
    Transport: RTP/AVP;multicast;mode="PLAY",
               RTP/AVP;unicast;dest_addr="192.0.2.5:3456"/
                 "192.0.2.5:3457";mode="PLAY"
    Accept-Ranges: NPT, SMPTE, UTC
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 302
    Date: 23 Jan 1997 15:35:06 GMT
    Session: 47112344
    Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:3456"/
                 "192.0.2.5:3457";src_addr="192.0.2.224:6256"/
                 "192.0.2.224:6257";mode="PLAY"
    Accept-Ranges: NPT

16.47. 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 16.32), 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 16.38 for a usage example.

16.48. 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.

16.49. Vary

See [H14.44].

16.50. Via

See [H14.45].
16.51. WWW-Authenticate

See [H14.47].
17. Proxies

RTSP Proxies are RTSP agents that sit in between a client and a server. A proxy can take on both the role as a client and as server depending on what it tries to accomplish. Proxies are also introduced for several different reasons.

Caching Proxy: This type of proxy is used to reduce the workload on servers and connections. By caching a presentation, both description and media streams the proxy can serve a client content without requesting it from the server once it has been cached and hasn’t become stale. See the caching Section 18.

Access Proxy: This type of proxy is used to ensure that a RTSP client get access to servers on an external network. Thus this proxy is placed on the border between two domains, e.g. a private address space and the public internet. The proxy performs the necessary translation, usually addresses, and often also media stream translation or redirection.

Security Proxy: This type of proxy is used to help facilitate security functions around RTSP. For example when having a firewalled network, the security proxy request that the necessary pinholes in the firewall is opened when a client in the protected network want to access media streams on the external side. It can also provide network owners with a logging and audit point for RTSP sessions, e.g. for corporations that tracks or limits their employees access to certain type of content.

All type of proxies can be used also when using secured communication with TLS as RTSP 2.0 allows the client to approve certificate chains used for connection establishment from a proxy, see Section 20.3.2. However that trust model may not be suitable for all type of deployment, and instead secured sessions do by-pass of the proxies.

Access proxies SHOULD NOT be used in equipment like NATs and firewalls that aren’t expected to be regularly maintained, like home or small office equipment. In these cases it is better to use the NAT traversal procedures defined for RTSP 2.0 [I-D.ietf-mmusic-rtsp-nat]. The reason for these recommendations is that any extensions of RTSP resulting in new media transport protocols or profiles, new parameters etc may fail in a proxy that isn’t maintained. Thus resulting in blocking further development of RTSP and its usage.

The existence of proxies must always be considered when developing new RTSP extensions. There must be definition of how proxies may
handle the extension, if it is required to understand it, thus requiring a feature-tag to be used in the Proxy-Require header.
18. 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 16.10. 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, e.g. 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.
19. Examples

This section contains several different examples trying to illustrate possible ways of using RTSP. The examples can also help with the understanding of how functions of RTSP work. However remember that this is examples and the normative and syntax description in the other sections takes precedence. Please also note that many of the example contain syntax illegal line breaks to accommodate the formatting restriction that the RFC series impose.

19.1. Media on Demand (Unicast)

The is an example of media on demand streaming of a media stored in 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 controlled media 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 URI.

The following is an example of using a single RTSP session to control multiple streams. It also illustrates the use of aggregate URIs. In a container file it is also desirable to not write any URI parts which is not kept, when the container is distributed, like the host and most of the path element. Therefore this example also uses the "*" and relative URI in the delivered SDP.

Client C requests a presentation from media server M. The movie is stored in a container file. The client has obtained an RTSP URI to the container file.
C->M: DESCRIBE rtsp://example.com/twister.3gp RTSP/2.0
CSeq: 1
User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
CSeq: 1
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 257
Content-Base: rtsp://example.com/twister.3gp/
Expires: 24 Jan 1997 15:35:06 GMT

v=0
o=- 2890844256 2890842807 IN IP4 192.0.2.5
s=RTSP Session
i=An Example of RTSP Session Usage
e=adm@example.com
a=control: *
a=range: npt=0-0:10:34.10
t=0 0
m=audio 0 RTP/AVP 0
a=control: trackID=1
m=video 0 RTP/AVP 26
a=control: trackID=4

C->M: SETUP rtsp://example.com/twister.3gp/trackID=1 RTSP/2.0
CSeq: 2
User-Agent: PhonyClient/1.2
Require: play.basic
Transport: RTP/AVP;unicast;dest_addr=":8000"/":8001"
Accept-Ranges: NPT, SMPTE, UTC

M->C: RTSP/2.0 200 OK
CSeq: 2
Server: PhonyServer/1.0
Transport: RTP/AVP; unicast; dest_addr=":8000"/":8001; src_addr="192.0.2.5:9000"/"192.0.2.5:9001"
   ssrc=93CB001E
Session: 12345678
Expires: 24 Jan 1997 15:35:12 GMT
Date: 23 Jan 1997 15:35:12 GMT
Accept-Ranges: NPT

C->M: SETUP rtsp://example.com/twister.3gp/trackID=4 RTSP/2.0
CSeq: 3
User-Agent: PhonyClient/1.2
Require: play.basic
Transport: RTP/AVP; unicast; dest_addr=":8002"/":8003"
Session: 12345678
Accept-Ranges: NPT, SMPTE, UTC

M->C: RTSP/2.0 200 OK
CSeq: 3
Server: PhonyServer/1.0
Transport: RTP/AVP; unicast; dest_addr=":8002"/":8003; src_addr="192.0.2.5:9002"/"192.0.2.5:9003";
   ssrc=A813FC13
Session: 12345678
Expires: 24 Jan 1997 15:35:13 GMT
Date: 23 Jan 1997 15:35:13 GMT
Accept-Range: NPT

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
CSeq: 4
User-Agent: PhonyClient/1.2
Range: npt=0-10, npt=30-
Session: 12345678

M->C: RTSP/2.0 200 OK
CSeq: 4
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:35:14 GMT
Session: 12345678
Range: npt=0-10, npt=30-623.10
RTP-Info: url="rtsp://example.com/twister.3gp/trackID=4"
   ssrc=0D12F123;seq=12345;rtptime=3450012,
   url="rtsp://example.com/twister.3gp/trackID=1"
   ssrc=4F312DD8;seq=54321;rtptime=2876889
19.2. Media on Demand using Pipelining

This example is basically the example above (Section 19.1) but now utilizing pipelining to speed up the setup. Into requiring only two round trip times until media starts flowing. First getting the session description to determine what media resources need to be setup. In the second one send the necessary SETUP requests and the PLAY request to initiate media delivery.

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

C->M: DESCRIBE rtsp://example.com/twister.3gp RTSP/2.0
CSeq: 1
User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
CSeq: 1
User-Agent: PhonyClient/1.2
Session: 12345678
Range: npt=34.57-623.10
RTP-Info: url="rtsp://example.com/twister.3gp/trackID=4"
  ssrc=0D12F123:seq=12555;rtptime=6330012,
  url="rtsp://example.com/twister.3gp/trackID=1"
  ssrc=4F312DD8:seq=55021;rtptime=3132889
CSeq: 1
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 257
Content-Base: rtsp://example.com/twister.3gp/
Expires: 24 Jan 1997 15:35:06 GMT

v=0
o=- 2890844256 2890842807 IN IP4 192.0.2.5
s=RTSP Session
i=An Example of RTSP Session Usage
e=adm@example.com
a=control: *
a=range: npt=0-0:10:34.10
t=0 0
m=audio 0 RTP/AVP 0
a=control: trackID=1
m=video 0 RTP/AVP 26
a=control: trackID=4

C->M: SETUP rtsp://example.com/twister.3gp/trackID=1 RTSP/2.0
CSeq: 2
User-Agent: PhonyClient/1.2
Require: play.basic
Transport: RTP/AVP;unicast;dest_addr=":8000"/":8001"
Accept-Ranges: NPT, SMPTE, UTC
Pipelined-Requests: 7654

C->M: SETUP rtsp://example.com/twister.3gp/trackID=4 RTSP/2.0
CSeq: 3
User-Agent: PhonyClient/1.2
Require: play.basic
Transport: RTP/AVP;unicast;dest_addr=":8002"/":8003"
Accept-Ranges: NPT, SMPTE, UTC
Pipelined-Requests: 7654

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
CSeq: 4
User-Agent: PhonyClient/1.2
Range: npt=0-10, npt=30-
Session: 12345678
Pipelined-Requests: 7654

M->C: RTSP/2.0 200 OK
CSeq: 2
Server: PhonyServer/1.0
Transport: RTP/AVP;unicast;dest_addr=":8000"/":8001;
An alternative example of media on demand with a bit more tweaks is the following. Client C requests a movie distributed from two different media servers A (tt audio.example.com) and V (tt 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
   Content-Length: 264
   Expires: 23 Jan 1998 15:35:06 GMT

   v=0
   o=- 2890844526 2890842807 IN IP4 192.0.2.5
   s=RTSP Session
   e=adm@example.com
   a=range:npt=0-1:49:34
   t=0 0
   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/2.0
   CSeq: 1
   User-Agent: PhonyClient/1.2
   Transport: RTP/AVP/UDP;unicast;dest_addr=":3056"/":3057",
              RTP/AVP/TCP;unicast;interleaved=0-1
   Accept-Ranges: NPT, SMPTE, UTC

A->C: RTSP/2.0 200 OK
   CSeq: 1
   Session: 12345678
   Transport: RTP/AVP/UDP;unicast;dest_addr=":3056"/":3057";
              src_addr="192.0.2.5:5000"/"192.0.2.5:5001"
   Date: 23 Jan 1997 15:35:12 GMT
   Server: PhonyServer/1.0
   Expires: 24 Jan 1997 15:35:12 GMT
   Cache-Control: public
   Accept-Ranges: NPT, SMPTE

C->V: SETUP rtsp://video.example.com/twister/video RTSP/2.0
   CSeq: 1
   User-Agent: PhonyClient/1.2
   Transport: RTP/AVP/UDP;unicast;dest_addr=":3058"/":3059",
              RTP/AVP/TCP;unicast;interleaved=0-1
   Accept-Ranges: NPT, SMPTE, UTC

V->C: RTSP/2.0 200 OK
   CSeq: 1
Session: 23456789
Transport: RTP/AVP/UDP;unicast;dest_addr=":3058"/":3059";
  src_addr="192.0.2.5:5002"/"192.0.2.5:5003"
Date: 23 Jan 1997 15:35:12 GMT
Server: PhonyServer/1.0
Cache-Control: public
Expires: 24 Jan 1997 15:35:12 GMT
Accept-Ranges: NPT, SMPTE

C->V: PLAY rtsp://video.example.com/twister/video RTSP/2.0
  CSeq: 2
  User-Agent: PhonyClient/1.2
  Session: 23456789
  Range: smpte=0:10:00-

V->C: RTSP/2.0 200 OK
  CSeq: 2
  Session: 23456789
  Range: smpte=0:10:00-1:49:23
  RTP-Info: url="rtsp://video.example.com/twister/video"
    ssrc=A17E189D:seq=12312232;rtptime=78712811
  Server: PhonyServer/2.0
  Date: 23 Jan 1997 15:35:13 GMT

C->A: PLAY rtsp://audio.example.com/twister/audio.en RTSP/2.0
  CSeq: 2
  User-Agent: PhonyClient/1.2
  Session: 12345678
  Range: smpte=0:10:00-

A->C: RTSP/2.0 200 OK
  CSeq: 2
  Session: 12345678
  Range: smpte=0:10:00-1:49:23
  RTP-Info: url="rtsp://audio.example.com/twister/audio.en"
    ssrc=3D124F01:seq=876655;rtptime=1032181
  Server: PhonyServer/1.0
  Date: 23 Jan 1997 15:35:13 GMT

C->A: TEARDOWN rtsp://audio.example.com/twister/audio.en RTSP/2.0
  CSeq: 3
  User-Agent: PhonyClient/1.2
  Session: 12345678

A->C: RTSP/2.0 200 OK
  CSeq: 3
Even though the audio and video track are on two different servers, may start at slightly different times, and may drift with respect to each other, the client can perform initial synchronize of the two media using RTP-Info and Range received in the PLAY responses. If the two servers are time synchronized the RTCP packets can also be used to maintain synchronization.

19.4. 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 needs to use the rules set forth in the session description for Request-URIs, rather than assuming that a consistent URI may always be used throughout. Below are 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/2.0
   Accept: application/x-rtsp-mh, application/sdp
   CSeq: 1
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 1
   Content-base: rtsp://foo.com/test.wav/
   Content-type: application/sdp
   Content-length: 148
   Server: PhonyServer/1.0
   Date: 23 Jan 1997 15:35:06 GMT
   Expires: 23 Jan 1997 17:00:00 GMT

   v=0
   o=- 872653257 872653257 IN IP4 192.0.2.5
   s=mu-law wave file
   i=audio test
   t=0 0
   a=control: *
   m=audio 0 RTP/AVP 0
   a=control:streamid=0

C->S: SETUP rtsp://foo.com/test.wav/streamid=0 RTSP/2.0
   Transport: RTP/AVP/UDP;unicast;
      dest_addr=":6970"/":6971";mode="PLAY"
   CSeq: 2
   User-Agent: PhonyClient/1.2
   Accept-Ranges: NPT, SMPTE, UTC

S->C: RTSP/2.0 200 OK
   Transport: RTP/AVP/UDP;unicast;dest_addr=":6970"/":6971";
      src_addr="192.0.2.5:6970"/"192.0.2.5:6971";
      mode="PLAY";ssrc=EAB98712
   CSeq: 2
   Session: 2034820394
   Expires: 23 Jan 1997 16:00:00 GMT
   Server: PhonyServer/1.0
   Date: 23 Jan 1997 15:35:07 GMT
Accept-Ranges: NPT

C->S: PLAY rtsp://foo.com/test.wav/ RTSP/2.0
    CSeq: 3
    User-Agent: PhonyClient/1.2
    Session: 2034820394

S->C: RTSP/2.0 200 OK
    CSeq: 3
    Server: PhonyServer/1.0
    Date: 23 Jan 1997 15:35:08 GMT
    Session: 2034820394
    Range: npt=0-600
    RTP-Info: url="rtsp://foo.com/test.wav/streamid=0"
               ssrc=0D12F123:seq=981888;rtptime=3781123

Note the different URI in the SETUP command, and then the switch back to the aggregate URI 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. However the server has declared that the aggregated control URI in the SDP and therefore this is legal.

In this case, it is also required that servers accept implementations that use the non-aggregated interpretation and use the individual media URI, like this:

C->S: PLAY rtsp://example.com/test.wav/streamid=0 RTSP/2.0
    CSeq: 3
    User-Agent: PhonyClient/1.2

19.5. Live Media Presentation Using Multicast

The media server M chooses the multicast address and port. Here, it is assumed that the web server only contains a pointer to the full description, while the media server M maintains the full description.
C->W: GET /sessions.html HTTP/2.0
    Host: www.example.com

W->C: HTTP/2.0 200 OK
    Content-Type: text/html

    <html>
    ...
    <href "Stremed Live Music performance"
        src="rtsp://live.example.com/concert/audio">
    ...
    </html>

C->M: DESCRIBE rtsp://live.example.com/concert/audio RTSP/2.0
    CSeq: 1
    Supported: play.basic, play.scale
    User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
    CSeq: 1
    Content-Type: application/sdp
    Content-Length: 182
    Server: PhonyServer/1.0
    Date: 23 Jan 1997 15:35:06 GMT
    Supported: play.basic

    v=0
    o=- 2890844526 2890842807 IN IP4 192.0.2.5
    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
    a=range:npt=0-

C->M: SETUP rtsp://live.example.com/concert/audio RTSP/2.0
    CSeq: 2
    Transport: RTP/AVP;multicast
    Accept-Ranges: NPT, SMPTE, UTC
19.6. Capability Negotiation

This example illustrates how the client and server determine their capability to support a special feature, in this case "play.scale". The server, through the client's request and the included Supported header, learns the client supports RTSP 2.0, and also supports the playback time scaling feature of RTSP. The server's response contains the following feature-related information to the client; it supports the basic playback (play.basic), the extended functionality of time scaling of content (play.scale), and one "example.com" proprietary feature (com.example.flight). The client also learns the methods supported (Public header) by the server for the indicated resource.

C->S: OPTIONS rtsp://media.example.com/movie/twister.3gp RTSP/2.0
CSeq: 1
Supported: play.basic, play.scale
User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
CSeq: 1
Public: OPTIONS, SETUP, PLAY, PAUSE, TEARDOWN
Server: PhonyServer/2.0
Supported: play.basic, play.scale, com.example.flight

When the client sends its SETUP request it tells the server that it is requires support of the play.scale feature for this session by including the Require header.

C->S: SETUP rtsp://media.example.com/twister.3gp/trackID=1 RTSP/2.0
    CSeq: 3
    User-Agent: PhonyClient/1.2
    Transport: RTP/AVP/UDP;unicast;dest_addr=":3056"/":3057",
              RTP/AVP/TCP;unicast;interleaved=0-1
    Require: play.scale
    Accept-Ranges: NPT, SMPTE, UTC
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 3
    Session: 12345678
    Transport: RTP/AVP/UDP;unicast;dest_addr=":3056"/":3057"
               src_addr="192.0.2.5:5000"/"192.0.2.5:5001"
    Server: PhonyServer/2.0
    Accept-Ranges: NPT, SMPTE
20. Security Framework

The RTSP security framework consists of two high level components: the pure authentication mechanisms based on HTTP authentication, and the transport protection based on TLS, which is independent of RTSP. Because of the similarity in syntax and usage between RTSP servers and HTTP servers, the security for HTTP is re-used to a large extent.

20.1. RTSP and HTTP Authentication

RTSP and HTTP share common authentication schemes, and thus follow the same usage guidelines as specified in [RFC2617] and also in [H15]. Servers SHOULD implement both basic and digest [RFC2617] authentication.

It should be stressed that using the HTTP authentication alone does not provide full control message security. Therefore, in environments requiring tighter security for the control messages, TLS SHOULD be used, see Section 20.2.

20.2. RTSP over TLS

RTSP SHALL follow the same guidelines with regards to TLS [RFC4346] usage as specified for HTTP, see [RFC2818]. RTSP over TLS is separated from unsecured RTSP both on URI level and port level. Instead of using the "rtsp" scheme identifier in the URI, the "rtsps" scheme identifier MUST be used to signal RTSP over TLS. If no port is given in a URI with the "rtsps" scheme, port 322 SHALL be used for TLS over TCP/IP.

When a client tries to setup an insecure channel to the server (using the "rtsp" URI), and the policy for the resource requires a secure channel, the server SHALL redirect the client to the secure service by sending a 301 redirect response code together with the correct Location URI (using the "rtsps" scheme). A user or client MAY upgrade a non secured URI to a secured by changing the scheme from "rtsp" to "rtsps". A server implementing support for "rtsps" SHALL allow this.

It should be noted that TLS allows for mutual authentication (when using both server and client certificates). Still, one of the more common way TLS is used is to only provide server side authentication (often to avoid client certificates). TLS is then used in addition to HTTP authentication, providing transport security and server authentication, while HTTP Authentication is used to authenticate the client.

RTSP includes the possibility to keep a TCP session up between the
client and server, throughout the RTSP session lifetime. It may be convenient to keep the TCP session, not only to save the extra setup time for TCP, but also the extra setup time for TLS (even if TLS uses the resume function, there will be almost two extra roundtrips). Still, when TLS is used, such behavior introduces extra active state in the server, not only for TCP and RTSP, but also for TLS. This may increase the vulnerability to DoS attacks.

In addition to these recommendations, Section 20.3 gives further recommendations of TLS usage with proxies.

20.3. Security and Proxies

The nature of a proxy is often to act as a "man-in-the-middle", while security is often about preventing the existence of a "man-in-the-middle". This section provides the clients with the possibility to use proxies even when applying secure transports (TLS) between the RTSP agents. The TLS proxy mechanism allows for server and proxy identification using certificates. However, the client can not be identified based on certificates. The client needs to select between using the procedure specified below or using a TLS connection directly (by-passing any proxies) to the server. The choice may be dependent on policies.

There are basically two categories of proxies, the transparent proxies (of which the client is not aware) and the non-transparent proxies (of which the client is aware). An infrastructure based on proxies requires that the trust model is such that both client and servers can trust the proxies to handle the RTSP messages correctly. To be able to trust a proxy, the client and server also needs to be aware of the proxy. Hence, transparent proxies cannot generally be seen as trusted and will not work well with security (unless they work only at transport layer). In the rest of this section any reference to proxy will be to a non-transparent proxy, which inspects or manipulate the RTSP messages.

HTTP Authentication is built on the assumption of proxies and can provide user-proxy authentication and proxy-proxy/server authentication in addition to the client-server authentication.

When TLS is applied and a proxy is used, the client will connect to the proxy’s address when connecting to any RTSP server. This implies that for TLS, the client will authenticate the proxy server and not the end server. Note that when the client checks the server certificate in TLS, it MUST check the proxy’s identity (URI or possibly other known identity) against the proxy’s identity as presented in the proxy’s Certificate message.
The problem is that for a proxy accepted by the client, the proxy needs to be provided information on which grounds it should accept the next-hop certificate. Both the proxy and the user may have rules for this, and the user have the possibility to select the desired behavior. To handle this case, the Accept-Credentials header (See Section 16.2) is used, where the client can force the proxy/proxies to relay back the chain of certificates used to authenticate any intermediate proxies as well as the server. Given the assumption that the proxies are viewed as trusted, it gives the user a possibility to enforce policies to each trusted proxy of whether it should accept the next entity in the chain.

A proxy MUST use TLS for the next hop if the RTSP request includes a "rtsps" URI. TLS MAY be applied on intermediate links (e.g. between client and proxy, or between proxy and proxy), even if the resource and the end server does not require to use it. The proxy SHALL when initiating the next hop TLS connection use the incoming TLS connections CipherSuite list, only modified by removing any cipher suits that the proxy does not support. In case a proxy fails to establish a TLS connection due to cipher suite mismatch between proxy and next hop proxy or server, this is indicated using error code 472 (Failure to establish secure connection).

20.3.1. Accept-Credentials

The Accept-Credentials header can be used by the client to distribute simple authorization policies to intermediate proxies. The client includes the Accept-Credentials header to dictate how the proxy treats the server/next proxy certificate. There are currently three methods defined:

Any, which means that the proxy (or proxies) SHALL accept whatever certificate presented. This is of course not a recommended option to use, but may be useful in certain circumstances (such as testing).

Proxy, which means that the proxy (or proxies) MUST use its own policies to validate the certificate and decide whether to accept it or not. This is convenient in cases where the user has a strong trust relation with the proxy. Reason why a strong trust relation may exist are; personal/company proxy, proxy has a out-of-band policy configuration mechanism.

User, which means that the proxy (or proxies) MUST send credential information about the next hop to the client for authorization. The client can then decide whether the proxy should accept the certificate or not. See Section 20.3.2 for further details.
If the Accept-Credentials header is not included in the RTSP request from the client, then the "Proxy" method SHALL be used as default. If an other method than the "Proxy" is to be used, then the Accept-Credentials header SHALL be included in all of the RTSP request from the client. This is because it cannot be assumed that the proxy always keeps the TLS state or the users previously preference between different RTSP messages (in particular if the time interval between the messages is long).

With the "Any" and "Proxy" methods the proxy will apply the policy as defined for respectively method. If the policy do not accept the credentials of the next hop, the entity SHALL respond with a message using status code 471 (Connection Credentials not accepted).

An RTSP request in the direction server to client MUST NOT include the Accept-Credential header. As for the non-secured communication, the possibility for these request depends on the presence of a client established connection. However if the server to client request is in relation to a session established over a TLS secured channel, if MUST be sent in a TLS secured connection. That secured connection MUST also be the one used by the last client to server request. If no such transport connection exist at the time when the server desire to send the request, it silently fails.

Further policies MAY be defined and registered, but should be done so with caution.

20.3.2. User approved TLS procedure

For the "User" method each proxy MUST perform the following procedure for each RTSP request:

1. Setup the TLS session to the next hop if not already present (i.e. run the TLS handshake, but do not send the RTSP request).

2. Extract the peer certificate chain for the TLS session.

3. Check if a matching identity and hash of the peer certificate is present in the Accept-Credentials header. If present, send the message to the next hop, and conclude these procedures. If not, go to the next step.

4. The proxy responds to the RTSP request with a 470 or 407 response code. The 407 response code MAY be used when the proxy requires both user and connection authorization from user or client. In this message the proxy SHALL include a Connection-Credentials header, see Section 16.12 with the next hop’s identity and certificate.
The client MUST upon receiving a 470 or 407 response with Connection-Credentials header take the decision on whether to accept the certificate or not (if it cannot do so, the user SHOULD be consulted). If the certificate is accepted, the client has to again send the RTSP request. In that request the client has to include the Accept-Credentials header including the hash over the DER encoded certificate for all trusted proxies in the chain.

Example:

C->P: SETUP rtsp://test.example.org/secret/audio RTSP/2.0
    CSeq: 2
    Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:4588"/
               "192.0.2.5:4589"
    Accept-Ranges: NPT, SMPTE, UTC
    Accept-Credentials: User

P->C: RTSP/2.0 470 Connection Authorization Required
    CSeq: 2
    Connection-Credentials: "rtsp://test.example.org";
    MIIDNTCCAp...

C->P: SETUP rtsp://test.example.org/secret/audio RTSP/2.0
    CSeq: 2
    Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:4588"/
               "192.0.2.5:4589"
    Accept-Credentials: User "rtsp://test.example.org";sha-256;
                        dPYD7txpoGTbAqZZQJ+vaeOkyH4=
    Accept-Ranges: NPT, SMPTE, UTC

P->S: SETUP rtsp://test.example.org/secret/audio RTSP/2.0
    CSeq: 2
    Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:4588"/
               "192.0.2.5:4589"
    Via: RTSP/2.0 proxy.example.org
    Accept-Credentials: User "rtsp://test.example.org";sha-256;
                        dPYD7txpoGTbAqZZQJ+vaeOkyH4=
    Accept-Ranges: NPT, SMPTE, UTC

One implication of this process is that the connection for secured RTSP messages may take significantly more round-trip times for the first message. An complete extra message exchange between the proxy connecting to the next hop and the client results because of the process for approval for each hop. However after the first message exchange the remaining message should not be delayed, if each message contains the chain of proxies that the requestor accepts. The procedure of including the credentials in each request rather than building state in each proxy, avoids the need for revocation.
procedures.
21. Syntax

The RTSP syntax is described in an Augmented Backus-Naur Form (ABNF) as defined in RFC 4234 [RFC4234]. It uses the basic definitions present in RFC 4234.

Please note that ABNF strings, e.g. "Accept", are case insensitive as specified in section 2.3 of RFC 4234.

21.1. Base Syntax

RTSP header field values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream. This is intended to behave exactly as HTTP/1.1 as described in RFC 2616 [RFC2616]. The SWS construct is used when linear white space is optional, generally between tokens and separators.

To separate the header name from the rest of value, a colon is used, which, by the above rule, allows whitespace before, but no line break, and whitespace after, including a linebreak. The HCOLON defines this construct.

OCTET = %x00-FF ; any 8-bit sequence of data
CHAR = %x01-7F ; any US-ASCII character (octets 1 - 127)
UPALPHA = %x41-5A ; any US-ASCII uppercase letter "A".."Z"
LOALPHA = %x61-7A ; any US-ASCII lowercase letter "a".."z"
ALPHA = UPALPHA / LOALPHA
DIGIT = %x30-39 ; any US-ASCII digit "0".."9"
CTL = %x00-1F / %x7F ; any US-ASCII control character (octets 0 - 31) and DEL (127)
CR = %x0D ; US-ASCII CR, carriage return (13)
LF = %x0A ; US-ASCII LF, linefeed (10)
SP = %x20 ; US-ASCII SP, space (32)
HT = %x09 ; US-ASCII HT, horizontal-tab (9)
DQ = %x22 ; US-ASCII double-quote mark (34)
BACKSLASH = %x5C ; US-ASCII backslash (92)
CRLF = CR LF
LWS             =  [CRLF] 1*( SP / HT )
SWS             =  [LWS] ; sep whitespace
HCOLON          =  *( SP / HT ) ":\" SWS
TEXT            =  %x20-7D / %x80-FF ; any OCTET except CTLs
tspecials       =  "(" / ")" / "<" / ">" / "@" / "," / ";" / "":" / \ / DQ / "/" / ",[" / "]" / ":?" / ":=" / "{" / "}" / SP / HT
token           =  1*%x21 / %x23-27 / %x2A-2B / %x30-39
                  / %x41-5A / %x5E-7A / %x7C / %x7E)
                  ; 1*<any CHAR except CTLS or special>
quoted-string   =  ( DQ *qdtext DQ )
qdtext          =  %x20-21 / %x23-27 / %x80-FF ; any TEXT except <">
quoted-pair     =  BACKSLASH CHAR
ctext          =  %x20-27 / %x2A-7D
                  / %x80-FF ; any OCTET except CTLS, "(" and ")"
generic-param   =  token [ EQUAL gen-value ]
gen-value       =  token / host / quoted-string
safe            =  "$" / "-" / ":" / "." / ":+"
extra           =  "!*" / "!*" / "" / "#" / ":" / ":+"
rtsp-extra      =  "!*" / "!*" / "!*" / ":" / ":+"
HEX             =  %x01-7F / "A" / "B" / "C" / "D" / "E" / "F" /
                  / "a" / "b" / "c" / "d" / "e" / "f"
LHEX            =  %x01-7F / lowercase a-f
reserved        =  ":;" / ":*" / ":?" / "::" / ":@" / ":&" / ":="
unreserved      =  ALPHA / DIGIT / safe / extra
rtsp-unreserved  =  ALPHA / DIGIT / safe / rtsp-extra
base64          =  *base64-unit [base64-pad]
base64-unit     =  4base64-char
base64-pad      =  (2base64-char ":=") / (3base64-char ":=")
base64-char     =  ALPHA / DIGIT / ":+" / ":/"
21.2. RTSP Protocol Definition

21.2.1. Generic Protocol elements
RTSP-IRI = schemes ":" IRI-rest
IRI-rest = ihier-part [ ":" iquery ] [ ":" ifragment ]

ihier-part = "//" iauthority ipath-abempty

RTSP-IRI-ref = RTSP-IRI / irelative-ref
irelative-ref = irelative-part [ ":" iquery ] [ ":" ifragment ]
irelative-part = "//" iauthority ipath-abempty

/ ipath-absolute
/ ipath-noscheme
/ ipath-empty

iauthority = < As defined in RFC 3987>

ipath = ipath-abempty ; begins with "/" or is empty
/ ipath-absolute ; begins with "/" but not "/://"
/ ipath-noscheme ; begins with a non-colon segment
/ ipath-rootless ; begins with a segment
/ ipath-empty ; zero characters

ipath-abempty = *( "/" isegment )
ipath-absolute = "/" [ isegment-nz *( "/" isegment ) ]
ipath-noscheme = isegment-nz-nc *( "/" isegment )
ipath-rootless = isegment-nz *( "/" isegment )
ipath-empty = 0<ipchar>

isegment = *ipchar [";" *ipchar]
isegment-nz = 1*ipchar [";" *ipchar]
/ ";" *ipchar

isegment-nz-nc = (1*ipchar-nc [";" *ipchar-nc])
/ ";" *ipchar-nc

; non-zero-length segment without any colon ":"

ipchar = iunreserved / pct-encoded / sub-delims / "@"
ipchar-nc = iunreserved / pct-encoded / sub-delims / "@"

iquery = < As defined in RFC 3987>
ifragment = < As defined in RFC 3987>
iunreserved = < As defined in RFC 3987>
pct-encoded = < As defined in RFC 3987>
RTSP-URI       =  schemes "::" URI-rest
RTSP-URI-Ref   =  RTSP-URI / RTSP-Relative
schemes        =  "rtsp" / "rtsps" / scheme
scheme         =  < As defined in RFC 3986>
URI-rest       =  hier-part [ "?" query ]
hier-part      =  "//" authority path-abempty
RTSP-Relative  =  relative-part [ "?" query ]
relative-part  =  "//" authority path-abempty
/ path-absolute
/ path-noscheme
/ path-empty
authority      =  < As defined in RFC 3986>
query          =  < As defined in RFC 3986>
path           =  path-abempty ; begins with "/" or is empty
/ path-absolute ; begins with "/" but not "//"
/ path-noscheme ; begins with a non-colon segment
/ path-rootless ; begins with a segment
/ path-empty ; zero characters
path-abempty   =  *( "/" segment )
path-absolute  =  "/" [ segment-nz *( "/" segment ) ]
path-noscheme  =  segment-nz-nc *( "/" segment )
path-rootless  =  segment-nz *( "/" segment )
path-empty     =  0<pchar>
segment        =  *pchar [";" *pchar]
segment-nz     =  ( 1*pchar [";" *pchar]) / (";" *pchar)
segment-nz-nc  =  ( 1*pchar-nc [";" *pchar-nc]) / (";" *pchar-nc)
               ; non-zero-length segment without any colon ":"
pchar          =  unreserved / pct-encoded / sub-delims / ":" / ":"
pchar-nc       =  unreserved / pct-encoded / sub-delims / ":"
sub-delims     =  ":" / "\" / "\" / ":" / "(" / ")"
               / ":" / ":" / ":" / ":"

smpte-range = smpte-type "=" smpte-range-spec
 ; See section 3.4
smpte-range-spec = ( smpte-time "-" [ smpte-time ] )
 / ( "-" smpte-time )
smpte-type = "smpte" / "smpte-30-drop"
 / "smpte-25" / smpte-type-extension
 ; other timecodes may be added
smpte-type-extension = token
smpte-time = 1*2DIGIT ":" 1*2DIGIT ":" 1*2DIGIT
 [ ":" 1*2DIGIT [ "." 1*2DIGIT ] ]

npt-range = "npt=" npt-range-spec
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 ">" utc-clock "Z"
utc-date = 8DIGIT
utc-clock = 6DIGIT [ "." fraction ]
fraction = 1*DIGIT

feature-tag = token

session-id = 1*256( ALPHA / DIGIT / safe )

extension-header = header-name HCOLON header-value
header-name = token
header-value = *(TEXT-UTF8char / UTF8-CONT / LWS)

21.2.2. Message Syntax
RTSP-message = Request / Response ;RTSP/2.0 messages

Request = Request-Line
  *(general-header
   / request-header
   / entity-header )
  CRLF
  [ message-body ]

Response = Status-Line
  *( general-header
   / response-header
   / entity-header )
  CRLF
  [ message-body ]

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

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

Method = "DESCRIBE"
  / "GET_PARAMETER"
  / "OPTIONS"
  / "PAUSE"
  / "PLAY"
  / "REDIRECT"
  / "SETUP"
  / "SET_PARAMETER"
  / "TEARDOWN"
  / extension-method

extension-method = token

Request-URI = "**" / RTSP-URI

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

message-body = 1*OCTET

Status-Code = "100" ; Continue
  / "200" ; OK
  / "300" ; Multiple Choices
  / "301" ; Moved Permanently
  / "302" ; Found
  / "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
/ "416" ; Parameter Not Understood
/ "417" ; reserved
/ "418" ; Not Enough Bandwidth
/ "419" ; Session Not Found
/ "420" ; Method Not Valid in This State
/ "421" ; Header Field Not Valid for Resource
/ "422" ; Invalid Range
/ "423" ; Parameter Is Read-Only
/ "424" ; Aggregate operation not allowed
/ "425" ; Only aggregate operation allowed
/ "426" ; Unsupported Transport
/ "427" ; Destination Unreachable
/ "428" ; Destination Prohibited
/ "429" ; Data Transport Not Ready Yet
/ "430" ; Connection Authorization Required
/ "431" ; Connection Credentials not accepted
/ "432" ; Failure to establish secure connection
/ "433" ; Internal Server Error
/ "434" ; Not Implemented
/ "435" ; Bad Gateway
/ "436" ; Service Unavailable
/ "437" ; Gateway Time-out
/ "438" ; RTSP Version not supported
/ "439" ; Option not supported

extension-code  =  3DIGIT

Reason-Phrase   =  *TEXT
general-header = Cache-Control
/ Connection
/ CSeq
/ Date
/ Pipelined-Requests
/ Proxy-Supported
/ Supported
/ Timestamp
/ Via
/ extension-header

request-header = Accept
/ Accept-Credentials
/ Accept-Encoding
/ Accept-Language
/ Authorization
/ Bandwidth
/ Blocksize
/ From
/ If-Match
/ If-Modified-Since
/ If-None-Match
/ Proxy-Require
/ Range
/ Referer
/ Require
/ Scale
/ Session
/ Speed
/ Supported
/ Transport
/ User-Agent
/ extension-header
response-header = Accept-Credentials
/ Accept-Ranges
/ Connection-Credentials
/ ETag
/ Location
/ Proxy-Authenticate
/ Public
/ Range
/ Retry-After
/ RTP-Info
/ Scale
/ Session
/ Server
/ Speed
/ Transport
/ Unsupported
/ Vary
/ WWW-Authenticate
/ extension-header

entity-header = Allow
/ Content-Base
/ Content-Encoding
/ Content-Language
/ Content-Length
/ Content-Location
/ Content-Type
/ Expires
/ Last-Modified
/ extension-header

21.2.3. Header Syntax

All header syntaxes not defined in this section are defined in section 14 of the HTTP 1.1 specification [RFC2616].

Accept = "Accept" HCOLON
[ accept-range *(COMMA accept-range) ]
accept-range = media-range *(SEMI accept-param)
media-range = ( "/" */" /
/ ( m-type SLASH "/" )
/ ( m-type SLASH m-subtype )
) *( SEMI m-parameter )
accept-param = ("q" EQUAL qvalue) / generic-param
qvalue = ( "/" "0" [ "." *DIGIT ] )
/ ( "/" "1" [ "." 3DIGIT ] )
Accept-Credentials = "Accept-Credentials" HCOLON cred-decision CRLF
cred-decision = ("User" [LWS cred-info])
   / "Proxy"
   / "Any"
   / token [LWS 1*TEXT] ; For future extensions
cred-info = cred-info-data *(COMMA cred-info-data)
cred-info-data = DQ RTSP-URI DQ SEMI hash-alg SEMI base64
hash-alg = "sha-256" / extension-alg
extension-alg = token
Accept-Encoding = "Accept-Encoding" HCOLON
   [ encoding *(COMMA encoding) ]
encoding = codings *(SEMI accept-param)
codings = content-coding / "*"
content-coding = "Accept-Language" HCOLON
   [ language *(COMMA language) ]
language = language-range *(SEMI accept-param)
language-range = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) / "*" )
Accept-Ranges = "Accept-Ranges" HCOLON acceptable-ranges CRLF
acceptable-ranges = (range-unit *(COMMA range-unit))
   / "none"
range-unit = "NPT" / "SMPTE" / "UTC" / extension-format
extension-format = token
Allow = "Allow" HCOLON [Method *(COMMA Method)]
Authorization = "Authorization" HCOLON credentials
credentials = ("Digest" LWS digest-response)
   / other-response
digest-response = dig-resp *(COMMA dig-resp)
dig-resp = username / realm / nonce / digest-uri
   / dresponse / algorithm / cnonce
   / opaque / message-qop
   / nonce-count / auth-param
username = "username" EQUAL username-value
username-value = quoted-string
digest-uri = "uri" EQUAL LDQUOT digest-uri-value RDQUOT
digest-uri-value = Request-URI
   ; by HTTP/1.1
message-qop = "qop" EQUAL qop-value
cnonce = "cnonce" EQUAL cnonce-value
cnonce-value = nonce-value
nonce-count = "nc" EQUAL nc-value
nc-value = 8LHEX
dresponse = "response" EQUAL request-digest
request-digest = LDQUOT 32LHEX RDQUOT
auth-param = auth-param-name EQUAL
   ( token / quoted-string )
auth-param-name = token
other-response = auth-scheme LWS auth-param *(COMMA auth-param)
auth-scheme = token

Bandwidth = "Bandwidth" HCOLON 1*DIGIT CRLF
Blocksize = "Blocksize" HCOLON 1*DIGIT CRLF
Cache-Control = "Cache-Control" HCOLON cache-directive CRLF *(COMMA cache-directive)
cache-directive = cache-rqst-directive / cache-rspns-directive
cache-rqst-directive = "no-cache" / "max-stale" [EQUAL delta-seconds] / "min-fresh" EQUAL delta-seconds / "only-if-cached" / cache-extension

cache-rspns-directive = "public" / "private" / "no-cache" / "no-transform" / "must-revalidate" / "proxy-revalidate" / "max-age" EQUAL delta-seconds / cache-extension
cache-extension = token [EQUAL (token / quoted-string)]
delta-seconds = 1*DIGIT

Connection-Credentials = "Connection-Credentials" HCOLON cred-chain CRLF
cred-chain = DQ RTSP-URI DQ SEMI base64
Connection = "Connection" HCOLON (connection-token) *(COMMA connection-token) CRLF
connection-token = token

Content-Base = "Content-Base" HCOLON RTSP-URI-Ref CRLF
Content-Encoding = "Content-Encoding" HCOLON content-coding *(COMMA content-coding)
Content-Language = "Content-Language" HCOLON language-tag *(COMMA language-tag)
language-tag = primary-tag *( "-" subtag )
primary-tag = 1*8ALPHA
subtag = 1*8ALPHA
Last-Modified = "Last-Modified" HCOLON RTSP-date
Location = "Location" HCOLON RTSP-URI
Pipelined-Requests = "Pipelined-Requests" HCOLON startup-id
startup-id = 1*8DIGIT
Proxy-Authenticate = "Proxy-Authenticate" HCOLON challenge
challenge = ("Digest" LWS digest-cln *(COMMA digest-cln))
/ other-challenge
other-challenge = auth-scheme LWS auth-param
/ (COMMA auth-param)
digest-cln = realm / domain / nonce
/ opaque / stale / algorithm
/ qop-options / auth-param
realm = "realm" EQUAL realm-value
realm-value = quoted-string
domain = "domain" EQUAL LDQUOT URI
*(1*SP URI)RDQUOT
URI = RTSP-URI / RTSP-URI-Ref
nonce = "nonce" EQUAL nonce-value
nonce-value = quoted-string
opaque = "opaque" EQUAL quoted-string
stale = "stale" EQUAL ("true" / "false")
algorithm = "algorithm" EQUAL ("MD5" / "MD5-sess" / token)
qop-options = "qop" EQUAL LDQUOT qop-value
*("", qop-value)RDQUOT
qop-value = "auth" / "auth-int" / token
Proxy-Require = "Proxy-Require" HCOLON feature-tag CRLF
*(COMMA feature-tag)
Proxy-Supported = "Proxy-Supported" HCOLON feature-tag
*(COMMA feature-tag)CRLF
Public = "Public" HCOLON Method*(COMMA Method)CRLF
Range = "Range" HCOLON ranges-list [exec-time] CRLF
ranges-list = ranges-spec*(COMMA ranges-spec)
exec-time = SEMI"time"EQUAL utc-time
ranges-spec = npt-range / utc-range / smpte-range
/ range-ext
range-ext = extension-format "=" range-value
range-value = 1*(rtsp-unreserved / quoted-string / ":" )
Referer = "Referer" HCOLON RTSP-URI-Ref
Require = "Require" HCOLON feature-tag-list CRLF
feature-tag-list = feature-tag*(COMMA feature-tag)
RTP-Info        =  "RTP-Info" HCOLON rtsp-info-spec
                   *(COMMA rtsp-info-spec) CRLF
rtsp-info-spec  =  stream-url 1*ssrc-parameter
stream-url      =  "url" EQUAL DQ RTSP-URI-Ref DQ
ssrc-parameter  =  LWS "ssrc" EQUAL ssrc HCOLON
                   ri-parameter *(SEMI ri-parameter)
ri-parameter     =  "seq" EQUAL 1*DIGIT
                   /  "rtptime" EQUAL 1*DIGIT
Retry-After     =  "Retry-After" HCOLON delta-seconds
                   [ comment ] *( SEI retry-param )
retry-param     =  ("duration" EQUAL delta-seconds)
                   /  generic-param
Scale           =  "Scale" HCOLON ["-"] 1*DIGIT [ "." *DIGIT ] CRLF
Speed           =  "Speed" HCOLON 1*DIGIT [ "." *DIGIT ] CRLF
Server          =  "Server" HCOLON ( product / comment )
                   *(LWS (product / comment)) CRLF
product         =  token [SLASH product-version]
product-version =  token
comment         =  LPAREN *( ctext / quoted-pair) RPAREN
Session         =  "Session" HCOLON session-id
                   [ SEMI "timeout" EQUAL delta-seconds ] CRLF
Supported       =  "Supported" HCOLON [feature-tag-list] CRLF
Timestamp       =  "Timestamp" HCOLON timestamp-value LWS [delay]
timestamp-value =  *DIGIT [ "." *DIGIT ]
delay           =  *DIGIT [ "." *DIGIT ]
Transport       =  "Transport" HCOLON transport-spec
                   *(COMMA transport-spec) CRLF
transport-spec  =  transport-id *tr-parameter
transport-id    =  trans-id-rtp / other-trans
trans-id-rtp    =  "RTP/" profile ["/" lower-transport]
                   ; no LWS is allowed inside transport-id
other-trans     =  token *(("/" token)
profile             = "AVP" / "SAVP" / "AVPF" / token
lower-transport     = "TCP" / "UDP" / token
tr-parameter        = SEMI ( "unicast" / "multicast" )
                     / SEMI "interleaved" EQUAL channel [ "-" channel ]
                     / SEMI "append"
                     / SEMI "ttl" EQUAL ttl
                     / SEMI "layers" EQUAL 1*DIGIT
                     / SEMI "ssrc" EQUAL ssrc *(SLASH ssrc)
                     / SEMI "client_ssrc" EQUAL ssrc
                     / SEMI "mode" EQUAL mode-spec
                     / SEMI "dest_addr" EQUAL addr-list
                     / SEMI "src_addr" EQUAL addr-list
                     / SEMI trn-param-ext
                     / SEMI "setup" EQUAL contrans-setup
                     / SEMI "connection" EQUAL contrans-con
contrans-setup      = "active" / "passive" / "actpass"
contrans-con        = "new" / "existing"
trn-param-ext       = par-name EQUAL trn-par-value
par-name            = token
trn-par-value       = *(rtsp-unreserved / DQ *TEXT DQ)
ttl                 = 1*3DIGIT ; 0 to 255
ssrc                = 8HEX
channel             = 1*3DIGIT
mode-spec           = ( DQ mode *(COMMA mode) DQ )
mode                = "PLAY" / token
addr-list           = quoted-addr *(SLASH quoted-addr)
quoted-addr         = DQ (host-port / extension-addr) DQ
host-port           = host [":" port]
                    / ";:" port
extension-addr      = 1*qdtext
host                = < As defined in RFC 3986>
port                = < As defined in RFC 3986>
Unsupported = "Unsupported" HCOLON feature-tag-list CRLF

User-Agent = "User-Agent" HCOLON ( product / comment )
0*(LWS (product / comment)) CRLF

Vary = "Vary" HCOLON ( "*" / field-name-list)
field-name-list = field-name *(COMMA field-name)
field-name = token
Via = "Via" HCOLON via-parm *(COMMA via-parm)
via-parm = sent-protocol LWS sent-by *( SEMI via-params )
via-params = via-ttl / via-maddr
/ via-received / via-branch
/ via-extension
ttl = "ttl" EQUAL tt1
via-maddr = "maddr" EQUAL host
via-received = "received" EQUAL (IPv4address / IPv6address)
IPv4address = < As defined in RFC 3986>
IPv6address = < As defined in RFC 3986>
via-branch = "branch" EQUAL token
via-extension = generic-param
sent-protocol = protocol-name SLASH protocol-version
SLASH transport-prot
protocol-name = "RTSP" / token
protocol-version = token
transport-prot = "UDP" / "TCP" / "TLS" / other-transport
other-transport = token
sent-by = host [ COLON port ]

WWW-Authenticate = "WWW-Authenticate" HCOLON challenge

21.3. SDP extension Syntax

This section defines in ABNF the SDP extensions defined for RTSP. See Appendix C for the definition of the extensions in text.

control-attribute = "a=control:" *SP RTSP-URI
a-range-def = "a=range:" ranges-spec CRLF
a-etag-def = "a=etag:" entity-tag CRLF
22. 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:

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 or controlled 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 URIs are opaque handles that do not necessarily have file system semantics, it is anticipated that many implementations will translate portions of the Request-URIs 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 sensitive. See [H15.1] for further recommendations.

Privacy Issues Connected to Accept Headers: Since many 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 each another, then it needs to 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 [RFC2616], as of this writing) and also of the previous RFC2068 [RFC2068], 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. See Section 22.1.

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. However, unless the RTSP signalling always are confidentiality protected, e.g. using TLS, an on-path attacker will be able to hijack a session. For real session security, client authentication needs to be performed.

Authentication: Servers SHOULD implement both basic and digest [RFC2617] authentication. In environments requiring tighter security for the control messages, the transport layer mechanism TLS (RFC 4346 [RFC4346]) 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.

Scope of Multicast: If RTSP is used to control the transmission of media onto a multicast network it is need to consider the scope that delivery has. RTSP supports the TTL Transport header parameter to indicate this scope. However such scope control is risk as it may be set to large and distribute media beyond the intended scope.

TLS through proxies: If one uses the possibility to connect TLS in multiple legs (Section 20.3 one really needs to be aware of the trust model. That procedure requires full faith and trust in all proxies that one allows to connect through. They are man in the middle and has access to all that goes on over the TLS connection. Thus it is important to consider if that trust model is acceptable in the actual application.

22.1. Remote 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 MUST only allow client-specified destinations for RTSP-initiated traffic flows if the server has ensured that the specified destination address accepts receiving media through different security mechanisms. Security mechanism that are acceptable in an increased generality are; verification of the client’s identity, either against a database of known users using RTSP authentication mechanisms (preferably digest authentication or stronger); a list of addresses that accept to be media destinations, especially considering user identity; and media path based verification.

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 media recipient (destination) 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.

One solution that performs the necessary verification of acceptance of media suitable for unicast based delivery is the ICE based NAT
traversal method described in [I-D.ietf-mmusic-rtsp-nat]. By using random passwords and username the probability of unintended indication as a valid media destination is very low. If the server include in its STUN requests a cookie (consisting of random material) that is the destination echo back the solution is also safe against having a off-path attacker being able to spoof the STUN checks. Leaving this solution vulnerable only to on-path attackers that can see the STUN requests go to the target of attack.

For delivery to multicast addresses there is need for another solution which is not specified here.
23. IANA Considerations

This section sets up a number of registries for RTSP 2.0 that should be maintained by IANA. For each registry there is a description on what it is required to 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 2.2 "Extending RTSP". There is also an IANA registration of two SDP attributes.

The sections describing how to register an item uses some of the requirements level described in RFC 2434 [RFC2434], 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, a particular company or group of companies, or an individual);
- 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);

23.1. Feature-tags

23.1.1. Description

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

The usage of feature-tags is explained in Section 11 and Section 13.1.
23.1.2. 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. The registration SHALL indicate if the feature-tag applies to clients, servers, or proxies only or any combinations of these. Any proprietary feature SHALL have as the first part of the name a vendor tag, which identifies the organization.

23.1.3. Registered entries

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

play.basic: The minimal implementation for playback operations according to Appendix D. Applies for both clients, servers and proxies.

play.scale: Support of scale operations for media playback. Applies only for servers.

play.speed: Support of the speed functionality for playback. Applies only for servers.

23.2. RTSP Methods

23.2.1. Description

What a method is, is described in section Section 13. Extending the protocol with new methods allow for totally new functionality.

23.2.2. 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 ABNF 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->S or S->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.

23.2.3. Registered Entries

This specification, RFCXXXX, registers 9 methods: DESCRIBE, GET_PARAMETER, OPTIONS, PAUSE, PLAY, REDIRECT, SETUP, SET_PARAMETER, and TEARDOWN.

23.3. RTSP Status Codes

23.3.1. Description

A status code is the three digit numbers used to convey information in RTSP response messages, see Section 8. The number space is limited and care should be taken not to fill the space.

23.3.2. 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.

23.3.3. Registered Entries

RFCXXX, registers the numbered status code defined in the ABNF entry "Status-Code" except "extension-code" in Section 21.2.2.

23.4. RTSP Headers

23.4.1. 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.
23.4.2. 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.
- An ABNF 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->S or S->C).
- How the header is to be handled by proxies.
- A description of the purpose of the header.

23.4.3. Registered entries

All headers specified in Section 16 in RFCXXXX are to be registered.

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

- x-wap-profile defined in [3gpp-26234].
- x-wap-profile-diff defined in [3gpp-26234].
- x-wap-profile-warning defined in [3gpp-26234].
- x-predecbufsize defined in [3gpp-26234].
- x-initpredecbufperiod defined in [3gpp-26234].
- x-initpostdecbufperiod defined in [3gpp-26234].
- 3gpp-videopostdecbufsize defined in [3gpp-26234].
- 3GPP-Link-Char defined in [3gpp-26234].
- 3GPP-Adaptation defined in [3gpp-26234].
- 3GPP-QoE-Metrics defined in [3gpp-26234].
3GPP-QoE-Feedback defined in [3gpp-26234].

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

23.5. Transport Header Registries

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

23.5.1. Transport Protocol Specification

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

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

This specification registers the following values:

RTP/AVP: Use of the RTP[RFC3550] protocol for media transport in combination with the "RTP profile for audio and video conferences with minimal control"[RFC3551] over UDP. The usage is explained in RFC XXXX, appendix Appendix B.1.

RTP/AVP/UDP: the same as RTP/AVP.


RTP/AVPF/UDP: the same as RTP/AVPF.

RTP/SAVP/UDP: the same as RTP/SAVP.


RTP/SAVPF/UDP: the same as RTP/SAVPF.

RTP/AVP/TCP: Use of the RTP[RFC3550] protocol for media transport in combination with the "RTP profile for audio and video conferences with minimal control"[RFC3551] over TCP. The usage is explained in RFC XXXX, appendix Appendix B.2.2.

RTP/AVPF/TCP: Use of the RTP[RFC3550] protocol for media transport in combination with the "Extended RTP Profile for RTCP-based Feedback (RTP/AVPF)"[RFC4585] over TCP. The usage is explained in RFC XXXX, appendix Appendix B.2.2.


RTP/SAVPF/TCP: Use of the RTP[RFC3550] protocol for media transport in combination with the "[I-D.ietf-avt-profile-savpf] over TCP. The usage is explained in RFC XXXX, appendix Appendix B.2.2.

23.5.2. Transport modes

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

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

This specification registers 1 value:

PLAY: See RFC XXXX.
23.5.3. Transport Parameters

A registry for parameters that may be included in the Transport header SHALL be defined with the following rules:

- Registering required a Open Standards document.
- A value definition that are following the ABNF token definition.
- A describing text that explains how the registered value are used in RTSP.

This specification registers all the transport parameters defined in Section 16.46.

23.6. 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 an IETF standard tracks document.
- A registration is required to contain a contact person.
- Name of the directive and a definition of the value, if any.
- Specification if it is an request or response directive.
- A describing text that explains how the cache directive is used for RTSP controlled media streams.

This specification registers the following values:

no-cache:

public:

private:

no-transform:

only-if-cached:

max-stale:
min-fresh:

must-revalidate:

proxy-revalidate:

max-age:

23.7. Accept-Credentials

The security framework’s TLS connection mechanism has two registerable entities.

23.7.1. Accept-Credentials policies

In Section 20.3.1 three policies for how to handle certificates. Further policies may be defined and SHALL be registered with IANA using the following rules:

- Registering requires an IETF standard tracks document.
- A registration is required name a contact person.
- Name of the policy.
- A describing text that explains how the policy works for handling the certificates.

This specification registers the following values:

Any
Proxy
User

23.7.2. Accept-Credentials hash algorithms

The Accept-Credentials header (See Section 16.2) allows for the usage of other algorithms for hashing the DER records of accepted entities. The registration of any future algorithm is expected to be extremely rare and could also be an interoperability problem. Therefore the XXX bare for registering new algorithms is placed intentional high.

Any registration of a new hash algorithm SHALL meet the following requirement:
23.8. Range header formats

The Range header allows for different range formats. New ones may be registered, but moderation should be applied as it makes interoperability more difficult. A registration SHALL fulfill the following requirements:

- A publicly available standards document.
- A ABNF definition of the range format that fulfils the "range-ext" definition.
- A Contact person for the registration.
- Rules for how one handles the range when using a negative Scale.

23.9. URI Schemes

This specification defines two URI schemes ("rtsp" and "rtsps") and reserves a third one ("rtspu"). Registrations are following RFC 4395.[RFC4395]

23.9.1. The rtsp URI Scheme

URI scheme name: rtsp

Status: Permanent

URI scheme syntax: See Section 21.2.1 of RFC XXXX.

URI scheme semantics: The rtsp scheme is used to indicate resources accessible through the usage of the Real-time Streaming Protocol (RTSP). RTSP allows different operations on the resource identified by the URI, but the primary purpose is the streaming delivery of the resource to a client. However the operations that are currently defined are: Describing the resource for the purpose of configuring the receiving entity (DESCRIBE), configuring the delivery method and its addressing (SETUP), controlling the delivery (PLAY and PAUSE), reading or setting of resource related parameters (SETPARAMETER and GET_PARAMETER, and termination of the session context created (TEARDOWN).
Encoding considerations: IRIs in this scheme are defined and need to be encoded as RTSP URIs when used within the RTSP protocol. That encoding is done according to RFC 3987 (XXX).

Applications/protocols that use this URI scheme name: RTSP 1.0 (RFC 2326), RTSP 2.0 (RFC XXXX)

Interoperability considerations: The change in URI syntax performed between RTSP 1.0 and 2.0 can create interoperability issues.

Security considerations: All the security threats identified in Section 7 of RFC 3986 applies also to this scheme. They needs to be reviewed and considered in any implementation utilizing this scheme.

Contact: Magnus Westerlund, magnus.westerlund@ericsson.com

Author/Change controller: IETF MMUSIC WG

References: RFC 2326, RFC 3986, RFC 3987, RFC XXXX

23.9.2. The rtsps URI Scheme

URI scheme name: rtsps

Status: Permanent

URI scheme syntax: See Section 21.2.1 of RFC XXXX.

URI scheme semantics: The rtsps scheme is used to indicate resources accessible through the usage of the Real-time Streaming Protocol (RTSP) over TLS. RTSP allows different operations on the resource identified by the URI, but the primary purpose is the streaming delivery of the resource to a client. However the operations that are currently defined are: Describing the resource for the purpose of configuring the receiving entity (DESCRIBE), configuring the delivery method and its addressing (SETUP), controlling the delivery (PLAY and PAUSE), reading or setting of resource related parameters (SETPARAMETER and GET_PARAMETER, and termination of the session context created (TEARDOWN).

Encoding considerations: IRIs in this scheme are defined and needs to be encoded as RTSP URIs when used within the RTSP protocol. That encoding is done according to RFC 3987.
Applications/protocols that use this URI scheme name: RTSP 1.0 (RFC 2326), RTSP 2.0 (RFC XXXX)

Interoperability considerations: The change in URI syntax performed between RTSP 1.0 and 2.0 can create interoperability issues.

Security considerations: All the security threats identified in Section 7 of RFC 3986 applies also to this scheme. They needs to be reviewed and considered in any implementation utilizing this scheme.

Contact: Magnus Westerlund, magnus.westerlund@ericsson.com

Author/Change controller: IETF MMUSIC WG

References: RFC 2326, RFC 3986, RFC 3987, RFC XXXX

23.9.3. The rtspu URI Scheme

URI scheme name: rtspu

Status: Permanent

URI scheme syntax: See Section 3.2 of RFC 2326.

URI scheme semantics: The rtspu scheme is used to indicate resources accessible through the usage of the Real-time Streaming Protocol (RTSP) over unreliable datagram transport. RTSP allows different operations on the resource identified by the URI, but the primary purpose is the streaming delivery of the resource to a client. However the operations that are currently defined are: Describing the resource for the purpose of configuring the receiving entity (DESCRIBE), configuring the delivery method and its addressing (SETUP), controlling the delivery (PLAY and PAUSE), reading or setting of resource related parameters (SETPARAMETER and GET_PARAMETER, and termination of the session context created (TEARDOWN).

Encoding considerations: IRIs in this scheme are defined and needs to be encoded as RTSP URIs when used within the RTSP protocol. That encoding is done according to RFC 3987.

Applications/protocols that use this URI scheme name: RTSP 1.0 (RFC 2326)
Interoperability considerations: The definition of the transport mechanism of RTSP over UDP has interoperability issues. That makes the usage of this scheme problematic.

Security considerations: All the security threats identified in Section 7 of RFC 3986 applies also to this scheme. They needs to be reviewed and considered in any implementation utilizing this scheme.

Contact: Magnus Westerlund, magnus.westerlund@ericsson.com

Author/Change controller: IETF MMUSIC WG

References: RFC 2326, RFC 3986, RFC 3987

23.10. SDP attributes

This specification defines two SDP [RFC4566] attributes that it is requested 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 URI
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 URIs.

Attribute name: etag
Long form: Entity Tag
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.
24. References

24.1. Normative References

[3gpp-26234]

[FIPS-pub-180-2]

[I-D.ietf-avt-profile-savpf]


24.2. Informative References


[ITU.H323.1996]

[NOSSDAV-1997-1]


[W3C.REC-PICS-labels]
Miller, J., Krauskopf, T., Resnick, P., and W. Treese, "PICS label distribution label syntax and communication protocols", W3C REC-PICS-labels-961031.

[W3C.REC-PICS-services]
Miller, J., Resnick, P., and D. Singer, "Rating services
and rating systems (and their machine readable
descriptions)", W3C REC-PICS-services-961031,
October 1996.
Appendix A. RTSP Protocol State Machine

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

The State machine is defined on a per session basis which is uniquely identified by the RTSP session identifier. The session may contain one 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.

The below state machine is a normative description of the protocols behavior. However, in case of ambiguity with the earlier parts of this specification, the description in the earlier parts SHALL take precedence.

A.1. States

The state machine contains three 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: Session is ready to start playing.

Play: Session is playing, i.e. sending media stream data in the direction S->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->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 changes to.

The response to valid request meeting the requisites is normally a 2xx (SUCCESS) unless other noted in the response column. The exceptions needs to 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 3rr 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 knows that the session has 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 URI is equal to the media URI or a specified presentation URI. For NRM > 1 the presentation URI MUST be other than any of the medias that are part of the session. This applies to all states.

<table>
<thead>
<tr>
<th>Event</th>
<th>Prerequisite</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE</td>
<td>Needs REDIRECT</td>
<td>3rr, Redirect</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td></td>
<td>200, Session description</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Session ID</td>
<td>200, Reset session timeout timer</td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
<td>200</td>
</tr>
<tr>
<td>SETPARAMETER</td>
<td>Valid parameter</td>
<td>200, change value of parameter</td>
</tr>
<tr>
<td>GET_PARAMETER</td>
<td>Valid parameter</td>
<td>200, return value of parameter</td>
</tr>
</tbody>
</table>

Table 13: None state-machine changing events

The methods in Table 13 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. Also all of these methods that allows Session header will also update the keep-alive timer for the session.

<table>
<thead>
<tr>
<th>Action</th>
<th>Requisite</th>
<th>New State</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP</td>
<td></td>
<td>Ready</td>
<td>NRM=1, RP=0.0</td>
</tr>
<tr>
<td>SETUP</td>
<td>Needs Redirect</td>
<td>Init</td>
<td>3rr Redirect</td>
</tr>
<tr>
<td>S -&gt; C: REDIRECT</td>
<td>No Session hdr</td>
<td>Init</td>
<td>Terminate all SES</td>
</tr>
</tbody>
</table>

Table 14: State: Init

The initial state of the state machine, see Table 14 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 URI and/or server by a 3rr response.
<table>
<thead>
<tr>
<th>Action</th>
<th>Requisite</th>
<th>New State</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP</td>
<td>New URI</td>
<td>Ready</td>
<td>NRM +=1</td>
</tr>
<tr>
<td>SETUP</td>
<td>URI Setup prior</td>
<td>Ready</td>
<td>Change transport param</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>Prs URI,</td>
<td>Init</td>
<td>No session hdr, NRM = 0</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>md URI,NRM=1</td>
<td>Init</td>
<td>No Session hdr, NRM = 0</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>md URI,NRM&gt;1</td>
<td>Ready</td>
<td>Session hdr, NRM -= 1</td>
</tr>
<tr>
<td>PLAY</td>
<td>Prs URI, No range</td>
<td>Play</td>
<td>Play from RP</td>
</tr>
<tr>
<td>PLAY</td>
<td>Prs URI, Range</td>
<td>Play</td>
<td>According to range</td>
</tr>
<tr>
<td>PAUSE</td>
<td>Prs URI</td>
<td>Ready</td>
<td>Return PP</td>
</tr>
<tr>
<td>SC:REDIRECT</td>
<td>Range hdr</td>
<td>Ready</td>
<td>Set RedP</td>
</tr>
<tr>
<td>SC:REDIRECT</td>
<td>no range hdr</td>
<td>Init</td>
<td>Session is removed</td>
</tr>
<tr>
<td>Timeout</td>
<td></td>
<td>Init</td>
<td></td>
</tr>
<tr>
<td>RedP reached</td>
<td></td>
<td>Init</td>
<td>TEARDOWN of session</td>
</tr>
</tbody>
</table>

Table 15: State: Ready

In the Ready state, see Table 15, 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 URI) 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 the presentations URI the whole session is torn down. If a media URI 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 URI the session is removed. The number of media streams remaining after tearing down
a media stream determines the new state.

<table>
<thead>
<tr>
<th>Action</th>
<th>Requisite</th>
<th>New State</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAUSE</td>
<td>PrsURI</td>
<td>Ready</td>
<td>Set RP to present point</td>
</tr>
<tr>
<td>PP reached</td>
<td></td>
<td>Ready</td>
<td>RP = PP</td>
</tr>
<tr>
<td>End of media</td>
<td>All media</td>
<td>Play</td>
<td>Set RP = End of media</td>
</tr>
<tr>
<td>End of range</td>
<td></td>
<td>Play</td>
<td>Set RP = End of range</td>
</tr>
<tr>
<td>PLAY</td>
<td>Prs URI, No</td>
<td>Play</td>
<td>Play from present point</td>
</tr>
<tr>
<td>PLAY</td>
<td>Prs URI, Range</td>
<td>Play</td>
<td>According to range</td>
</tr>
<tr>
<td>SETUP</td>
<td>New URI</td>
<td>Play</td>
<td>455</td>
</tr>
<tr>
<td>SETUP</td>
<td>Setuped URI</td>
<td>Play</td>
<td>455</td>
</tr>
<tr>
<td>SETUP</td>
<td>Setuped URI,</td>
<td>Play</td>
<td>Change transport param.</td>
</tr>
<tr>
<td></td>
<td>IFI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>Prs URI</td>
<td>Init</td>
<td>No session hdr</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>md URI,NRM=1</td>
<td>Init</td>
<td>No Session hdr, NRM=0</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>md URI</td>
<td>Play</td>
<td>455</td>
</tr>
<tr>
<td>SC:REDIRECT</td>
<td>Range hdr</td>
<td>Play</td>
<td>Set RedP</td>
</tr>
<tr>
<td>SC:REDIRECT</td>
<td>no range hdr</td>
<td>Init</td>
<td>Session is removed</td>
</tr>
<tr>
<td>RedP reached</td>
<td></td>
<td>Init</td>
<td>TEARDOWN of session</td>
</tr>
<tr>
<td>Timeout</td>
<td></td>
<td>Init</td>
<td>Stop Media playout</td>
</tr>
</tbody>
</table>

Table 16: State: Play

The Play state table, see Table 16, is the largest. The table contains an number of requests that has presentation URI as a
prerequisite on the Request-URI, 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 an 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 needs to be prepared for getting error message as the state may or may not have changed.
Appendix B. Media Transport Alternatives

This section defines how certain combinations of protocols, profiles and lower transports are used. This includes the usage of the Transport header’s source and destination address parameters “src_addr” and "dest_addr".

B.1. RTP

This section defines the interaction of RTSP with respect to the RTP protocol [RFC3550]. It also defines any necessary media transport signalling with regards to RTP.

The available RTP profiles and lower layer transports are described below along with rules on signalling the available combinations.

B.1.1. AVP

The usage of the "RTP Profile for Audio and Video Conferences with Minimal Control" [RFC3551] when using RTP for media transport over different lower layer transport protocols is defined below in regards to RTSP.

One such case is defined within this document, the use of embedded (interleaved) binary data as defined in Section 14. The usage of this method is indicated by include the "interleaved" parameter.

When using embedded binary data the "src_addr" and "dest_addr" 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 describes sending of RTP [RFC3550] over lower transport layer UDP [RFC0768] according to the profile "RTP Profile for Audio and Video Conferences with Minimal Control" defined in RFC 3551 [RFC3551]. This profiles requires 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. Embedding of RTP data with the RTSP messages, in accordance with Section 14, SHOULD NOT be performed when RTSP messages are transported over unreliable transport protocols, like UDP [RFC0768].

The RTP/UDP and RTCP/UDP flows can be established using the Transport header’s "src_addr", and "dest_addr" parameters.

In RTSP PLAY mode, the transmission of RTP packets from client to server is unspecified. The behavior in regards to such RTP packets
MAY be defined in future.

The "src_addr" and "dest_addr" parameters are used in the following way for media playback, i.e. Mode=PLAY:

- The "src_addr" and "dest_addr" parameters MUST contain either 1 or 2 address specifications.
- Each address specification for RTP/AVP/UDP or RTP/AVP/TCP MUST contain either:
  - both an address and a port number, or
  - a port number without an address.
- 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 "dest_addr" 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 "dest_addr" parameter. If no second pair is specified 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 "src_addr" parameter. If no second pair is given RTCP SHALL NOT be sent.
- The RTP/UDP packets from the client to the server SHALL be sent to the address and port given by the first address and port pair of the "src_addr" parameter.
- RTP and RTCP Packets SHOULD be sent from the corresponding receiver port, i.e. RTCP packets from server should be sent from the "src_addr" parameters second address port pair.

B.1.3. AVPF/UDP

The RTP profile "Extended RTP Profile for RTCP-based Feedback (RTP/AVPF)" [RFC4585] MAY be used as RTP profiles in session using RTP. All that is defined for AVP SHALL also apply for AVPF.

The usage of AVPF is indicated by the media initialization protocol.
used. In the case of SDP it is indicated by media lines (m=) containing the profile RTP/AVPF. That SDP MAY also contain further AVPF related SDP attributes configuring the AVPF session regarding reporting interval and feedback messages that shall be used that SHALL be followed.

B.1.4. SAVP/UDP

The RTP profile "The Secure Real-time Transport Protocol (SRTP)" [RFC3711] is an RTP profile (SAVP) that MAY be used in RTSP sessions using RTP. All that is defined for AVP SHALL also apply for SAVP.

The usage of SRTP requires that a security association is established. The RECOMMENDED mechanism for establishing that security association is to use MIKEY with RTSP as defined in RFC 4567 [RFC4567].

B.1.5. SAVPF/UDP

The RTP profile "Extended Secure RTP Profile for RTCP-based Feedback (RTP/SAVPF)" [I-D.ietf-avt-profile-savpf] is an RTP profile (SAVPF) that MAY be used in RTSP sessions using RTP. All that is defined for AVP SHALL also apply for SAVPF.

The usage of SRTP requires that a security association is established. The RECOMMENDED mechanism for establishing that security association is to use MIKEY[RFC3830] with RTSP as defined in RFC 4567 [RFC4567].

B.1.6. RTCP usage with RTSP

RTCP has several usages when RTP is used for media transport as explained below. Due to that RTCP SHALL be supported if an RTSP agent handles RTP.

B.1.6.1. Media synchronization

RTCP provides media synchronization and clock drift compensation. The first is available from RTP-Info header to accomplish the initial synchronization. But to be able to handle any clockdrift between the media streams, RTCP is needed.

B.1.6.2. RTSP Session keep-alive

RTCP traffic from the RTSP client to the RTSP server SHALL function as keep-alive. Which requires an RTSP server supporting RTP to use the received RTCP packets as indications that the client desires the related RTSP session to be kept alive.
B.1.6.3. Bit-rate adaptation

RTCP Receiver Reports and any additional feedback from the client SHALL be used adapt the bit-rate used over the transport for all cases when RTP is sent over UDP. A RTP sender without reserved resources SHALL NOT use more than its fair share of the available resources. This can be determined by comparing on short to medium term (some seconds) the used bit-rate and adapt it so that the RTP sender sends at a bit-rate comparable to what a TCP sender would achieve on average over the same path.

B.2. RTP over TCP

Transport of RTP over TCP can be done in two ways, over independent TCP connections using RFC 4571 [RFC4571] or interleaved in the RTSP control connection. In both cases the protocol SHALL be "rtp" and the lower layer SHALL be TCP. The profile may be any of the above specified ones; AVP, AVPF, SAVP or SAVPF.

B.2.1. Interleaved RTP over TCP

The use of embedded (interleaved) binary data transported on the RTSP connection is possible as specified in Section 14. When using this declared combination of interleaved binary data the RTSP messages MUST be transported over TCP. TLS may or may not be used.

One should however consider that this will result that all media streams go through any proxy. Using independent TCP connections can avoid that issue.

B.2.2. RTP over independent TCP

In this Appendix, we describe the sending of RTP [RFC3550] over lower transport layer TCP [RFC0793] according to "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport" [RFC4571]. This Appendix adapts the guidelines for using RTP over TCP within SIP/SDP [RFC4145] to work with RTSP.

A client codes the support of RTP over independent TCP by specifying an RTP/AVP/TCP transport option without an interleaved parameter in the Transport line of a SETUP request. This transport option MUST include the "unicast" parameter.

If the client wishes to use RTP with RTCP, two ports (or two address/port pairs) are specified by the dest_addr parameter. If the client wishes to use RTP without RTCP, one port (or one address/port pair) is specified by the dest_addr parameter. Ordering rules of dest_addr...
ports follow the rules for RTP/AVP/UDP.

If the client wishes to play the active role in initiating the TCP connection, it MAY set the "setup" parameter (See Section 16.46) on the Transport line to be "active", or it MAY omit the setup parameter, as active is the default. If the client signals the active role, the ports for all dest_addr values MUST be set to 9 (the discard port).

If the client wishes to play the passive role in TCP connection initiation, it MUST set the "setup" parameter on the Transport line to be "passive". If the client is able to assume the active or the passive role, it MUST set the "setup" parameter on the Transport line to be "actpass". In either case, the dest_addr port value for RTP MUST be set to the TCP port number on which the client is expecting to receive the RTP stream connection, and the dest_addr port value for RTCP MUST be set to the TCP port number on which the client is expecting to receive the RTCP stream connection.

If upon receipt of a non-interleaved RTP/AVP/TCP SETUP request, a server decides to accept this requested option, the 2xx reply MUST contain a Transport option that specifies RTP/AVP/TCP (without using the interleaved parameter, and with using the unicast parameter). The dest_addr parameter value MUST be echoed from the parameter value in the client request unless the destination address (only port) was not provided in which case the server MAY include the source address of the RTSP TCP connection with the port number unchanged.

In addition, the server reply MUST set the setup parameter on the Transport line, to indicate the role the server will play in the connection setup. Permissible values are "active" (if a client set "setup" to "passive" or "actpass") and "passive" (if a client set "setup" to "active" or "actpass").

If a server sets "setup" to "passive", the "src_addr" in the reply MUST indicate the ports the server is willing to receive an RTP connection (and if the client requested an RTCP connection by specifying two dest_addr ports or address/port pairs) and RTCP connection. If a server sets "setup" to "active", the ports specified in "src_addr" MUST be set to 9. The server MAY use the "ssrc" parameter, following the guidance in Section 16.46. Port ordering for src_addr follows the rules for RTP/AVP/UDP.

For cases when servers have a public IP-address it is RECOMMENDED that the server take the passive role and the client the active role. This helps in cases when the client is behind a NAT.

After sending (receiving) a 2xx reply for a SETUP method for a non-
interleaved RTP/AVP/TCP media stream, the active party SHOULD initiate the TCP connection as soon as possible. The client SHALL NOT send a PLAY request prior to the establishment of all the TCP connections negotiated using SETUP for the session. In case the server receives a PLAY request in a session that has not yet established all the TCP connections, it SHALL respond using the 464 "Data Transport Not Ready Yet" (Section 15.4.16) error code.

Once the PLAY request for a media resource transported over non-interleaved RTP/AVP/TCP occurs, media begins to flow from server to client over the RTP TCP connection, and RTCP packets flow bidirectionally over the RTCP TCP connection. As in the RTP/UDP case, client to server traffic on the TCP port is unspecified by this memo. The packets that travel on these connections SHALL be framed using the protocol defined in [RFC4571], not by the framing defined for interleaving RTP over the RTSP control connection defined in Section 14.

A successful PAUSE request for a media being transported over RTP/AVP/TCP pauses the flow of packets over the connections, without closing the connections. A successful TEARDOWN request signals that the TCP connections for RTP and RTCP are to be closed as soon as possible.

Subsequent SETUP requests on an already-SETUP RTP/AVP/TCP URI may be ambiguous in the following way: does the client wish to open up new TCP RTP and RTCP connections for the URI, or does the client wish to continue using the existing TCP RTP and RTCP connections? The client SHOULD use the "connection" parameter (defined in Section 16.46) on the Transport line to make its intention clear in the regard (by setting "connection" to "new" if new connections are needed, and by setting "connection" to "existing" if the existing connections are to be used). After a 2xx reply for a SETUP request for a new connection, parties should close the pre-existing connections, after waiting a suitable period for any stray RTP or RTCP packets to arrive.

Below, we rewrite part of the example media on demand example shown in Section 19.1 to use RTP/AVP/TCP non-interleaved:
C->M: DESCRIBE rtsp://example.com/twister.3gp RTSP/2.0
CSeq: 1
User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
CSeq: 1
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 257
Content-Base: rtsp://example.com/twister.3gp/
Expires: 24 Jan 1997 15:35:06 GMT

v=0
o= 2890844256 2890842807 IN IP4 192.0.2.5
s=RTSP Session
i=An Example of RTSP Session Usage
e=adm@example.com
a=control: *
a=range: npt=0-0:10:34.10
t=0 0
m=audio 0 RTP/AVP 0
a=control: trackID=1

C->M: SETUP rtsp://example.com/twister.3gp/trackID=1 RTSP/2.0
CSeq: 2
User-Agent: PhonyClient/1.2
Require: play.basic
Transport: RTP/AVP/TCP;unicast;dest_addr="":9"/":9"
setup=active;connection=new
Accept-Ranges: NPT, SMPTE, UTC

M->C: RTSP/2.0 200 OK
CSeq: 2
Server: PhonyServer/1.0
Transport: RTP/AVP/TCP;unicast;dest_addr=":9"/":9";
         src_addr="192.0.2.5:9000"/"192.0.2.5:9001"
         setup=passive;connection=new;ssrc=93CB001E
Session: 12345678
Expires: 24 Jan 1997 15:35:12 GMT
Date: 23 Jan 1997 15:35:12 GMT
Accept-Ranges: NPT

C->M: TCP Connection Establishment

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
CSeq: 4
User-Agent: PhonyClient/1.2
Range: npt=0-10, npt=30-
Session: 12345678

M->C: RTSP/2.0 200 OK
CSeq: 4
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:35:14 GMT
Session: 12345678
Range: npt=0-10, npt=30-623.10
RTP-Info: url="rtsp://example.com/twister.3gp/trackID=1";
         ssrc=4F312DD8:seq=54321;rtptime=2876889

B.2.3. Handling NPT Jumps in the RTP Media Layer

RTSP allows media clients to control selected, non-contiguous sections of media presentations, rendering those streams with an RTP media layer[RFC3550]. Such control allows jumps to be created in NPT timeline of the RTSP session. For example, jumps in NPT can be caused by multiple ranges in the range specifier of a PLAY request or through a "seek" operation on an RTSP session which involves a PLAY, PAUSE, PLAY scenario where a new NPT is set for the session. 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.

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.

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.

C->S: PLAY rtsp://example.com/fizzle RTSP/2.0
    CSeq: 4
    Session: abcdefg
    Range: npt=10-15
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 4
    Session: abcdefg
    Range: npt=10-15
    RTP-Info: url="rtsp://example.com/fizzle/audiotrack"
              ssrc=0D12F123:seq=0;rtptime=0

The ensuing RTP data stream is depicted below:

S -> C: RTP packet - seq = 0,  rtptime = 0,     NPT time = 10s
S -> C: RTP packet - seq = 1,  rtptime = 800,   NPT time = 10.1s
    . . .
S -> C: RTP packet - seq = 49, rtptime = 39200, NPT time = 14.9s

Immediately after the end of the play range, the client follows up
with a request to PLAY from a new NPT.

C->S: PLAY rtsp://example.com/fizzle RTSP/2.0
    CSeq: 5
    Session: abcdefg
    Range: npt=18-20
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 5
    Session: abcdefg
    Range: npt=18-20
    RTP-Info: url="rtsp://example.com/fizzle/audiotrack"
              ssrc=0D12F123:seq=50;rtptime=40100

The ensuing RTP data stream is depicted below:
S->C: RTP packet - seq = 50, rtptime = 40100, NPT time = 18s
S->C: RTP packet - seq = 51, rtptime = 40900, NPT time = 18.1s
... S->C: RTP packet - seq = 69, rtptime = 55300, NPT time = 19.9s

In this example, first, NPT 10 through 15 is played, then the client request the server to 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 40,100 through 55,200. While there is a gap in the NPT, there is no gap in the sequence number space of the RTP data stream.

The RTP timestamp gap is present in the above example due to the time it takes to perform the second play request, in this case 12.5 ms (100/8000). To avoid this gap in playback due to the time it takes to perform RTSP requests, a PLAY request with multiple ranges needs to be specified. That would result in the following example:

C->S: PLAY rtsp://example.com/fizzle RTSP/2.0
    CSeq: 4
    Session: abcdefg
    Range: npt=10-15;npt=18-20
    User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
    CSeq: 4
    Session: abcdefg
    Range: npt=10-15
    RTP-Info: url="rtsp://example.com/fizzle/audiotrack"
             ssrc=0D12F123:seq=0;rtptime=0

The ensuing RTP data stream is depicted below:

S -> C: RTP packet - seq = 0, rtptime = 0, NPT time = 10s
S -> C: RTP packet - seq = 1, rtptime = 800, NPT time = 10.1s
... S -> C: RTP packet - seq = 49, rtptime = 39200, NPT time = 14.9s
S -> C: RTP packet - seq = 50, rtptime = 40100, NPT time = 18s
S -> C: RTP packet - seq = 51, rtptime = 40900, NPT time = 18.1s
... S -> C: RTP packet - seq = 69, rtptime = 55300, NPT time = 19.9s
B.2.4. Handling RTP Timestamps after PAUSE

During a PAUSE / PLAY interaction in an RTSP session, the duration of time for which the RTP transmission was halted MUST be reflected in the RTP timestamp of each RTP stream. The duration can be calculated for each RTP stream as the time elapsed from when the last RTP packet was sent before the PAUSE request was received and when the first RTP packet was sent after the subsequent PLAY request was received. The duration includes all latency incurred and processing time required to complete the request.

The RTP RFC [RFC3550] states that: The RTP timestamp for each unit[packet] would be related to the wallclock time at which the unit becomes current on the virtual presentation timeline.

In order to satisfy the requirements of [RFC3550], the RTP timestamp space needs to increase continuously with real time. While this is not optimal for stored media, it is required for RTP and RTCP to function as intended. Using a continuous RTP timestamp space allows the same timestamp model for both stored and live media and allows better opportunity to integrate both types of media under a single control.

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.

C->S: PLAY rtsp://example.com/fizzle RTSP/2.0
   CSeq: 4
   Session: abcdefg
   Range: npt=10-15
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 4
   Session: abcdefg
   Range: npt=10-15
   RTP-Info: url="rtsp://example.com/fizzle/audiotrack"
            ssrc=0D12F123:seq=0;rtptime=0

The ensuing RTP data stream is depicted below:

S -> C: RTP packet - seq = 0, rtptime = 0,  NPT time = 10s
S -> C: RTP packet - seq = 1, rtptime = 800, NPT time = 10.1s
S -> C: RTP packet - seq = 2, rtptime = 1600, NPT time = 10.2s
S -> C: RTP packet - seq = 3, rtptime = 2400, NPT time = 10.3s

The client then sends a PAUSE request:
C->S: PAUSE rtsp://example.com/fizzle RTSP/2.0
   CSeq: 5
   Session: abdcdefg
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 5
   Session: abdcdefg
   Range: npt=10.4-15

20 seconds elapse and then the client sends a PLAY request.  In addition the server requires 15 ms to process the request:

C->S: PLAY rtsp://example.com/fizzle RTSP/2.0
   CSeq: 6
   Session: abdcdefg
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 6
   Session: abdcdefg
   Range: npt=10.4-15
   RTP-Info: url="rtsp://example.com/fizzle/audiotrack"
             ssrc=0D12F123:seq=4;rtptime=164400

The ensuing RTP data stream is depicted below:

   S -> C: RTP packet - seq = 4, rtptime = 164400, NPT time = 10.4s
   S -> C: RTP packet - seq = 5, rtptime = 165200, NPT time = 10.5s
   S -> C: RTP packet - seq = 6, rtptime = 166000, NPT time = 10.6s

First, NPT 10 through 10.3 is played, then a PAUSE is received by the server.  After 20 seconds a PLAY is received by the server which take 15ms to process.  The duration of time for which the session was paused is reflected in the RTP timestamp of the RTP packets sent after this PLAY request.

A client can use the RTSP range header and RTP-Info header to map NPT time of a presentation with the RTP timestamp.

Note: In RFC 2326 [RFC2326], this matter was not clearly defined and was misunderstood commonly.  However for RTSP 2.0 it is expected that this will be handled correctly and no exception handling will be required.
B.2.5.  RTSP / RTP Integration

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

B.2.6.  Scaling with RTP

For scaling (see Section 16.40), 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 16.41) 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.

Note: The above scaling puts requirements on the media codec or a
media stream to support it. For example motion JPEG or other non-
predictive video coding can easier handle the above example.

B.2.7.  Maintaining NPT synchronization with RTP timestamps

The client can maintain a correct display of NPT (Normal Play Time)
by noting the RTP timestamp value of the first packet arriving after
repositioning. The sequence parameter of the RTP-Info
(Section 16.39) header provides the first sequence number of the next
segment.

B.2.8.  Continuous Audio

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

B.2.9.  Multiple Sources in an RTP Session

Note that more than one SSRC MAY be sent in the media stream. If it
happens all sources are expected to be rendered simultaneously.

B.2.10.  Usage of SSRCs and the RTCP BYE Message During an RTSP Session

The RTCP BYE message indicates the end of use of a given SSRC. If
all sources leave an RTP session, it can, in most cases, be assumed
to have ended. Therefore, a client or server SHALL NOT send a RTCP
BYE message until it has finished using a SSRC. A server SHOULD keep
using a SSRC until the RTP session is terminated. Prolonging the use
of a SSRC allows the established synchronization context associated
with that SSRC to be used to synchronize subsequent PLAY requests even if the PLAY response is late.

An SSRC collision with the SSRC that transmits media does also have consequences, as it will force the media sender to change its SSRC in accordance with the RTP specification [RFC3550]. This will result in a loss of synchronization context, and require any receiver to wait for RTCP sender reports for all media requiring synchronization before being able to play out synchronized. Due to these reasons a client joining a session should take care to not select the same SSRC as the server. Any SSRC signalled in the Transport header SHOULD be avoided. A client detecting a collision prior to sending any RTP or RTCP messages can also select a new SSRC.

B.3. 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 section 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 (Section 23) for information how to register new attributes.
The Session Description Protocol (SDP, [RFC4566]) may be used to describe streams or presentations in RTSP. This description is typically returned in reply to a DESCRIBE request on an URI from a server to a client, or received via HTTP from a server to a client.

This appendix describes how an SDP file determines the operation of an RTSP session. SDP as is 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). However the SDP extension "Grouping of Media Lines in the Session Description Protocol (SDP)" [RFC3388] may provide such functionality depending on need. Also future grouping semantics may in the future be developed.

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 4566 [RFC4566]):

C.1.1. Control URI

The "a=control:" attribute is used to convey the control URI. This attribute is used both for the session and media descriptions. If used for individual media, it indicates the URI to be used for controlling that particular media stream. If found at the session level, the attribute indicates the URI for aggregate control (presentation URI). The session level URI SHALL be different from any media level URI. The presence of a session level control attribute SHALL be interpreted as support for aggregated control. The control attribute SHALL be present on media level unless the presentation only contains a single media stream, in which case the attribute MAY only be present on the session level.

ABNF for the attribute is defined in Section 21.3.

Example:

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

This attribute MAY contain either relative or absolute URIs, following the rules and conventions set out in RFC 3986 [RFC3986]. Implementations SHALL look for a base URI in the following order:

1. the RTSP Content-Base field;
2. the RTSP Content-Location field;

3. the RTSP Request-URI.

If this attribute contains only an asterisk (*), then the URI SHALL be treated as if it were an empty embedded URI, and thus inherit the entire base URI.

The URI handling for SDPs from container files need special consideration. For example let's assume that a container file has the URI: "rtsp://example.com/container.mp4". Let's further assume this URI is the base URI, and that there is a absolute media level URI: "rtsp://example.com/container.mp4/trackID=2". A relative media level URI that resolves in accordance with RFC 3986 [RFC3986] to the above given media URI is: "container.mp4/trackID=2". It is usually not desirable to need to include in or modify the SDP stored within the container file with the server local name of the container file. To avoid this, one can modify the base URI used to include a trailing slash, e.g. "rtsp://example.com/container.mp4/". In this case the relative URI for the media will only need to be: "trackID=2". However this will also mean that using "*" in the SDP will result in control URI including the trailing slash, i.e. "rtsp://example.com/container.mp4/".

Note: The usage of TrackID in the above is not an standardized form, but one example out of several similar strings such as TrackID, Track_ID, StreamID that is used by different server vendors to indicate a particular piece of media inside a container file.

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 over multicast, the port number indicated SHOULD be used for reception. The client MAY try to override the destination port, through the Transport header. The servers MAY allow this, the response will indicate if allowed or not. If the session is unicast, the port number is the ones RECOMMENDED by the server to the client, about which receiver ports to use; the client MUST still include its receiver ports in its SETUP request. The client MAY ignore this recommendation. If the server has no preference, it SHOULD set the port number value to zero.

The "m=" lines contain information about what transport protocol, profile, and possibly lower-layer is to be used for the media stream. The combination of transport, profile and lower layer, like RTP/AVP/UDP needs to be defined for how to be used with RTSP. The currently...
defined combinations are defined in Appendix B, further combinations MAY be specified.

Usage of grouping of media lines [RFC3388] to determine which media lines should or should not be included in a RTSP session is unspecified.

Example:

m=audio 0 RTP/AVP 31

C.1.3. Payload Type(s)

The payload type(s) are specified in the "m=" line. In case the payload type is a static payload type from RFC 3551 [RFC3551], no other information may be 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 3551 (Sections 5 and 6), or an MIME type registered with IANA, or an experimental encoding as specified in SDP (RFC 4566 [RFC4566]). Codec-specific parameters are not specified in this field, but rather in the "fmtp" attribute described below.

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 some of the format specific parameters may be specified outside of the fmtp parameters, like for example the "ptime" attribute for most audio encodings.

C.1.5. Directionality of media stream

The SDP attributes "a=sendrecv", "a=recvonly" and "a=sendonly" provides instructions on which direction the media streams flow within a session. When using RTSP the SDP can be delivered to a client using either RTSP DESCRIBE or a number of RTSP external methods, like HTTP, FTP, and email. Based on this the SDP applies to how the RTSP client will see the complete session. Thus for media streams delivered from the RTSP server to the client would be given the "a=recvonly" attribute.

The direction attributes are not commonly used in SDPs for RTSP, but may occur. "a=recvonly" in a SDP provided to the RTSP client SHALL indicate that media delivery will only occur in the direction from the RTSP server to the client. In SDP provided to the RTSP client that lacks any of the directionality attributes (a=recvonly,
a=sendonly, a=sendrecv) SHALL behave as if the "a=recvonly" attribute was received. Note that this overrules the normal default rule defined in SDP [RFC4566]. The usage of "a=sendonly" or "a=sendrecv" is not defined, nor is the interpretation of SDP by other entities than the RTSP client.

C.1.6. Range of Presentation

The "a=range" attribute defines the total time range of the stored session or an individual media. Non-seekable live sessions can be indicated, while the length of live sessions can be deduced from the "t" and "r" SDP parameters.

The attribute is both a session and a 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, and SHOULD NOT be given at session level. If the attribute is present at both media level and session level the media level values SHALL be used.

Note: Usually one will specify the same length for all media, even if there isn’t media available for the full duration on all media. However that requires that the server accepts PLAY requests within that range.

Servers SHALL take care to provide RTSP Range (see Section 16.35) values that are consistent with what is presented in the SDP for the content. There are no reason for non dynamic content, like media clips provided on demand to have inconsistent values. Inconsistent values between the SDP and the actual values for the content handled by the server is likely to generate some failure, like 457 "Invalid Range", in case the client uses PLAY requests with a Range header. In case the content is dynamic in length and it is infeasible to provide a correct value in the SDP the server is recommended to describe this as non-seekable content (see below). The server MAY override that property in the response to a PLAY request using the correct values in the Range header.

The unit is specified first, followed by the value range. The units and their values are as defined in Section 4.4, Section 4.5 and Section 4.6 and MAY be extended with further formats. Any open ended range (start-), i.e. without stop range, is of unspecified duration and SHALL be considered as non-seekable content unless this property is overridden. Multiple instances carrying different clock formats MAY be included at either session or media level.

ABNF for the attribute is defined in Section 21.3.
Examples:

a=range:npt=0-34.4368
a=range:clock=19971113T2115-19971113T2203
Non seekable stream of unknown duration:
a=range:npt=0-

C.1.7.  Time of Availability

The "t=" field MUST contain suitable values for the start and stop times for both aggregate and non-aggregate stream 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.

For sessions that are of live type, i.e. specific start time, unknown stop time, likely unseekable, the "t=" and "r=" field SHOULD be used to indicate the start time of the event. The stop time SHOULD be given so that the live event will have ended at that time, while still not be unnecessary long into the future.

C.1.8.  Connection Information

In SDP, the "c=" field contains the destination address for the media stream. For on-demand unicast streams and some multicast streams, the destination address MAY be specified by the client via the SETUP request, thus overriding any specified address. To identify streams without a fixed destination address, where the client is required to specify a destination address, the "c=" field SHOULD be set to a null value. For addresses of type "IP4", this value SHALL be "0.0.0.0", and for type "IP6", this value SHALL be "0:0:0:0:0:0:0:0", i.e. the unspecified address according to RFC 3513 [RFC3513].

C.1.9.  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 16.24) 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.

ABNF for the attribute is defined in Section 21.3.

Example:
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 no session level "a=control:" attribute is specified. For a SDP with multiple media sections specified, each section will have its own control URI specified via the "a=control:" attribute.

Example:

v=0
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 URI 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 URIs, and a session-level "a=control:" attribute which is used as the Request-URI for aggregate control. If the media-level URI is relative, it is resolved to absolute URIs according to Appendix C.1.1 above.
Example:

C->M: DESCRIBE rtsp://example.com/movie RTSP/2.0
    CSeq: 1
    User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
    CSeq: 1
    Date: 23 Jan 1997 15:35:06 GMT
    Content-Type: application/sdp
    Content-Base: rtsp://example.com/movie/
    Content-Length: 228

v=0
o=- 2890844256 2890842807 IN IP4 192.0.2.211
s=I contain
i=<more info>
e=adm@example.com
  c=IN IP4 0.0.0.0
t=0 0
a=control:*
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 URIs rtsp://example.com/movie/trackID=1 and rtsp://example.com/movie/trackID=2 to set up the video and audio streams, respectively. The URI rtsp://example.com/movie/, which is resolved from the "*", controls the whole presentation (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.

C.4. RTSP external SDP delivery

There are some considerations that needs to be made when the session description is delivered to client outside of RTSP, for example in HTTP or email.

First of all the SDP needs to contain absolute URIs, relative will in most cases not work as the delivery will not correctly forward the base URI. And as SDP might be temporarily stored on file system before being loaded into an RTSP capable client, thus if possible to
transport the base URI it still would need to be merged into the file.

The writing of the SDP session availability information, i.e. "t=" and "r=", needs to be carefully considered. When the SDP is fetched by the DESCRIBE method, the probability that it is valid is very high. However the same are much less certain for SDPs distributed using other methods. Therefore the publisher of the SDP should take care to follow the recommendations about availability in the SDP specification [RFC4656].
Appendix D. Minimal RTSP Implementation

This section defines the minimal implementation requirements for RTSP agents.

D.1. Minimal Core Implementation

The minimal core implementation is what is required to negotiate the usage of any other features. A minimal core implementation is not supporting any other feature set will be useless as the minimal implementation doesn’t deliver any service. All feature sets SHALL include the minimal core.

A minimal core implementation SHALL support the following functionalities:

 o Establishing a connection between RTSP agents using TCP.
 o Implement the reception and response to the OPTIONS method.
 o Implement the handling of all headers mandatory or conditional in regards to the usage of the OPTIONS method. See Table 9 and Table 10. This include at least the capability to ignore unknown headers.
 o Implement the headers related to capability negotiation and exchange:
   * Require
   * Supported
   * Proxy-Require
   * Proxy-Supported
   * Unsupported

D.2. Recommended Core Implementation

A RTSP Agent is also RECOMMENDED to support the following:

 o RTSP basic and digest authentication: The 401 response, the WWW-Authenticate and Authorization headers, and both Basic and Digest authentication methods as defined by [RFC2617].
 o Secure RTSP message transport as specified by Appendix D.4.
D.3. The Basic Playback Feature Support

This section defines what is required to be supported for clients, proxies and servers to be supporting the "play.basic" feature-tag.

D.3.1. Client

A play.basic supporting client SHALL implement the following:

- The RTSP methods as required by Table 7.
- All the RTSP headers that are required or conditional in requests or responses to methods required to be supported according to Table 9, Table 10, Table 11, and Table 12 and in addition the following headers:
  * Content-Base
  * Content-Encoding and at least the Identity method.
  * Content-Location
  * Location
  * Range and the npt time format
  * RTP-Info
- Handling of all Status code categories.
- Media delivery using RTP/AVP over UDP.

A play.basic supporting client is also RECOMMENDED to support the following:

- Expires header
- From header

D.3.2. Server

A play.basic supporting server SHALL implement the following:

- The RTSP methods as required by Table 7.
- Reception and responding to all headers specified in Section 16. The implementation of functionality provided by all these headers with the following exceptions:
* Scale
* Speed
* Blocksize

- Media delivery using RTP/AVP over UDP.

A play.basic supporting Server is also RECOMMENDED to support the following:

- XXX Editor’s note: empty element in minimal.text!

### D.3.3. Proxy

A play.basic supporting proxy SHALL implement the following:

- At least passing through all the methods listed in Table 7.

- The handling of all RTSP headers that are required to be handled by the server and clients supporting "play.basic" and in addition the following headers:
  
  * Cache-Control
  * Expires
  * Via

### D.4. Secure Transport

Any Client, Proxy or Server supporting secure transport of RTSP messages and usage of the "rtsp" URI scheme SHALL implement:

- The Accept-Credentials and Connection-Credentials headers;

- TLS over TCP.
Appendix E. Requirements for Unreliable Transport of RTSP

This section provides anyone intending to define how to transport of RTSP messages over an unreliable transport protocol with some information learned by the attempt in RFC 2326 [RFC2326]. RFC 2326 define both an URI scheme and some basic functionality for transport of RTSP messages over UDP, however it was not sufficient for reliable usage and successful interoperability.

The RTSP scheme defined for unreliable transport of RTSP messages was "rtspu". It has been reserved by this specification as at least one commercial implementation exist, thus avoiding any collisions in the name space.

The following considerations should exist for operation of RTSP over an unreliable transport protocol:

- Request shall be acknowledged by the receiver. If there is no acknowledgement, the sender may resend the same message after a timeout of one round-trip time (RTT). Any retransmissions due to lack of acknowledgement must carry the same sequence number as the original request.

- The round-trip time can be estimated as in TCP (RFC 1123) [RFC1123], 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 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) [RFC1644], can be beneficial.

- The Timestamp header (Section 16.45) is used to avoid the retransmission ambiguity problem XXY Need ref for Stev94:TCP and obviates the need for Karn’s algorithm.

- The registered default port for RTSP over UDP for the server is 554.

- 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.

- Source authentication, or at least validation that RTSP messages comes from the same entity becomes extremely important, as session hijacking may be substantially easier for RTSP message transport
using an unreliable protocol like UDP than for TCP.

There exist two RTSP headers that are primarily intended for being used by the unreliable handling of RTSP messages and which will be maintained:

- [CSeq] See Section 16.19
- [Timestamp] See Section 16.45
Appendix F. Backwards Compatibility Considerations

This section contains notes on issues about backwards compatibility with clients or servers being implemented according to RFC 2326 [RFC2326]. Note that there exist no requirement to implement RTSP 1.0, in fact we recommend against it as it is difficult to do in an interoperable way.

A server implementing RTSP/2.0 MUST include a RTSP-Version of RTSP/2.0 in all responses to requests containing RTSP-Version RTSP/2.0. If a server receives a RTSP/1.0 request, it MAY respond with a RTSP/1.0 response if it chooses to support RFC 2326. If the server chooses not to support RFC 2326, it SHOULD respond with a 505 (RTSP Version not supported) status code. A server MUST NOT respond to a RTSP-Version RTSP/1.0 request with a RTSP-Version RTSP/2.0 response.

Clients implementing RTSP/2.0 MAY use an OPTIONS request with a RTSP-Version of 2.0 to determine whether a server supports RTSP/2.0. If the server responds with either a RTSP-Version of 1.0 or a status code of 505 (RTSP Version not supported), the client will have to use RTSP/1.0 requests if it chooses to support RFC 2326.

F.1. Play Request in Play mode

The behavior in the server when a Play is received in Play mode has changed (Section 13.4). In RFC 2326, the new PLAY request would be queued until the current Play completed. Any new PLAY request now take effect immediately replacing the previous request.

F.2. Using Persistent Connections

Some server implementations of RFC 2326 maintain a one-to-one relationship between a connection and an RTSP session. Such implementations require clients to use a persistent connection to communicate with the server and when a client closes its connection, the server may remove the RTSP session. This is worth noting if a RTSP 2.0 client also supporting 1.0 connects to a 1.0 server.
Appendix G. Open Issues

This section contains a list of open issues that still needs to be resolved. However also any open issues in the bug tracker at http://rtsp spec.sourceforge.net should also be considered.

1. Should the SMPTE range format be updated to support the 50 and 60 frames per second modes?

2. Should we define a recommended format for error message bodies?

3. Today there is no recommended or required format for 300 response entities containing URI lists. Should one be defined?

4. Should the dest_addr parameter in the Transport header in responses include the destination used by the server?

5. Should a IPv6 multicast scope parameter for the Transport header be defined? This would be similar to TTL.

6. The Expires header (Section 16.22 contains the below paragraph:

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 16.10).

Is there any purpose for this in RTSP, or could we remove this statement and instead rely on the Cache-Control header?

7. Should proxies strip out the credentials for themselves when forwarding messages with Accept-Credentials?

8. Is Session ID combined with TLS a sufficient mechanism to prevent hijacking?

9. Move to start TLS mechanism like the one defined in RFC 2817?

10. Look into the GRID communities proxy-certs and see how this relates to the current TLS proxy solution.

11. Resolve Eric Rescorlas security comments on the Proxy TLS solution:

    1. There doesn’t seem to be any way to communicate your cipher suite preferences.
2. I don’t see how certificate-based client authentication works. Is it supposed to?

3. You need to provide the entire cert chain in Connection-Credentials, not just the certificate.

12. Consider to switch to SHA256 instead of SHA1 for the digest over the DER encoded certs.

13. Resolve the following Stephen Farrel issue: “C. I don’t understand how the client-side proxies can be expected to know enough about proxies existing toward the server. If they don’t then I’m not sure how they can be expected to make any decision that’s better than would be the case were policy to be dealt with solely on a hop-by-hop basis. Maybe I’m missing something that can provide that information?”

14. Resolve the following Stephen Farrel issue: “D. The "User" policy model is that a client presents acceptable name/URIs and digests to the proxy. TLS doesn’t really provide a way for that proxy, as a client, to ask the server for the "right" certificate, so I suspect there’s a gap here that’ll be hard to fill. (If the client imposed a constraint as to the root-CA that had to be used then that’d map to the next TLS connection, but maybe it’d be too coarse-grained)”
Appendix H. Changes

Compared to RTSP 1.0 (RFC 2326), the below changes has been made when defining RTSP 2.0. Note that this list does not reflect minor changes in wording or correction of typographical errors.

- The Transport header has been changed in the following way:
  - The ABNF has been changed to define that extensions are possible, and that unknown extension parameters are to be ignored.
  - To prevent backwards compatibility issues, any extension or new parameter requires the usage of a feature-tag combined with the Require header.
  - Syntax unclarieties with the Mode parameter has been resolved.
  - Syntax error with ";" for multicast and unicast has been resolved.
  - Two new addressing parameters has been defined, src_addr and dest_addr. These replaces the parameters "port", "client_port", "server_port", "destination", "source".
  - Support for IPv6 explicit addresses in all address fields has been included.
  - To handle URI definitions that contain ";" or "," a quoted URI format has been introduced and is required.
  - Defined IANA registries for the transport headers parameters, transport-protocol, profile, lower-transport, and mode.
  - The transport headers interleaved parameter’s text was made more strict and use formal requirements levels. It was also clarified that the interleaved channels are symmetric and that it is the server that sets the channel numbers.
  - It has been clarified that the client can’t request of the server to use a certain RTP SSRC, using a request with the transport parameter SSRC.
  - Syntax definition for SSRC has been clarified to require 8HEX. It has also been extend to allow multiple values for clients supporting this version.
* Clarified the text on the transport headers "dest_addr" parameters regarding what security precautions the server is required to perform.

o The Range formats has been changed in the following way:
  * The NPT format has been given a initial NPT identifier that must now be used.
  * All formats now support initial open ended formats of type "npt=-10".

o RTSP message handling has been changed in the following way:
  * RTSP messages now uses URIs rather then URLs.
  * It has been clarified that a 4xx message due to missing CSeq header shall be returned without a CSeq header.
  * Rules for how to handle timing out RTSP messages has been added.
  * Extended Pipelining rules allowing for quick session startup.

o The HTTP references has been updated to RFC 2616 and RFC 2617. This has resulted in that the Public, and the Content-Base header needed to be defined in the RTSP specification. Known effects on RTSP due to HTTP clarifications:
  * Content-Encoding header can include encoding of type "identity".

o The state machine section has completely been rewritten. It includes now more details and are also more clear about the model used.

o A IANA section has been included with contains a number of registries and their rules. This will allow us to use IANA to keep track of RTSP extensions.

o Than transport of RTSP messages has seen the following changes:
  * The use of UDP for RTSP message transport has been deprecated due to missing interest and to broken specification.
  * The rules for how TCP connections is to be handled has been clarified. Now it is made clear that servers should not close the TCP connection unless they have been unused for significant
time.

* Strong recommendations why server and clients should use persistent connections has also been added.

* There is now a requirement on the servers to handle non-persistent connections as this provides fault tolerance.

* Added wording on the usage of Connection:Close for RTSP.

* specified usage of TLS for RTSP messages, including a scheme to approve a proxies TLS connection to the next hop.

- The following header related changes have been made:
  
  * Accept-Ranges response header is added. This header clarifies which range formats that can be used for a resource.
  
  * Changed the Range header to allow multiple ranges for creating editing list.
  
  * 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.
  
  * Put requirement on CSeq header that the value is increased by one for each new RTSP request. A Recommendation to start at 1 has also been added.
  
  * Added requirement that the Date header must be used for all messages with entity and the Server should always include it.
  
  * Removed possibility of using Range header with Scale header to indicate when it is to be activated, since it can’t work as defined. Also added rule that lack of Scale header in response indicates lack of support for the header. Feature-tags for scaled playback has been 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.
* The Session header text has been expanded with an explanation on keep alive and which methods to use. SET_PARAMETER is now recommended to use if only keep-alive within RTSP is desired.

* It has been clarified how the Range header formats is used to indicate pause points in the PAUSE response.

* Clarified that RTP-Info URIs that are relative, uses the Request-URI as base URI. Also clarified that used URI must be that one that was used in the SETUP request. They are now also required to be quoted. The header also expresses the SSRC for the provided RTP timestamp and sequence number values.

* Added text that requires the Range to always be present in PLAY responses. Clarified what should be sent in case of live streams.

* The headers table has been updated using a structured borrowed from SIP. Those tables carries much more information and should provide a good overview of the available headers.

* It has been is clarified that any message with a message body is required to have a Content-Length header. This was the case in RFC 2326 but could be misinterpreted.

* To resolve functionality around ETag. The ETag and If-None-Match header has been added from HTTP with necessary clarification in regards to RTSP operation.

* Imported the Public header from HTTP RFC 2068 [RFC2068] since it has been removed from HTTP due to lack of use. Public is used quite frequently in RTSP.

* Clarified rules for populating the Public header so that it is an intersection of the capabilities of all the RTSP agents in a chain.

  o The Protocol Syntax has been changed in the following way:

* All BNF definitions are updated according to the rules defined in RFC 4234 [RFC4234] and has been gathered in a separate Section 21.

* The BNF for the User-Agent and Server headers has been corrected so now only the description is in the HTTP specification.
* Some definitions in the introduction regarding the RTSP session has been changed.

* The protocol has been made fully IPv6 capable. Certain of the functionality, like using explicit IPv6 addresses in fields requires that the protocol support this updated specification.

* Added a fragment part to the RTSP URI. This seem to be indicated by the note below the definition however it was not part of the BNF.

* The CHAR rule has been changed to exclude NULL.

  o The Status codes has been changed in the following way:

    * The use of status code 303 "See Other" has been deprecated as it does not make sense to use in RTSP.

    * When sending response 451 and 458 the response body should contain the offending parameters.

    * Clarification on when a 3rr redirect status code can be received has been added. This includes receiving 3rr as a result of request within a established session. This provides clarification to a previous unspecified behavior.

    * Removed the 201 (Created) and 250 (Low On Storage Space) status codes as they are only relevant to recording, which is deprecated.

  o The following functionality has been deprecated from the protocol:

    * The use of Queued Play.

    * The use of PLAY method for keep-alive in play state.

    * The RECORD and ANNOUNCE methods and all related functionality. Some of the syntax has been removed.

    * The possibility to use timed execution of methods with the time parameter in the Range header.

    * The description on how rtspu works is not part of the core specification and will require external description. Only that it exist is defined here and some requirements for the the transport is provided.
The following changes has been made in relation to methods:

* The OPTIONS method has been clarified with regards to the use of the Public and Allow headers.

* 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.

* Added text clarifying the usage of SET_PARAMETER for keep-alive and usage without any body.

* PLAY method is now allowed to be pipelined with the pipelining of one or more SETUP requests following the initial that generates the session for aggregated control.

Wrote a new section 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 section also includes guidelines what to think of when writing usage guidelines for new protocols and profiles.

Setup and usage of independent TCP connections for transport of RTP has been specified.

Added a new section describing the available mechanisms to determine if functionality is supported, called "Capability Handling". Renamed option-tags to feature-tags.

Added a contributors section with people who have contributed actual text to the specification.

Added a section Use Cases that describes the major use cases for RTSP.

Clarified the usage of a=range and how to indicate live content that are not seekable with this header.

Text specifying the special behavior of PLAY for live content.

H.1. Changes needing to be updated

The minimal implementation specification has been changed:

* Required Timestamp, Via, and Unsupported headers for a minimal server implementation.
- Recommended that Cache-Control, Expires and Date headers be supported by server implementations.
Appendix I. Acknowledgements

This memorandum defines RTSP version 2.0 which is a revision of the Proposed Standard RTSP version 1.0 which is defined in [RFC2326]. The authors of this RFC are Henning Schulzrinne, Anup Rao, and Robert Lanphier.

Both RTSP version 1.0 and RTSP version 2.0 borrow 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:


I.1. Contributors

The following people have made written contributions that were included in the specification:

- Tom Marshall contributed text on the usage of 3rr status codes.
- Thomas Zheng contributed text on the usage of the Range in PLAY responses.
- Sean Sheedy contributed text on the timeout behavior of RTSP messages and connections, and the 463 status code.
- Fredrik Lindholm contributed text about the RTSP security framework.
- John Lazzaro contributed the text for RTP over Independent TCP.
- Aravind Narasimhan contributed by rewriting Media Transport Alternatives (Appendix B) and editorial improvements on a number of places in the specification.
Appendix J. RFC Editor Consideration

Please replace RFC XXXX with the RFC number this specification receives.

Please replace RFC YYYY with the RFC number that SAVPF [I-D.ietf-avt-profile-savpf] receives.
Authors’ Addresses

Henning Schulzrinne
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
Email: schulzrinne@cs.columbia.edu

Anup Rao
Cisco
USA
Email: anrao@cisco.com

Rob Lanphier
Real Networks
Seattle, WA
USA
Email: robla@robla.net

Magnus Westerlund
Ericsson AB
Torshamsgatan 23
STOCKHOLM, SE-164 80
SWEDEN
Email: magnus.westerlund@ericsson.com

Martin Stiemerling
NEC Laboratories Europe, NEC Europe Ltd.
Kurfuersten-Anlage 36
Heidelberg 69115
Germany
Phone: +49 (0) 6221 4342 113
Email: stiemerling@netlab.nec.de