Real Time Streaming Protocol 2.0 (RTSP)
draft-ietf-mmusic-rfc2326bis-21

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

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 December 21, 2009.
Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
Abstract

This memorandum defines RTSP version 2.0 which obsoletes RTSP version 1.0 which is defined in RFC 2326.

The Real Time Streaming Protocol, or RTSP, is an application-level protocol for setup and control of 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 .................................................. 11
   1.1. Notes on Copyright ...................................... 11
2. Protocol Overview ........................................... 13
   2.1. Content Description ...................................... 13
   2.2. Session Establishment .................................. 14
   2.3. Media Delivery Control .................................. 15
   2.4. Session Parameter Manipulations .......................... 17
   2.5. Media Delivery .......................................... 17
   2.5.1. Media Delivery Manipulations .......................... 18
   2.6. Session Maintenance and Termination ...................... 20
   2.7. Extending RTSP ........................................... 21
3. Document Conventions ......................................... 23
   3.1. Notational Conventions .................................. 23
   3.2. Terminology ............................................. 23
4. Protocol Parameters .......................................... 27
   4.1. RTSP Version .............................................. 27
   4.2. RTSP IRI and URI ......................................... 27
   4.3. Session Identifiers ...................................... 29
   4.4. SMPTE Relative Timestamps ................................ 29
   4.5. Normal Play Time ......................................... 30
   4.6. Absolute Time ........................................... 31
   4.7. Feature-Tags .............................................. 31
   4.8. Message Body Tags ........................................ 31
   4.9. Media Properties ......................................... 32
       4.9.1. Random Access ....................................... 33
       4.9.2. Retention ............................................. 33
       4.9.3. Content Modifications ............................... 33
       4.9.4. Supported Scale Factors .............................. 34
       4.9.5. Mapping to the Attributes ............................ 34
5. RTSP Message ................................................ 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Message Types</td>
</tr>
<tr>
<td>5.2</td>
<td>Message Headers</td>
</tr>
<tr>
<td>5.3</td>
<td>Message Body</td>
</tr>
<tr>
<td>5.4</td>
<td>Message Length</td>
</tr>
<tr>
<td>6.</td>
<td>General Header Fields</td>
</tr>
<tr>
<td>7.</td>
<td>Request</td>
</tr>
<tr>
<td>7.1</td>
<td>Request Line</td>
</tr>
<tr>
<td>7.2</td>
<td>Request Header Fields</td>
</tr>
<tr>
<td>8.</td>
<td>Response</td>
</tr>
<tr>
<td>8.1</td>
<td>Status-Line</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Status Code and Reason Phrase</td>
</tr>
<tr>
<td>8.2</td>
<td>Response Headers</td>
</tr>
<tr>
<td>9.</td>
<td>Message Body</td>
</tr>
<tr>
<td>9.1</td>
<td>Message Body Header Fields</td>
</tr>
<tr>
<td>9.2</td>
<td>Message Body</td>
</tr>
<tr>
<td>10.</td>
<td>Connections</td>
</tr>
<tr>
<td>10.1</td>
<td>Reliability and Acknowledgements</td>
</tr>
<tr>
<td>10.2</td>
<td>Using Connections</td>
</tr>
<tr>
<td>10.3</td>
<td>Closing Connections</td>
</tr>
<tr>
<td>10.4</td>
<td>Timing Out Connections and RTSP Messages</td>
</tr>
<tr>
<td>10.5</td>
<td>Showing Liveness</td>
</tr>
<tr>
<td>10.6</td>
<td>Use of IPv6</td>
</tr>
<tr>
<td>11.</td>
<td>Capability Handling</td>
</tr>
<tr>
<td>12.</td>
<td>Pipelining Support</td>
</tr>
<tr>
<td>13.</td>
<td>Method Definitions</td>
</tr>
<tr>
<td>13.1</td>
<td>OPTIONS</td>
</tr>
<tr>
<td>13.2</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>13.3</td>
<td>SETUP</td>
</tr>
<tr>
<td>13.3.1</td>
<td>Changing Transport Parameters</td>
</tr>
<tr>
<td>13.4</td>
<td>PLAY</td>
</tr>
<tr>
<td>13.4.1</td>
<td>General Usage</td>
</tr>
<tr>
<td>13.4.2</td>
<td>Aggregated Sessions</td>
</tr>
<tr>
<td>13.4.3</td>
<td>Updating current PLAY Requests</td>
</tr>
<tr>
<td>13.4.4</td>
<td>Playing On-Demand Media</td>
</tr>
<tr>
<td>13.4.5</td>
<td>Playing Dynamic On-Demand Media</td>
</tr>
<tr>
<td>13.4.6</td>
<td>Playing Live Media</td>
</tr>
<tr>
<td>13.4.7</td>
<td>Playing Live with Recording</td>
</tr>
<tr>
<td>13.4.8</td>
<td>Playing Live with Time-Shift</td>
</tr>
<tr>
<td>13.5</td>
<td>PLAY_NOTIFY</td>
</tr>
<tr>
<td>13.5.1</td>
<td>End-of-Stream</td>
</tr>
<tr>
<td>13.5.2</td>
<td>Media-Properties-Update</td>
</tr>
<tr>
<td>13.5.3</td>
<td>Scale-Change</td>
</tr>
<tr>
<td>13.6</td>
<td>PAUSE</td>
</tr>
<tr>
<td>13.7</td>
<td>TEARDOWN</td>
</tr>
<tr>
<td>13.7.1</td>
<td>Client to Server</td>
</tr>
<tr>
<td>13.7.2</td>
<td>Server to Client</td>
</tr>
<tr>
<td>13.8</td>
<td>GET_PARAMETER</td>
</tr>
<tr>
<td>13.9</td>
<td>SET_PARAMETER</td>
</tr>
</tbody>
</table>
13.10. Redirect

14. Embedded (Interleaved) Binary Data

15. Status Code Definitions

15.1. Success 1xx

15.1.1. 100 Continue

15.2. Success 2xx

15.2.1. 200 OK

15.3. Redirection 3xx

15.3.1. 301 Moved Permanently

15.3.2. 302 Found

15.3.3. 303 See Other

15.3.4. 304 Not Modified

15.3.5. 305 Use Proxy

15.4. Client Error 4xx

15.4.1. 400 Bad Request

15.4.2. 401 Unauthorized

15.4.3. 402 Payment Required

15.4.4. 403 Forbidden

15.4.5. 404 Not Found

15.4.6. 405 Method Not Allowed

15.4.7. 406 Not Acceptable

15.4.8. 407 Proxy Authentication Required

15.4.9. 408 Request Timeout

15.4.10. 410 Gone

15.4.11. 411 Length Required

15.4.12. 412 Precondition Failed

15.4.13. 413 Request Message Body Too Large

15.4.14. 414 Request-URI Too Long

15.4.15. 415 Unsupported Media Type

15.4.16. 416 ReturnETag

15.4.17. 417 Reserved

15.4.18. 418 Request-URI Too Long

15.4.19. 420 Accepted

15.4.20. 422 Unprocessable Entity

15.4.21. 423 Unstorable

15.4.22. 424 Unsupported Media Type

15.4.23. 425 Unavailable For Legal Reason

15.4.24. 426 Upgrade Required

15.4.25. 428 Precondition Required

15.4.26. 431 Request Header FieldsToo Large

15.4.27. 432 Request Entity Too Large

15.4.28. 433 Request URI Too Long

15.4.29. 434 Too Many Requests

15.4.30. 451 Not Acceptable

15.4.31. 452 Not Enough Bandwidth

15.4.32. 453 Not Acceptable

15.4.33. 454 Connection Authorization Required

15.4.34. 455 Connection Credentials not accepted

15.4.35. 456 Failure to establish secure connection

15.5. Server Error 5xx
15.5.1.  500 Internal Server Error  
15.5.2.  501 Not Implemented  
15.5.3.  502 Bad Gateway  
15.5.4.  503 Service Unavailable  
15.5.5.  504 Gateway Timeout  
15.5.6.  505 RTSP Version Not Supported  
15.5.7.  551 Option not supported  

16.  Header Field Definitions  
16.1.  Accept  
16.2.  Accept-Credentials  
16.3.  Accept-Encoding  
16.4.  Accept-Language  
16.5.  Accept-Ranges  
16.6.  Allow  
16.7.  Authorization  
16.8.  Bandwidth  
16.9.  Blocksize  
16.10.  Cache-Control  
16.11.  Connection  
16.12.  Connection-Credentials  
16.13.  Content-Base  
16.15.  Content-Language  
16.16.  Content-Length  
16.17.  Content-Location  
16.18.  Content-Type  
16.19.  CSeq  
16.20.  Date  
16.21.  Expires  
16.22.  From  
16.23.  If-Match  
16.24.  If-Modified-Since  
16.25.  If-None-Match  
16.26.  Last-Modified  
16.27.  Location  
16.28.  Media-Properties  
16.29.  Media-Range  
16.30.  MTag  
16.31.  Notify-Reason  
16.32.  Pipelined-Requests  
16.33.  Proxy-Authenticate  
16.34.  Proxy-Authorization  
16.35.  Proxy-Require  
16.36.  Proxy-Supported  
16.37.  Public  
16.38.  Range  
16.39.  Referer  
16.40.  Retry-After
22.2.  RTSP Methods ................................................. 198
  22.2.1.  Description ........................................ 198
  22.2.2.  Registering New Methods with IANA .................. 198
  22.2.3.  Registered Entries .................................. 199
22.3.  RTSP Status Codes .......................................... 199
  22.3.1.  Description ........................................ 199
  22.3.2.  Registering New Status Codes with IANA ............ 199
  22.3.3.  Registered Entries .................................. 199
22.4.  RTSP Headers ................................................. 199
  22.4.1.  Description ........................................ 199
  22.4.2.  Registering New Headers with IANA .................. 200
  22.4.3.  Registered entries .................................. 200
22.5.  Accept-Credentials ......................................... 201
  22.5.1.  Accept-Credentials policies ......................... 201
  22.5.2.  Accept-Credentials hash algorithms .................. 201
22.7.  Media Properties ........................................... 202
  22.7.1.  Description ........................................ 203
  22.7.2.  Registration Rules .................................. 203
  22.7.3.  Registered Values .................................. 203
22.8.  Notify-Reason header ....................................... 203
  22.8.1.  Description ........................................ 203
  22.8.2.  Registration Rules .................................. 203
  22.8.3.  Registered Values .................................. 204
22.9.  Range header formats ....................................... 204
22.10. Terminate-Reason Header .................................... 204
  22.10.1.  Redirect Reasons .................................. 204
  22.10.2.  Terminate-Reason Header Parameters ................. 205
22.11. RTP-Info header parameters .................................. 205
  22.11.1.  Description ........................................ 205
  22.11.2.  Registration Rules .................................. 205
  22.11.3.  Registered Values .................................. 205
22.12. Seek-Style Policies ........................................ 206
  22.12.1.  Description ........................................ 206
  22.12.2.  Registration Rules .................................. 206
  22.12.3.  Registered Values .................................. 206
22.13. Transport Header Registries ................................ 206
  22.13.2.  Transport modes .................................... 208
  22.13.3.  Transport Parameters ................................ 208
22.14. URI Schemes .................................................. 209
  22.14.1.  The rtsp URI Scheme ................................ 209
  22.14.2.  The rtsps URI Scheme ................................ 210
  22.14.3.  The rtspu URI Scheme ................................ 211
22.15. SDP attributes .............................................. 211
22.16. Media Type Registration for text/parameters ............. 212
23. References ..................................................... 214
  23.1.  Normative References ...................................... 214
Appendix E. RTSP Use Cases .......................... 263
  E.1. On-demand Playback of Stored Content .......... 263
  E.2. Unicast Distribution of Live Content .......... 264
  E.3. On-demand Playback using Multicast .......... 265
  E.4. Inviting an RTSP server into a conference ... 265
  E.5. Live Content using Multicast ................. 266
Appendix F. Text format for Parameters .............. 268
Appendix G. Requirements for Unreliable Transport of RTSP ... 269
Appendix H. Backwards Compatibility Considerations ... 271
  H.1. Play Request in Play mode ..................... 271
  H.2. Using Persistent Connections ................. 271
Appendix I. Open Issues ............................. 272
Appendix J. Changes .................................. 273
Appendix K. Acknowledgements ....................... 280
  K.1. Contributors .................................. 280
Appendix L. RFC Editor Consideration ................. 282
Authors’ Addresses .................................. 283
1. Introduction

This memo defines version 2.0 of the Real Time Streaming Protocol (RTSP 2.0). RTSP 2.0 is an application-level protocol for setup and control over the delivery of data with real-time properties, typically streaming media. Streaming media is, for instance, video on demand or audio live 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 proxies placed between clients and servers. Clients can request information about streaming media from servers, by asking for a description of the media or use media description provided externally. Then the media delivery protocol is used to establish the media streams described by the media description. Clients can then request to play out the media, pause it, or stop it completely, as known from a regular DVD player remote control. The requested media can consist of multiple audio and video streams that are delivered as a time-synchronized streams from servers to clients.

RTSP 2.0 is an replacement of RTSP 1.0 [RFC2326] that obsoletes that specification. This protocol is based on RTSP 1.0 but not backwards compatible other than in the basic version negotiation mechanism. The changes are documented in Appendix J. There are many reasons why RTSP 2.0 can’t be backwards compatible with RTSP 1.0 but some of the main ones are; that most header that needed to be extensible did not define the allowed syntax preventing safe deployment of extensions; the changed behavior of the PLAY method when received in playing state; changed behavior of the extensibility model and its mechanism; the change of syntax for some headers. The summary is that there are so many small details that changing version become necessary to enable clarification and consistent behavior.

This document is structured in the way that it begins with an overview of the protocol operations and its functions in an informal way. Then a set of definitions of used terms and document conventions is introduced. Then comes the actual protocol specification. In the appendix some functionality that isn’t core RTSP defined, but still important to enable some usage, like RTP and SDP usage with RTSP. This is followed by a number of informational parts discussing the changes, use cases, different considerations or motivations.

1.1. Notes on Copyright

The IETF has adopted new IPR contributor rules in [RFC5378], which results in a changed model of copyright. The baseline is that "The
IETF Trust and the IETF must obtain the right to publish an IETF Contribution as an RFC or an Internet-Draft from the Contributors." (taken from Section 3.1 of [RFC5378]).

This memo has plenty of text taken from [RFC2326] and thus the associated copyright. Magnus Westerlund has solicited the authors of [RFC2326] and this memo to transfer the copyright to the new model, i.e., to the IETF trust and the IETF. Most of the authors have responded and transferred their copyright. However, not all of them have. This is the first reason for the currently used boiler plate (and thus the current status), i.e., with pre5378Trust200902. See also this document [IETF-Trust-License-Policy] for more information.

Furthermore, this memo does contain text that has been copied and modified from [RFC2616]. Older versions of this memo solely linked to the particular places. Linking to the HTTP/1.1 specification was not appropriate anymore, as the text was not fitting to RTSP 2.0 needs and had to be adapted. Thus text copied from HTTP/1.1 is still under copyright prior to [RFC5378].
2. Protocol Overview

This section provides an informative overview of the different mechanisms in the RTSP 2.0 protocol, to give the reader a high level understanding before getting into all the different details. In case of conflict with this description and the later sections, the later sections take precedence. For more information about considered use cases for RTSP see Appendix E.

RTSP 2.0 is a bi-directional request and response protocol that first establishes a context including content resources (the media) and then controls the delivery of these content resources from the server to the client. RTSP has three fundamental parts of interest: Session Establishment, Playback Control, and an extensibility model described below. The protocol is based on some assumptions on existing functionality to provide a complete solution for client controlled real-time media delivery.

RTSP uses text-based messages, requests and responses, that may contain a binary message body. An RTSP request starts with a method line that identifies the method, the protocol and version and the resource to act on. Following the method line follows a number of RTSP headers. This part is ended by two consecutive carriage return line feed (CRLF) character pairs. The message body if present follows the two CRLFs and the bodies length are described by a message header. RTSP responses are similar, but start with a response line with protocol and version, followed by a status code and a reason phrase. RTSP messages are sent over a reliable transport protocol between the client and server. RTSP 2.0 requires clients and servers to implement TCP, and TLS over TCP, as mandatory transports for RTSP messages.

2.1. Content Description

RTSP exists to provide access to multi-media content, but tries to be agnostic to the media type or the actual media delivery protocol that is used. To enable a client to implement a complete system, an RTSP-external mechanism for describing the content and the delivery protocol(s) is used. RTSP assumes that this description is either delivered completely out of bands or as a data object in the response to a client’s request using the DESCRIBE method (Section 13.2).

Parameters that commonly have to be included in the Content Description are the following:

- Number of media streams
o The resource identifier for each media stream/resource that is to be controlled by RTSP

o The protocol that each media stream is to be delivered over

o Transport protocol parameters that are not negotiated or varies with each client

o Media encoding information enabling client to correctly decode it upon reception

o An aggregate control resource identifier

RTSP uses its own URI schemes ("rtsp" and "rtsps") to reference media resources and aggregates under common control.

This specification describes in Appendix D how one uses SDP [RFC4566] for Content Description

2.2. Session Establishment

The RTSP client can request the establishment of an RTSP session after having used the content description to determine which media streams are available, and also which media delivery protocol is used and their particular resource identifiers. The RTSP session is a common context between the client and the server that consist of one or more media resource that is to be under common playback control.

The client creates an RTSP session by sending an request using the SETUP method (Section 13.3) to the server. In the SETUP request the client also includes all the transport parameter necessary to enable the media delivery protocol to function in the "Transport" header (Section 16.52). This includes parameters that are pre-established by the content description but necessary for any middlebox to correctly handle the media delivery protocols. The Transport header in a request may contain multiple alternatives for media delivery in a prioritized list, which the server can select from. These alternatives are typically based on information in the content description.

The server determines if the media resource is available upon receiving a SETUP request and if any of the transport parameter specifications are acceptable. If that is successful, an RTSP session context is created and the relevant parameters and state is stored. An identifier is created for the RTSP session and included in the response in the Session header (Section 16.48). The SETUP response includes a Transport header that specifies which of the alternatives that have been selected and relevant parameters.
A SETUP request that references an existing RTSP session but identifies a new media resource is a request to add that media resource under common control with the already present media resources in an aggregated session. A client can expect this to work for all media resources under RTSP control within a multi-media content. However, aggregating resources from different content are likely to be refused by the server. The RTSP session as aggregate is referenced by the aggregate control URI, even if the RTSP session only contains a single media.

To avoid an extra round trip in the session establishment of aggregated RTSP sessions, RTSP 2.0 supports pipelined requests, i.e., the client can send multiple requests back to back without waiting first for the completion of any of them. The client uses client selected identifier in the Pipelined-Requests header to instruct the server to bind multiple requests together as if they included the session identifier.

The SETUP response also provides additional information about the established sessions in a couple of different headers. The Media-Properties header include a number of properties that apply for the aggregate that is valuable when doing playback control and configuring user interface. The Accept-Ranges header inform the client about which range formats that the server supports with these media resources. The Media-Range header inform the client about the time range of the media currently available.

2.3. Media Delivery Control

After having established an RTSP session, the client can start controlling the media delivery. The basic operations are Start by using the PLAY method (Section 13.4) and Halt by using the PAUSE method (Section 13.6). PLAY also allows for choosing the starting media position from which the server should deliver the media. The positioning is done using the Range header (Section 16.38) that supports several different time formats: Normal Play Time (Section 4.5), SMPTE Timestamps (Section 4.4) and absolute time (Section 4.6). The Range header does further allow the client to specify a position where delivery should end, thus allowing a specific interval to be delivered.

The support for positioning/searching within a content depends on the content’s media properties. Content exists in a number of different types, such as: on-demand, live, and live with simultaneous recording. Even within these categories there are differences in how the content is generated and distributed, which affect how it can be accessed for playback. The properties applicable for the RTSP session are provided by the server in the SETUP response using the
Media-Properties header (Section 16.28). These are expressed using one or several independent attributes. A first attribute is Random Access, which expresses if positioning can be done, and with what granularity. Another aspect is whether the content will change during the lifetime of the session. While on-demand content will provided in its completeness from the beginning, a live stream being recorded results in that the length of the accessible content grows as the session goes on. There also exist content that is dynamically built by another protocol than RTSP and thus also changes in steps during the session, but maybe not continuously. Furthermore, when content is recorded, there are cases where not the complete content is maintained, but, for example, only the last hour. All these properties result in the need for mechanisms that will be discussed below.

When the client accesses on-demand content, that is possible to perform random access in, the client can issue the PLAY request for any point in the content between the start and the end. The server will deliver media from the closest random access point prior to the requested point and indicate that in its PLAY response. If the client issues a pause the delivery will be halted and the point at which the server stopped will be reported back in the response. The client can later resume by a PLAY request without a range header. When the server is about to completed the PLAY request by delivering the end of the content or the requested range the server will send a PLAY_NOTIFY request indicating this.

When playing live content with no extra functions, such as recording, the client will receive the live media from the server after having sent a PLAY request. Seeking in such content is not working as the server does not store it, but only forwards it from the source of the session. Thus delivery continues until the client sends a PAUSE request, tears down the session, or the content ends.

For live sessions that are being recorded the client will need to keep track of how the recording progresses. Upon session establishment the client will learn the current duration of the recording from the Media-Range header. As the recording is ongoing the content grows in direct relation to the passed time. Therefore, each server’s response to a PLAY request will contain the current Media-Range header. The server should also send regularly every 5 minutes the current media range in a PLAY_NOTIFY request. If the live transmission ends, the server must send a PLAY_NOTIFY request with the updated Media-Properties indicating that the content stopped being a recorded live session and instead become a on-demand content. The request also contains the final media range. While the live delivery continues the client can request to play what is delivered just now by using the NPT timescale symbol "now", or it can request a
specific point in the available content by an explicit range request for that point. If the requested point is outside of the available interval the server will adjust the position to the closest available point, i.e., either at the beginning or the end.

A special case of recording is, where the recording is not retained longer than a specific time period, thus as the live delivery continues the client can access any media within a moving window that covers for example "now" to "now" minus 1 hour. A client that pauses on a specific point within the content may not be able to retrieve the content anymore. If the client waits too long before resuming the pause point, the content may no longer be available. In this case the pause point will be adjusted to the end of the available media.

### 2.4. Session Parameter Manipulations

A session may have additional state or functionality that effects how the server or client treats the session, content, how it functions, or feedback on how well the session works. Such extensions are not defined in this specification, but may be done in various extensions. RTSP has two methods for retrieving and setting parameter values on either the client or the server: GET_PARAMETER (Section 13.8) and SET_PARAMETER (Section 13.9). These methods carry the parameters in a message body of the appropriate format. One can also headers to query state with the GET_PARAMETER method. As an example, clients needing to know the current Media-Range for a time-progressing session can use the GET_PARAMETER method and include the media-range. Furthermore, synchronization information can be requested by using a combination of RTP-Info and Range.

RTSP 2.0 does not have a strong mechanism for providing negotiation of which headers, or parameters and their formats, that can be used. However, responses will indicate request headers or parameters that are not supported. A priori determination of what features are available needs to be done through out-of-band mechanisms, like the session description, or through the usage of feature tags (Section 4.7).

### 2.5. Media Delivery

The delivery of media to the RTSP client is done with a protocol outside of RTSP and this protocol is determined during the session establishment. This document specifies how media is delivered with RTP over UDP, TCP or the RTSP control connection. Additional protocols may be specified in the future based on demand.

The usage of RTP as media delivery protocol requires some additional
information to function well. The PLAY responses contains synchronization information to enable reliable and timely deliver of how a client should synchronize different sources in the different RTP sessions. It also provides a mapping between RTP timestamps and the content time scale. When the server is notifying the client about the end of the media delivery requested using PLAY, it sends a PLAY_NOTIFY request to the client. The PLAY_NOTIFY request includes information about the last RTP sequence numbers for each stream, and thus enables correct handling of the buffer drainage at the end.

2.5.1. Media Delivery Manipulations

The basic playback functionality of RTSP is to request content for a particular range to be delivered to the client in a pace that enables playback as intended by the creator. However, RTSP can also manipulate how this delivery is done to the client in two ways.

Scale: The ratio of media content time delivered per unit playback time.

Speed: The ratio of playback time delivered per unit of wallclock time.

Both affect the media delivery per time unit. However, they manipulate two independent time scales and the effects are possible to combine.

Scale is used for fast forward or slow motion control as it changes the amount of content timescale that should be played back per time unit. Scale > 1.0, means fast forward, e.g. Scale=2.0 results in that 2 seconds of content is played back every second of playback. Scale = 1.0 is the default value that is used if no Scale is specified, i.e. playback at the contents original rate. Scale values between 0 and 1.0 is providing for slow motion. Scale can be negative to allow for reverse playback in either regular pace (Scale = -1.0) or fast backwards (Scale < -1.0) or slow motion backwards (-1.0 < Scale < 0). Scale = 0 is equal to pause and is not allowed.

In most cases the realization of scale means server side manipulation of the media to ensure that the client can actually play it back. These media manipulation and when they are needed are highly media type dependent. Lets exemplify with two common media types audio and video.

It is very difficult to modify the playback rate of audio. A maximum of 10-30% is possible by changing the pitch-rate of speech. Music goes out of tune if one tries to manipulate the playback rate by resampling it. This is a well known problem and audio is commonly
muted or played back in short segments with skips to keep up with the current playback point.

For video is possible to manipulate the number of frames that is displayed per second, but the rendering capabilities are often limited to certain frame rates. The decoding, handling capabilities and bitrate of received encoded content also limits the number of frames that can be delivered. Therefore, when providing fast forward, one generally picks a subset of the frames from the original content to be displayed. However, the video encoding methods used will commonly limit the possibilities on which frames that can be chosen and still be decoded by the receiver.

Due to the media restrictions, a particular content will commonly be restricted to a limited set of possible scale ratios. To handle this correctly, RTSP has a mechanism to indicate the supported Scale ratios for the content. To support aggregated or dynamic content, where this may change during the ongoing session and dependent on the location within the content, a mechanism for updating the media properties and the current used scale factor exist.

Speed affects how much of the playback timeline that is delivered in a given wallclock period. The default is Speed = 1 which is to deliver at the same rate the media is consumed. Speed > 1 means that the receiver will get content faster than it regularly would consume it. Speed < 1 means that delivery is slower than the regular media rate. Speed values of 0 or lower has no meaning and are not allowed. This mechanism enables two general functionalities. Client side scale operations, i.e. the client receives all the frames and makes the adjustment to the playback locally. The second usage is to control delivery for buffering of media. By specifying a speed over 1.0 the client can build up the amount of playback time it has present in its buffers to a level that is sufficient for its needs.

A naive implementation of Speed would only affect the transmission schedule of the media and has a clear impact on the needed bandwidth. This would result in the data rate being proportional to the speed factor. Speed = 1.5, i.e. 50% faster than normal delivery, will then result in a 50% increase in the data transport rate. If that can be supported or not depends solely on the underlying network path. Scale may also have some impact on the required bandwidth due to the manipulation of the content in the new playback schedule. An example is fast forward where only the independently decodable intra frames are included in the media stream. This usage of solely intra frames increase the data rate significantly compared to a normal sequence with the same number of frames where most frames are encoded using prediction.
This potential increase of the data rate needs to be handled by the media sender. The client has requested that the media is delivered in a specific way, which should be honored. However, the media sender cannot ignore if the network path between the sender and the receiver cannot handle the resulting media stream. In that case the media stream needs to be adapted to fit the available resources of the path. This can result in that media quality has be reduced due to the delivery modifications that the client has requested.

The need for bitrate adaptation becomes especially problematic in connection to Speed. If the goal is to fill up the buffer, the client may not want to do that at the cost of reduced quality. If you like to do local playout changes then you may actually require that the requested speed is honored. To resolve this issue, the usage of speed specifies a range so that both usages can be supported. The server is requested to use the highest possible speed value within the range which is compatible with the available bandwidth. As long as the server can maintain a speed value within the range it shall not change the media quality, but instead modify the speed value in response to available bandwidth. However, if this is not possible, the server should instead modify the media quality to respect the lowest speed value and the available bandwidth.

This functionality enables the local scaling implementation to use a tight range, or even a range where the lower bound equals the upper bound, to identify that it requires the server to deliver the requested amount of media time per delivery time independent of how much it needs to adapt the media quality to fit within the available path bandwidth. For buffer fill up, it is suitable to use a range with a reasonable span and with a lower bound at the nominal media rate 1.0, such as 1.0 - 2.5. If the client wants to reduce the buffer, it can specify an upper bound that is below 1.0 to force the server to deliver slower than the nominal media rate.

2.6. Session Maintenance and Termination

The session context that has been established is kept alive by having the client show liveness. This is done in two main ways:

- Media transport protocol keep-alive. RTCP is possible to use when using RTP.

- Any RTSP request referencing the session context.

Section 10.5 discusses the methods for showing liveness in more depth. If the client fails to show liveness for more than the established session timeout value (normally 60 seconds), the server may terminate the context. Other values may be selected by the
server through the inclusion of the timeout parameter in the session header.

The session context is normally terminated by the client by sending a TEARDOWN request to the server referencing the aggregated control URI. An individual media resource can be removed from a session context by a TEARDOWN request referencing that particular media resource. If all media resources are removed from a session context, the session context is terminated.

A client may keep the session alive indefinitely if allowed by the server, however it is recommend to release the session context when an extended period of time without media delivery activity has passed. It can re-establish the session context if required later. One issue is that what is extended periods of time is dependent on the server and its usage. It is recommended that the client terminates the session before 10*times the session timeout value has passed. A server may terminate the session after one session timeout period without any client activity beyond keep-alive. When a server terminates the session context, it does that by sending a TEARDOWN request indicating the reason.

A server can also request that the client tear down the session and re-establish it at an alternative server, as may be needed for maintenance. This is done by using the REDIRECT method. The Terminate-Reason header is used to indicate when and why. The Location header indicates where it should connect if there is an alternative server available. When the deadline expires, the server simply stops providing the service. To achieve a clean closure, the client needs to initiate session termination prior to the deadline. In case the server has no other server to redirect to, and likes to close the session for maintenance, it shall use the TEARDOWN method with a Terminate-Reason header.

2.7. Extending RTSP

RTSP is quite a versatile protocol which supports extensions in many different directions. Even this core specification contains several blocks of functionality that are optional to implement. The use case and need for the protocol deployment is what should determine what is implemented. Allowing for extensions makes it possible for RTSP to reach out to additional use cases. However, extensions will affect the interoperability of the protocol and therefore it is important that it can be done in a structured way.

The client can learn the servers capability through the usage of the OPTIONS method (Section 13.1) and the Supported header (Section 16.49). It can also try and possibly fail by using new
methods or require that particular features are supported using the Require or Proxy-Require header.

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

- Existing methods can be extended with new parameters, for example, 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 16.35).

- 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.

- 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 a detailed explanation of this see Section 11.

New media delivery protocols may be added and negotiated at session establishment, in addition to extension to the core protocol. Certain types of protocol manipulations can be done through parameter formats using SET_PARAMETER and GET_PARAMETER.
3. Document Conventions

3.1. Notational Conventions

Since a few of the definitions 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 [RFC5234].

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.

3.2. Terminology

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.

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.

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) 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 20 and transmitted over a connection or a connectionless transport. A message is either a Request or a Response.

Message Body: The information transferred as the payload of a message. A message body consists of meta-information in the form of message-header and content in the form of an message-body, as described in Section 9.

Non-Aggregated Control: Control of a single media stream.

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 to a Request. One type of RTSP message. If an HTTP response is meant, it is indicated explicitly.

Request: An RTSP request. One type of RTSP message. If an HTTP request is meant, it 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.
4. Protocol Parameters

4.1. RTSP Version

This specification defines version 2.0 of RTSP.

RTSP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further RTSP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values.

The <minor> number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The <major> number is incremented when the format of a message within the protocol is changed. The version of an RTSP message is indicated by an RTSP-Version field in the first line of the message. Note that the major and minor numbers MUST be treated as separate integers and that each MAY be incremented higher than a single digit. Thus, RTSP/2.4 is a lower version than RTSP/2.13, which in turn is lower than RTSP/12.3. Leading zeros MUST be ignored by recipients and MUST NOT be sent.

4.2. RTSP IRI and URI

RTSP 2.0 defines and registers three URI schemes "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 MUST 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 IRI by the requester and not included in the request URI. The user agent 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 requester’s perspective, an opaque string and needs to be handled as such. Please note that relative URI with queries are difficult to handle due to the RFC 3986 relative URI handling rules. Any change of the path element using a relative URI results in the stripping of the query. Which means the relative part needs to contain the query.

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 [RFC5246], see (Section 19).

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

A presentation or a stream is identified by a textual media identifier, using the character set and escape conventions of URIs [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 but with a minimum length of 8 characters. A session identifier MUST be chosen cryptographically random (see [RFC4086]) and MUST be at least 8 characters long (can contain a maximum of 48 bits of entropy) 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 21) 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

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:

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

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 MUST 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 16.42), Proxy-Require (Section 16.35), Proxy-Supported (Section 16.36), and Unsupported (Section 16.53) 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 22).

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

4.8. Message Body Tags

Message body tags are opaque strings that are used to compare two message bodies from the same resource, for example in caches or to optimize setup after a redirect. Message body tags can be carried in the MTag header (see Section 16.30) or in SDP (see Appendix D.1.9).

A message body tag MUST be unique across all versions of all message bodies associated with a particular resource. A given message body tag value MAY be used for message body obtained by requests on different URIs. The use of the same message body tag value in conjunction with message bodies obtained by requests on different URIs does not imply the equivalence of those message bodies.

Message body tags are used in RTSP to make some methods conditional. The methods are made conditional through the inclusion of headers, see Section 16.23 and Section 16.25. Note that RTSP message body tags apply to the complete presentation; i.e., both the session
description and the individual media streams. Thus message body 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.

4.9. Media Properties

When RTSP handles media, it is important to consider the different properties a media instance for playback can have. This specification considers the below listed media properties in its protocol operations. They are derived from the differences between a number of supported usages.

On-demand: Media that has a fixed (given) duration that doesn’t change during the life time of the RTSP session and are known at the time of the creation of the session. It is expected that the content of the media will not change, even if the representation, i.e., encoding, quality, etc, may change. Generally one can seek within the media, i.e., randomly access any range of the media stream to playback.

Dynamic On-demand: This is a variation of the on-demand case where external methods are used to manipulate the actual content of the media setup for the RTSP session. The main example is where a playlist determines the content of the session.

Live: Live media represents a progressing content stream (such as broadcast TV) where the duration may or may not be known. It is not seekable, only the content presently being delivered can be accessed.

Live with Recording: A Live stream that is combined with a server side capability to store and retain the content of the live session for random access playback within the part of the already recorded content. The actual behavior of the media stream is very much depending on the retention policy for the media stream. Either the server will be able to capture the complete media stream, or it will have a limitation in how much will be retained. The media range will dynamically change as the session progress. For servers with a limited amount of storage available for recording, there will be a sliding window that goes forwards while data is made available and content that is older than the limitation will be discarded.

Considering the above usages one get the following media properties and their different instance values.
### 4.9.1. Random Access

Random Access, i.e. if one can request that the playback point is moved from one point in the media duration to another. The following different values are considered:

- **Random Access**: Yes the media are seekable to any out of a large number of points within the media. Due to media encoding limitations a particular point may not be reachable, but seeking to a point close by is enabled. A floating point number of seconds may be provided to express the worst case distance between random access points.

- **Return To Start**: Seeking is only possible to beginning of the content.

- **No seeking**: Seeking is not possible at all.

### 4.9.2. Retention

Media may have different retention policy in place that affect the operation on the media. The following different media retention policies are envisioned and taken into consideration where applicable.

- **Unlimited**: The media will not be removed as long as the RTSP session is in existence.

- **Time Limited**: The media will at least not be removed before given wallclock time. After that time it may or may not be available any more.

- **Duration limited**: Each individual unit of the media will be retained for the specified duration.

### 4.9.3. Content Modifications

There is also the question of how the content may change during time for a give media resource:

- **Immutable**: The content of the media will not change, even if the representation, i.e. encoding, quality, etc, may change.

- **Dynamic**: Between explicit updates the media content will not change, but the content may change due to external methods or triggers, such as playlists.
Time Progressing: As times progress new content will become
available. If the content also is retained it will become longer
and longer as everything between the start point and the point in
currently being made available can be accessed.

4.9.4. Supported Scale Factors

The content is often limiting the possible rates of scale that can be
supported when delivering the media. To enable the client to know
what values or ranges of scale operations that the whole content or
the current position supports a media properties attribute for this
is defined. It contains a list with the values and/or ranges that
are supported. The attribute is named "Scales". It may be updated
at any point in the content due to content consisting of spliced
pieces or content being dynamically updated by out of bands
mechanisms.

4.9.5. Mapping to the Attributes

This section exemplifies how one would map the above listed usages to
the properties and their values.

On-demand: Random Access: Random Access=5s, Content Modifications:
Immutable, Retention: unlimited or time limited.

Dynamic On-demand: Random Access: Random Access=3s, Content
Modifications: Dynamic, Retention: unlimited or time limited.

Live: Random Access: No seeking, Content Modifications: Time
Progressing, Retention: Duration limited=0.0s

Live with Recording: Random Access: Random Access=3s, Content
Modifications: Time Progressing, Retention: Duration limited=2H
5. RTSP Message

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

RTSP messages consist of requests from client to server, or server to client, and responses in the reverse direction. Request (Section 7) and Response (Section 8) messages use the generic message format of RFC 822 [RFC0822] for transferring entities (the payload of the message). Both types of message consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header, and possibly a message-body.

generic-message = start-line *(message-header CRLF)  
CRLF [ message-body ]

In the interest of robustness, servers SHOULD ignore any empty line(s) received where a Request-Line is expected. In other words, if the server is reading the protocol stream at the beginning of a message and receives a CRLF first, it should ignore the CRLF.
5.2. Message Headers

RTSP header fields (see Section 16) include general-header, request-header, response-header, and entity-header fields.

The order in which header fields with differing field names are received is not significant. However, it is "good practice" to send general-header fields first, followed by request-header or response-header fields, and ending with the entity-header fields.

Multiple message-header fields with the same field-name MAY be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list [i.e., #(values)]. It MUST be possible to combine the multiple header fields into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The order in which header fields with the same field-name are received is therefore significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

Unknown message headers MUST be ignored by a RTSP server or client. An RTSP Proxy MUST forward unknown message headers. Message headers defined outside of this specification that are required to be interpret by the RTSP agent will need to use feature tags (Section 4.7) and include it in the appropriate Require (Section 16.42) or Proxy-Require (Section 16.35) header.

5.3. Message Body

The message-body (if any) of an RTSP message is used to carry further information for a particular resource associated with the request or response. An example for a message body is the Session Description Protocol (SDP).

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

The presence of a message-body in a request is signaled by the inclusion of a Content-Length header field in the RTSP message. A message-body MUST NOT be included in a request or response if the specification of the particular method (see Method Definitions (Section 13)) does not allow sending a message body. A server SHOULD read and forward a message-body on any request; if the request method does not include defined semantics for a message body, then the message-body SHOULD be ignored when handling the request.
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 message-header fields present in the message. (Note: An empty line is a line with nothing preceding the CRLF.)

2. If a Content-Length header (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 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

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>Media-Properties</td>
<td>Section 16.28</td>
</tr>
<tr>
<td>Media-Range</td>
<td>Section 16.29</td>
</tr>
<tr>
<td>Pipelined-Requests</td>
<td>Section 16.32</td>
</tr>
<tr>
<td>Proxy-Supported</td>
<td>Section 16.36</td>
</tr>
<tr>
<td>Seek-Style</td>
<td>Section 16.45</td>
</tr>
<tr>
<td>Supported</td>
<td>Section 16.49</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Section 16.51</td>
</tr>
<tr>
<td>Via</td>
<td>Section 16.56</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 message body (Section 9.1);

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

- Optionally a message-body, consisting of one or more lines. The length of the message body in bytes is indicated by the Content-Length message 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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Defined in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE</td>
<td>Section 13.2</td>
</tr>
<tr>
<td>GET_PARAMETER</td>
<td>Section 13.8</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Section 13.1</td>
</tr>
<tr>
<td>PAUSE</td>
<td>Section 13.6</td>
</tr>
<tr>
<td>PLAY</td>
<td>Section 13.4</td>
</tr>
<tr>
<td>PLAY_NOTIFY</td>
<td>Section 13.5</td>
</tr>
<tr>
<td>REDIRECT</td>
<td>Section 13.10</td>
</tr>
<tr>
<td>SETUP</td>
<td>Section 13.3</td>
</tr>
<tr>
<td>SET_PARAMETER</td>
<td>Section 13.9</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>Section 13.7</td>
</tr>
</tbody>
</table>

Table 2: The RTSP Methods

The syntax of the RTSP request line is the following:

```
<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 16.22</td>
</tr>
<tr>
<td>If-Match</td>
<td>Section 16.23</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>Section 16.24</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>Section 16.25</td>
</tr>
<tr>
<td>Notify-Reason</td>
<td>Section 16.31</td>
</tr>
<tr>
<td>Headers</td>
<td>Sections</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Proxy-Require</td>
<td>Section 16.35</td>
</tr>
<tr>
<td>Range</td>
<td>Section 16.38</td>
</tr>
<tr>
<td>Terminate-Reason</td>
<td>Section 16.50</td>
</tr>
<tr>
<td>Referer</td>
<td>Section 16.39</td>
</tr>
<tr>
<td>Request-Status</td>
<td>Section 16.41</td>
</tr>
<tr>
<td>Require</td>
<td>Section 16.42</td>
</tr>
<tr>
<td>Scale</td>
<td>Section 16.44</td>
</tr>
<tr>
<td>Session</td>
<td>Section 16.48</td>
</tr>
<tr>
<td>Speed</td>
<td>Section 16.46</td>
</tr>
<tr>
<td>Supported</td>
<td>Section 16.49</td>
</tr>
<tr>
<td>Transport</td>
<td>Section 16.52</td>
</tr>
<tr>
<td>User-Agent</td>
<td>Section 16.54</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 processing of the header.
8. Response

After receiving and interpreting a request message, the recipient responds with an RTSP response message. The final response is exactly one message, and final responses are any using the response code classes from the list: 2xx, 3xx, 4xx and 5xx classes. Only for responses using the response code class 1xx, is it allowed to send one or more 1xx response messages prior to the final response message.

The valid response codes and the methods they can be used with are listed in Table 4.

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

3rr: 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 message body returned with the response, since that message body 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>301</td>
<td>Moved Permanently</td>
<td>all</td>
</tr>
<tr>
<td>302</td>
<td>Found</td>
<td>all</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</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>Module(s)</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>410</td>
<td>Gone</td>
<td>all</td>
</tr>
<tr>
<td>411</td>
<td>Length Required</td>
<td>all</td>
</tr>
<tr>
<td>412</td>
<td>Precondition Failed</td>
<td>DESCRIBE, SETUP</td>
</tr>
<tr>
<td>413</td>
<td>Request Message Body Too Large</td>
<td>all</td>
</tr>
<tr>
<td>414</td>
<td>Request-URI Too Long</td>
<td>all</td>
</tr>
<tr>
<td>415</td>
<td>Unsupported Media Type</td>
<td>all</td>
</tr>
<tr>
<td>451</td>
<td>Parameter Not Understood</td>
<td>SET_PARAMETER</td>
</tr>
<tr>
<td>452</td>
<td>reserved</td>
<td>n/a</td>
</tr>
<tr>
<td>453</td>
<td>Not Enough Bandwidth</td>
<td>SETUP</td>
</tr>
<tr>
<td>454</td>
<td>Session Not Found</td>
<td>all</td>
</tr>
<tr>
<td>455</td>
<td>Method Not Valid In This State</td>
<td>all</td>
</tr>
<tr>
<td>456</td>
<td>Header Field Not Valid</td>
<td>all</td>
</tr>
<tr>
<td>457</td>
<td>Invalid Range</td>
<td>PLAY, PAUSE</td>
</tr>
<tr>
<td>458</td>
<td>Parameter Is Read-Only</td>
<td>SET_PARAMETER</td>
</tr>
<tr>
<td>459</td>
<td>Aggregate Operation Not Allowed</td>
<td>all</td>
</tr>
<tr>
<td>460</td>
<td>Only Aggregate Operation Allowed</td>
<td>all</td>
</tr>
<tr>
<td>Code</td>
<td>Status Code</td>
<td>Usage</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>465</td>
<td>Notification Reason Unknown</td>
<td>PLAY_NOTIFY</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 support</td>
<td>all</td>
</tr>
</tbody>
</table>

Table 4: Status codes and their usage with RTSP methods

8.2. Response Headers

The response-header allow the request recipient to pass additional information about the response which cannot be placed in the Status-Line. This header 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>MTag</td>
<td>Section 16.30</td>
</tr>
<tr>
<td>Location</td>
<td>Section 16.27</td>
</tr>
<tr>
<td>Proxy-Authenticate</td>
<td>Section 16.33</td>
</tr>
<tr>
<td>Public</td>
<td>Section 16.37</td>
</tr>
<tr>
<td>Range</td>
<td>Section 16.38</td>
</tr>
<tr>
<td>Retry-After</td>
<td>Section 16.40</td>
</tr>
<tr>
<td>RTP-Info</td>
<td>Section 16.43</td>
</tr>
<tr>
<td>Scale</td>
<td>Section 16.44</td>
</tr>
<tr>
<td>Session</td>
<td>Section 16.48</td>
</tr>
<tr>
<td>Server</td>
<td>Section 16.47</td>
</tr>
<tr>
<td>Speed</td>
<td>Section 16.46</td>
</tr>
<tr>
<td>Transport</td>
<td>Section 16.52</td>
</tr>
<tr>
<td>Unsupported</td>
<td>Section 16.53</td>
</tr>
<tr>
<td>Vary</td>
<td>Section 16.55</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Section 16.57</td>
</tr>
</tbody>
</table>

Table 5: The RTSP response headers

Response-headers 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 MAY be given the semantics of response-header if all parties in the communication recognize them to be response-header.
Unrecognized headers in responses are treated as message-headers.
9. Message Body

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

The SET_PARAMETER and GET_PARAMETER request and response, and DESCRIBE response MAY have an message body. All 4xx and 5xx responses MAY also have an message body.

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

9.1. Message Body Header Fields

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

<table>
<thead>
<tr>
<th>Header</th>
<th>Defined in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Section 16.6</td>
</tr>
<tr>
<td>Content-Base</td>
<td>Section 16.13</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>Section 16.14</td>
</tr>
<tr>
<td>Content-Language</td>
<td>Section 16.15</td>
</tr>
<tr>
<td>Content-Length</td>
<td>Section 16.16</td>
</tr>
<tr>
<td>Content-Location</td>
<td>Section 16.17</td>
</tr>
<tr>
<td>Content-Type</td>
<td>Section 16.18</td>
</tr>
<tr>
<td>Expires</td>
<td>Section 16.21</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>Section 16.26</td>
</tr>
</tbody>
</table>

Table 6: The RTSP message body headers

The extension-header mechanism allows additional message-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. Message Body

RTSP message with a message body MUST include the Content-Type and Content-Length headers if a message body is included.

When an message body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding.

Content-Type specifies the media type of the underlying data. Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource. There is no default encoding.

The Content-Length of a message is the length of the message-body.
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 MUST 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 is RECOMMENDED be used for all transactions between the server and client, including messages for multiple RTSP sessions. However a persistent connection MAY 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.

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 silently discard RTSP messages, and may even 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. A server that attempt to send a request to a client that has no connection currently to the server SHOULD discard the request directly, it MAY queue it for later delivery. However, if the server queue the request it should when adding additional requests to the queue ensure to remove older requests that are now redundant.

Without a persistent connection between the client and the server, the media server has no reliable way of reaching the client. Because the likely failure of server to client established connections the server will not even attempt establishing any connection.

The client and server sending requests can be asynchronous events. To avoid deadlock situations both client and server MUST be able to send and receive requests simultaneously. As an RTSP response may be queue up for transmission, reception or processing behind the peer RTSP agent’s own requests, all RTSP agents are required to have a certain capability of handling outstanding messages. The issue is that outstanding requests may timeout despite them being processed by the peer due to the response is caught in the queue behind a number of request that the RTSP agent is processing but that take some time to complete. To avoid this problem an RTSP agent is recommended to buffer incoming messages locally so that any response messages can be processed immediately upon reception. If responses are separated from requests and directly forwarded for processing can not only the result be used immediately, the state associated with that outstanding request can also be released. However, buffering a number of requests on the receiving RTSP agent consumes resources and enables a resource exhaustion attack on the agent. Therefore this buffer should be limited so that an unreasonable number of requests or total message size is not allowed to consume the receiving agents resources. In most APIs having the receiving agent stop reading from the TCP socket will result in TCP’s window being clamped. Thus forcing the buffering on the sending agent when the load is larger than expected. However, as both RTSP message sizes and frequency may be changed in the future by protocol extension an agent should be careful against taking harsher measurements against a potential attack. When under attack an RTSP agent can close TCP connections and release state associated with that TCP connection.

To provide some guidance on what is reasonable the following guidelines are given. An RTSP agent should not have more than 10 outstanding requests per RTSP session. An RTSP agent should not have more than 10 outstanding requests that aren’t related to an RTSP session or that are requesting to create an RTSP session.
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 16.48). 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 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 15.4.25) 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 21).

If a server closes a connection 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 (requester) 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 requester. After sending a 100 response, the receiver MUST send a final response indicating the success or failure of the request.

A requester 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 requester SHOULD continue waiting for further responses. If more than 10 seconds elapses without receiving any response, the requester MAY assume that the responder is unresponsive and abort the connection.

A requester SHOULD wait longer than 10 seconds for a response if it is experiencing significant transport delays on its connection to the responder. The requester is capable of determining the RTT of the request/response cycle using the Timestamp header (Section 16.51) in any RTSP request.

10 seconds was chosen for the following reasons. It gives TCP time to perform a couple of retransmissions, even if operating on default values. It is short enough that users may not abandon the process themselves. However, it should be noted that 10 seconds can be aggressive on certain type of networks. The 5 seconds value for 1xx messages is half the timeout giving a reasonable change of successful delivery before timeout happens on the requestor side.

10.5. Showing Liveness

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 MUST 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.

SET_PARAMETER: When using SET_PARAMETER 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 is also usable, but it causes the server to perform more unnecessary processing and result in bigger responses than necessary for the task. The reason is that the server needs to determine the capabilities associated with the media resource to correctly populate the Public and Allow headers.

The timeout parameter MAY be included in a SETUP response, and MUST 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 B). The timeout is measured in seconds, with a default of 60 seconds. The length of the session timeout MUST NOT be changed in a established session.

10.6. 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 available capability handling mechanism 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 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: This 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 requester declares all its capabilities by including all supported feature-tags. This results in that the receiver learns the requesters feature support. The receiver then includes its set of features in the response.
Proxy-Supported: This header is used similar to the Supported header, but instead of giving the supported functionality of the client or server it provides both the requester 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 requester add it.

Require: The 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 header 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 features were 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 requester 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 MUST 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 MUST 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 16.32). This header allows a client to request that two or more requests are 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.

If a pipelined request builds on the successful completion of one or more prior requests the requester must verify that all requests were 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 MUST NOT start with a $ character (decimal 24) and MUST be a token as defined by the ABNF [RFC5234] in the syntax chapter Section 20. 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>PLAY_NOTIFY</td>
<td>S -&gt; C</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>SET_PARAMETER</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>
<tr>
<td></td>
<td>S -&gt; C</td>
<td></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 optional, but SET_PARAMETER is required due to its usage for keep-alive. PAUSE is now required due to 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. An RTSP proxy who’s main function is to log or audit and not modify transport or media handling in any way MAY forward RTSP messages with unknown methods. Note, the proxy still needs to perform the minimal required processing, like adding the Via header.

13.1. OPTIONS

The semantics of the RTSP OPTIONS method is similar 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 16.48. 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:

C->S:  OPTIONS * RTSP/2.0
       CSeq: 1
       User-Agent: PhonyClient/1.2
       Require:
       Proxy-Require: gzipped-messages
       Supported: play.basic

S->C:  RTSP/2.0 200 OK
       CSeq: 1
       Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
       Supported: play.basic, implicit-play, gzipped-messages
       Server: PhonyServer/1.1

Note that some of the feature-tags in Require and Proxy-Require are fictional features.

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 MUST respond with a description of the requested resource and return the description in the message body of the response. The DESCRIBE reply-response pair constitutes the media initialization phase of RTSP.

Example:
C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/2.0
   CSeq: 312
   User-Agent: PhonyClient 1.2
   Accept: application/sdp, application/example

S->C: RTSP/2.0 200 OK
   CSeq: 312
   Date: Thu, 23 Jan 1997 15:35:06 GMT
   Server: PhonyServer 1.1
   Content-Base: rtsp://server.example.com/fizzle/foo/
   Content-Type: application/sdp
   Content-Length: 358

   v=0
   o=mhandley 2890844526 2890842807 IN IP4 192.0.2.46
   s=SDP Seminar
   i=A Seminar on the session description protocol
   u=http://www.example.com/lectures/sdp.ps
   e=seminar@example.com (Seminar Management)
   c=IN IP4 0.0.0.0
   a=recvonly
   a=control:*
   t=2873397496 2873404696
   m=audio 3456 RTP/AVP 0
   a=control:audio
   m=video 2232 RTP/AVP 31
   a=control:video

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:

   o via an RTSP DESCRIBE request

   o 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 two different cases; Create an RTSP session and change the transport parameters of already set up media stream. SETUP can be used in all three states; INIT, and READY, for both purposes and in PLAY to change the transport parameters. There is also a third possible usage for the SETUP method which is not specified in this memo: adding a media to a session. Using SETUP to add media to an existing session, when the session is in PLAY state, is unspecified.

The Transport header, see Section 16.52, specifies the media 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 middleboxes, such as firewalls or NATs.

For the benefit of any intervening firewalls, a client MUST 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 MUST 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 MUST 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 MUST include the Accept-Ranges header (see Section 16.5) to indicate which time formats that are acceptable to use for this media resource.

The SETUP response 200 OK MUST include the Media-Properties header (see Section 16.28). The combination of the parameters of the Media-Properties header indicate the nature of the content present in the session (see also Section 4.9). For example, a live stream with time shifting is indicated by

- Random Access set to Random-Access,
- Content Modifications set to Time Progressing,
- Retention set to Time-Duration (with specific recording window time value).

The SETUP response 200 OK MUST include the Media-Range header (see Section 16.29) if the media is Time-Progressing.

A basic example for SETUP:

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: Thu, 23 Jan 1997 15:35:06 GMT
    Server: PhonyServer 1.1
    Session: 47112344;timeout=60
    Transport: RTP/AVP;unicast;dest_addr="192.0.2.53:4588"/
      "192.0.2.53:4589"; src_addr="192.0.2.241:6256"/
      "192.0.2.241:6257"; ssrc=2A3F93ED
    Accept-Ranges: NPT
    Media-Properties: Random-Access=3.2, Time-Progressing,
      Time-Duration=3600.0
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 15.4.24). 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 carries enough information for aggregate control of a session, the Aggregate control URI is still important for some methods such as SET_PARAMETER where the control URI enables the resource in question to be easily identified. The Aggregate control URI is also useful for proxies, enabling them to route the request to the appropriate server, and for logging, where it is useful to note the actual resource that a request was operating on.

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 16.48. 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 MUST 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 vise versa or change delivery address.

In a SETUP response for a request to change the transport parameters while in Play state, the server MUST include the Range to indicate from what point the new transport parameters are used. Further, if RTP is used for delivery, the server MUST 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 start point in the Range header 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

This section describes the usage of the PLAY method in general, for aggregated sessions, and in different usage scenarios.

13.4.1. General Usage

The PLAY method tells the server to start sending data via the mechanism specified in SETUP and which part of the media should be played out. 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.

Upon receipt of the PLAY request, the server MUST 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. If no Range header is present in the PLAY request the server shall play from current pause point until the end of media. The pause point defaults at start to the beginning of the media. For media that is time-progressing and has no retention, the pause point will always be set equal to NPT "now", i.e. current playback point. The pause point may also be set to a particular point in the media by the PAUSE method, see Section 13.6. The pause point for media that is currently playing is equal to the current media position. For time-progressing media with time-limited retention, if the pause point represents a position that is older than what is retained by the server, the pause point will be moved to the oldest retained.

What range values is valid depends on the type of content. For content that isn’t time progressing the range value is valid if the given range is part of any media within the aggregate. In other words the valid media range for the aggregate is the super-set of all of the media components in the aggregate. If a given range value points outside of the media, the response MUST be the 457 (Invalid Range) error code and include the Media-Range header (Section 16.29) with the valid range for the media. For time progressing content where the client request a start point prior to what is retained, the start point is adjusted to the oldest retained content. For a start point that is beyond the media front edge, i.e. beyond the current value for "now", the server shall adjust the start value to the current front edge. The Range headers end point value may point beyond the current media edge. In that case the server shall deliver media from the requested (and possibly adjusted) start point until the provided end-point, or the end of the media is reached prior to
the specified stop point. Please note that if one simply want to play from a particular start point until the end of media using an Range header with an implicit stop point is recommended.

For media with random access properties a client may express its preference on which policy for start point selection the server shall use. This is done by including the Seek-Style header (Section 16.45) in the PLAY 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. 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 MUST be included.

All range specifiers in this specification allow for ranges with implicit start point (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 MUST be given.

The below example will play seconds 10 through 25. It also request the server to deliver media from the first Random Access Point prior to the indicated start point.

```
C->S: PLAY rtsp://audio.example.com/audio RTSP/2.0
CSeq: 835
Session: 12345678
Range: npt=10-25
Seek-Style: RAP
User-Agent: PhonyClient/1.2
```

Server MUST include a "Range" header in any PLAY response, even if no Range header was present in the request. The response MUST use the same format as the request’s range header contained. If no Range header was in the request, the format used in any previous PLAY request within the session SHOULD be used. If no format has been indicated in a previous request the server MAY use any time format supported by the media and indicated in the Accept-Ranges header in the SETUP response. It is RECOMMENDED that NPT is used if supported by the media.

For any error response to a PLAY request, the server’s response depends on the current session state. If the session is in ready state, the current pause-point is returned using Range header with the pause point as the explicit start-point and an implicit end-
point. For time-progressing content where the pause-point moves with real-time due to limited retention, the current pause point is returned. For sessions in playing state, the current playout point and the remaining parts of the range request is returned. For any media with retention longer than 0 seconds the currently valid Media-Range header shall also be included in the response.

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 16.43, and used in the following example.

Here is a simple example for a single audio stream where the client requests the media starting from 3.52 seconds and to the end. The server sends a 200 OK response with the actual play time which is 10 m prior (3.51) and the RTP-Info header that contains the necessary parameters for the RTP stack.

C->S: PLAY rtsp://example.com/audio RTSP/2.0
   CSeq: 836
   Session: 12345678
   Range: npt=3.52-
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 836
   Date: Thu, 23 Jan 1997 15:35:06 GMT
   Server: PhonyServer 1.0
   Range: npt=3.51-324.39
   Seek-Style: First-Prior
   RTP-Info:url="rtsp://example.com/audio"
       ssrc=0D12F123:seq=14783;rtptime=2345962545

S->C: RTP Packet TS=2345962545 => NPT=3.51
   Duration=0.16 seconds

The server reply with the actual start point that will be 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. The server shall also indicate which policy it uses for selecting the actual start point by including a Seek-Style header.

In the following 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.

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: Thu, 23 Jan 1997 15:35:06 GMT
    Server: PhonyServer 1.0
    Range: npt=3.52-
    Seek-Style: First-Prior
    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

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 MUST 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 range in Range header has a implicit start time (-endtime), the server MUST continue to play from where it currently was until the specified end point. This is useful to change end at another point than in the previous request.

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.
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: Thu, 23 Jan 1997 15:35:06 GMT
    Server: PhonyServer 1.0
    Range: clock=19961108T142300Z-19961108T143520Z
    Seek-Style: Next
    RTP-Info:url="rtsp://example.com/meeting.en"
    ssrc=0D12F123:seq=53745;rtptime=484589019

13.4.2. Aggregated Sessions

PLAY requests can operate on sessions controlling a single media and on aggregated sessions controlling multiple media.

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

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 procedure can reduce the
delay from start of session establishment until media play-out has
started with one round trip time. However, a 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 can 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.

13.4.3. Updating current PLAY Requests

Clients can issue PLAY requests while the stream is in PLAYING state
and thus updating their request.

The important difference compared to a PLAY request in ready state is
the handling of the current play point and how the range header in
request is constructed. The session is actively playing media and
the play point will be moving making the exact time a request will
take action is hard to predict. Depending on how the PLAY header
appears two different cases exist: total replacement or continuation.
A total replacement is signalled by having the first range
specification have an explicit start value, e.g. npt=45- or
npt=45-60, in which case the server stops playout at the current
playout point and then starts delivering media according to the Range
header. This is equivalent to having the client first send a PAUSE
and then a new play request that isn’t based on the pause point. In
the case of continuation the first range specifier has an implicit
start point and a explicit stop value (Z), e.g. npt=-60, which
indicate that it MUST convert the range specifier being played prior
to this PLAY request (X to Y) into (X to Z) and continue as this was
the request originally played.

An example of this behavior. The server has received requests to
play ranges 10 to 15. If the new PLAY request arrives at the server
4 seconds after the previous one, it will take effect while the
server still plays the first range (10-15). Thus changing the
behavior of this range to continue to play to 25 seconds, i.e. the
equivalent single request would be PLAY with range: npt=10-25.
C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
   CSeq: 834
   Session: 12345678
   Range: npt=10-15
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 834
   Date: Thu, 23 Jan 1997 15:35:06 GMT
   Server: PhonyServer 1.0
   Range: npt=10-15
   Seek-Style: Next
   RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
            ssrc=0D12F123:seq=5712;rtptime=934207921,
            url="rtsp://example.com/fizzle/videotrack"
            ssrc=789DAF12:seq=57654;rtptime=2792482193
   Session: 12345678

C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
   CSeq: 835
   Session: 12345678
   Range: npt=-25
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 835
   Date: Thu, 23 Jan 1997 15:35:09 GMT
   Server: PhonyServer 1.0
   Range: npt=14-25
   Seek-Style: Next
   RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
            ssrc=0D12F123:seq=5712;rtptime=934239921,
            url="rtsp://example.com/fizzle/videotrack"
            ssrc=789DAF12:seq=57654;rtptime=2792842193
   Session: 12345678

13.4.4. Playing On-Demand Media

On-demand media is indicated by the content of the Media-Properties
header in the SETUP response by (see also Section 16.28):

- Random-Access property is set to Random Access;
- Content Modifications set to Immutable;
- Retention set Unlimited or Time-Limited.

Playing on-demand media follows the general usage as described in
13.4.5. Playing Dynamic On-Demand Media

Dynamic on-demand media is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to Random Access;
- Content Modifications set to dynamic;
- Retention set Unlimited or Time-Limited.

Playing on-demand media follows the general usage as described in Section 13.4.1 as long as the media has not been changed.

There are ways for the client to get informed about changed of media resources in play state, if the resource was changed. The client will receive a PLAY_NOTIFY request with Notify-Reason header set to media-properties-update (see Section 13.5.2). The client can use the value of the Media-Range to decide further actions, if the Media-Range header is present in the PLAY_NOTIFY request. The second way is that the client issues a GET_PARAMETER request without a body but including a Media-Range header. The 200 OK response MUST include the current Media-Range header (see Section 16.29).

13.4.6. Playing Live Media

Live media is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to no-seeking;
- Content Modifications set to Time-Progressing;
- Retention with Time-Duration set to 0.0.

For live media, the SETUP response 200 OK MUST include the Media-Range header (see Section 16.29).

A client MAY send PLAY requests without the Range header, if the request include the Range header it MUST use a symbolic value representing "now". For NPT that range specification is "npt=now-". The server MUST include the Range header in the response and it MUST indicate an explicit time value and not a symbolic value. In other words npt=now- is not a valid to use in the response. Instead the time since session start is recommended expressed 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 can only be used if client has shown support for it using the Accept-Ranges header.

13.4.7. Playing Live with Recording

Certain media server may offer recording services of live sessions to their clients. This recording would normally be from the beginning of the media session. Clients can randomly access the media between now and the beginning of the media session. This live media with recording is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to random-access;
- Content Modifications set to Time-Progressing;
- Retention set to Time-limited or Unlimited

The SETUP response 200 OK MUST include the Media-Range header (see Section 16.29) for this type of media. For live media with recording the Range header indicates the current playback time in the media and the Media-Range header indicates the currently available media window around the current time. This window can cover recorded content in the past (seen from current time in the media) or recorded content in the future (seen from current time in the media). The server adjusts the play point to the requested border of the window, if the client requests a play point that is located outside the recording windows, e.g., if requested to far in the past, the server selects the oldest range in the recording. The considerations in Section 13.5.3 apply, if a client requests playback at Scale (Section 16.44) values other than 1.0 (Normal playback rate) while playing live media with recording.

13.4.8. Playing Live with Time-Shift

Certain media server may offer time-shift services to their clients. This time shift records a fixed interval in the past, i.e., a sliding window recording mechanism, but not past this interval. Clients can randomly access the media between now and the interval. This live media with recording is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to random-access;
- Content Modifications set to Time-Progressing;
Retention set to Time-Duration and a value indicating the recording interval (>0).

The SETUP response 200 OK MUST include the Media-Range header (see Section 16.29) for this type of media. For live media with recording the Range header indicates the current time in the media and the Media Range indicates a window around the current time. This window can cover recorded content in the past (seen from current time in the media) or recorded content in the future (seen from current time in the media). The server adjusts the play point to the requested border of the window, if the client requests a play point that is located outside the recording windows, e.g., if requested too far in the past, the server selects the oldest range in the recording. The considerations in Section 13.5.3 apply, if a client requests playback at Scale (Section 16.44) values other than 1.0 (Normal playback rate) while playing live media with time-shift.

13.5. PLAY_NOTIFY

The PLAY_NOTIFY method is issued by a server to inform a client about an asynchronously event for a session in play state. The Session header MUST be presented in a PLAY_NOTIFY request and indicates the scope of the request. Sending of PLAY_NOTIFY requests requires a persistent connection between server and client, otherwise there is no way for the server to send this request method to the client.

PLAY_NOTIFY requests have an end-to-end (i.e. server to client) scope, as they carry the Session header, and apply only to the given session. The client SHOULD immediately return a response to the server.

PLAY_NOTIFY requests MAY be used with a message body, depending on the value of the Notify-Reason header. It is described in the particular section for each Notify-Reason if a message body is used. However, currently there is no Notify-Reason that allows using a message body. There is in this case a need to obey some limitations when adding new Notify-Reasons that intend to use a message body: The server can send any type of message body, but it is not ensured that the client can understand the received message body. This is related to DESCRIBE (see Section 13.2), but in this particular case the client can state its acceptable message bodies by using the Accept header. In the case of PLAY_NOTIFY, the server does not know which message bodies are understood by the client.

The Notify-Reason header (see Section 16.31) specifies the reason why the server sends the PLAY_NOTIFY request. This is extensible and new reasons MAY be added in the future. In case the client does not understand the reason for the notification it MUST respond with an
465 (Notification Reason Unknown) (Section 15.4.30) error code. Servers can send PLAY_NOTIFY with these types:

- end-of-stream (see Section 13.5.1);
- media-properties-update (see Section 13.5.2);
- scale-change (see Section 13.5.3).

13.5.1. End-of-Stream

A PLAY_NOTIFY request with Notify-Reason header set to end-of-stream indicates the completion or near completion of the PLAY request and the ending delivery of the media stream(s). The request MUST NOT be issued unless the server is in the playing state. The end of the media stream delivery notification may be used to indicate either a successful completion of the PLAY request currently being served, or to indicate some error resulting in failure to complete the request. The Request-Status header (Section 16.41) MUST be included to indicate which request the notification is for and its completion status. The message response status codes (Section 8.1.1) are used to indicate how the PLAY request concluded. The sender of a PLAY_NOTIFY can issue an updated PLAY_NOTIFY, in the case of a PLAY_NOTIFY sent with wrong information. For instance, a PLAY_NOTIFY was issued before reaching the end-of-stream, but some error occurred resulting in that the previously sent PLAY_NOTIFY contained a wrong time when the stream will end. In this case a new PLAY_NOTIFY MUST be sent including the correct status for the completion and all additional information.

PLAY_NOTIFY requests with Notify-Reason header set to end-of-stream MUST include a Range header and the Scale header if the scale value is not 1. The Range header indicates the point in the stream or streams where delivery is ending with the timescale that was used by the server in the PLAY response for the request being fulfilled. The server MUST NOT use the "now" constant in the Range header; it MUST use the actual numeric end position in the proper timescale. When end-of-stream notifications are issued prior to having sent the last media packets, this is evident as the end time in the Range header is beyond the current time in the media being received by the client, e.g., npt=-15, if npt is currently at 14.2 seconds. The Scale header is to be included so that it is evident if the media time scale is moving backwards and/or have a non-default pace.

If RTP is used as media transport, a RTP-Info header MUST be included, and the RTP-Info header MUST indicate the last sequence number in the seq parameter.
A PLAY_NOTIFY request with Notify-Reason header set to end-of-stream MUST NOT carry a message body.

This example request notifies the client about a future end-of-stream event:

S->C: PLAY_NOTIFY rtsp://example.com/fizzle/foo RTSP/2.0
     CSeq: 854
     Notify-Reason: end-of-stream
     Request-Status: cseq=853 status=200 reason="OK"
     Range: npt=-145
     RTP-Info:url="rtsp://example.com/audio"
          ssrc=0D12F123:seq=14783;rtptime=2345962545
     Session: uZ3ci0K+Ld-M

C->S: RTSP/2.0 200 OK
     CSeq: 854
     User-Agent: PhonyClient/1.2

13.5.2. Media-Properties-Update

A PLAY_NOTIFY request with Notify-Reason header set to media-properties-update indicates an update of the media properties for the given session (see Section 16.28) and/or the available media range that can be played as indicated by Media-Range (Section 16.29).

PLAY_NOTIFY requests with Notify-Reason header set to media-properties-update MUST include a Media-Properties and Date header and SHOULD include a Media-Range header.

This notification MUST be sent for media that are time-progressing every time an event happens that changes the basis for making estimations on how the media range progress. In addition it is RECOMMENDED that the server sends these notifications every 5 minutes for time-progressing content to ensure the long term stability of the client estimation and allowing for clock skew detection by the client. Requests for the just mentioned reasons MUST include Media-Range header to provide current Media duration and the Range header to indicate the current playing point and any remaining parts of the requested range.

The recommendation for sending updates every 5 minutes is due to any clock skew issues. In 5 minutes the clock skew should not become too significant as this is not used for media playback and synchronization, only for determining which content is available to the user.

A PLAY_NOTIFY request with Notify-Reason header set to media-properties-update MUST NOT carry a message body.
13.5.3. Scale-Change

The server may be forced to change the rate, when a client requests playback at Scale (Section 16.44) values other than 1.0 (normal playback rate). For time progressing media with some retention, i.e. the server stores already sent content, a client requesting to play with Scale values larger than 1 may catch up with the front end of the media. The server will then be unable to continue to provide with content at Scale larger than 1 as content is only made available by the server at Scale=1. Another case is when Scale < 1 and the media retention is time-duration limited. In this case the playback point can reach the oldest media unit available, and further playback at this scale becomes impossible as there will be no media available. To avoid having the client loose any media, the scale will need to be adjusted to the same rate which the media is removed from the storage buffer, commonly scale = 1.0.

Another case is when the content itself consist of spliced pieces or is dynamically updated. In these cases the server may be required to change from one supported scale value (different than Scale=1.0) to another. In this case the server will pick the closest value and inform the client of what it has picked. In these case the media properties will also be sent updating the supported Scale values. This enables a client to adjust the used Scale value.

To minimize impact on playback in any of the above cases the server MUST modify the playback properties and set Scale to a supportable value and continue delivery the media. When doing this modification it MUST send a PLAY_NOTIFY message with the Notify-Reason header set to "scale-change". The request MUST contain a Range header with the media time where the change took effect, a Scale header with the new value in use, Session header with the ID for the session it applies to and a Date header with the server wallclock time of the change. For time progressing content also the Media-Range and the Media-
Properties at this point in time MUST be included. The Media-
Properties header MUST be included if the scale change was due to the
content changing what scale values that is supported.

For media streams being delivered using RTP also a RTP-Info header
MUST be included. It MUST contain the rtptime parameter with a value
corresponding to the point of change in that media and optionally
also the sequence number.

A PLAY_NOTIFY request with Notify-Reason header set to "Scale-Change"
MUST NOT carry a message body.

S->C: PLAY_NOTIFY rtsp://example.com/fizzle/foo RTSP/2.0
Date: Tue, 14 Apr 2008 15:48:06 GMT
CSeq: 854
Notify-Reason: scale-change
Session: uZ3ci0K+Ld-M
Media-Properties: Time-Progressing,
    Time-Limited=20080415T153919.36Z, Random-Access=5.0
Media-Range: npt=0-1:37:21.394
Range: npt=1:37:21.394-
Scale: 1
RTP-Info: url="rtsp://example.com/fizzle/foo/audio"
    ssrc=0D12F123:rtptime=2345962545

C->S: RTSP/2.0 200 OK
CSeq: 854
User-Agent: PhonyClient/1.2

13.6. 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 MUST 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: Thu, 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 resume from the pause point and play until media end.

The pause point after any PAUSE request MUST be returned to the client by adding a Range header with what remains unplayed of the PLAY request’s range. For media with random access properties, if one desires to resume playing a ranged request, one simply includes the Range header from the PAUSE response and include the Seek-Style header with the Next policy in the PLAY request. For media that is time-progressing and has retention duration=0 the follow-up PLAY request to start media delivery again, will need to use "npt=now-" and not the answer given in the response to PAUSE.
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: Thu, 23 Jan 1997 15:35:06 GMT
   Server: PhonyServer 1.0
   Range: npt=10-30
   Seek-Style: First-Prior
   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
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 835
   Date: 23 Jan 1997 15:35:09 GMT
   Server: PhonyServer 1.0
   Range: npt=21-30
   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:
13.7. TEARDOWN

13.7.1. Client to Server

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 MUST 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 MUST result in that media delivery is immediately halted and the session state is destroyed. This MUST 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 MUST 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

13.7.2.  Server to Client

The server can send TEARDOWN requests in the server to client direction to indicate that the server has been forced to terminate the ongoing session. This may happen for several reasons such as, server maintenance without available backup, or session have been inactive for extended periods of time. The reason is provided in the Terminate-Reason header (Section 16.50).

When a RTSP client has maintained a RTSP session that otherwise is inactive for an extended period of time the server may reclaim the resources. That is done by issuing a REDIRECT request with the Terminate-Reason set to "Session-Timeout". This MAY be done when the client has been inactive in the RTSP session for more than one Session Timeout period (Section 16.48). However, the server is RECOMMENDED to not perform this operation until an extended period of inactivity has passed. The time period is considered extended when it is 10 times the Session Timeout period. Consideration of the application of the server and its content should be performed when configuring what is considered as extended periods of time.

In case the server needs to stop provide service to the established sessions and their is no server to point at in a REDIRECT request TEARDOWN shall be used to terminate the session. This method can also be used when non-recoverable internal errors have happened and the server has no other option then to terminate the sessions.

The TEARDOWN request is normally done on the session aggregate control URI and MUST include the following headers; Session and Terminate-Reason headers. The request only applies to the session identified in the Session header. The server may include a message to the client’s user with the "user-msg" parameter.

The TEARDOWN request may alternatively be done on the wild card URI * and without any session header. 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. This request indicates that all sessions and pending requests being managed via the control connection are terminated. Any intervening proxies SHOULD do all of the following in the order listed:

1. respond to the TEARDOWN request
2. disconnect the control channel from the requesting server
3. pass the TEARDOWN request to each applicable client (typically those clients with an active session or an unanswered request)

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

13.8. GET_PARAMETER

The GET_PARAMETER request retrieves the value of any specified 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. There are two ways of specifying the parameters to be retrieved. The first is by including headers which have been defined such that you can use them for this purpose. Header for this purpose should allow empty, or stripped value parts to avoid having to specify bogus data when indicating the desire to retrieve a value. The successful completion of the request should also be evident from any filled out values in the response. The Media-Range header (Section 16.29) is one such header. The other is to specify a message body that lists the parameter(s) that are desirable to retrieve. The Content-Type header (Section 16.18) is used to specify which format the message body has.

The headers that MAY be used for retrieving their current value using GET_PARAMETER are:

- Accept-Ranges
- Media-Range
- Media-Properties
- Range
- RTP-Info
The method MAY also be used without a message body or any header that request parameters for keep-alive purpose. Any request that 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. Normally the presence of filled out values in the header will be indication that the header has been processed. However, for cases when this is difficult to determine, it is recommended to use a feature-tag and the Require header. Due to this reason it is usually easier if any parameters to be retrieved are sent in the body, rather than using any header.

Parameters specified within the body of the message must all be understood by the request receiving agent. If one or more parameters are not understood a 451 (Parameter Not Understood) MUST be sent including a body listing these parameters that wasn’t understood. If all parameters are understood their value is filled in and returned in the response message body.

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
Session: 12345678
Content-Length: 38
Content-Type: text/parameters

packets_received: 10
jitter: 0.3838

13.9. 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 message body. 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 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 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) MUST be used. In the case a parameter is not allowed to change, the error code is 458 (Parameter Is Read-Only). The response body MUST contain only the parameters that have errors. Otherwise no body MUST be returned.

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

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

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

Example:

C->S: SET_PARAMETER rtsp://example.com/fizzle/foo RTSP/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
13.10. REDIRECT

The REDIRECT method is issued by a server to inform a client that the service provided will be terminated and where a corresponding service can be provided instead. This happens for different reasons. One is that the server is being administrated such that it must stop providing service. Thus the client is required to connect to another server location to access the resource indicated by the Request-URI.

The REDIRECT request SHALL contain a Terminate-Reason header (Section 16.50) to inform the client of the reason for the request. Additional parameters related to the reason may also be included. The intention here is to allow an server administrator to do a controlled shutdown of the RTSP server. That requires sufficient time to inform all entities having associated state with the server and for them to perform a controlled migration from this server to a fall back server.

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 REQUIRED Location header MUST contain a complete absolute URI pointing to the resource to which the client SHOULD reconnect. Specifically, the Location MUST 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 to all sessions controlled, as well as 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 REQUIRED 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.

When the server lacks any alternative server and needs to terminate a session or all sessions the TEARDOWN request SHALL be used instead.

When no Terminate-Reason "time" parameter are included in a REDIRECT request, the client SHALL perform the redirection immediately and return a response to the server. The server shall 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.

The Terminate-Reason header "time" parameter MAY be used to indicate the wallclock time by when the redirection MUST have take place. To allow a client to determine that redirect time without being time synchronized with the server, the server MUST include a Date header in the request. The client should have before the redirection time-line terminated the session and close the control connection. The server MAY simple cease to provide service when the deadline time has been reached, or it may issue TEARDOWN requests to the remaining sessions.

The differentiation of REDIRECT requests with and without range header 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 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 MTag 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
     Terminate-Reason: Server-Admin ;time=19960213T143205Z
     Session: uZ3ci0K+Ld-M
     Date: Thu, 13 Feb 1996 14:30:43 GMT

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 MUST contain exactly one upper-layer protocol data unit, e.g., one RTP packet.

```
+-------------------+-------------------+-------------------+
| "$" = 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 16.52).

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 second channel in the interleaved parameter of the Transport header, see Section 16.52. If RTCP is used, packets MUST 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.
Internet-Draft

Real Time Streaming Protocol 2.0 (RTSP)

June 2009

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: Thu, 05 Jun 1997 18:57:18 GMT
Transport: RTP/AVP/TCP;unicast;interleaved=5-6
Session: 12345678
Accept-Ranges: NPT
Media-Properties: Random-Access=0.2, Unmutable, Unlimited
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: Thu, 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
Seek-Style: RAP
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}

Schulzrinne, et al.

Expires December 21, 2009

[Page 93]


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

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed.

15.2. Success 2xx

This class of status code indicates that the client’s request was successfully received, understood, and accepted.

15.2.1. 200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request.

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.

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.

An 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 a request in relation to
an 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 Location header is used in a response it MUST contain an
absolute URI pointing out the media resource the client is redirected
to, the URI MUST NOT only contain the host name.

15.3.1. 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.

15.3.2. 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

15.3.3. 303 See Other

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

15.3.4. 304 Not Modified

If the client has performed a conditional DESCRIBE or SETUP (see
Section 16.24) and the requested resource has not been modified, the
server SHOULD send a 304 response. This response MUST NOT contain a
The response MUST include the following header fields:

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

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

15.3.5. 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses MUST only be generated by origin servers.

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. If the request does not have a CSeq header, the server MUST NOT include a CSeq in the response.

15.4.2. 401 Unauthorized

The request requires user authentication. The response MUST include a WWW-Authenticate header (Section 16.57) field containing a challenge applicable to the requested resource. The client MAY repeat the request with a suitable Authorization header field. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the entity that was given in the response, since that entity might
include relevant diagnostic information. HTTP access authentication is explained in [RFC2617].

15.4.3. 402 Payment Required

This code is reserved for future use.

15.4.4. 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the server wishes to make public why the request has not been fulfilled, it SHOULD describe the reason for the refusal in the entity. If the server does not wish to make this information available to the client, the status code 404 (Not Found) can be used instead.

15.4.5. 404 Not Found

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

15.4.6. 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.

15.4.7. 406 Not Acceptable

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

The response SHOULD include an message body containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed.
automatically. However, this specification does not define any
standard for such automatic selection.

If the response could be unacceptable, a user agent SHOULD
temporarily stop receipt of more data and query the user for a
decision on further actions.

15.4.8. 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized) (Section 15.4.2), but
indicates that the client must first authenticate itself with the
proxy. The proxy MUST return a Proxy-Authenticate header field
(Section 16.33) containing a challenge applicable to the proxy for
the requested resource.

15.4.9. 408 Request Timeout

The client did not produce a request within the time that the server
was prepared to wait. The client MAY repeat the request without
modifications at any later time.

15.4.10. 410 Gone

The requested resource is no longer available at the server and the
forwarding address is not known. This condition is expected to be
considered permanent. If the server does not know, or has no
facility to determine, whether or not the condition is permanent, the
status code 404 (Not Found) SHOULD be used instead. This response is
cacheable unless indicated otherwise.

The 410 response is primarily intended to assist the task of
repository maintenance by notifying the recipient that the resource
is intentionally unavailable and that the server owners desire that
remote links to that resource be removed. Such an event is common
for limited-time, promotional services and for resources belonging to
individuals no longer working at the server’s site. It is not
necessary to mark all permanently unavailable resources as "gone" or
to keep the mark for any length of time -- that is left to the
discretion of the owner of the server.

15.4.11. 411 Length Required

The server refuses to accept the request without a defined Content-
Length. The client MAY repeat the request if it adds a valid
Content-Length header field containing the length of the message-body
in the request message.
15.4.12. 412 Precondition Failed

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource meta information (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.

15.4.13. 413 Request Message Body Too Large

The server is refusing to process a request because the request message body is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

15.4.14. 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has used a request with long query information, when the client has descended into a URI "black hole" of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the Request-URI.

15.4.15. 415 Unsupported Media Type

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

15.4.16. 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 message body containing the offending parameter(s).

15.4.17. 452 reserved

This error code was removed from RFC 2326 [RFC2326] as it is obsolete. This error code MUST NOT be used anymore.

15.4.18. 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.

15.4.19. 454 Session Not Found

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

15.4.20. 455 Method Not Valid in This State

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

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

15.4.22. 457 Invalid Range

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

15.4.23. 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 message body containing the offending parameter(s).

15.4.24. 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.

15.4.25. 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.
15.4.26. 461 Unsupported Transport

The Transport field did not contain a supported transport specification.

15.4.27. 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.

15.4.28. 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.

15.4.29. 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.

15.4.30. 465 Notification Reason Unknown

This indicates that the client has received a PLAY_NOTIFY (Section 13.5) with a Notify-Reason header (Section 16.31) unknown to the client.

15.4.31. 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.
15.4.32. 471 Connection Credentials not accepted

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

15.4.33. 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

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

15.5.1. 500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request.

15.5.2. 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

15.5.3. 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

15.5.4. 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.
Note: The existence of the 503 status code does not imply that a server must use it when becoming overloaded. Some servers may wish to simply refuse the connection.

15.5.5. 504 Gateway Timeout

The server, while acting as a proxy, did not receive a timely response from the upstream server specified by the URI or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.

15.5.6. 505 RTSP Version Not Supported

The server does not support, or refuses to support, the RTSP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client other than with this error message. The response SHOULD contain a message body describing why that version is not supported and what other protocols are supported by that server.

15.5.7. 551 Option not supported

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

<table>
<thead>
<tr>
<th>method</th>
<th>direction</th>
<th>object</th>
<th>acronym</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>DES</td>
<td>r</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>OPT</td>
<td></td>
</tr>
<tr>
<td>PAUSE</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>PSE</td>
<td></td>
</tr>
<tr>
<td>PLAY</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>PLY</td>
<td></td>
</tr>
<tr>
<td>PLAY_NOTIFY</td>
<td>S -&gt; C</td>
<td>P,S</td>
<td>PNY</td>
<td>R</td>
</tr>
<tr>
<td>REDIRECT</td>
<td>S -&gt; C</td>
<td>P,S</td>
<td>RDR</td>
<td></td>
</tr>
<tr>
<td>SETUP</td>
<td>C -&gt; S</td>
<td>S</td>
<td>STP</td>
<td></td>
</tr>
<tr>
<td>SET_PARAMETER</td>
<td>C -&gt; S, S -&gt; C</td>
<td>P,S</td>
<td>SPR</td>
<td>R,r</td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>C -&gt; S</td>
<td>P,S</td>
<td>TRD</td>
<td></td>
</tr>
</tbody>
</table>

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 20.2.3. Throughout this section, we use [HX.Y] to informational 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 MUST 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 8:

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 MUST 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 headers that require
processing are the Require and Proxy-Require header fields discussed
in Section 16.42 and Section 16.35. 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 MUST 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 quotes.
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>SETUP</th>
<th>PLAY</th>
<th>PAUSE</th>
<th>TRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>R</td>
<td>o</td>
<td>-</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-Encoding</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>-</td>
<td>-</td>
<td>m</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>r</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>456</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</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>Authorization</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>----------------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Blocksize</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Connection</td>
<td></td>
<td>o</td>
<td>o</td>
<td>o</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>
<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>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Content-Base</td>
<td>4xx, 5xx</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</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>o</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>o</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>Content-Locality</td>
<td>r</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Content-Locality</td>
<td>4xx, 5xx</td>
<td>o</td>
<td>o</td>
<td>o</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>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Header</td>
<td>Where</td>
<td>Proxy</td>
<td>DES</td>
<td>OPT</td>
<td>SETU</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>Media-Properties</td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>-</td>
</tr>
<tr>
<td>Media-Range</td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>r</td>
<td>r</td>
<td>r</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>
</tbody>
</table>

Table 9: Overview of RTSP header fields (A–L) related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.
<table>
<thead>
<tr>
<th>Field</th>
<th>Requirements</th>
<th>Public</th>
<th>Public</th>
<th>Range</th>
<th>Range</th>
<th>Terminate-Reason</th>
<th>Referer</th>
<th>Request-Status</th>
<th>Require</th>
<th>Retry-After</th>
<th>RTP-Info</th>
<th>Scale</th>
<th>Seek-Style</th>
<th>Seek-Style</th>
<th>Session</th>
<th>Session</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy-Require</td>
<td>R ar o o o o o o o</td>
<td>r r c c c c c c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proxy-Require</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proxy-Supported</td>
<td>R amr c c c c c c</td>
<td>r c c c c c c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proxy-Supported</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public</td>
<td>r admr - m - - - -</td>
<td>501 admr m m m m m m m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>R - - - o - - -</td>
<td>r - - - c m m -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminate-Reason</td>
<td>R r - - - - - - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Referer</td>
<td>R o o o o o o o o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request-Status</td>
<td>R - - - - - - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Require</td>
<td>R o o o o o o o o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retry-After</td>
<td>3rr,5 03 o o o - - -</td>
<td>413 o o o o o o o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTP-Info</td>
<td>r - - c c c - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td>- - - o - - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seek-Style</td>
<td>R - - - o - - -</td>
<td>r - - - m - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seek-Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td>R r - o o m m m</td>
<td>r r - c m m m o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>R r - o - - - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header</td>
<td>Where</td>
<td>Proxy</td>
<td>GPR</td>
<td>SPR</td>
<td>RDR</td>
<td>PNY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allow</td>
<td>405</td>
<td>amr</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td></td>
<td></td>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bandwidth</td>
<td>R</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocksize</td>
<td>R</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection-Credentials</td>
<td>470,407</td>
<td>ar</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Overview of RTSP header fields (P-W) related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.
<p>| Content-Base | R       | o | o | - | - |
| Content-Base | r | o | o | - | - |
| Content-Base | 4xx,5xx | o | o | o | - |
| Content-Encoding | R | r | o | o | - | - |
| Content-Encoding | r | r | o | o | - | - |
| Content-Encoding | 4xx,5xx | r | o | o | o | - |
| Content-Language | R | r | o | o | - | - |
| Content-Language | r | r | o | o | - | - |
| Content-Language | 4xx,5xx | r | o | o | o | - |
| Content-Length | R | r | * | * | - | - |
| Content-Length | r | r | * | * | - | - |
| Content-Length | 4xx,5xx | r | * | * | * | - |
| Content-Location | R | o | o | - | - |
| Content-Location | r | o | o | - | - |
| Content-Location | 4xx,5xx | o | o | o | - |
| Content-Type | R | * | * | - | - |
| Content-Type | r | * | * | - | - |
| Content-Type | 4xx | * | * | * | - |
| CSeq | R,c | mr | m | m | m | m |
| Date | R | a | o | o | m | - |
| Date | r | am | o | o | o | - |
| From | R | r | o | o | o | - |
| Last-Modified | R | r | - | - | - | - |
| Last-Modified | r | r | o | - | - | - |</p>
<table>
<thead>
<tr>
<th>Header</th>
<th>Where</th>
<th>Proxy</th>
<th>GPR</th>
<th>SPR</th>
<th>RDR</th>
<th>PNY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Terminate-Reason</td>
<td>R</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>Referer</td>
<td>R</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Request-Status</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>Require</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Retry-After</td>
<td>3rr,413,503</td>
<td>o</td>
<td>o</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Retry-After</td>
<td>413</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 11: Overview of RTSP header fields (A-P) related to methods GET_PARAMETER, SET_PARAMETER, PLAY_NOTIFY, and REDIRECT.
### Table 12: Overview of RTSP header fields (R-W) related to methods GET_PARAMETER, SET_PARAMETER, PLAY_NOTIFY, and REDIRECT.

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>r</th>
<th>o</th>
<th>o</th>
<th>o</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>Seek-Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>R</td>
<td>r</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Server</td>
<td>r</td>
<td>r</td>
<td>o</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>
<td>-</td>
</tr>
<tr>
<td>Supported</td>
<td>r</td>
<td>adrm</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>-</td>
</tr>
<tr>
<td>Timestamp</td>
<td>R</td>
<td>adrm</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>Timestamp</td>
<td>c</td>
<td>adrm</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>Unsupported</td>
<td>r</td>
<td>arm</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>-</td>
</tr>
<tr>
<td>User-Agent</td>
<td>R</td>
<td>r</td>
<td>m*</td>
<td>m*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>User-Agent</td>
<td>r</td>
<td>r</td>
<td>-</td>
<td>-</td>
<td>m*</td>
<td>-</td>
</tr>
<tr>
<td>Vary</td>
<td>r</td>
<td>c</td>
<td>c</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Vary</td>
<td>c</td>
<td>dr</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>-</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>401</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>-</td>
</tr>
</tbody>
</table>

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 Section 20.2.3 for the 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 19 for the usage of this header. It MUST NOT be included in server to client requests.

In a request the header MUST contain the method (User, Proxy, or Any) for approving credentials selected by the requester. The method MUST 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 accepts. 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 MUST consist of one URI identifying the proxy or server, the hash algorithm identifier, and the hash over that entity’s DER encoded certificate [RFC5280] in Base64 [RFC4648]. All RTSP clients and proxies MUST 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.

Example:
Accept-Credentials:User
   "rtsps://proxy2.example.com/";sha-256;exaIl9VMbQMOFGClx5rXnPJKvNI=,
   "rtsps://server.example.com/";sha-256;lurbjj5khhB0Nh1uOXTt4bBRH1M=

16.3. Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but restricts the content-codings that are acceptable in the response.

A server tests whether a content-coding is acceptable, according to an Accept-Encoding field, using these rules:

1. If the content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable, unless it is accompanied by a qvalue of 0. (As defined in section 3.9, a qvalue of 0 means "not acceptable.")
2. The special "*" symbol in an Accept-Encoding field matches any available content-coding not explicitly listed in the header field.

3. If multiple content-codings are acceptable, then the acceptable content-coding with the highest non-zero qvalue is preferred.

4. The "identity" content-coding is always acceptable, unless specifically refused because the Accept-Encoding field includes "identity;q=0", or because the field includes "*;q=0" and does not explicitly include the "identity" content-coding. If the Accept-Encoding field-value is empty, then only the "identity" encoding is acceptable.

If an Accept-Encoding field is present in a request, and if the server cannot send a response which is acceptable according to the Accept-Encoding header, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

If no Accept-Encoding field is present in a request, the server MAY assume that the client will accept any content coding. In this case, if "identity" is one of the available content-codings, then the server SHOULD use the "identity" content-coding, unless it has additional information that a different content-coding is meaningful to the client.

16.4. Accept-Language

The Accept-Language request-header field is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request. Note that the language specified applies to the presentation description and any reason phrases, but not the media content.

A language tag identifies a natural language spoken, written, or otherwise conveyed by human beings for communication of information to other human beings. Computer languages are explicitly excluded. The syntax and registry of RTSP 2.0 language tags is the same as that defined by [RFC4646].

Each language-range MAY be given an associated quality value which represents an estimate of the user’s preference for the languages specified by that range. The quality value defaults to "q=1". For example:
Accept-Language: da, en-gb;q=0.8, en;q=0.7

would mean: "I prefer Danish, but will accept British English and other types of English." A language-range matches a language-tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first tag character following the prefix is "-".
The special range "*", if present in the Accept-Language field, matches every tag not matched by any other range present in the Accept-Language field.

Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags for which this tag is a prefix. The prefix rule simply allows the use of prefix tags if this is the case.

The language quality factor assigned to a language-tag by the Accept-Language field is the quality value of the longest language-range in the field that matches the language-tag. If no language-range in the field matches the tag, the language quality factor assigned is 0. If no Accept-Language header is present in the request, the server SHOULD assume that all languages are equally acceptable. If an Accept-Language header is present, then all languages which are assigned a quality factor greater than 0 are acceptable.

16.5. Accept-Ranges

The Accept-Ranges request and response-header field allows indication of the format supported in the Range header. The client MUST include the header in SETUP requests to indicate which formats it support to receive in PLAY and PAUSE responses, and REDIRECT requests. The server MUST include the header in SETUP and 456 error responses to indicate the formats supported for the resource indicated by the request URI. The header MAY be included in GET_PARAMETER request and response pairs. The GET_PARAMETER request MUST contain a Session header to identify the session context the request are related to. The requester and responder will indicate their capabilities regarding Range formats respectively.

Accept-Ranges: NPT, SMPTE

The syntax is defined in Section 20.2.3.

16.6. Allow

The Allow message-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. 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 MUST 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

An RTSP client that wishes to authenticate itself with a server, usually, but not necessarily, after receiving a 401 response, does so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

If a request is authenticated and a realm specified, the same credentials SHOULD be valid for all other requests within this realm (assuming that the authentication scheme itself does not require otherwise, such as credentials that vary according to a challenge value or using synchronized clocks).

When a shared cache (see Section 18) receives a request containing an Authorization field, it MUST NOT return the corresponding response as a reply to any other request, unless one of the following specific exceptions holds:

1. If the response includes the "maxage" cache-control directive, the cache MAY use that response in replying to a subsequent request. But (if the specified maximum age has passed) a proxy cache MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request. (This is the defined behavior for maxage.) If the response includes "maxage=0", the proxy MUST always revalidate it before re-using it.

2. If the response includes the "must-revalidate" cache-control directive, the cache MAY use that response in replying to a subsequent request. But if the response is stale, all caches MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request.
3. If the response includes the "public" cache-control directive, it MAY be returned in reply to any subsequent request.

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 is 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

The Connection general-header field allows the sender to specify options that are desired for that particular connection and MUST NOT be communicated by proxies over further connections.

RTSP 2.0 proxies MUST parse the Connection header field before a message is forwarded and, for each connection-token in this field, remove any header field(s) from the message with the same name as the connection-token. Connection options are signaled by the presence of a connection-token in the Connection header field, not by any corresponding additional header field(s), since the additional header field may not be sent if there are no parameters associated with that connection option.

Message headers listed in the Connection header MUST NOT include end-to-end headers, such as Cache-Control.

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 requester. It MUST only be used in server to client responses.

The Connection-Credentials header in an RTSP response MUST, 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 [RFC4648] encoded binary structure containing a sequence of DER encoded X.509v3 certificates[RFC5280].

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:"rtsp://proxy2.example.com/";MIIDNTCC...

Where MIIDNTCC... is a BASE64 encoding of the following structure:

```
<table>
<thead>
<tr>
<th>Number of certificates</th>
<th>Size of certificate #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Size of certificate #2</td>
<td>Size of certificate #3</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------</td>
</tr>
</tbody>
</table>
```

16.13. Content-Base

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

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

If no Content-Base field is present, the base URI of an message body 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 message-body may be redefined within that message-body.


The Content-Encoding header field is used as a modifier to the media-type. When present, its value indicates what additional content codings have been applied to the message body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

The content-coding is a characteristic of the entity identified by the Request-URI. Typically, the message body is stored with this encoding and is only decoded before rendering or analogous usage.
However, a non-transparent proxy MAY modify the content-coding if the new coding is known to be acceptable to the recipient, unless the "no-transform" cache-control directive is present in the message.

If the content-coding of an message body is not "identity", then the response MUST include a Content-Encoding entity-header that lists the non-identity content-coding(s) used.

If the content-coding of an message body in a request message is not acceptable to the origin server, the server SHOULD respond with a status code of 415 (Unsupported Media Type).

If multiple encodings have been applied to a message body, the content codings MUST be listed in the order in which they were applied. Additional information about the encoding parameters MAY be provided by other header fields not defined by this specification.

16.15. Content-Language

The Content-Language header field describes the natural language(s) of the intended audience for the enclosed message body. Note that this might not be equivalent to all the languages used within the message body.

Language tags are mentioned in Section 16.4. The primary purpose of Content-Language is to allow a user to identify and differentiate entities according to the user’s own preferred language. Thus, if the body content is intended only for a Danish-literate audience, the appropriate field is

    Content-Language: da

If no Content-Language is specified, the default is that the content is intended for all language audiences. This might mean that the sender does not consider it to be specific to any natural language, or that the sender does not know for which language it is intended.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi," presented simultaneously in the original Maori and English versions, would call for

    Content-Language: mi, en

However, just because multiple languages are present within an entity does not mean that it is intended for multiple linguistic audiences. An example would be a beginner’s language primer, such as "A First Lesson in Latin," which is clearly intended to be used by an English-
literate audience. In this case, the Content-Language would properly only include "en".

Content-Language MAY be applied to any media type -- it is not limited to textual documents.

16.16. Content-Length

The Content-Length general-header field contains the length of the message body of the RTSP message (i.e. after the double CRLF following the last header). Unlike HTTP, it MUST be included in all messages that carry a message body beyond the header portion of the RTSP message. If it is missing, a default value of zero is assumed. Any Content-Length greater than or equal to zero is a valid value.

16.17. Content-Location

The Content-Location header field MAY be used to supply the resource location for the entity enclosed in the message when that entity is accessible from a location separate from the requested resource’s URI. A server SHOULD provide a Content-Location for the variant corresponding to the response entity; especially in the case where a resource has multiple entities associated with it, and those entities actually have separate locations by which they might be individually accessed, the server SHOULD provide a Content-Location for the particular variant which is returned.

The Content-Location value is not a replacement for the original requested URI; it is only a statement of the location of the resource corresponding to this particular entity at the time of the request. Future requests MAY specify the Content-Location URI as the request-URI if the desire is to identify the source of that particular entity.

A cache cannot assume that an entity with a Content-Location different from the URI used to retrieve it can be used to respond to later requests on that Content-Location URI. However, the Content-Location can be used to differentiate between multiple entities retrieved from a single requested resource.

If the Content-Location is a relative URI, the relative URI is interpreted relative to the Request-URI.

16.18. Content-Type

The Content-Type header indicates the media type of the message body sent to the recipient. 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 MUST 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

The Date header field represents the date and time at which the message was originated. The inclusion of the Date header in RTSP message follows these rules:

- An RTSP message, sent either by the client or the server, containing a body MUST include a Date header, if the sending host has a clock;
- Clients and servers are RECOMMENDED to include a Date header in all other RTSP messages, if the sending host has a clock;
- If the server does not have a clock that can provide a reasonable approximation of the current time, its responses MUST NOT include a Date header field. In this case, this rule MUST be followed: Some origin server implementations might not have a clock available. An origin server without a clock MUST NOT assign Expires or Last-Modified values to a response, unless these values were associated with the resource by a system or user with a reliable clock. It MAY assign an Expires value that is known, at or before server configuration time, to be in the past (this allows "pre-expiration" of responses without storing separate
Expires values for each resource).

A received message that does not have a Date header field MUST be assigned one by the recipient if the message will be cached by that recipient. An RTSP implementation without a clock MUST NOT cache responses without revalidating them on every use. An RTSP cache, especially a shared cache, SHOULD use a mechanism, such as NTP, to synchronize its clock with a reliable external standard.

The RTSP-date sent in a Date header SHOULD NOT represent a date and time subsequent to the generation of the message. It SHOULD represent the best available approximation of the date and time of message generation, unless the implementation has no means of generating a reasonably accurate date and time. In theory, the date ought to represent the moment just before the entity is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

16.21. Expires

The Expires message-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 message body). 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 RTSP-date:

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.

16.22. From

The From request-header field, if given, SHOULD contain an Internet e-mail address for the human user who controls the requesting user agent. The address SHOULD be machine-usable, as defined by "mailbox" in [RFC1123].

This header field MAY be used for logging purposes and as a means for identifying the source of invalid or unwanted requests. It SHOULD NOT be used as an insecure form of access protection. The interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsibility for the method performed. In particular, robot agents SHOULD include this header so that the person responsible for running the robot can be contacted if problems occur on the receiving end.

The Internet e-mail address in this field MAY be separate from the Internet host which issued the request. For example, when a request is passed through a proxy the original issuer’s address SHOULD be used.

The client SHOULD NOT send the From header field without the user’s approval, as it might conflict with the user’s privacy interests or their site’s security policy. It is strongly recommended that the user be able to disable, enable, and modify the value of this field at any time prior to a request.

16.23. 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 MTag 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 MTag
validation check fails, MUST response using 412 (Precondition Failed).

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

16.24. 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 MUST be returned without any message-body.

An example of the field is:

16.25. If-None-Match

This request header can be used with one or several message body tags to make DESCRIBE requests conditional. A client that has one or more message bodies previously obtained from the resource, can verify that none of those entities is current by including a list of their associated message body tags in the If-None-Match header field. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. As a special case, the value "*" matches any current entity of the resource.

If any of the message body tags match the message body tag of the message body that would have been returned in the response to a similar DESCRIBE request (without the If-None-Match header) on that resource, or if "*" is given and any current entity exists for that resource, then the server MUST NOT perform the requested method, unless required to do so because the resource’s modification date fails to match that supplied in an If-Modified-Since header field in the request. Instead, if the request method was DESCRIBE, the server SHOULD respond with a 304 (Not Modified) response, including the cache-related header fields (particularly MTag) of one of the message bodies that matched. For all other request methods, the server MUST respond with a status of 412 (Precondition Failed).

See Section 18.1.3 for rules on how to determine if two message body tags match.

If none of the message body tags match, then the server MAY perform the requested method as if the If-None-Match header field did not
exist, but MUST also ignore any If-Modified-Since header field(s) in the request. That is, if no message body tags match, then the server MUST NOT return a 304 (Not Modified) response.

If the request would, without the If-None-Match header field, result in anything other than a 2xx or 304 status, then the If-None-Match header MUST be ignored. (See Section 18.1.4 for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)

The meaning of "If-None-Match: *" is that the method MUST NOT be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see Section 16.55) exists, and SHOULD be performed if the representation does not exist. This feature is intended to be useful in preventing races between PUT operations.

The result of a request having both an If-None-Match header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

16.26. Last-Modified

The Last-Modified message-header field indicates the date and time at which the origin server believes the presentation description or media stream was last modified. For the method DESCRIBE, the header field indicates the last modification date and time of the description, for SETUP that of the media stream.

An origin server MUST NOT send a Last-Modified date which is later than the server’s time of message origination. In such cases, where the resource’s last modification would indicate some time in the future, the server MUST replace that date with the message origination date.

An origin server SHOULD obtain the Last-Modified value of the entity as close as possible to the time that it generates the Date value of its response. This allows a recipient to make an accurate assessment of the entity’s modification time, especially if the entity changes near the time that the response is generated.

RTSP servers SHOULD send Last-Modified whenever feasible.

16.27. Location

The Location response-header field is used to redirect the recipient to a location other than the Request-URI for completion of the request or identification of a new resource. For 3xx responses, the
location SHOULD indicate the server’s preferred URI for automatic redirection to the resource. The field value consists of a single absolute URI.

Note: The Content-Location header field (Section 16.17) differs from Location in that the Content-Location identifies the original location of the entity enclosed in the request. It is therefore possible for a response to contain header fields for both Location and Content-Location. Also see Section 18.2 for cache requirements of some methods.

16.28. Media-Properties

This general header is used in SETUP response or PLAY_NOTIFY requests to indicate the media’s properties that currently are applicable to the RTSP session. PLAY_NOTIFY MAY be used to modify these properties at any point. However, the client SHOULD have received the update prior to any action related to the new media properties take affect. For aggregated sessions the Media-Properties header will be returned in each SETUP response. The header received in the latest response is the one that applies on the whole session from this point until any future update. The header MAY be included without value in GET_PARAMETER requests to the server with a Session header included to query the current Media-Properties for the session. The responder MUST include the current session’s media properties.

The media properties expressed by this header is the one applicable to all media in the RTSP session. So for aggregated sessions the header expressed the combined media-properties. As a result aggregation of media MAY result in a change of the media properties, and thus the content of the Media-Properties header contained in subsequent SETUP responses.

The header contains a list of property values that are applicable to the currently setup media or aggregate of media as indicated by the RTSP URI in the request. No ordering are enforced within the header. Property values should be grouped into a single group that handles a particular orthogonal property. Values or groups that express multiple properties SHOULD NOT be used. The list of properties that can be expressed MAY be extended at any time. Unknown property values MUST be ignored.

This specification defines the following 4 groups and their property values:
Random Access:

Random-Access: Indicates that random access is possible. May optionally include a floating point value in seconds indicating the longest duration between any two random access points in the media.

Beginning-Only: Seeking is limited to the beginning only.

No-Seeking: No seeking is possible.

Content Modifications

Immutable: The content will not be changed during the life-time of the RTSP session.

Dynamic: The content may be changed based on external methods or triggers.

Time-Progressing: The media accessible progress as wallclock time progresses.

Retention

Unlimited: Content will be retained for the duration of the life-time of the RTSP session.

Time-Limited: Content will be retained at least until the specified wallclock time. The time must be provided in the absolute time format specified in Section 4.6.

Time-Duration: Each individual media unit is retained for at least the specified time duration. This definition allows for retaining data with a time based sliding window. The time duration is expressed as floating point number in seconds. 0.0 is a valid value as this indicates that no data is retained in a time-progressing session.

Supported Scale:

Scales: A quoted comma separated list of one or more decimal values or ranges of scale values supported by the content. A range has a start and stop value separated by a colon. A range indicates that the content supports fine grained selection of scale values. Fine grained allows for steps at least as small as one tenth of a scale value. Negative values are supported. The value 0 have no meaning and must not be used.
An Example of this header for first an on-demand content and then a live stream without recording.

On-demand:
Media-Properties: Random-Access=2.5s, Unlimited, Immutable,
Scale="-20, -10, -4, 0.5:1.5, 4, 8, 10, 15, 20"

Live stream without recording/timeshifting:
Media-Properties: No-Seeking, Time-Progressing, Time-Duration=0.0

16.29. Media-Range

The Media-Range general header is used to give the range of the media at the time of sending the RTSP message. This header MUST be included in SETUP response, and PLAY and PAUSE response for media that are Time-Progressing, and PLAY and PAUSE response after any change for media that are Dynamic, and in PLAY_NOTIFY request that are sent due to Media-Property-Update. Media-Range header without any range specifications MAY be included in GET_PARAMETER requests to the server to request the current range. The server MUST in this case include the current range at the time of sending the response.

The header MUST include range specifications for all time formats supported for the media, as indicated in Accept-Ranges header (Section 16.5) when setting up the media. The server MAY include more than one range specification of any given time format to indicate media that has non-continuous range.

For media that has the Time-Progressing property, the Media-Range values will only be valid for the particular point in time when it was issued. As wallclock progresses so will also the media range. However it shall be assumed that media time progress in direct relationship to wallclock time (with the exception of clock skew) so that a reasonably accurate estimation of the media range can be calculated.

16.30. MTag

The MTag response header MAY be included in DESCRIBE or SETUP responses. The message body 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 MTag is provided both inside the message body, e.g. within the
"a=mtag" attribute in SDP, and in the response message, then both tags MUST be identical. It is RECOMMENDED that the MTag is primarily given in the RTSP response message, to ensure that caches can use the MTag 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 message body tag in the session description as specified in Appendix D.1.9.

SETUP and DESCRIBE requests can be made conditional upon the MTag using the headers If-Match (Section 16.23) and If-None-Match (Section 16.25).

16.31. Notify-Reason

The Notify Reason header is solely used in the PLAY_NOTIFY method. It indicates the reason why the server has sent the asynchronous PLAY_NOTIFY request (see Section 13.5).

16.32. 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 does not 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 MUST look up if there exist a binding between this Pipelined-Requests identifier for the current persistent connection and an RTSP session ID. If that exists 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 MUST 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 MUST be created. In case the request contains both a Session header and the Pipelined-Requests header the Pipelined-Requests MUST 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 MUST 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 MUST 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.

16.33. Proxy-Authenticate

The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI.

The HTTP access authentication process is described in [RFC2617]. Unlike WWW-Authenticate, the Proxy-Authenticate header field applies only to the current connection and SHOULD NOT be passed on to downstream clients. However, an intermediate proxy might need to obtain its own credentials by requesting them from the downstream client, which in some circumstances will appear as if the proxy is forwarding the Proxy-Authenticate header field.

16.34. Proxy-Authorization

The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy which requires authentication. The Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

The HTTP access authentication process is described in [RFC2617]. Unlike Authorization, the Proxy-Authorization header field applies only to the next outbound proxy that demanded authentication using the Proxy-Authenticate field. When multiple proxies are used in a
chain, the Proxy-Authorization header field is consumed by the first outbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

16.35. 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 MUST 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 16.42 for more details on the mechanics of this message and a usage example. See discussion in the proxies section (Section 17.1) about when to consider that a feature requires proxy support.

Example of use:

    Proxy-Require: play.basic

16.36. 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 MUST 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 MUST 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 pro.example.com

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

16.37. 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 16.6) 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.
16.38. Range

The Range header specifies a time range in PLAY (Section 13.4), PAUSE (Section 13.6), SETUP (Section 13.3), REDIRECT (Section 13.10), and PLAY_NOTIFY (Section 13.5) requests and responses. It MAY be included in GET_PARAMETER request from client to the server with only a Range format and no value to request the current media position independent if the session is in playing or ready state in the included format. The server SHALL if supporting that range format respond with the current playing point or pause point as the start of the range. If an explicit stop point was used in the previous PLAY request, then that value shall be included as stop point. Note that if the server is currently under any type of media playback manipulation affecting the interpretation of Range, like Scale, that is also required to be included in any GET_PARAMETER response to provide complete information.

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.

Example:

Range: clock=19960213T143205Z-

The Range header contains a range of one single range format. A range is a half-open interval with a start and an end point, including the start point, but excluding the end point. A range may either be fully specified with explicit values for start point and end point, or have either start or end point be implicit. An implicit start point indicates the session's pause point, and if no pause point is set the start of the content. An implicit end point indicates the end of the content. The usage of both implicit start and end point is not allowed in the same range header, however, the exclusion of the range header has that meaning, i.e. from pause point (or start) until end of content.

Regarding the half-open intervals; a range of A-B starts exactly at time A, but ends 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.

Please note the difference between NPT time scales’ "now" and an implicit start value. Implicit value reference the current pause-point. While "now" is the currently ongoing time. In a time-progressing session with recording (retention for some or full time) the pause point may be 2 min into the session while now could be 1 hour into the session.

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 16.44) 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.

16.39. Referer

The Referer request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained (the "referer", although the header field is misspelled.) The URI refers to that of the presentation
description, typically retrieved via HTTP. The Referer request-header allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field MUST NOT be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

If the field value is a relative URI, it SHOULD be interpreted relative to the Request-URI. The URI MUST NOT include a fragment.

See [H15.1.3] for security considerations on Referer.

16.40. Retry-After

The Retry-After response-header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. This field MAY also be used with any 3xx (Redirection) response to indicate the minimum time the user-agent is asked wait before issuing the redirected request. The value of this field can be either an RTSP-date or an integer number of seconds (in decimal) after the time of the response.

Example:
Retry-After: 120

In the latter example, the delay is 2 minutes.

16.41. Request-Status

This request header is used to indicate the end result for requests that takes time to complete, such a PLAY (Section 13.4). It is sent in PLAY_NOTIFY (Section 13.5) with the end-of-stream reason to report how the PLAY request concluded, either in success or in failure. The header carries a reference to the request is reports on using the CSeq number for the session indicated by the Session header in the request. It provides both a numerical status code (according to Section 8.1.1) and a human readable reason phrase.

Example:
Request-Status: cseq=63 status=500 reason="Media data unavailable"

16.42. 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 16.49) 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 MUST use the error code 551 (Option Not Supported). This header does not apply to proxies, for the same functionality in respect to proxies see Proxy-Require header (Section 16.35) with the exception of media modifying proxies. Media modifying proxies due to their nature of handling media in a way that is very similar to what a server, do need to understand also the server features to correctly serve the client.

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 MUST 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 16.35). See discussion in the proxies section (Section 17.1) about when to consider that a feature requires proxy support.

16.43. 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 be included in SETUP responses to provide synchronization information when changing transport parameters, see Section 13.3. The RTP-Info header MAY also be included in GET_PARAMETER requests from client to server without any value to indicate a request for this information. In such a case the Range header MUST also be included in the request. The server SHALL respond if the session is in playing state with the Range header filled in with the current playback point and with the corresponding RTP-Info values.

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 MUST use the Request-URI as base URI. This parameter MUST be present.

ssrc: The Synchronization source (SSRC) that the RTP timestamp and sequence number provide applies to. This parameter MUST 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: MUST 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 (wallclock) 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

Lets 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

16.44. 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 content time increase at twice the playback time. 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 C.1 for description on how RTP handles this.

The transmitted data rate SHOULD NOT be changed by selection of a different scale value. The resulting bit-rate should be in reasonably close to the nominal bit-rate of the content for Scale = 1. The server has to actively manipulate the data when needed to meet the bitrate constraints. 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, for example, it may time-scale the audio while preserving pitch or, less desirably, deliver fragments of audio, or completely mute the audio.

The server and content may restrict the range of scale values that it supports. The supported values are indicated by the Media-Properties header (Section 16.28). The client SHOULD only indicate values indicated to be supported. However, as the values may change as the content progresses a requested value may no longer be valid when the request arrives. Thus an non-supported value in a request does not generate an error, only forces the server to choose the closest value. The response MUST always 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 MUST indicate this with the use of the "play.scale" feature-tag.

When indicating a negative scale for a reverse playback, the Range header MUST indicate a decreasing range as described in Section 16.38.

Example of playing in reverse at 3.5 times normal rate:
  Scale: -3.5
  Range: npt=15-10

16.45. Seek-Style

When a client sends a PLAY request with a Range header to perform a random access to the media, the client does not know if the server will pick the first media samples or the first random access point prior to the request range. Depending on use case, the client may have a strong preference. To express this preference and provide the client with information on how the server actually acted on that preference the Seek-Style header is defined.
Seek-Style is a general header that MAY be included in any PLAY request to indicate the client’s preference for any media stream that has random access properties. The server MUST always include the header in any PLAY response for media with random access properties to indicate what policy was applied. A Server that receives an unknown Seek-Style policy MUST ignore it and select the server default policy.

This specification defines the following seek policies that may be requested:

**RAP:** Random Access Point (RAP) is the behavior of requesting the server to locate the closest previous random access point that exist in the media aggregate and deliver from that. By requesting a RAP media quality will be the best possible as all media will be delivered from a point where full media state can be established in the media decoder.

**First-Prior:** The first-prior policy will start delivery with the media unit that has a playout time first prior to the requested time. For discrete media that would only include media units that would still be rendered at the request time. For continuous media that is media that will be render during the requested start time of the range.

**Next:** The next media units after the provided start time of the range. For continuous framed media that would mean the first next frame after the provided time. For discrete media the first unit that is to be rendered after the provided time. The main usage is for this case is when the client knows it has all media up to a certain point and would like to continue delivery so that a complete non-interrupted media playback can be achieved. Example of such scenarios include switching from a broadcast-multicast delivery to a unicast based delivery. This policy MUST only be used on the client’s explicit request.

Please note that these expressed preferences exist for optimizing the startup time or the media quality. The "Next" policy breaks the normal definition of the Range header to enable a client to request media with minimal overlap, although some may still occur for aggregated sessions. RAP and First-Prior both fulfill the requirement of providing media from the requested range and forward. However, unless RAP is used, the media quality for many media codecs using predictive methods can be severely degraded unless additional data is available as, for example, already buffered, or through other side channels.
16.46. Speed

The Speed request-header field requests the server to deliver specific amounts of nominal media time per unit of delivery time, contingent on the server’s ability and desire to serve the media stream at the given speed. The client requests the delivery speed to be within a given range with an upper and lower bound. The server SHALL deliver at the highest possible speed within the range, but not faster than the upper-bound, for which the underlying network path can support the resulting transport data rates. As long as any speed value within the given range can be provided the server SHALL NOT modify the media quality. Only if the server is unable to deliver media at the speed value provided by the lower bound shall it reduce the media quality.

Implementation of the Speed functionality by the server is OPTIONAL. The server can indicate its support through a feature-tag, play.scale. The lack of a Speed header in the response is an indication of lack of support of this functionality.

The speed parameter values are expressed as a positive decimal value, e.g., a value of 2.0 indicates that data is to be delivered twice as fast as normal. A speed value of zero is invalid. The range is specified in the form "lower bound - upper bound". The lower bound value may be smaller or equal to the upper bound. All speeds may not be possible to support. Therefore the server MAY modify the requested values to the closest supported. The actual supported speed MUST be included in the response. Note however that the use cases may vary and that Speed value ranges such as 0.7 - 0.8, 0.3-2.0, 1.0-2.5, 2.5-2.5 all have their usage.

Example:
    Speed: 1.0 - 2.5

Use of this header changes the bandwidth used for data delivery. It is meant for use in specific circumstances where delivery of the presentation at a higher or lower rate is desired. The main use cases are buffer operations or local scale operations. 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. To perform Speed operations the server needs to ensure that the network path can support the resulting bit-rate. Thus the media transport needs to support feedback so that the server can react and adapt to the available bitrate.

16.47. Server

The Server response-header field contains information about the software used by the origin server to handle the request. The field
can contain multiple product tokens and comments identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application.

Example:
Server: PhonyServer/1.0

If the response is being forwarded through a proxy, the proxy application MUST NOT modify the Server response-header. Instead, it SHOULD include a Via field (Section 16.56).

16.48. 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, REDIRECT 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 MUST 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, SET_PARAMETER, GET_PARAMETER, and REDIRECT, and MUST NOT be included in DESCRIBE. In an RTSP response the session header MUST 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 MUST also be included in the response, OPTIONS, GET_PARAMETER, and SET_PARAMETER, and MUST NOT be included in DESCRIBE.

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) MUST be returned if the session identifier is invalid.

The header MAY include the session timeout period. If not explicitly
provided this value is set to 60 seconds. As this affects how often session keep-alives are needed values smaller than 30 seconds are not recommended. However larger than default values can be useful in applications of RTSP that have inactive but established sessions for longer time periods.

60 seconds was chosen as session timeout value due to: Resulting in not to frequent keep-alive messages and having low sensitivity to variations in request response timing. If one reduces the timeout value to below 30 seconds the corresponding request response timeout becomes a significant part of the session timeout. 60 seconds also allows for reasonably rapid recovery of committed server resources in case of client failure.

16.49. Supported

The Supported header 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 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

16.50. Terminate-Reason

The Terminate-Reason request header allows the server when sending a REDIRECT or TERMINATE request to provide a reason for the session termination and any additional information. This specification identifies three reasons for Redirections and may be extended in the future:

Server-Admin: The server needs to be shutdown for some administrative reason.
Session-Timeout: A client’s session is kept alive for extended periods of time and the server has determined that it needs to reclaim the resources associated with this session.

Internal-Error An internal error that is impossible to recover from has occurred forcing the server to terminate the session.

The Server may provide additional parameters containing information around the redirect. This specification defines the following ones.

time: Provides a wallclock time when the server will stop provide any service.

user-msg: An UTF-8 text string with a message from the server to the user. This message SHOULD be displayed to the user.

16.51. Timestamp

The Timestamp general-header 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.

16.52. Transport

The Transport request and response header 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.

A Transport request header MAY contain a list of transport options acceptable to the client, in the form of multiple transport specification entries. Transport specifications are comma separated, listed in decreasing order of preference. Parameters may be added to each transport specification, separated by a semicolon. The server MUST return a Transport response-header in the response to indicate the values actually chosen if any. If not transport specification is supported no transport header is returned and the request MUST be responded using the status code 461 (Unsupported Transport) (Section 15.4.26). In case more than one transport specification was present in the request, the server MUST return the single (transport-spec) which was actually chosen if any. 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.

The Transport header MAY also be used in subsequent SETUP requests to change transport parameters. A server MAY refuse to change parameters of an existing stream.

A transport specification may only contain one of any given parameter within it. Parameters MAY be given in any order. Additionally, it may only contain either of the unicast or the multicast transport type parameter. All parameters need to be understood in a transport specification, if not, the transport specification MUST be ignored. RTSP proxies of any type that uses or modifies the transport specification, e.g., access proxy or security proxy, MUST remove specifications with unknown parameters before forwarding the RTSP message. If that result in no remaining transport specification the proxy shall send a 461 (Unsupported Transport) (Section 15.4.26) response without any Transport header.

The Transport header 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 for unicast transport:

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 MUST 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 some 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.

For Multicast there is several methods for specifying addresses but they are different in how they work compared with unicast:

dest_addr with client picked address: The address and relevant parameters like TTL (scope) for the actual multicast group to deliver the media to. There are security implications (Section 21) with this method that needs to be addressed if using this method because a RTSP server can be used as a DoS attacker on a existing multicast group.

dest_addr using Session Description Information: The information included in the transport header can all be coming from the session description, e.g. the SDP c= and m= line. This mitigates some of the security issues of the previous methods as it is the session provider that picks the multicast group and scope. The client MUST include the information if it is available in the session description.

No dest_addr: The behavior when no explicit multicast group is present in a request is not defined.

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 21.1) 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 MUST 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. Responses to request including the Transport header with a dest_addr parameter SHOULD include the full destination address that is actually used by the server. The server MUST NOT remove address information present already in the request when responding unless the protocol requires it.

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
originates. 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 interval, e.g., interleaved=4-5 in cases where the transport choice for the media stream requires it, e.g. for RTP with RTCP. The channel number given in the request are only a guidance from the client to the server on what channel number(s) to use. The server MAY set any valid channel number in the response. The declared channel(s) are bi-directional, so both end-parties MAY send data on the given channel. One example of such usage is the second channel used for RTCP, where both server and client sends RTCP packets on the same channel.

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

Multicast-specific:

ttl: multicast time-to-live for IPv4. When included in requests the value indicate the TTL value that the client request the server to use. In a response, the value actually being used by the server is returned. A server will need to consider what values that are reasonable and also the authority of the user to set this value. Corresponding function are not needed for IPv6 as the scoping is part of the address.
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 MUST 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. 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 of 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 C.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).

RTCP-mux: Use to negotiate the usage of RTP and RTCP multiplexing on a single underlying transport stream. The presence of this parameter in a SETUP request indicates the clients support and desire to use RTP and RTCP multiplexing. The client MAY still include two transport streams in the Transport header specification to handle cases if RTP and RTCP multiplexing is not supported by the server. If the server supports the usage of RTP and RTCP multiplexing it SHALL include this parameter in the response and strip down the transport address negotiation to a single src_addr and dest_addr. If the server does not support RTP and RTCP multiplexing it removes this parameter from the transport specification in response and treat the specification as if the parameter was not included.

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

```plaintext
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: Thu, 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
     Media-Properties: Random-Access=0.6, Dynamic,
                        Time-Limited=20081128T165900
```

16.53. Unsupported

The Unsupported response-header lists the features not supported by the server. In the case where the feature was specified via the Proxy-Require field (Section 16.35), 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 MUST NOT be forwarded.

See Section 16.42 for a usage example.

16.54. User-Agent

The User-Agent request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests. The field can contain multiple product tokens and comments identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.
Example:
User-Agent: PhonyClient/1.2

16.55. Vary

The Vary field value indicates the set of request-header fields that fully determines, while the response is fresh, whether a cache is permitted to use the response to reply to a subsequent request without revalidation. For uncacheable or stale responses, the Vary field value advises the user agent about the criteria that were used to select the representation. A Vary field value of "*" implies that a cache cannot determine from the request headers of a subsequent request whether this response is the appropriate representation.

An RTSP server SHOULD include a Vary header field with any cacheable response that is subject to server-driven negotiation. Doing so allows a cache to properly interpret future requests on that resource and informs the user agent about the presence of negotiation on that resource. A server MAY include a Vary header field with a non-cacheable response that is subject to server-driven negotiation, since this might provide the user agent with useful information about the dimensions over which the response varies at the time of the response.

A Vary field value consisting of a list of field-names signals that the representation selected for the response is based on a selection algorithm which considers ONLY the listed request-header field values in selecting the most appropriate representation. A cache MAY assume that the same selection will be made for future requests with the same values for the listed field names, for the duration of time for which the response is fresh.

The field-names given are not limited to the set of standard request-header fields defined by this specification. Field names are case-insensitive.

A Vary field value of "*" signals that unspecified parameters not limited to the request-headers (e.g., the network address of the client), play a role in the selection of the response representation. The "*" value MUST NOT be generated by a proxy server; it may only be generated by an origin server.

16.56. Via

The Via general-header field MUST be used by proxies to indicate the intermediate protocols and recipients between the user agent and the server on requests, and between the origin server and the client on responses. The field is intended to be used for tracking message
forwards, avoiding request loops, and identifying the protocol
capabilities of all senders along the request/response chain.

Multiple Via field values represents each proxy that has forwarded
the message. Each recipient MUST append its information such that
the end result is ordered according to the sequence of forwarding
applications.

Proxies (e.g., Access Proxy or Translator Proxy) SHOULD NOT, by
default, forward the names and ports of hosts within the private/
protected region. This information SHOULD only be propagated if
explicitly enabled. If not enabled, the via-received of any host
behind the firewall/NAT SHOULD be replaced by an appropriate
pseudonym for that host.

For organizations that have strong privacy requirements for hiding
internal structures, a proxy MAY combine an ordered subsequence of
Via header field entries with identical sent-protocol values into a
single such entry. Applications MUST NOT combine entries which have
different received-protocol values.

16.57. WWW-Authenticate

The WWW-Authenticate response-header field MUST be included in 401
(Unauthorized) response messages. The field value consists of at
least one challenge that indicates the authentication scheme(s) and
parameters applicable to the Request-URI.

The HTTP access authentication process is described in [RFC2617].
User agents are advised to take special care in parsing the WWW-
Authenticate field value as it might contain more than one challenge,
or if more than one WWW-Authenticate header field is provided, the
contents of a challenge itself can contain a comma-separated list of
authentication parameters.
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 and the below are often combined.

Caching Proxy: This type of proxy is used to reduce the workload on servers and connections. By caching the description and media streams, i.e., the presentation, the proxy can serve a client with content, but without requesting it from the server once it has been cached and has not become stale. See the caching Section 18. This type of proxy is also expected to understand RTSP end-point functionality, i.e., functionality identified in the Require header in addition to what Proxy-Require demands.

Translator Proxy: This type of proxy is used to ensure that an RTSP client get access to servers and content on an external network or using content encodings not supported by the client. The proxy performs the necessary translation of addresses, protocols or encodings. This type of proxy is expected to also understand RTSP end-point functionality, i.e. functionality identified in the Require header in addition to what Proxy-Require demands.

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. This type of proxies are required to redirect the media to themselves or a controlled gateway that perform the translation before the media can reach the client.

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. This proxy can also limit the clients access to certain type of content. This proxy can perform its function without redirecting the media between the server and client. However, in deployments with private address spaces this proxy is likely to be combined with the access proxy. Anyway, the functionality of this proxy is usually closely tied into understand all aspects of how the media transport.
Auditing Proxy: RTSP proxies can also provide network owners with a logging and audit point for RTSP sessions, e.g. for corporations that tracks their employees usage of the network. This type of proxy can perform its function without inserting itself or any other node in the media transport. This proxy type can also accept unknown methods as it doesn’t interfere with the clients requests.

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 19.3.2. 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.

17.1. Proxies and Protocol Extensions

The existence of proxies must always be considered when developing new RTSP extensions. Most type of proxies will need to implement any new method to operate correct in the presence of that extension. New headers will be possible to introduce without being blocked by proxies not yet updated. However, it is important to consider if this header and its function is required to be understood by the proxy or can be forwarded. If the header needs to be understood a feature-tag representing the functionality needs to be included in the Proxy-Require header. Below are guidelines for analysis if the header needs to be understood. The transport header and its parameters also shows that headers that are extensible and requires correct interpretation in the proxy also requires handling rules.

When defining a new RTSP header it needs to be considered if RTSP proxies are required to understand them to achieve correct functionality. Determining this is not easy as the functionality for proxies are widely varied as can be understood from the above list of functionality. When evaluating this one can dived the functionality into three main categories:
Media modifying: The caching and translator proxies are modifying the actual media and therefore needs to understand also request directed to the server that affects how the media is rendered. Thus this type of proxies needs to also understand the server side functionality.

Transport modifying: The access and the security proxy both need to understand the how the transport is performed, either for opening pinholes or to translate the outer headers, e.g. IP and UDP.

Non-modifying: The audit proxy is special in that it do not modify the messages in other ways than to insert the Via header. That makes it possible for this type to forward RTSP message that contains different type of unknown methods, headers or header parameters.

Based on the above classification one should evaluate if ones functionality requires the Transport modifying type of proxies to understand it or not.
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 requester, 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.
18.1. Validation Model (HTTP)

When a cache has a stale entry that it would like to use as a response to a client’s request, it first has to check with the origin server (or possibly an intermediate cache with a fresh response) to see if its cached entry is still usable. We call this "validating" the cache entry. Since we do not want to have to pay the overhead of retransmitting the full response if the cached entry is good, and we do not want to pay the overhead of an extra round trip if the cached entry is invalid, the RTSP protocol supports the use of conditional methods.

The key protocol features for supporting conditional methods are those concerned with "cache validators." When an origin server generates a full response, it attaches some sort of validator to it, which is kept with the cache entry. When a client (user agent or proxy cache) makes a conditional request for a resource for which it has a cache entry, it includes the associated validator in the request.

The server then checks that validator against the current validator for the entity, and, if they match (see Section 18.1.3), it responds with a special status code (usually, 304 (Not Modified)) and no message body. Otherwise, it returns a full response (including message body). Thus, we avoid transmitting the full response if the validator matches, and we avoid an extra round trip if it does not match.

In RTSP, a conditional request looks exactly the same as a normal request for the same resource, except that it carries a special header (which includes the validator) that implicitly turns the method (usually DESCRIBE) into a conditional.

The protocol includes both positive and negative senses of cache-validating conditions. That is, it is possible to request either that a method be performed if and only if a validator matches or if and only if no validators match.

Note: a response that lacks a validator may still be cached, and served from cache until it expires, unless this is explicitly prohibited by a cache-control directive (see Section 16.10). However, a cache cannot do a conditional retrieval if it does not have a validator for the entity, which means it will not be refreshable after it expires.
18.1.1. Last-Modified Dates

The Last-Modified header (Section 16.26) value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the entity has not been modified since the Last-Modified value.

18.1.2. Message Body Tag Cache Validators

The MTag response-header field value, a message body tag, provides for an "opaque" cache validator. This might allow more reliable validation in situations where it is inconvenient to store modification dates, where the one-second resolution of RTSP-date values is not sufficient, or where the origin server wishes to avoid certain paradoxes that might arise from the use of modification dates.

Message body tags are described in Section 5.3

18.1.3. Weak and Strong Validators

Since both origin servers and caches will compare two validators to decide if they represent the same or different entities, one normally would expect that if the message body (i.e., the presentation description) or any associated message body headers changes in any way, then the associated validator would change as well. If this is true, then we call this validator a "strong validator." We call message body (i.e., the presentation description) or any associated message body headers an entity for a better understanding.

However, there might be cases when a server prefers to change the validator only on semantically significant changes, and not when insignificant aspects of the entity change. A validator that does not always change when the resource changes is a "weak validator."

Message body tags are normally "strong validators," but the protocol provides a mechanism to tag an message body tag as "weak." One can think of a strong validator as one that changes whenever the bits of an entity changes, while a weak value changes whenever the meaning of an entity changes. Alternatively, one can think of a strong validator as part of an identifier for a specific entity, while a weak validator is part of an identifier for a set of semantically equivalent entities.
Note: One example of a strong validator is an integer that is incremented in stable storage every time an entity is changed.

An entity’s modification time, if represented with one-second resolution, could be a weak validator, since it is possible that the resource might be modified twice during a single second.

Support for weak validators is optional. However, weak validators allow for more efficient caching of equivalent objects; for example, a hit counter on a site is probably good enough if it is updated every few days or weeks, and any value during that period is likely "good enough" to be equivalent.

A "use" of a validator is either when a client generates a request and includes the validator in a validating header field, or when a server compares two validators.

Strong validators are usable in any context. Weak validators are only usable in contexts that do not depend on exact equality of an entity. For example, either kind is usable for a conditional DESCRIBE of a full entity. However, only a strong validator is usable for a sub-range retrieval, since otherwise the client might end up with an internally inconsistent entity.

Clients MAY issue DESCRIBE requests with either weak validators or strong validators. Clients MUST NOT use weak validators in other forms of request.

The only function that the RTSP protocol defines on validators is comparison. There are two validator comparison functions, depending on whether the comparison context allows the use of weak validators or not:

- The strong comparison function: in order to be considered equal, both validators MUST be identical in every way, and both MUST NOT be weak.

- The weak comparison function: in order to be considered equal, both validators MUST be identical in every way, but either or both of them MAY be tagged as "weak" without affecting the result.

An message body tag is strong unless it is explicitly tagged as weak.

A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:
o The validator is being compared by an origin server to the actual current validator for the entity and,

o That origin server reliably knows that the associated entity did not change twice during the second covered by the presented validator.

OR

o The validator is about to be used by a client in an If-Modified-Since, because the client has a cache entry for the associated entity, and

o That cache entry includes a Date value, which gives the time when the origin server sent the original response, and

o The presented Last-Modified time is at least 60 seconds before the Date value.

OR

o The validator is being compared by an intermediate cache to the validator stored in its cache entry for the entity, and

o That cache entry includes a Date value, which gives the time when the origin server sent the original response, and

o The presented Last-Modified time is at least 60 seconds before the Date value.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-Modified values are generated from different clocks, or at somewhat different times during the preparation of the response. An implementation MAY use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

If a client wishes to perform a sub-range retrieval on a value for which it has only a Last-Modified time and no opaque validator, it MAY do this only if the Last-Modified time is strong in the sense described here.
18.1.4. Rules for When to Use Entity Tags and Last-Modified Dates

We adopt a set of rules and recommendations for origin servers, clients, and caches regarding when various validator types ought to be used, and for what purposes.

RTSP origin servers:

- SHOULD send an message body tag validator unless it is not feasible to generate one.
- MAY send a weak message body tag instead of a strong message body tag, if performance considerations support the use of weak message body tags, or if it is unfeasible to send a strong message body tag.
- SHOULD send a Last-Modified value if it is feasible to send one, unless the risk of a breakdown in semantic transparency that could result from using this date in an If-Modified-Since header would lead to serious problems.

In other words, the preferred behavior for an RTSP origin server is to send both a strong message body tag and a Last-Modified value.

In order to be legal, a strong message body tag MUST change whenever the associated entity value changes in any way. A weak message body tag SHOULD change whenever the associated entity changes in a semantically significant way.

Note: in order to provide semantically transparent caching, an origin server must avoid reusing a specific strong message body tag value for two different entities, or reusing a specific weak message body tag value for two semantically different entities. Cache entries might persist for arbitrarily long periods, regardless of expiration times, so it might be inappropriate to expect that a cache will never again attempt to validate an entry using a validator that it obtained at some point in the past.

RTSP clients:

- If an message body tag has been provided by the origin server, MUST use that message body tag in any cache-conditional request (using If-Match or If-None-Match).
- If only a Last-Modified value has been provided by the origin server, SHOULD use that value in non-subrange cache-conditional requests (using If-Modified-Since).
o If both an message body tag and a Last-Modified value have been provided by the origin server, SHOULD use both validators in cache-conditional requests.

An RTSP origin server, upon receiving a conditional request that includes both a Last-Modified date (e.g., in an If-Modified-Since header) and one or more message body tags (e.g., in an If-Match, If-None-Match, or If-Range header field) as cache validators, MUST NOT return a response status of 304 (Not Modified) unless doing so is consistent with all of the conditional header fields in the request.

Note: The general principle behind these rules is that RTSP servers and clients should transmit as much non-redundant information as is available in their responses and requests. RTSP systems receiving this information will make the most conservative assumptions about the validators they receive.

18.1.5. Non-validating Conditionals

The principle behind message body tags is that only the service author knows the semantics of a resource well enough to select an appropriate cache validation mechanism, and the specification of any validator comparison function more complex than byte-equality would open up a can of worms. Thus, comparisons of any other headers are never used for purposes of validating a cache entry.

18.2. Invalidation After Updates or Deletions (HTTP)

The effect of certain methods performed on a resource at the origin server might cause one or more existing cache entries to become non-transparently invalid. That is, although they might continue to be "fresh," they do not accurately reflect what the origin server would return for a new request on that resource.

There is no way for the RTSP protocol to guarantee that all such cache entries are marked invalid. For example, the request that caused the change at the origin server might not have gone through the proxy where a cache entry is stored. However, several rules help reduce the likelihood of erroneous behavior.

In this section, the phrase "invalidate an entity" means that the cache will either remove all instances of that entity from its storage, or will mark these as "invalid" and in need of a mandatory revalidation before they can be returned in response to a subsequent request.

Some HTTP methods MUST cause a cache to invalidate an entity. This is either the entity referred to by the Request-URI, or by the
Location or Content-Location headers (if present). These methods are:

- DESCRIBE
- SETUP

In order to prevent denial of service attacks, an invalidation based on the URI in a Location or Content-Location header MUST only be performed if the host part is the same as in the Request-URI.

A cache that passes through requests for methods it does not understand SHOULD invalidate any entities referred to by the Request-URI.
19. 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.

19.1. 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. Client MUST implement both basic and digest authentication [RFC2617] so that Server who requires the client to authenticate can trust that the capability is present.

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 19.2.

19.2. RTSP over TLS

RTSP MUST follow the same guidelines with regards to TLS [RFC5246] 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 MUST 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 MUST 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" MUST 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 round trips). 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 19.3 gives further recommendations of TLS usage with proxies.

19.3. 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 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 MUST when initiating the next hop TLS connection use the incoming TLS connections cipher suite 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).

19.3.1. 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) MUST 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 19.3.2 for further details.
If the Accept-Credentials header is not included in the RTSP request from the client, then the "Proxy" method MUST be used as default. If another method than the "Proxy" is to be used, then the Accept-Credentials header MUST 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 previous 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 does not accept the credentials of the next hop, the entity MUST 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 requests 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, it 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 desires to send the request, it silently fails.

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

19.3.2. User approved TLS procedure

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

- 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).

- Extract the peer certificate chain for the TLS session.

- 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.

- 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 MUST 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 rtsps://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: "rtsps://test.example.org";
    MDINTCCAp...

C->P: SETUP rtsps://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 "rtsps://test.example.org";sha-256;
    dPYD7txoGTbAqZZQJ+vaekyH4=
    Accept-Ranges: NPT, SMPTE, UTC
P->S: SETUP rtsps://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 "rtsps://test.example.org";sha-256;
    dPYD7txoGTbAqZZQJ+vaekyH4=
    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 requester accepts. The procedure of including the credentials in each request rather than building state in each proxy, avoids the need for revocation procedures.
20. Syntax

The RTSP syntax is described in an Augmented Backus-Naur Form (ABNF) as defined in RFC 5234 [RFC5234]. It uses the basic definitions present in RFC 5234.

Please note that ABNF strings, e.g. "Accept", are case insensitive as specified in section 2.3 of RFC 5234.

20.1. Base Syntax

RTSP header 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 line break. 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 = %x20-7E / %x80-FF ; any OCTET except CTls
tspecials = "(" / ")" / "<" / ">" / ":" / BACKSLASH / DQ
/ ":" / "[" / "]" / "?" / ":="
/ "(" / ")" / SP / HT
token = 1*(%x21 / %x23-27 / %x2A-2B / %x2D-2E / %x30-39
/ %x41-5A / %x5E-7A / %x7C / %x7E)
/ ; 1*<any CHAR except CTls or special>
quoted-string = ( DQ *qdtext DQ )
qdtext = %x20-21 / %x23-7E / %x80-FF ; any TEXT except <">
quoting-pair = BACKSLASH CHAR
ctext = %x20-27 / %x2A-7E
/ %x80-FF ; any OCTET except CTls, ")" and "(")
generic-param = token [ EQUAL gen-value ]
gen-value = token / host / quoted-string
safe = "$" / "_" / ":" / ":" / ":"
extra = ":" / "%" / "*" / "#' / "(" / ")" / ")" / "("
rtsp-extra = ":" / "%" / "*" / "#" / "("
HEX = DIGIT / ":A" / ":B" / ":C" / ":D" / ":E" / ":F"
/ ":a" / ":b" / ":c" / ":d" / ":e" / "f"
LHEX = DIGIT / %x61-66 ; 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 / "+" / "/"
20.2. RTSP Protocol Definition

20.2.1. 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-REQ-URI   =  schemes "::" URI-req-rest
RTSP-URI-Ref   =  RTSP-URI / RTSP-Relative
RTSP-REQ-Ref   =  RTSP-REQ-URI / RTSP-REQ-Rel
schemes       =  "rtsp" / "rtsps" / scheme
scheme        =  < As defined in RFC 3986>
URI-rest      =  hier-part [ "?" query ] [="#" fragment ]
URI-req-rest  =  hier-part [ "?" query ]
               ; Note fragment part not allowed in requests
hier-part      =  "//" authority path-abempty
RTSP-Relative =  relative-part [ "?" query ] [="#" fragment ]
RTSP-REQ-Rel   =  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>
fragment      =  < 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 = "smpte" 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 "T" 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)

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

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

Response      = Status-Line
                *((general-header
                         / response-header
                         / message-header) CRLF)
                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"
         / "PLAY_NOTIFY"
         / "REDIRECT"
         / "SETUP"
         / "SET_PARAMETER"
         / "TEARDOWN"
         / extension-method

extension-method = token

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

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

message-body = 1*OCTET

Status-Code  = "100" ; Continue
               / "200" ; OK
               / "301" ; Moved Permanently
               / "302" ; Found
               / "303" ; See Other
               / "304" ; Not Modified
               / "305" ; Use Proxy
               / "400" ; Bad Request
               / "401" ; Unauthorized
               / "402" ; Payment Required
Internet-Draft   Real Time Streaming Protocol 2.0 (RTSP)       June 2009

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

extension-code = 3DIGIT

Reason-Phrase = *TEXT

general-header = Cache-Control
/ Connection
/ CSeq
/ Date
/ Media-Properties
/ Media-Range
/ Pipelined-Requests
/ Proxy-Supported
/ Seek-Style
/ 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
/ Notify-Reason
/ Proxy-Require
/ Range
/ Referer
/ Request-Status
/ Require
/ Scale
/ Session
/ Speed
/ Supported
/ Terminate-Reason
/ Transport
/ User-Agent
/ extension-header
response-header = Accept-Credentials
                / Accept-Ranges
                / Connection-Credentials
                / MTag
                / Location
                / Proxy-Authenticate
                / Public
                / Range
                / Retry-After
                / RTP-Info
                / Scale
                / Session
                / Server
                / Speed
                / Transport
                / Unsupported
                / Vary
                / WWW-Authenticate
                / extension-header

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

20.2.3.  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-type-range *(SEMI accept-param)
media-type-range = ( "/" "/"
                  / ( m-type SLASH "*" )
                  / ( m-type SLASH m-subtype )
                  ) *( SEMI m-parameter )
accept-param = ("q" EQUAL qvalue) / generic-param
qvalue = ( "0" [ "." *3DIGIT ]
         / ( "1" [ "." *3("0") ] )
Accept-Credentials = "Accept-Credentials" HCOLON cred-decision
cred-decision = ("User" [LWS cred-info])
cred-info = cred-info-data *(COMMA cred-info-data)
cred-info-data = DQ RTSP-REQ-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 = token
Accept-Language = "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
acceptable-ranges = (range-unit *(COMMA range-unit))
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

Blocksize = "Blocksize" HCOLON 1*DIGIT

Cache-Control = "Cache-Control" HCOLON cache-directive *(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
cred-chain = DQ RTSP-REQ-URI DQ SEMI base64

Connection = "Connection" HCOLON connection-token *(COMMA connection-token)
connection-token = token

Content-Base = "Content-Base" HCOLON RTSP-URI-Ref
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

Content-Length = "Content-Length" HCOLON 1*DIGIT
Content-Location = "Content-Location" HCOLON RTSP-REQ-Ref
Content-Type = ( "Content-Type" / "c" ) HCOLON media-type
media-type = m-type SLASH m-subtype *(SEMI m-parameter)
m-type = discrete-type / composite-type
discrete-type = "text" / "image" / "audio" / "video"
/ "application" / extension-token
composite-type = "message" / "multipart" / extension-token
extension-token = ietf-token / x-token
ietf-token = token
x-token = "x-" token
m-subtype = extension-token / iana-token
iana-token = token
m-parameter = m-attribute EQUAL m-value
m-attribute = token
m-value = token / quoted-string

CSeq = "CSeq" HCOLON cseq-nr
cseq-nr = 1*9DIGIT
Date = "Date" HCOLON RTSP-date
RTSP-date = rfc1123-date ; HTTP-date
rfc1123-date = wkday ";" SP date1 SP time SP "GMT"
date1 = 2DIGIT SP month SP 4DIGIT
; day month year (e.g., 02 Jun 1982)
time = 2DIGIT ":" 2DIGIT ":" 2DIGIT
; 00:00:00 - 23:59:59
wkday = "Mon" / "Tue" / "Wed"
/ "Thu" / "Fri" / "Sat" / "Sun"
month = "Jan" / "Feb" / "Mar" / "Apr"
/ "May" / "Jun" / "Jul" / "Aug"
/ "Sep" / "Oct" / "Nov" / "Dec"
Expires = "Expires" HCOLON RTSP-date
From = "From" HCOLON from-spec
from-spec = ( name-addr / addr-spec ) *( SEMI from-param )
name-addr = [ display-name ] LAQUOT addr-spec RAQUOT
addr-spec = RTSP-REQ-URI / absolute-URI
absolute-URI = < As defined in RFC 3986>
display-name = *(token LWS) / quoted-string
from-param = tag-param / generic-param
tag-param = "tag" EQUAL token
If-Match = "If-Match" HCOLON ("*" / message-tag-list)
message-tag-list = message-tag *(COMMA message-tag)
message-tag = [ weak ] opaque-tag
weak = "W/"
opaque-tag = quoted-string
If-Modified-Since = "If-Modified-Since" HCOLON RTSP-date
If-None-Match = "If-None-Match" HCOLON ("*" / message-tag-list)
Last-Modified = "Last-Modified" HCOLON RTSP-date
Location = "location" HCOLON RTSP-REQ-URI
Media-Properties = "Media-Properties" HCOLON [media-prop-list]
media-prop-list = media-prop-value *(COMMA media-prop-value)
media-prop-value = ("Random-Access" [EQUAL POS-FLOAT])
  / "Beginning-Only"
  / "No-Seeking"
  / "Immutable"
  / "Dynamic"
  / "Time-Progressing"
  / "Unlimited"
  / ("Time-Limited" EQUAL utc-range-spec)
  / ("Time-Duration" EQUAL POS-FLOAT)
  / ("Scales" EQUAL scale-value-list)
  / media-prop-ext
media-prop-ext = token [EQUAL (1*rtsp-unreserved / quoted-string)]
scale-value-list = DQ scale-entry *(COMMA scale-entry) DQ
scale-entry = scale-value / (scale-value COLON scale-value)
scale-value = ["-" ] 1*8DIGIT ["." 1*8DIGIT]
Media-Range = "Media-Range" HCOLON [ranges-list]
ranges-list = ranges-spec *(COMMA ranges-spec)
MTag = "MTag" HCOLON message-tag
Notify-Reason = "Notify-Reason" HCOLON Notify-Reas-val
Notify-Reas-val = "end-of-stream"
  / "media-properties-update"
  / "scale-change"
  / Notify-Reason-extension
Notify-Reason-extension = token
Pipelined-Requests = "Pipelined-Requests" HCOLON startup-id
startup-id = 1*8DIGIT
Proxy-Authenticate  =  "Proxy-Authenticate" HCOLON challenge-list
challenge-list    =  challenge *(COMMA 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 RTSP-REQ-Ref *(1*SP RTSP-REQ-Ref ) RDQUOT
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 *(COMMA feature-tag)
Proxy-Supported   =  "Proxy-Supported" HCOLON feature-tag *(COMMA feature-tag)
Public            =  "Public" HCOLON Method *(COMMA Method)
Range             =  "Range" HCOLON ranges-spec
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-REQ-Ref
Request-Status    =  "Request-Status" HCOLON req-status-info
req-status-info   =  cseq-info LWS status-info LWS reason-info
cseq-info         =  "cseq" EQUAL cseq-nr
status-info       =  "status" EQUAL Status-Code
reason-info       =  "reason" EQUAL DQ Reason-Phrase DQ
Require           =  "Require" HCOLON feature-tag-list
feature-tag-list  =  feature-tag *(COMMA feature-tag)
RTP-Info = "RTP-Info" HCOLON [rtsp-info-spec *(COMMA rtsp-info-spec)]
rtsp-info-spec = stream-url 1*ssrc-parameter
stream-url = "url" EQUAL DQ RTSP-REQ-Ref DQ
ssrc-parameter = LWS "ssrc" EQUAL ssrc HCOLON
                  ri-parameter *(SEMI ri-parameter)
ri-parameter = ("seq" EQUAL 1*DIGIT)
                  / ("rtptime" EQUAL 1*DIGIT)
                  / generic-param
Retry-After = "Retry-After" HCOLON delta-seconds
                  [ comment ] *( SEMI retry-param )
retry-param = ("duration" EQUAL delta-seconds)
                  / generic-param
Scale = "Scale" HCOLON ["-"] 1*DIGIT [ "." *DIGIT ]
Seek-Style = "Seek-Style" HCOLON Seek-S-values
Seek-S-values = "RAP"
                  / "First-Prior"
                  / "Next"
                  / Seek-S-value-ext
Seek-S-value-ext = token
Speed = "Speed" HCOLON POS-FLOAT MINUS POS-FLOAT
Server = "Server" HCOLON ( product / comment )
       *(LWS (product / comment))
product = token [SLASH product-version]
product-version = token
comment = LPAREN *( ctext / quoted-pair) RPAREN
Session = "Session" HCOLON session-id
          [ SEMI "timeout" EQUAL delta-seconds ]
Supported = "Supported" HCOLON [feature-tag-list]
Terminate-Reason = "Terminate-Reason" HCOLON TR-Info
TR-Info = TR-Reason *(SEMI TR-Parameter)
TR-Reason = "Session-Timeout"
  / "Server-Admin"
  / "Internal-Error"
  / token
TR-Parameter = TR-time / TR-user-msg / generic-param
TR-time = "time" EQUAL utc-time
TR-user-msg = "user-msg" EQUAL quoted-string

Timestamp = "Timestamp" HCOLON timestamp-value LWS [delay]
timestamp-value = *DIGIT [ "." *DIGIT ]
delay = *DIGIT [ "." *DIGIT ]

Transport = "Transport" HCOLON transport-spec
  *(COMMA transport-spec)
transport-spec = transport-id *trns-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
trns-parameter = (SEMI ( "unicast" / "multicast" ))
/ (SEMI "interleaved" EQUAL channel [ "-" channel ])
/ (SEMI "ttl" EQUAL ttl)
/ (SEMI "layers" EQUAL 1*DIGIT)
/ (SEMI "ssrc" EQUAL ssrc *(SLASH 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)
/ (SEMI "RTCP-mux")
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 / quoted-string)
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

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

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
via-ttl = "ttl" EQUAL ttl
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-list

20.3. SDP extension Syntax

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

control-attribute = "a=control:" *SP RTSP-REQ-REF
a-range-def = "a=range:" ranges-spec CRLF
a-mtag-def = "a=mtag:" message-tag CRLF
21. Security Considerations

Because of the similarity in syntax and usage between RTSP servers and HTTP servers, the security considerations outlined in [H15] apply also.

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

DNS Spoofing: Presumably, given the longer connection times typically associated to RTSP sessions relative to HTTP sessions, RTSP client DNS optimizations should be less prevalent. Nonetheless, the recommendations provided in [H15.3] are still relevant to any implementation which attempts to rely on a DNS-to-IP mapping to hold beyond a single use of the mapping.
Location Headers and Spoofing: If a single server supports multiple organizations that do not trust 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 21.1.

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 [RFC5246] 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
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 19.3) 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.

Resource Exhaustion As RTSP is a stateful protocol and establish resource usages on the server there is a clear possibility to attack the server by trying to overbook these resources to perform an denial of service attack. This attack can be both against ongoing sessions and to prevent others from establishing sessions. RTSP agents will need to have mechanism to prevent single peers from consuming extensive amounts of resources.

21.1. 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 mechanisms 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
- 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 the
server may choose 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.
22. 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 2.7 "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 5226 [RFC5226], namely "First Come, First Served", "Expert Review, "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 decreasing 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);

22.1. Feature-tags

22.1.1. 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.
22.1.2. 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 MUST indicate if the feature-tag applies to clients, servers, or proxies only or any combinations of these. Any proprietary feature MUST have as the first part of the name a vendor tag, which identifies the organization.

22.1.3. 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 this specification. 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.

22.2. RTSP Methods

22.2.1. Description

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

22.2.2. Registering New Methods with IANA

A new method MUST be registered through an IETF Standards Action. 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.

22.2.3. Registered Entries

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

22.3. RTSP Status Codes

22.3.1. 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.

22.3.2. Registering New Status Codes with IANA

A new status code can only be registered by an IETF Standards Action. 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.

22.3.3. Registered Entries

RFCXXXX, registers the numbered status code defined in the ABNF entry "Status-Code" except "extension-code" in Section 20.2.2.

22.4. RTSP Headers

22.4.1. 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.
22.4.2. Registering New Headers with IANA

Registrations in the registry can be done following the Expert Review policy. A specification SHOULD be provided, preferable an IETF RFC or other Standards Developing Organization specification. The minimal information in a registration request is the header name and the contact information.

The specification SHOULD 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.

22.4.3. 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].
The use of "x-" is NOT RECOMMENDED but the above headers in the register list was defined prior to the clarification.

22.5. Accept-Credentials

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

22.5.1. Accept-Credentials policies

In Section 19.3.1 three policies for how to handle certificates are specified. Further policies may be defined and MUST be registered with IANA using the following rules:

- Registering requires an IETF Standards Action
- A registration is required to 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

22.5.2. 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 cause interoperability problems. Therefore the bar for registering new algorithms is intentionally placed high.

Any registration of a new hash algorithm MUST fulfill the following requirement:

- Follow the IETF Standards Action policy.
22.6. Cache-Control  Cache Directive Extensions

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

- Registering requires an IETF Standards Action.
- 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:

22.7. Media Properties
22.7.1. Description

The media streams being controlled by RTSP can have many different properties. The media properties required to cover the use cases that was in mind when writing the specification are defined. However, it can be expected that further innovation will result in new use cases or media streams with properties not covered by the ones specified here. Thus new media properties can be specified. As new media properties may need a substantial amount of new definitions to correctly specify behavior for this property the bar is intended to be high.

22.7.2. Registration Rules

Registering new media property MUST fulfill the following requirements

- Follow the Specification Required policy and get the approval of the designated Expert.
- Have an ABNF definition of the media property value name that meets "media-prop-ext" definition
- A Contact Person for the Registration
- Description of all changes to the behavior of the RTSP protocol as result of these changes.

22.7.3. Registered Values

This specification registers the 9 values listed in Section 16.28.

22.8. Notify-Reason header

22.8.1. Description

Notify-Reason values used to indicate why a notification was sent. They may also imply that certain headers are required for the client to act properly upon the information the notification carries. New notification behaviors need to be described to result in interoperable usage, thus a specification of each new value is required.

22.8.2. Registration Rules

Registrations for new Notify-Reason value MUST fulfill the following requirements
Follow the Specification Required policy and get the approval of the designated Expert.

Have a ABNF definition of the Notify reason value name that meets "Notify-Reason-extension" definition

A Contact Person for the Registration

Description of which headers shall be included in the request and response, when it should be sent, and any effect it has on the server client state.

22.8.3. Registered Values

This specification registers 3 values defined in the Notify-Reas-val ABNF

- end-of-stream
- media-properties-update
- scale-change

22.9. 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 MUST fulfill the following requirements:

Follow the Specification Required policy.

An ABNF definition of the range format that fulfills the "range-ext" definition.

A Contact person for the registration.

Rules for how one handles the range when using a negative Scale.

22.10. Terminate-Reason Header

The Terminate-Reason header (Section 16.50) has two registries for extensions.

22.10.1. Redirect Reasons

Registrations are done under the policy of Expert Review. The registered value needs to follow syntax, i.e. be a token. The
specification needs to provide definition of what the procedures that is to be followed when a client receives this redirect reason. This specification registers two values:

- Session-Timeout
- Server-Admin

### 22.10.2. Terminate-Reason Header Parameters

Registrations are done under the policy of Specification Required. The registrations must define a syntax for the parameter that also follows the allowed by the RTSP 2.0 specification. A contact person is also required. This specification registers:

- time
- user-msg

### 22.11. RTP-Info header parameters

#### 22.11.1. Description

The RTP-Info header (Section 16.43) carries one or more parameter value pairs with information about a particular point in the RTP stream. RTP extensions or new usages may need new types of information. As RTP information that could be needed is likely to be generic enough and to maximize the interoperability registration requires specification required.

#### 22.11.2. Registration Rules

Registrations for new Notify-Reason value MUST fulfill the following requirements

- Follow the Specification Required policy and get the approval of the designated Expert.
- Have a ABNF definition that meets the "generic-param" definition
- A Contact Person for the Registration

#### 22.11.3. Registered Values

This specification registers 2 parameter value pairs:
22.12. Seek-Style Policies

22.12.1. Description

New seek policies may be registered, however a large number of these will complicate implementation substantially. The impact of unknown policies is that the server will not honor the unknown and use the server default policy instead.

22.12.2. Registration Rules

Registrations of new Seek-Style polices MUST fulfill the following requirements

- Follow the Specification Required policy.
- Have a ABNF definition of the Seek-Style policy name that meets "Seek-S-value-ext" definition
- A Contact Person for the Registration
- Description of which headers shall be included in the request and response, when it should be sent, and any affect it has on the server client state.

22.12.3. Registered Values

This specification registers 3 values:

- RAP
- First-Prior
- Next

22.13. 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.

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

- Registering uses the policy of Specification Required.
- 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 C.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: Use of the RTP[RFC3550] protocol for media transport in combination with the "[RFC5124] over UDP. The usage is explained in RFC XXXX, appendix Appendix C.1.

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 C.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 C.2.2.


RTP/SAVPF/TCP: Use of the RTP[RFC3550] protocol for media transport in combination with the "[RFC5124] over TCP. The usage is explained in RFC XXXX, appendix Appendix C.2.2.

22.13.2. Transport modes

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

- Registering requires an IETF Standards Action.
- 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.

22.13.3. Transport Parameters

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

- Registering uses the Specification Required policy.
- 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 16.52.
22.14. URI Schemes

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

22.14.1. The rtsp URI Scheme

URI scheme name: rtsp

Status: Permanent

URI scheme syntax: See Section 20.2.1 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 (SET_PARAMETER 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 need to be reviewed and considered in any implementation utilizing this scheme.

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

Author/Change controller: IETF
22.14.2. The rtsps URI Scheme

URI scheme name: rtsps

Status: Permanent

URI scheme syntax: See Section 20.2.1 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 (SET_PARAMETER 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 need to be reviewed and considered in any implementation utilizing this scheme.

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

Author/Change controller: IETF

References: RFC 2326, RFC 3986, RFC 3987, RFC XXXX
22.14.3. 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 (SET_PARAMETER 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

References: RFC 2326, RFC 3986, RFC 3987

22.15. SDP attributes

This specification defines three 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: mtag
Long form: Message 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

22.16. Media Type Registration for text/parameters

Type name: text

Subtype name: parameters

Required parameters:

Optional parameters:

Encoding considerations:

Security considerations: This format may carry any type of parameters. Some can clear have security requirements, like privacy, confidentiality or integrity requirements. The format has no built in security protection. For the usage it was defined the transport can be protected between server and client using TLS. However, care must be take to consider if also the proxies...
are trusted with the parameters in case hop-by-hop security is used. If stored as file in file system the necessary precautions needs to be taken in relation to the parameters requirements including object security such as S/MIME [RFC3851].

Interoperability considerations: This media type was mentioned as a fictional example in RFC 2326 but was not formally specified. This have resulted in usage of this media type which may not match its formal definition.

Published specification: RFC XXXX, Appendix F.

Applications that use this media type: Applications that use RTSP and have additional parameters they like to read and set using the RTSP GET_PARAMETER and SET_PARAMETER methods.

Additional information:

Magic number(s):

File extension(s):

Macintosh file type code(s):

Person & email address to contact for further information: Magnus Westerlund (magnus.westerlund@ericsson.com)

Intended usage: Common

Restrictions on usage: None

Author: Magnus Westerlund (magnus.westerlund@ericsson.com)

Change controller: IETF

Addition Notes:
23. References

23.1. Normative References

[3gpp-26234]

[FIPS-pub-180-2]

[I-D.ietf-avt-rtp-and-rtcp-mux]


Internet-Draft   Real Time Streaming Protocol 2.0 (RTSP)   June 2009


23.2. Informative References


[ISO.8601.2000]  

[RFC0822]  

[RFC1123]  

[RFC1305]  

[RFC1644]  

[RFC2068]  

[RFC2326]  

[RFC2663]  

[RFC2974]  

[RFC3261]  

[RFC3388]  

[RFC4145]  

[Stevens98]  
Appendix A. 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.

A.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: Thu, 23 Jan 1997 15:35:06 GMT
    Content-Type: application/sdp
    Content-Length: 271
    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
c=IN IP4 0.0.0.0
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; ssrc=93CB001E;
              dest_addr="192.0.2.53:8000"/"192.0.2.53:8001";
              src_addr="192.0.2.5:9000"/"192.0.2.5:9001"
   Session: 12345678
   Expires: 24 Jan 1997 15:35:12 GMT
   Date: 23 Jan 1997 15:35:12 GMT
   Accept-Ranges: NPT
   Media-Properties: Random-Access=0.02, Immutable, Unlimited

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; ssrc=A813FC13;
              dest_addr="192.0.2.53:8002"/"192.0.2.53:8003";
              src_addr="192.0.2.5:9002"/"192.0.2.5:9003";
   Session: 12345678
   Expires: 24 Jan 1997 15:35:13 GMT
   Date: 23 Jan 1997 15:35:13 GMT
   Accept-Range: NPT
   Media-Properties: Random-Access=0.8, Immutable, Unlimited

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
   CSeq: 4
   User-Agent: PhonyClient/1.2
   Range: npt=30-
   Seek-Style: RAP
   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=30-623.10
Seek-Style: RAP
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

C->M: PAUSE rtsp://example.com/twister.3gp/ RTSP/2.0
CSeq: 5
User-Agent: PhonyClient/1.2
Session: 12345678

M->C: RTSP/2.0 200 OK
CSeq: 5
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:36:01 GMT
Session: 12345678
Range: npt=34.57-623.10

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
CSeq: 6
User-Agent: PhonyClient/1.2
Range: npt=34.57-623.10
Seek-Style: Next
Session: 12345678

M->C: RTSP/2.0 200 OK
CSeq: 6
Server: PhonyServer/1.0
Date: 23 Jan 1997 15:36:01 GMT
Session: 12345678
Range: npt=34.57-623.10
Seek-Style: Next
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

A.2. Media on Demand using Pipelining

This example is basically the example above (Appendix A.1), but now utilizing pipelining to speed up the setup. It requires only two round trip times until the media starts flowing. First of all, the
session description is retrieved to determine what media resources need to be setup. In the second step, one sends 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
  Server: PhonyServer/1.0
  Date: Thu, 23 Jan 1997 15:35:06 GMT
  Content-Type: application/sdp
  Content-Length: 271
  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
c=IN IP4 0.0.0.0
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-
   Seek-Style: RAP
   Session: 12345678
   Pipelined-Requests: 7654

M->C: RTSP/2.0 200 OK
   CSeq: 2
   Server: PhonyServer/1.0
   Transport: RTP/AVP; unicast;
             dest_addr="192.0.2.53:8000"/"192.0.2.53:8001";
             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
   Pipelined-Requests: 7654
   Media-Properties: Random-Access=0.2, Unmutable, Unlimited

M->C: RTSP/2.0 200 OK
   CSeq: 3
   Server: PhonyServer/1.0
   Transport: RTP/AVP; unicast;
             dest_addr="192.0.2.53:8002"/"192.0.2.53:8003";
             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
   Pipelined-Requests: 7654
   Media-Properties: Random-Access=0.8, Unmutable, Unlimited

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=0-623.10
   Seek-Style: RAP
   RTP-Info: url="rtsp://example.com/twister.3gp/trackID=4"
             ssrc=0D12F123;seq=12345;rtptime=3450012,
A.3. Media on Demand (Unicast)

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

In this example, the client is only interested in the last part of the movie.
C->W: GET /twister.sdp HTTP/1.1
    Host: www.example.com
    Accept: application/sdp

W->C: HTTP/1.0 200 OK
    Date: Thu, 23 Jan 1997 15:35:06 GMT
    Content-Type: application/sdp
    Content-Length: 278
    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
    c=IN IP4 0.0.0.0
    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="192.0.2.53:3056"/"192.0.2.53:3057";
               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
    Media-Properties: Random-Access=0.02, Immutable, Unlimited
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="192.0.2.53:3058"/"192.0.2.53:3059",
              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="192.0.2.53:3058"/"192.0.2.53:3059";
              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
   Media-Properties: Random-Access=1.2, Unmutable, Unlimited

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
   Seek-Style: First-Prior
   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
Even though the audio and video track are on two different servers
that may start at slightly different times and may drift with respect
to each other over time, the client can perform initial
synchronization 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.
A.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 need 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: 163
    Server: PhonyServer/1.0
    Date: Thu, 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
    c=IN IP4 0.0.0.0
    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="192.0.2.53:6970"/"192.0.2.53:6971";
    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
Media-Properties: Begining-Only, Unmutable, Unlimited

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
Seek-Style: RAP
RTP-Info: url="rtsp://foo.com/test.wav/streamid=0"
    ssrc=OD12F123: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:
A.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/1.1
   Host: www.example.com

W→C: HTTP/1.1 200 OK
   Content-Type: text/html

   <html>
   ...
   <href "Streamed 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: 183
   Server: PhonyServer/1.0
   Date: Thu, 23 Jan 1997 15:35:06 GMT
   Supported: play.basic

   a=control: rtsp://live.example.com/concert/audio
   a=range:npt=0-
A.6. Capability Negotiation

This examples illustrate how the client and server determines their capability to support a special feature, in this case "play.scale". The server, through the clients 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.
When the client sends its SETUP request it tells the server that it 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="192.0.2.53:3056"/"192.0.2.53:3057",
              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="192.0.2.53:3056"/"192.0.2.53:3057";
              src_addr="192.0.2.5:5000"/"192.0.2.5:5001"
   Server: PhonyServer/2.0
   Accept-Ranges: NPT, SMPTE
   Media-Properties: Random-Access=0.8, Unmutable, Unlimited
Appendix B. 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 MUST take precedence.

B.1. States

The state machine contains three states, described below. For each state there exist a table which shows which requests and events that are 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.

B.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.

B.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.

B.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 need 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 MUST NOT result in any change of the session state, except if the error is not possible to recover from. A unrecoverable error MUST 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>SET_PARAMETER</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 range</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>PLAY_NOTIFY</td>
<td></td>
<td>Play</td>
<td>200</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, IFI</td>
<td>Play</td>
<td>Change transport param.</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 C. 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".

C.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.

C.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" MUST NOT be used. This addressing and multiplexing is used as defined with use of channel numbers and the interleaved parameter.

C.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 profile 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
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 MUST 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 MUST 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 MUST NOT be sent.
- The RTCP/UDP packets from the client to the server MUST 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 MUST NOT be sent.
- The RTP/UDP packets from the client to the server MUST 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.

C.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 MUST 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 MUST be followed.

C.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 MUST 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].

C.1.5. SAVPF/UDP

The RTP profile "Extended Secure RTP Profile for RTCP-based Feedback (RTP/SAVPF)" [RFC5124] is an RTP profile (SAVPF) that MAY be used in RTSP sessions using RTP. All that is defined for AVP MUST 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].

C.1.6. RTCP usage with RTSP

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

C.1.6.1. Media synchronization

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

C.1.6.2. RTSP Session keep-alive

RTCP traffic from the RTSP client to the RTSP server MUST 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.
C.1.6.3. Bit-rate adaption

RTCP Receiver reports and any additional feedback from the client MUST be used adapt the bit-rate used over the transport for all cases when RTP is sent over UDP. An RTP sender without reserved resources MUST 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.

C.1.6.4. RTP and RTCP Multiplexing

RTSP can be used to negotiate the usage of RTP and RTCP multiplexing as described in [I-D.ietf-avt-rtp-and-rtcp-mux]. This allows servers and client to reduce the amount of resources required for the session by only requiring one underlying transport stream per media stream instead of two when using RTP and RTCP. This lessens the server port consumption and also the necessary state and keep-alive work when operating across Network and Address Translators [RFC2663].

Content must be prepared with some consideration for RTP and RTCP multiplexing, mainly ensuring that the RTP payload types used does not collide with the ones used for RTCP packet types this option likely needs explicit support from the content unless the RTP payload types can be remapped by the server and that is correctly reflected in the session description. Beyond that support of this feature should come at little cost and much gain.

It is recommended that if the content and server supports RTP and RTCP multiplexing that this is indicated in the session description, for example using the SDP attribute "a=rtcp-mux". If the SDP message contains the a=rtcp-mux attribute for a media stream, the server MUST support RTP and RTCP multiplexing. If indicated or otherwise desired by the client it can include the Transport parameter "RTCP-mux" in any transport specification where it desires to use RTCP-mux. The server will indicate if it supports RTCP-mux. Server and Client SHOULD support RTP and RTCP multiplexing.

C.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 MUST be "rtp" and the lower layer MUST be TCP. The profile may be any of the above specified ones; AVP, AVPF, SAVP or SAVPF.
C.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.

C.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 16.52) 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 16.52. 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 help 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 MUST 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 MUST respond using the 464 "Data Transport Not Ready Yet" (Section 15.4.29) 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 MUST 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 16.52) 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 Appendix A.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: Thu, 23 Jan 1997 15:35:06 GMT
    Content-Type: application/sdp
    Content-Length: 227
    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
c=IN IP4 0.0.0.0
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
  Media-Properties: Random-Access=0.8, Unmutable, Unlimited

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=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=30-623.10
  Seek-Style: First-Prior
  RTP-Info: url="rtsp://example.com/twister.3gp/trackID=1"
            ssrc=4F312DD8:seq=54321;rtptime=2876889

C.3. Handling Media Clock Time 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]. Two cases occur, the first is when a new PLAY request replaces an old ongoing request and the new request results in a jump in the media. This should produce in the RTP layer a continuous media stream. A client may also directly following a completed PLAY request perform a new PLAY request. This will result in some gap in the media layer. The below text will look into both cases.

A PLAY request that replaces a ongoing request allows the media layer rendering the RTP stream without being affected by jumps in media clock time. The RTP timestamps for the new media range is set so that they become continuous with the previous media range in the previous request. The RTP sequence number for the first packet in
the new range will be the next following the last packet in the previous range, i.e. monotonically increasing. The goal is to allow the media rendering layer to work without interruption or reconfiguration across the jumps in media clock. This should be possible in all cases of replaced PLAY requests for media that has random-access properties. In this case care is needed to align frames or similar media dependent structures.

In cases where jumps in media clock time are a result of RTSP signalling operations arriving after a completed PLAY operation, the request timing will result in that media becomes non-continuous. The server becomes unable to send the media so that it arrive timely and still carry timestamps to make the media stream continuous. In these cases the server will produce RTP streams where there are gaps in the RTP timeline for the media. In such cases, if the media has frame structure, aligning the timestamp for the next frame with the previous structure reduces the burden to render this media. The gap should represent the time the server hasn’t been serving media, e.g. the time between the end of the media stream or a PAUSE request and the new PLAY request. In these cases the RTP sequence number would normally be monotonically increasing across the gap.

For RTSP sessions with media that lacks random access properties, like live streams, any media clock jump is commonly result of correspondingly long pause of delivery. The RTP timestamp will have increased in direct proportion to the duration of the paused delivery. Note also that in this case the RTP sequence number should be the next packet number. If not, the RTCP packet loss reporting will indicate as loss all packets not received between the point of pausing and later resuming. This may trigger congestion avoidance mechanisms. An allowed exception from the above recommendation on monotonically increasing RTP sequence number is live media streams, likely being relayed. In this case, when the client resumes delivery, it will get the media that is currently being delivered to the server itself. For this type of basic delivery of live streams to multiple users over unicast, individual rewriting of RTP sequence numbers becomes quite a burden. For solutions that anyway caches media, timeshifts, etc, the rewriting should be a minor issue.

The goal when handling jumps in media clock time is that the provided stream is continuous without gaps in RTP timestamp or sequence number. However, when delivery has been halted for some reason the RTP timestamp when resuming MUST represent the duration the delivery was halted. RTP sequence number MUST generally be the next number, i.e. monotonically increasing modulo 65536. For media resources with the properties Time-Progressing and Time-Duration=0.0 the server MAY create RTP media streams with RTP sequence number jumps in them due to client first halting delivery and later resuming it (PAUSE and
then later PLAY). However, servers utilizing this exception must
take into consideration the resulting RTCP receiver reports that
likely contains loss report for all the packets part of the
discontinuity. A client can not rely on that a server will align
when resuming playing even if it is RECOMMENDED. The RTP-Info header
will provide information on how the server acts in each case.

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. Having the RTP timestamp jump will also affect the
RTCP measurements based on this.

As an example, assume a RTP timestamp 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: abcdefgh
   Range: npt=10-15
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 4
   Session: abcdefgh
   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

Upon the completion of the requested playback the server sends a
PLAY_NOTIFY

Upon the completion 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).
C.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: abcdefg
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 5
   Session: abcdefg
   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: abcdefg
   User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
   CSeq: 6
   Session: abcdefg
   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.

Note Further: To ensure correct media decoding and usually jitter-buffer handling resetting some of the state when issuing a PLAY request is needed.
C.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.

C.6. Scaling with RTP

For scaling (see Section 16.44), 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 16.46) 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.

C.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 16.43) header provides the first sequence number of the next segment.

C.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.

C.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.

C.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 MUST 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.

C.11. 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 or 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, profiles and lower layer transport for this extension needs to be defined. 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. May need new headers similar to RTP info to carry information.

- Discuss congestion control for media, especially if transport without built in congestion control is used.

See the IANA section (Section 22) for information how to register new attributes.
Appendix D. Use of SDP for RTSP Session Descriptions

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.

D.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]):

D.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 MUST be different from any media level URI. The presence of a session level control attribute MUST be interpreted as support for aggregated control. The control attribute MUST 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 20.3.

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 MUST 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 MUST be treated as if it were an empty embedded URI, and thus inherit the entire base URI.

Note, RFC 2326 was very unclear on the processing of relative URI and several RTSP 1.0 implementations at the point of publishing this document did not perform RFC 3986 processing to determine the resulting URI, instead simple concatenation is common. To avoid this issue completely it is recommended to use absolute URI in the SDP.

The URI handling for SDPs from container files need special consideration. For example lets assume that a container file has the URI: "rtsp://example.com/container.mp4". Lets 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.

D.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 numbers are 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 which 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 C, 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

D.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 "fmt" attribute described below.

D.1.4. Format-Specific Parameters

Format-specific parameters are conveyed using the "fmt" media attribute. The syntax of the "fmt" 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 fmt parameters, like for example the "ptime" attribute for most audio encodings.

D.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.
D.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 MUST 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 MUST take care to provide RTSP Range (see Section 16.38) values that are consistent with what is presented in the SDP for the content. There is 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 MUST 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 20.3.

Examples:
  a=range:npt=0-34.4368
  a=range:clock=19971113T211503Z-19971113T220300Z
  Non seekable stream of unknown duration:
  a=range:npt=0-

D.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.

D.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 MUST be "0.0.0.0", and for type "IP6", this value MUST be "0:0:0:0:0:0:0:0" (can also be written as "::"), i.e. the unspecified address according to RFC 4291 [RFC4291].

D.1.9. Message Body Tag

The optional "a=mtag" 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 16.23) 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 20.3.

Example:
\[ a=mtag:"158bb3e7c7fd62ce67f12b533f06b83a" \]

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

D.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:
\[ v=0
\]
\[ o=- 2890844256 2890842807 IN IP4 192.0.2.56
\]
\[ s=I came from a web page
\]
\[ e=adm@example.com
\]
\[ c=IN IP4 0.0.0.0
\]
\[ t=0 0
\]
\[ m=video 8002 RTP/AVP 31
\]
\[ a=control:rtsp://audio.com/movie.aud
\]
\[ m=audio 8004 RTP/AVP 3
\]
\[ a=control:rtsp://video.com/movie.vid
\]

Note that the position of the control 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.

D.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 D.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: Thu, 23 Jan 1997 15:35:06 GMT
   Content-Type: application/sdp
   Content-Base: rtsp://example.com/movie/
   Content-Length: 227

   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
   a=control:*
   t=0 0
   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.

D.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 [RFC4566].
Appendix E. RTSP Use Cases

This Appendix 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.

E.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).

E.2. Unicast Distribution of Live Content

This use case is similar to the above on-demand content case (see Appendix E.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.

E.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. This memo has to rely on a external solution to solve this issue.

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

E.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 [RFC2974] 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 known 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 the PLAY_NOTIFY method (see Section 13.5) with a suitable extension could provide clear benefits.
Appendix F. Text format for Parameters

A resource of type "text/parameters" consists of either 1) a list of parameters (for a query) or 2) a list of parameters and associated values (for an response or setting of the parameter). Each entry of the list is a single line of text. Parameters are separated from values by a colon. The parameter name MUST only use US-ASCII visible characters while the values are UTF-8 text strings. The media type registration template is in Section 22.16.

There exist a potential interoperability issue for this format. It was named in RFC 2326 but never defined, even if used in examples that hint at the syntax. This format matches the purpose and its syntax supports the examples provided. However, it goes further by allowing UTF-8 in the value part, thus usage of UTF-8 strings may not be supported. However, as individual parameters are not defined, the using application anyway needs to have out-of-band agreement or using feature-tag to determine if the end-point supports the parameters.

The ABNF [RFC5234] grammar for "text/parameters" content is:

```plaintext
file             = *((parameter / parameter-value) CRLF)
parameter       = 1*visible-except-colon
parameter-value  = parameter "WSP ":" value
visible-except-colon = %x21-39 / %x3B-7E    ; VCHAR - ":"
value            = *(TEXT-UTF8char / WSP)
TEXT-UTF8char    = %x21-7E / UTF8-NONASCII
UTF8-NONASCII    = %xC0-DF 1UTF8-CONT
                  / %xE0-EF 2UTF8-CONT
                  / %xF0-F7 3UTF8-CONT
                  / %x80-BF 5UTF8-CONT
UTF8-CONT        = %x80-BF
WSP              = <See RFC 5234> ; Space or HTAB
VCHAR            = <See RFC 5234>
CRLF             = <See RFC 5234>
```

Appendix G. 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 16.51) is used to avoid the retransmission ambiguity problem [Stevens98].

- 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 primarily are intended for being used by the unreliable handling of RTSP messages and which will be maintained:

- [CSeq] See Section 16.19
- [Timestamp] See Section 16.51
Appendix H. 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 MUST 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.

H.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.

H.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 I. Open Issues

Open issues are filed and tracked in the bug and feature trackers at http://rtspspec.sourceforge.net. Open issues are discussed on MMUSIC list.
Appendix J. 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 section on minimal implementation was deleted without substitution.

- 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 unclarities 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.

- The Range formats has been changed in the following way:
  * The NPT format has been given an initial NPT identifier that must now be used.
  * All formats now support initial open ended formats of type "npt=-10".

- RTSP message handling has been changed in the following way:
  * RTSP messages now uses URIs rather than URLs.
  * It has been clarified that a 4xx message due to missing CSeq header shall be returned without a CSeq header.
  * The 300 (Multiple Choices) response code has been removed.
  * Rules for how to handle timing out RTSP messages has been added.
  * Extended Pipelining rules allowing for quick session startup.

- 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".

- The state machine section has completely been rewritten. It includes now more details and are also more clear about the model used.

- 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.

- 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.

  * 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 message body 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 [RFC3261] 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 a 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 MTag. The MTag 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.

* Added the Media-Range header for listing the current availability of the media range.

* Added the Notify-Reason header for giving the reason when sending PLAY_NOTIFY requests.

o The Protocol Syntax has been changed in the following way:

* All ABNF definitions are updated according to the rules defined in RFC 5234 [RFC5234] and has been gathered in a separate Section 20.
The ABNF 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 ABNF.

The CHAR rule has been changed to exclude NULL.

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.

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 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 an 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.
  - REDIRECT has been expanded and diversified for different situations.

- 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.
Added a new method PLAY_NOTIFY. This method is used by the RTSP server to asynchronously notify clients about session changes.
Appendix K. 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:


K.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 and proposed an earlier version of the PLAY_NOTIFY method.
- Sean Sheedy contributed text on the timeout behavior of RTSP messages and connections, the 463 status code, and proposed an earlier version of the PLAY_NOTIFY method.
- Greg Sherwood proposed an earlier version of the PLAY_NOTIFY method.
o Fredrik Lindholm contributed text about the RTSP security framework.

o John Lazzaro contributed the text for RTP over Independent TCP.

o Aravind Narasimhan contributed by rewriting Media Transport Alternatives (Appendix C) and editorial improvements on a number of places in the specification.
Appendix L. RFC Editor Consideration

Please replace RFC XXXX with the RFC number this specification 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
Seattle, WA
USA

Email: robla@robla.net

Magnus Westerlund
Ericsson AB
Faeroegatan 6
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@nw.neclab.eu