Internet Engineering Task Force (IETF)
Request for Comments: 7826
Obsoletes: 2326
Category: Standards Track
ISSN: 2070-1721

H. Schulzrinne
Columbia University
A. Rao
Cisco
R. Lanphier

M. Westerlund
Ericsson
M. Stiemerling, Ed.
University of Applied Sciences Darmstadt

December 2016

# Real-Time Streaming Protocol Version 2.0

## Abstract

This memorandum defines the Real-Time Streaming Protocol (RTSP) version 2.0, which obsoletes RTSP version 1.0 defined in RFC 2326.

RTSP is an application-layer protocol for the 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).

Copyright Notice



Table of Contents

## 1. Introduction

This memo defines version 2.0 of the Real-Time Streaming Protocol (RTSP 2.0). RTSP 2.0 is an application-layer protocol for the 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.

The protocol operates between RTSP 2.0 clients and servers, but it also supports the use 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. 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. The requested media can consist of multiple audio and video streams that are delivered as time-synchronized streams from servers to clients.

RTSP 2.0 is a replacement of RTSP 1.0 [RFC2326] and this document obsoletes that specification. This protocol is based on RTSP 1.0 but is not backwards compatible other than in the basic version negotiation mechanism. The changes between the two documents are listed in Appendix I. There are many reasons why RTSP 2.0 can't be backwards compatible with RTSP 1.0; some of the main ones are as follows:

- Most headers 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 Play state;
- the changed behavior of the extensibility model and its mechanism; and
- the change of syntax for some headers.

There are so many small updates that changing versions became necessary to enable clarification and consistent behavior. Anyone implementing RTSP for a new use case in which they have not installed RTSP 1.0 should only implement RTSP 2.0 to avoid having to deal with RTSP 1.0 inconsistencies.

This document is structured as follows. It begins with an overview of the protocol operations and its functions in an informal way. Then, a set of definitions of terms used and document conventions is introduced. These are followed by the actual RTSP 2.0 core protocol specification. The appendices describe and define some functionalities that are not part of the core RTSP specification, but which are still important to enable some usages. Among them, the RTP usage is defined in Appendix C, the Session Description Protocol (SDP) usage with RTSP is defined in Appendix D, and the "text/parameters" file format Appendix F, are three normative specification appendices. Other appendices include a number of informational parts discussing the changes, use cases, different considerations or motivations.

## 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 specific details. In case of conflict with this description and the later sections, the later sections take precedence. For more information about use cases considered for RTSP, see Appendix E.

RTSP 2.0 is a bidirectional 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 provider to the consumer. RTSP has three fundamental parts: Session Establishment, Media Delivery Control, and an extensibility model described below. The protocol is based on some assumptions about 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 on which to act. The resource is identified by a URI and the hostname part of the URI is used by RTSP client to resolve the IPv4 or IPv6 address of the RTSP server. Following the method line are a number of RTSP headers. These lines are ended by two consecutive carriage return line feed (CRLF) character pairs. The message body, if present, follows the two CRLF character pairs, and the body's length is described by a message header. RTSP responses are similar, but they start with a response line with the 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. Presentation Description

RTSP exists to provide access to multimedia presentations and content but tries to be agnostic about 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 presentation and the delivery protocol(s) is used. RTSP assumes that this description is either delivered completely out of band 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 presentation description are the following:

- The number of media streams;
- the resource identifier for each media stream/resource that is to be controlled by RTSP;
- the protocol that will be used to deliver each media stream;
- the transport protocol parameters that are not negotiated or vary with each client;
- the media-encoding information enabling a client to correctly decode the media upon reception; and
- an aggregate control resource identifier.

RTSP uses its own URI schemes ("rtsp" and "rtsps") to reference media resources and aggregates under common control (see Section 4.2).

This specification describes in Appendix D how one uses SDP [RFC4566] for describing the presentation.

## 2.2. Session Establishment

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

The client creates an RTSP session by sending a request using the SETUP method (Section 13.3) to the server. In the Transport header (Section 18.54) of the SETUP request, the client also includes all the transport parameters necessary to enable the media delivery protocol to function. This includes parameters that are preestablished by the presentation 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 presentation description.

When receiving a SETUP request, the server determines if the media resource is available and if one or more of the 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 18.49). The SETUP response includes a Transport header that specifies which of the alternatives has 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 multimedia content container. However, a server will likely refuse to aggregate resources from different content containers. Even if an RTSP session contains only a single media stream, the RTSP session can be referenced by the aggregate control URI.

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 a client-selected identifier in the Pipelined-Requests header (Section 18.33) 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 (Section 18.29) includes a number of properties that apply for the aggregate that is valuable when doing media delivery control and configuring user interface. The Accept-Ranges header (Section 18.5) informs the client about range formats that the server supports for these media resources. The Media-Range header (Section 18.30) informs 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 "begin playback", using the PLAY method (Section 13.4) and "suspend (pause) playback" 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 by using the Range header (Section 18.40) that supports several different time formats: Normal Play Time (NPT) (Section 4.4.2), Society of Motion Picture and Television Engineers (SMPTE) Timestamps (Section 4.4.1), and absolute time (Section 4.4.3). The Range header also allows the client to specify a position where delivery should end, thus allowing a specific interval to be delivered.

The support for positioning/searching within media 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 18.29). These are expressed using one or several independent attributes. A first attribute is Random-Access, which indicates whether positioning is possible, and with what granularity. Another aspect is whether the content will change during the lifetime of the session. While on-demand content will be provided in full from the beginning, a live stream being recorded results in the length of the accessible content growing as the session goes on. There also exists content that is dynamically built by a protocol other than RTSP and, thus, also changes in steps during the session, but maybe not continuously. Furthermore, when content is recorded, there are cases where the complete content is not maintained, but, for example, only the last hour. All of these properties result in the need for mechanisms that will be discussed below.

When the client accesses on-demand content that allows random access, 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 sending a PLAY request without a Range header. When the server is about to complete the PLAY request by delivering the end of the content or the requested range, the server will send a PLAY_NOTIFY request (Section 13.5) 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 possible 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. Because the recording is ongoing, the content grows in direct relation to the time passed. Therefore, each server’s response to a PLAY request will contain the current Media-Range header. The server should also regularly send (approximately every 5 minutes) the current media range in a PLAY_NOTIFY request (Section 13.5.2). 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 became on-demand content; the request also contains the final media range. While the live delivery continues, the client can request to play the current live point 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 that 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 closest point in the available media.

## 2.4. Session Parameter Manipulations

A session may have additional state or functionality that affects how the server or client treats the session or content, how it functions, or feedback on how well the session works. Such extensions are not defined in this specification, but they may be covered 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 use 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 (Section 18.45) and Range (Section 18.40).

RTSP 2.0 does not have a strong mechanism for negotiating the headers or parameters and their formats. 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 4.5).

## 2.5. Media Delivery

This document specifies how media is delivered with RTP [RFC3550] over UDP [RFC768], TCP [RFC793], or the RTSP connection. Additional protocols may be specified in the future as needed.

The usage of RTP as a media delivery protocol requires some additional information to function well. The PLAY response contains information to enable reliable and timely delivery 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 wants to notify the client about the completion of the media delivery, it sends a PLAY_NOTIFY request to the client. The PLAY_NOTIFY request includes information about the stream end, including the last RTP sequence number for each stream, thus enabling the client to empty the buffer smoothly.

### 2.5.1. Media Delivery Manipulations

The basic playback functionality of RTSP enables delivery of a range of requested content to the client at the pace intended by the content's creator. However, RTSP can also manipulate the delivery to the client in two ways.

Scale: The ratio of media-content time delivered per unit of 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 timescales and the effects are possible to combine.

Scale (Section 18.46) 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 being 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 content's original rate. Scale values between 0 and 1.0 provide for slow motion. Scale can be negative to allow for reverse playback in either regular pace (scale = -1.0), fast backwards (scale < -1.0), or slow-motion backwards (-1.0 < scale < 0). Scale = 0 would be 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. The nature of these media manipulations and when they are needed is highly media-type dependent. Let's consider two common media types, audio and video.

It is very difficult to modify the playback rate of audio. Typically, no more than a factor of two is possible while maintaining intelligibility by changing the pitch and 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, it is possible to manipulate the frame rate, although the rendering capabilities are often limited to certain frame rates. Also, the allowed bitrates in decoding, the structure used in the encoding, and the dependency between frames and other capabilities of the rendering device limits the possible manipulations. Therefore, the basic fast-forward capabilities often are implemented by selecting certain subsets of frames.

Due to the media restrictions, the possible scale values are commonly restricted to the set of realizable scale ratios. To enable the clients to select from the possible scale values, RTSP can signal 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 scale factor currently in use, exists.

Speed (Section 18.50) affects how much of the playback timeline is delivered in a given wallclock period. The default is Speed = 1 which means 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 have no meaning and are not allowed. This mechanism enables two general functionalities. One is client-side scale operations, i.e., the client receives all the frames and makes the adjustment to the playback locally. The second is delivery control for the 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, would result in a 50% increase in the data-transport rate. Whether or not that can be supported 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 increases 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 be 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 can't 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 a reduced media quality.

The need for bitrate adaptation becomes especially problematic in connection with the Speed semantics. If the goal is to fill up the buffer, the client may not want to do that at the cost of reduced quality. If the client wants to make local playout changes, then it may actually require that the requested speed be honored. To resolve this issue, Speed uses a range so that both cases 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 actual delivery rate in response to available bandwidth and reflect this in the Speed value in the response. 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 filling, 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:

- o Media-transport protocol keep-alive. RTP Control Protocol (RTCP) may be used when using RTP.

- o 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 sending a TEARDOWN request (Section 13.7) 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, a client is advised to release the session context when an extended period of time without media delivery activity has passed. The client can re-establish the session context if required later. What constitutes an extended period of time is dependent on the client, server, and their usage. It is recommended that the client terminate the session before ten 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 so 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 (Section 13.10). The Terminate-Reason header (Section 18.52) 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 it wants 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 that 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 should determine what parts are implemented. Allowing for extensions makes it possible for RTSP to address additional use cases. However, extensions will affect the interoperability of the protocol; therefore, it is important that they can be added in a structured way.

The client can learn the capability of a server by using the OPTIONS method (Section 13.1) and the Supported header (Section 18.51). It can also try and possibly fail using new methods or require that particular features be supported using the Require (Section 18.43) or Proxy-Require (Section 18.37) header.

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

- o 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 acknowledgment 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.

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

- o A new version of the protocol can be defined, allowing almost all aspects (except the position of the protocol version number) to change. A new version of the protocol must be registered through a Standards 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 extensions 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

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 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", "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 one 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 that 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 is the requester of 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 that may contain multiple media streams that often constitute 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 it can be streaming where the relationship is less strict.

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

IRI: An Internationalized Resource Identifier is similar to a URI but allows characters from the whole Universal Character Set (Unicode/ISO 10646), rather than the US-ASCII only. See [RFC3987] for more information.

Live: A live presentation or session originates media from an event taking place at the same time as the media delivery. Live sessions often have an unbound or only loosely defined duration and seek operations may not be possible.

Media initialization: The datatype- or codec-specific initialization. This includes such things as clock rates, color tables, etc. Any transport-independent information that is required by a client for playback of a media stream occurs in the media initialization phase of stream setup.

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

Media server: The server providing media-delivery 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 media 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 transport between RTSP agents. A message is either a request or a response.

Message body: The information transferred as the payload of a message (request or response). A message body consists of meta-information in the form of message body headers and content in the form of an arbitrary number of data octets, 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 SDP format.

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: 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 common context; it is created and maintained on a client's request and can be destroyed by either the client or 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 labeled 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 B) 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.

Origin server: The server on which a given resource resides.

Seeking: Requesting playback from a particular point in the content time line.

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

URI: A 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 a URL.

URL: A Universal Resource Locator is a URI that 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 that do not affect communication behavior or that only add to extensible field values.

The <minor> number is incremented when the changes made to the protocol add features that do not change the general message parsing algorithm but that 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 SHALL NOT be sent and MUST be ignored by recipients.

## 4.2. RTSP IRI and URI

RTSP 2.0 defines and registers or updates 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 the URI scheme that was defined in RTSP 1.0. The "rtspu" scheme indicates unspecified transport of the RTSP messages over unreliable transport means (UDP in RTSP 1.0). An RTSP server MUST respond with an error code indicating the "rtspu" scheme is not implemented (501) to a request that carries a "rtspu" URI scheme.

The details of the syntax of "rtsp" and "rtsps" URIs have been changed from RTSP 1.0. These changes include the addition of:

- o Support for an IPv6 literal in the host part and future IP literals through a mechanism defined in [RFC3986].
- o A new relative format to use in the RTSP elements that is not required to start with "/".

Neither should have any significant impact on interoperability. If IPv6 literals are needed in the RTSP URI, then that RTSP server must be IPv6 capable, and RTSP 1.0 is not a fully IPv6 capable protocol. If an RTSP 1.0 client attempts to process the URI, the URI will not match the allowed syntax, it will be considered invalid, and processing will be stopped. This is clearly a failure to reach the resource; however, it is not a signification issue as RTSP 2.0 support was needed anyway in both server and client. Thus, failure will only occur in a later step when there is an RTSP version mismatch between client and server. The second change will only occur inside RTSP message headers, as the Request-URI must be an absolute URI. Thus, such usages will only occur after an agent has accepted and started processing RTSP 2.0 messages, and an agent using RTSP 1.0 only will not be required to parse such types of relative URIs.

This specification also defines the format of RTSP IRIs [RFC3987] that can be used as RTSP resource identifiers and locators on 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 to URIs, as defined in [RFC3987]. This allows a URI that matches the RTSP 2.0 specification, and so is suitable for use in a request, to be created from an RTSP IRI.

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

o An absolute URI requires the authority part; i.e., a host identity MUST be provided.

o Parameters in the path element are prefixed with the reserved separator ";".

The "scheme" and "host" parts of all URIs [RFC3986] and IRIs [RFC3987] are case insensitive. All other parts of RTSP URIs and IRIs are case sensitive, and they MUST NOT be case mapped.

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 a DESCRIBE request.

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 URIs with queries are difficult to handle due to the relative URI handling rules of RFC 3986. 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 be 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, the port number 554 MUST be used. For the scheme "rtsps", if no port number is provided in the authority part of the URI port number, the TCP port 322 MUST be used.

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, such as 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 a length between 8-128 characters. A session identifier MUST be generated using methods that make it cryptographically random (see [RFC4086]). It is RECOMMENDED that a session identifier contain 128 bits of entropy, i.e., approximately 22 characters from a high-quality generator (see Section 21). However, note that the session identifier does not provide any security against session hijacking unless it is kept confidential by the client, server, and trusted proxies.

### 4.4. Media-Time Formats

RTSP currently supports three different media-time formats defined below. Additional time formats may be specified in the future. These time formats can be used with the Range header (Section 18.40) to request playback and specify at which media position protocol requests actually will or have taken place. They are also used in description of the media's properties using the Media-Range header (Section 18.30). The unqualified format identifier is used on its own in Accept-Ranges header (Section 18.5) to declare supported time formats and also in the Range header (Section 18.40) to request the time format used in the response.

#### 4.4.1. SMPTE-Relative Timestamps

A timestamp may use a format derived from a Society of Motion Picture and Television Engineers (SMPTE) specification and expresses time offsets anchored at the start of the media clip. Relative timestamps are expressed as SMPTE time codes [SMPTE-TC] 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 a frame rate of 29.97 frames per second. Other SMPTE codes MAY be supported (such as "SMPTE 25") through the 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 hundredths 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
```

### 4.4.2. Normal Play Time

Normal Play Time (NPT) indicates the stream-absolute position relative to the beginning of the presentation. The timestamp consists of two parts: The mandatory first part may be expressed in either seconds only or in hours, minutes, and seconds. The optional second part consists of a decimal point and decimal figures and indicates 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 Digital Storage Media Command and Control (DSMb;CC) [ISO.13818-6.1995]:

> Intuitively, NPT is the clock the viewer associates with a program. It is often digitally displayed on a DVD player. 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 (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 is based on ISO 8601 [ISO.8601.2000] and expresses the time elapsed since presentation start, with two different notations allowed:

o The npt-hhmmss notation uses an ISO 8601 extended complete representation of the time of the day format (Section 5.3.1.1 of [ISO.8601.2000] ) using colons (":") as separators between hours, minutes, and seconds (hh:mm:ss). The hour counter is not limited to 0-24 hours; up to nineteen (19) hour digits are allowed.

  * In accordance with the requirements of the ISO 8601 time format, the hours, minutes, and seconds MUST all be present, with two digits used for minutes and for seconds and with at least two digits for hours. An NPT of 7 minutes and 0 seconds is represented as "00:07:00", and an NPT of 392 hours, 0 minutes, and 6 seconds is represented as "392:00:06".

  * RTSP 1.0 allowed NPT in the npt-hhmmss notation without any leading zeros to ensure that implementations don't fail; for backward compatibility, all RTSP 2.0 implementations are REQUIRED to support receiving NPT values, hours, minutes, or seconds, without leading zeros.

o The npt-sec notation expresses the time in seconds, using between one and nineteen (19) digits.

Both notations allow decimal fractions of seconds as specified in Section 5.3.1.3 of [ISO.8601.2000], using at most nine digits, and allowing only "." (full stop) as the decimal separator.

The npt-sec notation is optimized for automatic generation; the npt-hhmmss notation is optimized 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.

### 4.4.3. Absolute Time

Absolute time is expressed using a timestamp based on ISO 8601 [ISO.8601.2000]. The date is a complete representation of the calendar date in basic format (YYYYMMDD) without separators (per Section 5.2.1.1 of [ISO.8601.2000]). The time of day is provided in the complete representation basic format (hhmmss) as specified in Section 5.3.1.1 of [ISO.8601.2000], allowing decimal fractions of seconds following Section 5.3.1.3 requiring "." (full stop) as decimal separator and limiting the number of digits to no more than nine. The time expressed MUST use UTC (GMT), i.e., no time zone offsets are allowed. The full date and time specification is the eight-digit date followed by a "T" followed by the six-digit time value, optionally followed by a full stop followed by one to nine fractions of a second and ended by "Z", e.g., YYYYMMDDThhmmss.ssZ.

The reasons for this time format rather than using "Date and Time on the Internet: Timestamps" [RFC3339] are historic. We continue to use the format specified in RTSP 1.0. The motivations raised in RFC 3339 apply to why a selection from ISO 8601 was made; however, a different and even more restrictive selection was applied in this case.

Below are three examples of media time formats, first, a request for a clock format range request for a starting time of November 8, 1996 at 14 h 37 min and 20 1/4 seconds UTC playing for 10 min and 5 seconds, followed by a Media-Properties header's "Time-Limited" UTC property for the 24th of December 2014 at 15 hours and 00 minutes, and finally a Terminate-Reason header "time" property for the 18th of June 2013 at 16 hours, 12 minutes, and 56 seconds:

```
clock=19961108T143720.25Z-19961108T144725.25Z
Time-Limited=20141224T1500Z
time=20130618T161256Z
```

## 4.5. Feature Tags

Feature tags are unique identifiers used to designate features in RTSP. These tags are used in Require (Section 18.43), Proxy-Require (Section 18.37), Proxy-Supported (Section 18.38), Supported (Section 18.51), and Unsupported (Section 18.55) header fields.

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

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 Section 22, "IANA Considerations".)

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

## 4.6. 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 18.31) or in SDP (see Appendix D.1.9). MTag is similar to ETag in HTTP/1.1 (see Section 3.11 of [RFC2068]).

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 bodies 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 18.24 and Section 18.26 for information on the If-Match and If-None-Match headers, respectively. Note that RTSP message body tags apply to the complete presentation, i.e., both the presentation 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.

## 4.7. Media Properties

When an RTSP server handles media, it is important to consider the different properties a media instance for delivery and playback can have. This specification considers the media properties listed below 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 lifetime of the RTSP session and is 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, such as encoding, or quality, may change. Generally, one can seek, i.e., request any range, within the media.

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 content defined by a playlist.

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 and allow for random access delivery within the part of the already-recorded content. The actual behavior of the media stream is very much dependent 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 typically be a sliding window that moves forward while new data is made available and older data is discarded.

To cover the above usages, the following media properties with appropriate values are specified.

### 4.7.1. Random Access and Seeking

Random access is the ability to specify and get media delivered starting from any time (instant) within the content, an operation called "seeking". The Media-Properties header will indicate the general capability for a media resource to perform random access.

Random-Access: The media is 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.

Beginning-Only: Seeking is only possible to the beginning of the content.

No-Seeking: Seeking is not possible at all.

If random access is possible, as indicated by the Media-Properties header, the actual behavior policy when seeking can be controlled using the Seek-Style header (Section 18.47).

### 4.7.2. Retention

The following retention policies are used by media to limit possible protocol operations:

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

Time-Limited: The media will not be removed before the given wallclock time. After that time, it may or may not be available anymore.

Time-Duration: The media (on fragment or unit basis) will be retained for the specified duration.

### 4.7.3. Content Modifications

The media content and its timeline can be of different types, e.g. pre-produced content on demand, a live source that is being generated as time progresses, or something that is dynamically altered or recomposed during playback. Therefore, a media property for content modifications is needed and the following initial values are defined:

Immutable: The content of the media will not change, even if the representation, such as encoding or quality changes.

Dynamic: The content can change due to external methods or triggers, such as playlists, but this will be announced by explicit updates.

Time-Progressing: As time progresses, new content will become available. If the content is also retained, it will become longer as everything between the start point and the point currently being made available can be accessed. If the media server uses a sliding-window policy for retention, the start point will also change as time progresses.

### 4.7.4. Supported Scale Factors

A particular media content item often supports only a limited set or range of scales 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 that contains a list with the values or ranges that are supported. The attribute is named "Scales". The "Scales" attribute may be updated at any point in the content due to content consisting of spliced pieces or content being dynamically updated by out-of-band mechanisms.

### 4.7.5. Mapping to the Attributes

This section shows examples of how one would map the above usages to the properties and their values.

Example of On-Demand:
Random Access: Random-Access=5.0, Content Modifications: Immutable, Retention: Unlimited or Time-Limited.

Example of Dynamic On-Demand:
Random Access: Random-Access=3.0, Content Modifications: Dynamic, Retention: Unlimited or Time-Limited.

Example of Live:
Random Access: No-Seeking, Content Modifications: Time-Progressing, Retention: Time-Duration=0.0

Example of Live with Recording:
Random Access: Random-Access=3.0, Content Modifications: Time-Progressing, Retention: Time-Duration=7200.0

## 5. RTSP Message

RTSP is a text-based protocol that uses the ISO 10646 character set in UTF-8 encoding per RFC 3629 [RFC3629]. Lines MUST be terminated by a 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 used carefully, also allow easy implementation of research prototypes in scripting languages such as Python, PHP, Perl and TCL.

The ISO 10646 character set avoids character-set switching, but is invisible to the application as long as US-ASCII is being used. This is also the encoding used for text fields in RTCP [RFC3550].

A request contains a method, 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 are either requests from client to server or from server to client, and responses in the reverse direction. Request (Section 7) and response (Section 8) messages use a format based on the generic message format of RFC 5322 [RFC5322] for transferring bodies (the payload of the message). Both types of messages 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 headers, and possibly the data of the message body. The ABNF [RFC5234] below is for illustration only; the formal message specification is presented in Section 20.2.2.

```
generic-message = start-line
                *(rtsp-header CRLF)
                  CRLF
                [ message-body-data ]
start-line = Request-Line / Status-Line
```

In the interest of robustness, agents MUST ignore any empty line(s) received where a Request-Line or Status-Line is expected. In other words, if the agent is reading the protocol stream at the beginning of a message and receives any number of CRLFs first, it MUST ignore all of the CRLFs.

## 5.2. Message Headers

RTSP header fields (see Section 18) include general-header, request-header, response-header, and message body 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 a request-header or response-header field, and ending with the message body header fields.

Multiple 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. 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; thus, a proxy MUST NOT change the order of these field-values when a message is forwarded.

Unknown message headers MUST be ignored (skipping over the header to the next protocol element, and not causing an error) by an RTSP server or client. An RTSP proxy MUST forward unknown message headers. Message headers defined outside of this specification that are required to be interpreted by the RTSP agent will need to use feature tags (Section 4.5) and include them in the appropriate Require (Section 18.43) or Proxy-Require (Section 18.37) 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 of a message body is an SDP message.

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 18.17) and Content-Type header (see Section 18.19). 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. In case a message body is received in a message when not expected, the message body data SHOULD be discarded. This is to allow future extensions to define optional use of a message body.

## 5.4. Message Length

An RTSP message that does not contain any message body is terminated by the first empty line after the header fields (note: an empty line is a line with nothing preceding the CRLF.). In RTSP messages that contain message bodies, the empty line is followed by the message body. The length of that body is determined by the value of the Content-Length header (Section 18.17). The value in the header represents the length of the message body in octets. If this header field is not present, a value of zero is assumed, i.e., no message body present in the message. 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 Section 4.1 of [RFC7230]).

> 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

General headers are headers that may be used in both requests and responses. The general-headers are listed in Table 1:

| Header Name | Defined in |
| --- | --- |
| Accept-Ranges | Section 18.5 |
| Cache-Control | Section 18.11 |
| Connection | Section 18.12 |
| CSeq | Section 18.20 |
| Date | Section 18.21 |
| Media-Properties | Section 18.29 |
| Media-Range | Section 18.30 |
| Pipelined-Requests | Section 18.33 |
| Proxy-Supported | Section 18.38 |
| Range | Section 18.40 |
| RTP-Info | Section 18.45 |
| Scale | Section 18.46 |
| Seek-Style | Section 18.47 |
| Server | Section 18.48 |
| Session | Section 18.49 |
| Speed | Section 18.50 |
| Supported | Section 18.51 |
| Timestamp | Section 18.53 |
| Transport | Section 18.54 |
| User-Agent | Section 18.56 |
| Via | Section 18.57 |

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, whether 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, which can be of the following types: general-headers (Section 6), request-headers (Section 7.2), or message body headers (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 octets 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.

| Method | Defined in |
|---|---|
| DESCRIBE | Section 13.2 |
| GET_PARAMETER | Section 13.8 |
| OPTIONS | Section 13.1 |
| PAUSE | Section 13.6 |
| PLAY | Section 13.4 |
| PLAY_NOTIFY | Section 13.5 |
| REDIRECT | Section 13.10 |
| SETUP | Section 13.3 |
| SET_PARAMETER | Section 13.9 |
| TEARDOWN | Section 13.7 |

Table 2: The RTSP Methods

The syntax of the RTSP request line has the following:

```
<Method> SP <Request-URI> SP <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 [RFC7230], RTSP requests identify the resource through an absolute RTSP URI (including 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.

| Header | Defined in |
|---|---|
| Accept | Section 18.1 |
| Accept-Credentials | Section 18.2 |
| Accept-Encoding | Section 18.3 |
| Accept-Language | Section 18.4 |
| Authorization | Section 18.8 |
| Bandwidth | Section 18.9 |
| Blocksize | Section 18.10 |
| From | Section 18.23 |
| If-Match | Section 18.24 |
| If-Modified-Since | Section 18.25 |
| If-None-Match | Section 18.26 |
| Notify-Reason | Section 18.32 |
| Proxy-Authorization | Section 18.36 |
| Proxy-Require | Section 18.37 |
| Referrer | Section 18.41 |
| Request-Status | Section 18.42 |
| Require | Section 18.43 |
| Terminate-Reason | Section 18.52 |

Table 3: The RTSP Request-Headers

Detailed header definitions are provided in Section 18.

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. Normally, there is only one, final, response. Responses using the response code class 1xx is the only class for which there MAY be sent one or more responses 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 17. 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 five 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 (3rr rather than 3xx is used as 304 is excluded; see Section 17.3)

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 adopted most HTTP/1.1 [RFC2068] status codes and then added RTSP-specific status codes starting at x50 to avoid conflicts with future HTTP status codes that are desirable to import into RTSP. All these codes are RTSP specific and RTSP has its own registry separate from HTTP for 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 an exception for unknown 3xx codes, which MUST be treated as a 302 (Found). The reason for that exception is that the status code 300 (Multiple Choices in HTTP) is not defined for RTSP. A response with an unrecognized status code 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 that will explain the unusual status.

| Code | Reason | Method |
|---|---|---|
| 100 | Continue | all |
| 200 | OK | all |
| 301 | Moved Permanently | all |
| 302 | Found | all |
| 303 | See Other | n/a |
| 304 | Not Modified | all |

| | | |
|---|---|---|
| 305 | Use Proxy | all |
| 400 | Bad Request | all |
| 401 | Unauthorized | all |
| 402 | Payment Required | all |
| 403 | Forbidden | all |
| 404 | Not Found | all |
| 405 | Method Not Allowed | all |
| 406 | Not Acceptable | all |
| 407 | Proxy Authentication Required | all |
| 408 | Request Timeout | all |
| 410 | Gone | all |
| 412 | Precondition Failed | DESCRIBE, SETUP |
| 413 | Request Message Body Too Large | all |
| 414 | Request-URI Too Long | all |
| 415 | Unsupported Media Type | all |
| 451 | Parameter Not Understood | SET_PARAMETER, GET_PARAMETER |
| 452 | reserved | n/a |
| 453 | Not Enough Bandwidth | SETUP |
| 454 | Session Not Found | all |
| 455 | Method Not Valid in This State | all |
| 456 | Header Field Not Valid for Resource | all |
| 457 | Invalid Range | PLAY, PAUSE |
| 458 | Parameter Is Read-Only | SET_PARAMETER |

| | | |
|---|---|---|
| 459 | Aggregate Operation Not Allowed | all |
| 460 | Only Aggregate Operation Allowed | all |
| 461 | Unsupported Transport | all |
| 462 | Destination Unreachable | all |
| 463 | Destination Prohibited | SETUP |
| 464 | Data Transport Not Ready Yet | PLAY |
| 465 | Notification Reason Unknown | PLAY_NOTIFY |
| 466 | Key Management Error | all |
| 470 | Connection Authorization Required | all |
| 471 | Connection Credentials Not Accepted | all |
| 472 | Failure to Establish Secure Connection | all |
| 500 | Internal Server Error | all |
| 501 | Not Implemented | all |
| 502 | Bad Gateway | all |
| 503 | Service Unavailable | all |
| 504 | Gateway Timeout | all |
| 505 | RTSP Version Not Supported | all |
| 551 | Option Not Supported | all |
| 553 | Proxy Unavailable | all |

Table 4: Status Codes and Their Usage with RTSP Methods

## 8.2. Response Headers

The response-headers allow the request recipient to pass additional information about the response that cannot be placed in the Status-Line. This header gives 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.

| Header | Defined in |
|---|---|
| Authentication-Info | Section 18.7 |
| Connection-Credentials | Section 18.13 |
| Location | Section 18.28 |
| MTag | Section 18.31 |
| Proxy-Authenticate | Section 18.34 |
| Public | Section 18.39 |
| Retry-After | Section 18.44 |
| Unsupported | Section 18.55 |
| WWW-Authenticate | Section 18.58 |

Table 5: The RTSP Response Headers

Response-header 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. A new or experimental header can be given the semantics of response-header if all parties in the communication recognize them to be a response-header. Unrecognized headers in responses MUST be ignored.

# 9. Message Body

Some request and response messages include a message body, if not otherwise restricted by the request method or response status code. The message body consists of the content data itself (see also Section 5.3).

The SET_PARAMETER and GET_PARAMETER requests and responses, and the DESCRIBE response as defined by this specification, can have a message body; the purpose of the message body is defined in each case. All 4xx and 5xx responses MAY also have a message body to carry additional response information. Generally, a message body MAY be attached to any RTSP 2.0 request or response, but the content of the message body MAY be ignored by the receiver. Extensions to this specification can specify the purpose and content of message bodies, including requiring their inclusion.

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 body header fields define meta-information about the content data in the message body. The message body header fields are listed in Table 6.

| Header | Defined in |
|---|---|
| Allow | Section 18.6 |
| Content-Base | Section 18.14 |
| Content-Encoding | Section 18.15 |
| Content-Language | Section 18.16 |
| Content-Length | Section 18.17 |
| Content-Location | Section 18.18 |
| Content-Type | Section 18.19 |
| Expires | Section 18.22 |
| Last-Modified | Section 18.27 |

Table 6: The RTSP Message Body Headers

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

## 9.2. Message Body

An RTSP message with a message body MUST include the Content-Type and Content-Length headers. When a message body is included with a message, the data type of that content data is determined via the Content-Type and Content-Encoding header fields.

Content-Type specifies the media type of the underlying data. There is no default media format and the actual format used in the body is required to be explicitly stated in the Content-Type header. By being explicit and always requiring the inclusion of the Content-Type header with accurate information, one avoids the many pitfalls in a heuristic-based interpretation of the body content. The user experience of HTTP and email have suffered from relying on such heuristics.

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. The default encoding is 'identity', i.e. no transformation of the message body.

The Content-Length of a message is the length of the content, measured in octets.

## 9.3. Message Body Format Negotiation

The content format of the message body is provided using the Content-Type header (Section 18.19). To enable the responder of a request to determine which media type it should use, the requester may include the Accept header (Section 18.1) in a request to identify supported media types or media type ranges suitable to the response. In case the responder is not supporting any of the specified formats, then the request response will be a 406 (Not Acceptable) error code.

The media types that may be used on requests with message bodies need to be determined through the use of feature tags, specification requirement, or trial and error. Trial and error works because when the responder does not support the media type of the message body, it will respond with a 415 (Unsupported Media Type).

The formats supported and their negotiation is done individually on a per method and direction (request or response body) direction. Requirements on supporting particular media types for use as message bodies in requests and response SHALL also be specified on a per-method and per-direction basis.

# 10. Connections

RTSP messages are transferred between RTSP agents and proxies using a transport connection. This transport connection uses TCP or TCP/TLS. This transport connection is referred to as the "connection" or "RTSP connection" within this document.

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

- o persistent - a transport connection is used for several request/response transactions;

- o transient - a transport connection is used for each 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 18.20), which may appear to be relevant only to connectionless transport scenarios, are still retained and MUST be implemented according to this 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

Since RTSP messages are transmitted using reliable transport protocols, they MUST NOT be retransmitted at the RTSP level. Instead, the implementation must rely on the underlying transport to provide reliability. The RTSP implementation may use any indication of reception acknowledgment 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 acknowledgment 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) allows the client to specify the port it will contact the server on, and defines the default port to use if one is not explicitly given.

In addition to the registered default ports, i.e., 554 (rtsp) and 322 (rtsps), there is an alternative port 8554 registered. This port may provide some benefits over non-registered ports if an RTSP server is unable to use the default ports. The benefits may include preconfigured security policies as well as classifiers in network monitoring tools.

An RTSP client opening a TCP connection to access a particular resource as identified by a URI uses the IP address and port derived from the host and port parts of the URI. The IP address is either the explicit address provided in the URI or any of the addresses provided when performing A and AAAA record DNS lookups of the hostname in the URI.

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 supports 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 to 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 be either transient or persistent.

An RTSP agent MAY use one connection to handle multiple RTSP sessions on the same server. The RTSP agent SHALL NOT use more than one connection per RTSP session at any given point.

Having only one connection in use at any time avoids confusion regarding on which connection any server-to-client requests shall be sent. Using a single connection for multiple RTSP sessions also saves complexity by enabling the server to maintain less state about its connection resources on the server. Not using more than one connection at a time for a particular RTSP session avoids wasting connection resources and allows the server to track only the most recently used client-to-server connection for each RTSP session as being the currently valid server-to-client connection.

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 signaling channel, the server may be forced to silently discard RTSP messages, and it 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 attempts to send a request to a client that has no connection currently to the server SHALL discard the request.

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

Queuing of server-to-client requests has been considered. However, a security issue exists as to how it might be possible to authorize a client establishing a new connection as being a legitimate receiver of a request related to a particular RTSP session, without the client first issuing requests related to the pending request. Thus, it would be likely to make any such requests even more delayed and less useful.

The sending of client and server 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 queued 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. A potential issue is that outstanding requests may time out despite being processed by the peer; this can be due to the response being caught in the queue behind a number of requests that the RTSP agent is processing but that take some time to complete. To avoid this problem, an RTSP agent should 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, not only can 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 agent's 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 onto 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 extensions, an agent should be careful about 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. It is RECOMMENDED that:

o an RTSP agent should not have more than 10 outstanding requests per RTSP session;

o an RTSP agent should not have more than 10 outstanding requests that are not 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).

RTSP agents can send requests to multiple different destinations, either server or client contexts over the same connection to a proxy. Then, the proxy forks the message to the different destinations over proxy-to-agent connections. In these cases when multiple requests are outstanding, the requesting agent MUST be ready to receive the responses out of order compared to the order they where sent on the connection. The order between multiple messages for each destination will be maintained; however, the order between response from different destinations can be different.

> The reason for this is to avoid a head-of-line blocking situation. In a sequence of requests, an early outstanding request may take time to be processed at one destination. Simultaneously, a response from any other destination that was later in the sequence of requests may have arrived at the proxy; thus, allowing out-of-order responses avoids forcing the proxy to buffer this response and instead deliver it as soon as possible. Note, this will not affect the order in which the messages sent to each separate destination were processed at the request destination.

This scenario can occur in two cases involving proxies. The first is a client issuing requests for sessions on different servers using a common client-to-proxy connection. The second is for server-to-client requests, like REDIRECT being sent by the server over a common transport connection the proxy created for its different connecting clients.

### 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 18.49). 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, the server should wait for a reasonable amount of time for the client to receive and act upon the TEARDOWN response and then 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. Ten seconds should give the client ample opportunity to get its message to the server.

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

Certain error responses such as 460 (Only Aggregate Operation Allowed) (Section 17.4.24) 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 or special types of requests or result in extra overhead for the client when testing for new features. On the other hand, keeping connections open after sending an error response poses a Denial-of-Service (DoS) security risk (Section 21).

The server MAY close a connection if it receives an incomplete message and if the message is not completed within a reasonable amount of time. It is RECOMMENDED that the server wait at least 10 seconds for the completion of a message or for the next part of the message to arrive (which is an indication that the transport and the client are still alive). Servers believing they are under attack or that are otherwise starved for resources during that event MAY consider using a shorter timeout.

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 (responders) 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 the 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 responder 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 elapse without receiving any response, the requester MAY assume that the responder is unresponsive and abort the connection by closing the TCP connection.

In some cases, multiple RTSP sessions share the same transport connection; abandoning a request and closing the connection may have significant impact on those other sessions. First of all, other RTSP requests may have become queued up due to the request taking a long time to process. Secondly, those sessions also lose the possibility to receive server-to-client requests. To mitigate that situation, the RTSP client or server SHOULD establish a new connection and send any requests that are queued up or that haven't received a response on this new connection. Thirdly, to ensure that the RTSP server knows which connection is valid for a particular RTSP session, the RTSP agent SHOULD send a keep-alive request, if no other request will be sent immediately for that RTSP session, for each RTSP session on the old connection. The keep-alive request will normally be a SET_PARAMETER with a session header to inform the server that this agent cares about this RTSP session.

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 Round-Trip Time (RTT) of the request/response cycle using the Timestamp header (Section 18.53) in any RTSP request.

The 10-second wait 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 types of networks. The 5-second value for 1xx messages is half the timeout giving a reasonable chance of successful delivery before timeout happens on the requester side.

## 10.5. Showing Liveness

RTSP requires the client to periodically show its liveness to the server or the server may terminate any session state. Several different protocol mechanism include in their usage a liveness proof from the client. These mechanisms are RTSP requests with a Session header to the server; if RTP & RTCP is used for media data transport and the transport is established, the RTCP message proves liveness; or through any other used media-transport protocol capable of indicating liveness of the RTSP client. It is RECOMMENDED that a client not wait to the last second of the timeout before trying to send a liveness message. The RTSP message may take some time to arrive safely at the receiver, due to packet loss and TCP retransmissions. To show liveness between RTSP requests being 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 will necessarily also function as a keep-alive. The server can determine the client by network address and port together with the fact that the client is reporting on the server's RTP sender sources (synchronization source (SSRCs)). A downside of using RTCP is that it only gives statistical guarantees of reaching the server. However, the probability of a false client timeout is so low that it can be ignored in most cases. For example, assume a session with a 60-second timeout and enough bitrate assigned to RTCP messages to send a message from client to server on average every 5 seconds. That client has, for a network with 5% packet loss, a probability of failing to confirm liveness within the timeout interval for that session of 2.4*E-16. Sessions with shorter timeouts, much higher packet loss, or small RTCP bandwidths SHOULD also implement one or more of the mechanisms below.

SET_PARAMETER: When using SET_PARAMETER for keep-alives, a body SHOULD NOT be included. This method is the RECOMMENDED RTSP method to use for a request intended only to perform keep-alives. RTSP servers MUST support the SET_PARAMETER method, so that clients can always use this mechanism.

GET_PARAMETER: When using GET_PARAMETER for keep-alives, a body SHOULD NOT be included, dependent on implementation support in the server. Use the OPTIONS method to determine if there is method support or simply try.

OPTIONS: This method is also usable, but it causes the server to perform more unnecessary processing and results 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 of the Session header (Section 18.49) 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 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 an established session.

## 10.6. Use of IPv6

Explicit IPv6 [RFC2460] support was not present in RTSP 1.0. RTSP 2.0 has been updated for explicit IPv6 support. Implementations of RTSP 2.0 MUST understand literal IPv6 addresses in URIs and RTSP headers. Although the general URI format envisages potential future new versions of the literal IP address, usage of any such new version would require other modifications to the RTSP specification (e.g., address fields in the Transport header (Section 18.54)).

## 10.7. Overload Control

Overload in RTSP can occur when servers and proxies have insufficient resources to complete the processing of a request. An improper handling of such an overload situation at proxies and servers can impact the operation of the RTSP deployment, and probably worsen the situation. RTSP defines the 503 (Service Unavailable) response (Section 17.5.4) to let servers and proxies notify requesting proxies and RTSP clients about an overload situation. In conjunction with the Retry-After header (Section 18.44), the server or proxy can indicate the time after which the requesting entity can send another request to the proxy or server.

There are two scopes of such 503 answers. The first scope is for an established RTSP session, where the request resulting in the 503 response as well as the response itself carries a Session header identifying the session that is suffering overload. This response only applies to this particular session. The other scope is the general RTSP server as identified by the host in the Request-URI. Such a 503 answer with any Retry-After header applies to all requests that are not session specific to that server, including a SETUP request intended to create a new RTSP session.

Another scope for overload situations exists: the RTSP proxy. To enable an RTSP proxy to signal that it is overloaded, or otherwise unavailable and unable to handle the request, a 553 response code has been defined with the meaning "Proxy Unavailable". As with servers, there is a separation in response scopes between requests associated with existing RTSP sessions and requests to create new sessions or general proxy requests.

Simply implementing and using the 503 (Service Unavailable) and 553 (Proxy Unavailable) response codes is not sufficient for properly handling overload situations. For instance, a simplistic approach would be to send the 503 response with a Retry-After header set to a fixed value. However, this can cause a situation in which multiple RTSP clients again send requests to a proxy or server at roughly the same time, which may again cause an overload situation. Another situation would be if the "old" overload situation is not yet resolved, i.e., the length indicated in the Retry-After header was too short for the overload situation to subside.

An RTSP server or proxy in an overload situation must select the value of the Retry-After header carefully, bearing in mind its current load situation. It is REQUIRED to increase the timeout period in proportion to the current load on the server, i.e., an increasing workload should result in an increased length of the indicated unavailability. It is REQUIRED not to send the same value in the Retry-After header to all requesting proxies and clients, but to add a variation to the mean value of the Retry-After header.

A more complex case may arise when a load-balancing RTSP proxy is in use. This is the case when an RTSP proxy is used to select amongst a set of RTSP servers to handle the requests or when multiple server addresses are available for a given server name. The proxy or client may receive a 503 (Service Unavailable) or 553 (Proxy Unavailable) response code from one of its RTSP servers or proxies, or a TCP timeout (if the server is even unable to handle the request message). The proxy or client simply retries the other addresses or configured proxies, but it may also receive a 503 (Service Unavailable) or 553 (Proxy Unavailable) response or TCP timeouts from those addresses. In such a situation, where none of the RTSP servers/proxies/addresses can handle the request, the RTSP agent has to wait before it can send any new requests to the RTSP server. Any additional request to a specific address MUST be delayed according to the Retry-After headers received. For addresses where no response was received or TCP timeout occurred, an initial wait timer SHOULD be set to 5 seconds. That timer MUST be doubled for each additional failure to connect or receive response until the value exceeds 30 minutes when the timer's mean value may be set to 30 minutes. It is REQUIRED not to set the same value in the timer for each scheduling, but instead to add a variation to the mean value, resulting in picking a random value within the range of 0.5 to 1.5 times the mean value.

## 11. Capability Handling

This section describes the available capability-handling mechanism that allows RTSP to be extended. Extensions to this version of the protocol are basically done in two ways. Firstly, 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 an OPTIONS request to the other party. Depending on the URI, it will either apply in regard to a certain media resource, the whole server in general, or simply the next hop. The OPTIONS response MUST contain a Public header that declares all methods supported for the indicated resource.

It is not necessary to use OPTIONS to discover support of a method, as 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. For example, a feature tag can represent the functionality a single RTSP header provides. Another feature tag can represent much more functionality, such as the "play.basic" feature tag (Section 11.1), which represents the minimal media delivery for 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, in general, and is not dependent on a specific resource. The intended usage is to determine before one needs to use a functionality that it is supported. It can be used in any method, but OPTIONS is the most suitable as it simultaneously 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 the receiver learning the requester's feature support. The receiver then includes its set of features in the response.

Proxy-Supported: This header is used in a similar fashion as 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 the common functionality supported in general by all members of the proxy chain between the client and server; it does not depend on the resource. Proxies are required to add this header whenever the Supported header is present, but proxies may also add it independently of the requester.

Require: This header can be included in any request where the endpoint, 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 endpoint 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 is 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 behavior as Require except that it only applies to proxies and not the endpoint. Features that need to be supported by both proxies and endpoints need to be included in both the Require and Proxy-Require header.

Unsupported: This header is used in a 551 (Option Not Supported) error response, to indicate which features were not supported. Such a response is only the result of the usage of the Require or Proxy-Require headers where one or more features were not supported. This information allows the requester to make the best of situations as it knows which features are not supported.

## 11.1. Feature Tag: play.basic

An implementation supporting all normative parts of this specification for the setup and control of playback of media uses the feature tag "play.basic" to indicate this support. The appendices (starting with letters) are not part of the functionality included in the feature tag unless the appendix is explicitly specified in a main section as being a required appendix.

> Note: This feature tag does not mandate any media delivery protocol, such as RTP.
>
> In RTSP 1.0, there was a minimal implementation section. However, that was not consistent with the rest of the specification. So, rather than making an attempt to explicitly enumerate the features for play.basic, this specification has to be taken as a whole and the necessary features normatively defined as being required are included.

# 12. Pipelining Support

Pipelining is a general method to improve performance of request/response protocols by allowing the requesting agent to have more than one request outstanding and to 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 agent 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, between two agents, as the sending order. The processing of the request MUST also have been finished before processing the next request from the same agent. The responses MUST be sent in the order the requests were processed.

RTSP 2.0 has extended support for pipelining beyond the capabilities in RTSP 1.0. As a major improvement, all requests involved in setting up and initiating media delivery can now be pipelined, indicated by the Pipelined-Request header (see Section 18.33). This header allows a client to request that two or more requests be processed in the same RTSP session context that the first request creates. In other words, a client can request that two or more media streams be set up and then played without needing to wait for a single response. This speeds up the initial start-up time for an RTSP session by 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 requests fails unexpectedly, the PLAY request can still be successfully executed. However, the resulting presentation will not be 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 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 36) and MUST be a token as defined by the ABNF [RFC5234] in Section 20. The methods are summarized in Table 7.

| method | direction | object | Server req. | Client req. |
|---|---|---|---|---|
| DESCRIBE | C -> S | P,S | recommended | recommended |
| GET_PARAMETER | C -> S | P,S | optional | optional |
| | S -> C | P,S | optional | optional |
| OPTIONS | C -> S | P,S | required | required |
| | S -> C | P,S | optional | optional |
| PAUSE | C -> S | P,S | required | required |
| PLAY | C -> S | P,S | required | required |
| PLAY_NOTIFY | S -> C | P,S | required | required |
| REDIRECT | S -> C | P,S | optional | required |
| SETUP | C -> S | S | required | required |
| SET_PARAMETER | C -> S | P,S | required | optional |
| | S -> C | P,S | optional | optional |
| TEARDOWN | C -> S | P,S | required | required |
| | S -> C | P | required | required |

Table 7: Overview of RTSP Methods

Note on Table 7: This table covers RTSP methods, their direction, and on what objects (P: presentation, S: stream) they operate. Further, it indicates whether a server or a client implementation is required (mandatory), recommended, or optional.

Further note on Table 7: the GET_PARAMETER is optional. For example, a fully functional server can be built to deliver media without any parameters. However, SET_PARAMETER is required, i.e., mandatory to implement for the server; this is due to its usage for keep-alive. PAUSE is required because it is the only way of leaving the Play state without terminating the whole session.

If an RTSP agent does not support a particular method, it MUST return a 501 (Not Implemented) response code and the requesting RTSP agent, in turn, SHOULD NOT try this method again for the given agent/resource combination. An RTSP proxy whose 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 that 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 Section 4.3.7 of [RFC7231]. However, in RTSP, OPTIONS is bidirectional in that a client can send the request 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. In addition to this "MUST-implement" functionality, clients, servers and proxies MAY provide support both for sending OPTIONS requests and for generating responses to the requests.

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 the 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, this is not the preferred way of session keep-alive signaling; see Section 18.49. An OPTIONS request intended for keeping alive an RTSP session MUST include the Session header with the associated session identifier. Such a request SHOULD also use the media or the aggregated control URI as the Request-URI.

Example:

```
  C->S:  OPTIONS rtsp://server.example.com RTSP/2.0
         CSeq: 1
         User-Agent: PhonyClient/1.2
         Proxy-Require: gzipped-messages
         Supported: play.basic

  S->C:  RTSP/2.0 200 OK
         CSeq: 1
         Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE, OPTIONS
         Supported: play.basic, setup.rtp.rtcp.mux, play.scale
         Server: PhonyServer/1.1
```

Note that the "gzipped-messages" feature tag in the Proxy-Require is a fictitious feature.

## 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, if the DESCRIBE method request can be successfully fulfilled. The DESCRIBE reply-response pair constitutes the media initialization phase of RTSP.

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, using the 3rr responses is RECOMMENDED.

> By forcing a DESCRIBE response to contain all media initialization information 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.

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=MNobody 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=control:*
        t=2873397496 2873404696
        m=audio 3456 RTP/AVP 0
        a=control:audio
        m=video 2232 RTP/AVP 31
        a=control:video
```

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.)

o via some form of 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. The client should use any MTag to either validate the presentation description or make the session establishment conditional on being valid.

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, or other means that are appropriate to the operating environment of the clients.

### 13.3. SETUP

The description below uses the following states in a protocol state machine that is related to a specific session when that session has been created. The state transitions are driven by protocol interactions. For additional information about the state machine, see Appendix B.

Init: Initial state. No session exists.

Ready: Session is ready to start playing.

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

The SETUP request for a URI specifies the transport mechanism to be used for the streamed media. The SETUP method may be used in two different cases, namely, creating an RTSP session and changing the transport parameters of media streams that are already set up. SETUP can be used in all three states, Init, Ready, and Play, to change the transport parameters. Additionally, Init and Ready can also be used for the creation of the RTSP session. The usage of the SETUP method in the Play state to add a media resource to the session is unspecified.

The Transport header, see Section 18.54, 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, so 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 of possible configurations that are offered as choices to the server. All transport-related parameters SHALL be included in the Transport header; the use of other headers for this purpose is NOT RECOMMENDED 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 formats it may use in future session-related responses, such as a 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 18.5) to indicate which time formats are acceptable to use for this media resource.

The SETUP 200 OK response MUST include the Media-Properties header (see Section 18.29). The combination of the parameters of the Media-Properties header indicates the nature of the content present in the session (see also Section 4.7). For example, a live stream with time shifting is indicated by

o Random access set to Random-Access,

o Content Modifications set to Time-Progressing, and

o Retention set to Time-Duration (with specific recording window time value).

The SETUP 200 OK response MUST include the Media-Range header (see Section 18.30) 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, clock
        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: QKyjN8nt2WqbWw4tIYof52;timeout=60
        Transport: RTP/AVP;unicast;dest_addr="192.0.2.53:4588"/
                   "192.0.2.53:4589"; src_addr="198.51.100.241:6256"/
                   "198.51.100.241:6257"; ssrc=2A3F93ED
        Accept-Ranges: npt
        Media-Properties: Random-Access=3.2, Time-Progressing,
                          Time-Duration=3600.0
        Media-Range: npt=0-2893.23
```

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 are either RTP/AVP/UDP (UDP per default) to be received on client port 4588 and 4589 at the address the RTSP setup connection comes from or RTP/AVP interleaved on the RTSP control channel. The server selects the RTP/AVP/UDP transport and adds the address and ports it will send and receive 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 or a Pipelined-Requests header referencing an existing session context. In that latter case, the server MUST bundle this SETUP request into the existing session (aggregated session) or return a 459 (Aggregate Operation Not Allowed) error code (see Section 17.4.23). 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 (see Section 13.4.2 for aggregated sessions and for the particular URIs see Appendix D.1.1). 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 choose not to 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 that 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 on which a request was operating.

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 18.49. Signs of liveness for an RTSP session include any RTSP requests from a client that contain a Session header with the ID for that session, as well as RTCP sender or receiver reports if RTP is used to transport the underlying media stream. RTCP sender reports may, for example, be received in session where the server is invited into a conference session and are thus valid as a liveness indicator.

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 the changing of parameters, it MUST respond with error 455 (Method Not Valid in This State). The reasons to support changing transport parameters include allowing application-layer mobility and flexibility to utilize the best available transport as it becomes available. If a client receives a 455 error 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 that also fails, the changing of transport parameters will require that the client perform a TEARDOWN of the affected media and then set it up again. For an aggregated session, not 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 the transport protocol completely, like switching from Interleaved TCP mode to UDP or vice versa, or to change the delivery address.

In a SETUP response for a request to change the transport parameters while in Play state, the server MUST include the Range header to indicate at what point the new transport parameters will be used. Further, if RTP is used for delivery, the server MUST also include the RTP-Info header to indicate at what timestamp and RTP sequence number the change will take place. If both RTP-Info and Range are 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 that happened while in Play state results in a change of synchronization-related information, for example, changing RTP SSRC, the server MUST include the necessary synchronization information in the SETUP response. However, the server SHOULD 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 state. A PLAY request MUST include a Session header to indicate to which session the request applies.

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, within the limits of the media resource and in accordance with the Seek-Style header (Section 18.47). It MUST deliver stream data until the end of the range if given, until a new PLAY request is received, until a PAUSE request (Section 13.5) is received, or until 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 session 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., the current delivery 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 position.

What range values are 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 union 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 18.30) with the valid range for the media. Except for time-progressing content where the client requests 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 header's stop 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 first of either the provided stop point or the end of the media. Please note that if one simply wants to play from a particular start point until the end of media, using a Range header with an implicit stop point is RECOMMENDED.

If a client requests to start playing at the end of media, either explicitly with a Range header or implicitly with a pause point that is at the end of media, a 457 (Invalid Range) error MUST be sent and include the Media-Range header (Section 18.30). It is specified below that the Range header also must be included in the response and that it will carry the pause point in the media, in the case of the session being in Ready State. Note that this also applies if the pause point or requested start point is at the beginning of the media and a Scale header (Section 18.46) is included with a negative value (playing backwards).

For media with random access properties, a client may indicate which policy for start point selection the server should use. This is done by including the Seek-Style header (Section 18.47) in the PLAY request. The Seek-Style applied will affect the content of the Range header as it will be adjusted to indicate from what point the media actually is delivered.

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 and 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 MUST be included in a Range header.

All range specifiers in this specification allow for ranges with an implicit start point (e.g., "npt=-30"). When used in a PLAY request, the server treats this as a request to start or resume delivery 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 returned.

The example below will play seconds 10 through 25. It also requests that the server 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: ULExwZCXh2pd0xuFgkgZJW
      Range: npt=10-25
      Seek-Style: RAP
      User-Agent: PhonyClient/1.2
```

Servers 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 request. 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 a Range header with the pause point as the explicit start point and an implicit stop 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 Play state, the current playout point and the remaining parts of the range request are 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 carrying synchronization information. As the information necessary is dependent on the media-transport format, further rules specifying the header and its usage are needed. For RTP the RTP-Info header is specified, see Section 18.45, 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 ms 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: ULExwZCXh2pd0xuFgkgZJW
      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
          Session: ULExwZCXh2pd0xuFgkgZJW
      RTP-Info:url="rtsp://example.com/audio"
         ssrc=0D12F123:seq=14783;rtptime=2345962545

S->C: RTP Packet TS=2345962545 => NPT=3.51
      Media duration=0.16 seconds
```

The server replies 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 formats 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 whether 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 not to render that time period.

```
C->S: PLAY rtsp://example.com/audio RTSP/2.0
      CSeq: 836
      Session: ZGGyCJOs8xaLkdNK2dmxQO
      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
          Session: ZGGyCJOs8xaLkdNK2dmxQO
      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 change to the Ready state, media delivery simply stops. If it is necessary to put the stream into the Ready state, a PAUSE request MUST be issued. A PLAY request while the stream is still in the Play 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 handled in the same way as if as the request was received in Ready state. In the case that the range in the Range header has an implicit start time ("-endtime"), the server MUST continue to play from where it currently was until the specified endpoint. This is useful to change the end to 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 have been broken into several lines, where subsequent lines start with whitespace as allowed by the syntax.

```
C->S: PLAY rtsp://audio.example.com/twister.en RTSP/2.0
      CSeq: 833
      Session: N465Wvsv0cjUy6tLqINkcf
      Range: smpte=0:10:20-
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 833
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Session: N465Wvsv0cjUy6tLqINkcf
      Server: PhonyServer/1.0
      Range: smpte=0:10:22-0:15:45
      Seek-Style: Next
      RTP-Info:url="rtsp://example.com/twister.en"
         ssrc=0D12F123:seq=14783;rtptime=2345962545
```

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

```
C->S: PLAY rtsp://audio.example.com/meeting.en RTSP/2.0
      CSeq: 835
      Session: N465Wvsv0cjUy6tLqINkcf
      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
      Session: N465Wvsv0cjUy6tLqINkcf
      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 stream and on aggregated sessions controlling multiple media streams.

In an aggregated session, the PLAY request MUST contain an aggregated control URI. A server MUST respond with a 460 error (Only Aggregate Operation Allowed) if the client PLAY Request-URI is for a single 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 requests, a PLAY request MAY be pipelined (Section 12) after those additional SETUP requests without awaiting their responses. This procedure can reduce the delay from the start of session establishment until media playout has started with one RTT. 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 state may not be the intended one due to failure of any of the prior requests. 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 Play 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 the request is constructed. The session is actively playing media and the play point will be moving, making the exact time a request will take effect hard to predict. Depending on how the PLAY header appears, two different cases exist: total replacement or continuation. A total replacement is signaled 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 an 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 if this was the request originally played. If the current delivery point is beyond the stop point, the server SHALL immediately pause delivery. As the request has been completed successfully, it shall be responded to with a 200 OK response. A PLAY_NOTIFY with end-of-stream is also sent to indicate the actual stop point. The pause point is set to the requested stop point.

The following is 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). The server changes the current play to continue 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: apzA8LnjQ5KWTdw0kUkiRh
      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
      Session: apzA8LnjQ5KWTdw0kUkiRh
      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


C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 835
      Session: apzA8LnjQ5KWTdw0kUkiRh
      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
      Session: apzA8LnjQ5KWTdw0kUkiRh
      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
```

A common use of a PLAY request while in Play state is changing the scale of the media, i.e., entering or leaving fast forward or fast rewind. The client can issue an updating PLAY request that is either a continuation or a complete replacement, as discussed above this section. Below is an example of a client that is requesting a fast forward (scale = 2) without giving a stop point and then a change from fast forward to regular playout (scale = 1). In the second PLAY request, the time is set explicitly to be wherever the server currently plays out (npt=now-) and the server responds with the actual playback point where the new scale actually takes effect (npt=02:17:27.144-).

```
  C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
        CSeq: 2034
        Session: apzA8LnjQ5KWTdw0kUkiRh
        Range: npt=now-
        Scale: 2.0
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 2034
        Date: Thu, 23 Jan 1997 15:35:06 GMT
        Session: apzA8LnjQ5KWTdw0kUkiRh
        Server: PhonyServer/1.0
        Range: npt=02:17:21.394-
        Seek-Style: Next
        Scale: 2.0
        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
```

[playing in fast forward and now returning to scale = 1]

```
  C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
        CSeq: 2035
        Session: apzA8LnjQ5KWTdw0kUkiRh
        Range: npt=now-
        Scale: 1.0
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 2035
        Date: Thu, 23 Jan 1997 15:35:09 GMT
        Session: apzA8LnjQ5KWTdw0kUkiRh
        Server: PhonyServer/1.0
        Range: npt=02:17:27.144-
        Seek-Style: Next
        Scale: 1.0
        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
```

### 13.4.4. Playing On-Demand Media

On-demand media is indicated by the content of the Media-Properties header in the SETUP response when (see also Section 18.29):

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

Playing on-demand media follows the general usage as described in Section 13.4.1.

### 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 when (see also Section 18.29):

- o the Random Access property is set to Random-Access;
- o the Content Modifications property is set to Dynamic;
- o the Retention property is set to 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 two ways for the client to be informed about changes of media resources in Play state. The first being that the client will receive a PLAY_NOTIFY request with the Notify-Reason header set to media-properties-update (see Section 13.5.2). The client can use the value of the Media-Range header 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 18.30).

### 13.4.6. Playing Live Media

Live media is indicated by the content of the Media-Properties header in the SETUP response when (see also Section 18.29):

- o the Random Access property is set to No-Seeking;
- o the Content Modifications property is set to Time-Progressing;

o the Retention property's Time-Duration is set to 0.0.

For live media, the SETUP 200 OK response MUST include the Media-Range header (see Section 18.30).

A client MAY send PLAY requests without the Range header. If the request includes 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-" cannot be used 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 Absolute Time format can only be used if the client has shown support for it using the Accept-Ranges header.

### 13.4.7. Playing Live with Recording

Certain media servers 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 when (see also Section 18.29):

o the Random Access property is set to Random-Access;

o the Content Modifications property is set to Time-Progressing;

o the Retention property is set to Time-Limited or Unlimited

The SETUP 200 OK response MUST include the Media-Range header (see Section 18.30) for this type of media. For live media with recording, the Range header indicates the current delivery point 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 delivery point to the requested border of the window. If the client requests a delivery point that is located outside the recording window, e.g., if the requested point is too far in the past, the server selects the oldest point in the recording. The considerations in Section 13.5.3 apply if a client requests delivery with scale (Section 18.46) values other than 1.0 (normal playback rate) while delivering live media with recording.

### 13.4.8. Playing Live with Time-Shift

Certain media servers 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 when (see also Section 18.29):

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

The SETUP 200 OK response MUST include the Media-Range header (see Section 18.30) for this type of media. For live media with recording, the Range header indicates the current time in the media and the Media-Range header 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 delivery using a scale (Section 18.46) value other than 1.0 (normal playback rate) while delivering live media with time-shift.

## 13.5. PLAY_NOTIFY

The PLAY_NOTIFY method is issued by a server to inform a client about an asynchronous 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 use both an aggregate control URI and individual media resource URIs, depending on the scope of the notification. This scope may have important distinctions for aggregated sessions, and each reason for a PLAY_NOTIFY request needs to specify the interpretation as well as if aggregated control URIs or individual URIs may be used in requests.

PLAY_NOTIFY requests can 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 the use of a message body. In this case, there is 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 18.32) specifies the reason why the server sends the PLAY_NOTIFY request. This is extensible and new reasons can be added in the future (see Section 22.8). In case the client does not understand the reason for the notification, it MUST respond with a 465 (Notification Reason Unknown) (Section 17.4.29) error code. This document defines how servers can send PLAY_NOTIFY with Notify-Reason values of these types:

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

### 13.5.1. End-of-Stream

A PLAY_NOTIFY request with the 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 Play state. The end of the media stream delivery notification may be used either to indicate 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 18.42) 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 MAY 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 the 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 made evident because 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 timescale is moving backwards or has a non-default pace. The end-of-stream notification does not prevent the client from sending a new PLAY request.

If RTP is used as media transport, an RTP-Info header MUST be included, and the RTP-Info header MUST indicate the last sequence number in the sequence parameter.

For an RTSP Session where media resources are under aggregated control, the media resources will normally end at approximately the same time, although some small differences may exist, on the scale of a few hundred milliseconds. In those cases, an RTSP session under aggregated control SHOULD send only a single PLAY_NOTIFY request. By using the aggregate control URI in the PLAY_NOTIFY request, the RTSP server indicates that this applies to all media resources within the session. In cases in which RTP is used for media delivery, corresponding RTP-Info needs to be included for all media resources. In cases where one or more media resources have a significantly shorter duration than some other resources in the aggregated session, the server MAY send end-of-stream notifications using individual media resource URIs to indicate to agents that there will be no more media for this particular media resource related to the current active PLAY request. In such cases, when the remaining media resources come to the end of the stream, they MUST send a PLAY_NOTIFY request using the aggregate control URI to indicate that no more resources remain.

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/fizzle/foo/audio"
         ssrc=0D12F123:seq=14783;rtptime=2345962545,
         url="rtsp://example.com/fizzle/video"
         ssrc=789DAF12:seq=57654;rtptime=2792482193
      Session: CDtUJfDQXJWtJ7Iqua2xOi
      Date: Mon, 08 Mar 2010 13:37:16 GMT

C->S: RTSP/2.0 200 OK
      CSeq: 854
      User-Agent: PhonyClient/1.2
      Session: CDtUJfDQXJWtJ7Iqua2xOi
```

### 13.5.2. Media-Properties-Update

A PLAY_NOTIFY request with a Notify-Reason header set to media-properties-update indicates an update of the media properties for the given session (see Section 18.29) or the available media range that can be played as indicated by the Media-Range header (Section 18.30). 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. The Media-Properties header has session scope; thus, for aggregated sessions, the PLAY_NOTIFY request MUST use the aggregated control URI.

This notification MUST be sent for media that are time-progressing every time an event happens that changes the basis for making estimates on how the available for play-back media range will progress with wall clock time. In addition, it is RECOMMENDED that the server send these notifications approximately every 5 minutes for time-progressing content to ensure the long-term stability of the client estimation and allow for clock skew detection by the client. The time between notifications should be greater than 1 minute and less than 2 hours. For the reasons just explained, requests MUST include a Media-Range header to provide current Media duration and a 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, it is 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.

```
 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: media-properties-update
       Session: CDtUJfDQXJWtJ7Iqua2xOi
       Media-Properties: Time-Progressing,
             Time-Limited=20080415T153919.36Z, Random-Access=5.0
       Media-Range: npt=00:00:00-01:37:21.394
       Range: npt=01:15:49.873-

  C->S: RTSP/2.0 200 OK
       CSeq: 854
       User-Agent: PhonyClient/1.2
       Session: CDtUJfDQXJWtJ7Iqua2xOi
```

### 13.5.3. Scale-Change

The server may be forced to change the rate of media time per playback time when a client requests delivery using a scale (Section 18.46) value 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 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 delivery 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 lose any media, the scale will need to be adjusted to the same rate at which the media is removed from the storage buffer, commonly scale = 1.0.

Another case is when the content itself consists 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 cases, the media properties will also be sent, updating the supported scale values. This enables a client to adjust the scale value used.

To minimize impact on playback in any of the above cases, the server MUST modify the playback properties, set scale to a supportable value, and continue delivery of 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 when the change took effect, a Scale header with the new value in use, a Session header with the identifier for the session to which it applies, and a Date header with the server wallclock time of the change. For time-progressing content, the Media-Range and the Media-Properties headers at this point in time also MUST be included. The Media-Properties header MUST be included if the scale change was due to the content changing what scale values ("Scales") are supported.

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

PLAY_NOTIFY requests for aggregated sessions MUST use the aggregated control URI in the request. The scale change for any aggregated session applies to all media streams that are part of the aggregate.

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: CDtUJfDQXJWtJ7Iqua2xOi
      Media-Properties: Time-Progressing,
            Time-Limited=20080415T153919.36Z, Random-Access=5.0
      Media-Range: npt=00:00:00-01:37:21.394
      Range: npt=01:37:21.394-
      Scale: 1
      RTP-Info: url="rtsp://example.com/fizzle/foo/audio"
          ssrc=0D12F123:rtptime=2345962545,
          url="rtsp://example.com/fizzle/foo/videotrack"
          ssrc=789DAF12:seq=57654;rtptime=2792482193

C->S: RTSP/2.0 200 OK
      CSeq: 854
      User-Agent: PhonyClient/1.2
      Session: CDtUJfDQXJWtJ7Iqua2xOi
```

## 13.6. PAUSE

The PAUSE request causes the stream delivery to immediately be interrupted (halted). A PAUSE request MUST be made either with the aggregated control URI for aggregated sessions, resulting in all media being halted, or with the media URI for non-aggregated sessions. Any attempt to mute a single media with a PAUSE request in an aggregated session MUST be responded to with a 460 (Only Aggregate Operation Allowed) error. 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: OoOUPyUwt0VeY9fFRHuZ6L
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 834
      Date: Thu, 23 Jan 1997 15:35:06 GMT
              Session: OoOUPyUwt0VeY9fFRHuZ6L
      Range: npt=45.76-75.00
```

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 a Range header resumes from the pause point and plays 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 includes 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 MUST 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: OccldOFFq23KwjYpAnBbUr
      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: OccldOFFq23KwjYpAnBbUr
```

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: OccldOFFq23KwjYpAnBbUr
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 835
      Date: 23 Jan 1997 15:35:17 GMT
      Server: PhonyServer/1.0
      Range: npt=21-30
      Session: OccldOFFq23KwjYpAnBbUr
```

If a client issues a PAUSE request and the server acknowledges and enters the Ready state, the proper server response, if the player issues another PAUSE, is still 200 OK. The 200 OK response MUST include the Range header with the current pause point. See examples below:

```
C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 834
      Session: OccldOFFq23KwjYpAnBbUr
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 834
      Session: OccldOFFq23KwjYpAnBbUr
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Range: npt=45.76-98.36

C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 835
      Session: OccldOFFq23KwjYpAnBbUr
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 835
      Session: OccldOFFq23KwjYpAnBbUr
      Date: 23 Jan 1997 15:35:07 GMT
      Range: npt=45.76-98.36
```

## 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 can 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 to what session the request applies. The TEARDOWN request MUST NOT include a Terminate-Reason header.

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 being immediately halted and the session state being 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 can 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 a session returning to non-aggregated control, because it only contains a single media after the request’s 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 extant or has been removed.

Example:

```
C->S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 892
      Session: OccldOFFq23KwjYpAnBbUr
      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 that the session has been inactive for extended periods of time. The reason is provided in the Terminate-Reason header (Section 18.52).

When an RTSP client has maintained an RTSP session that otherwise is inactive for an extended period of time, the server may reclaim the resources. That is done by issuing a TEARDOWN 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 18.49). However, the server is NOT RECOMMENDED to perform this operation until an extended period of inactivity of 10 times the Session-Timeout period has passed. It is up to the operator of the RTSP server to actually configure how long this extended period of inactivity is. An operator should take into account, when doing this configuration, what the served content is and what this means for the extended period of inactivity.

In case the server needs to stop providing service to the established sessions and there is no server to point at in a REDIRECT request, then 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 than to terminate the sessions.

The TEARDOWN request MUST be made only on the session aggregate control URI (i.e., it is not allowed to terminate individual media streams, if it is a session aggregate), and it MUST include the following headers: Session and Terminate-Reason. 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 wildcard 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 RTSP connection between the next-hop RTSP agent and the server. This request indicates that all sessions and pending requests being managed via the 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 target server in the redirect.

## 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 approach includes headers that have been defined to be usable for this purpose. Headers 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 headers in this specification that MAY be used for retrieving their current value using GET_PARAMETER are listed below; additional headers MAY be specified in the future:

- Accept-Ranges

- Media-Range

- Media-Properties

- Range

- RTP-Info

The other way is to specify a message body that lists the parameter(s) that are desired to be retrieved. The Content-Type header (Section 18.19) is used to specify which format the message body has. If the receiver of the request does not support the media type used for the message body, it SHALL respond using the error code 415 (Unsupported Media Type). The responder to a GET_PARAMETER request MUST use the media type of the request for the response. For additional considerations regarding message body negotiation, see Section 9.3.

RTSP agents implementing support for responding to GET_PARAMETER requests SHALL implement the "text/parameters" format (Appendix F). This to ensure that at least one known format for parameters is implemented and, thus, prevent parameter format negotiation failure.

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 the parameters that weren't understood. If all parameters are understood, their values are filled in and returned in the response message body.

The method can also be used without a message body or any header that requests parameters for keep-alive purposes. The keep-alive timer has been updated for any request that is successful, i.e., a 200 OK response is received. Any non-required header present in such a request may or may not have 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. For this reason, it is usually easier if any parameters to be retrieved are sent in the body, rather than using any header.

Example:

```
S->C: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 431
      User-Agent: PhonyClient/1.2
      Session: OccldOFFq23KwjYpAnBbUr
      Content-Length: 26
      Content-Type: text/parameters

      packets_received
      jitter

C->S: RTSP/2.0 200 OK
      CSeq: 431
      Session: OccldOFFq23KwjYpAnBbUr
      Server: PhonyServer/1.1
      Date: Mon, 08 Mar 2010 13:43:23 GMT
      Content-Length: 38
      Content-Type: text/parameters

      packets_received: 10
      jitter: 0.3838
```

## 13.9. SET_PARAMETER

This method requests the setting of the value of a parameter or a set of parameters for a presentation or stream specified by the URI. If the Session header is present in a request, the value of a parameter MUST be retrieved in the specified session context. The method MAY also be used without a message body. It is the RECOMMENDED method to be used in a 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 have 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.

When using a message body to list the parameter(s) desired to be set, the Content-Type header (Section 18.19) is used to specify which format the message body has. If the receiver of the request is not supporting the media type used for the message body, it SHALL respond using the error code 415 (Unsupported Media Type). For additional considerations regarding message body negotiation, see Section 9.3. The responder to a SET_PARAMETER request MUST use the media type of the request for the response. For additional considerations regarding message body negotiation, see Section 9.3.

RTSP agents implementing support for responding to SET_PARAMETER requests SHALL implement the text/parameters format (Appendix F). This is to ensure that at least one known format for parameters is implemented and, thus, prevent parameter format negotiation failure.

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. When 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, a body MUST NOT be returned. The response body MUST use the media type of the request for the response.

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 connected to border RTSP proxies.
>
> 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
      Session: iixT43KLc
      Date: Mon, 08 Mar 2010 14:45:04 GMT
      Content-length: 20
      Content-type: text/parameters

      barparam: barstuff

S->C: RTSP/2.0 451 Parameter Not Understood
      CSeq: 421
      Session: iixT43KLc
      Server: PhonyServer/1.0
      Date: Mon, 08 Mar 2010 14:44:56 GMT
      Content-length: 20
      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 may happen for different reasons. One is that the server is being administered 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 18.52) 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 a 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 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 connection MUST be redirected. The 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.

A server that needs to terminate a session or all its sessions and lacks an alternative server to redirect to, SHALL instead use TEARDOWN requests.

When no Terminate-Reason "time" parameter is 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 to be terminated and can free any associated state after it receives the successful (2xx) response. The server MAY close the signaling connection upon receiving the response, and the client SHOULD close the signaling 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 signaling 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 signaling connection.

The Terminate-Reason header "time" parameter MAY be used to indicate the wallclock time by which the redirection MUST have taken 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 terminated the session and closed the connection before the redirection time-line terminated. The server MAY simply cease to provide service when the deadline time has been reached, or it can issue a TEARDOWN requests to the remaining sessions.

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 refuse to respond to a REDIRECT request. This action removes any incentive of not acknowledging the reception of a REDIRECT request.

After a REDIRECT request has been processed, a client that wants to continue to 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 is RECOMMENDED as a means 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/fizzle/foo
      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
      Session: uZ3ci0K+Ld-M
```

## 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. Interleaved data is not allowed inside RTSP messages.

Stream data, such as RTP packets, is encapsulated by an ASCII dollar sign (36 decimal) followed by a one-octet channel identifier and the length of the encapsulated binary data as a binary, two-octet unsigned integer in network octet order (Appendix B of [RFC791]). The stream data follows immediately afterwards, without a CRLF, but including the upper-layer protocol headers. Each dollar sign block MUST contain exactly one upper-layer protocol data unit, e.g., one RTP packet.

Note that this mechanism does not support PDUs larger than 65535 octets, which matches the maximum payload size of regular, non-jumbo IPv4 and IPv6 packets. If the media delivery protocol intended to be used has larger PDUs than that, a definition of a PDU fragmentation mechanism will be required to support embedded binary data.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| "$" = 36      | Channel ID    | Length in octets              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
: Binary data (Length according to Length field)                :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Embedded Interleaved Binary Data Format

The channel identifier is defined in the Transport header with the interleaved parameter (Section 18.54).

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 an interval containing a second channel in the interleaved parameter of the Transport header (see Section 18.54). If RTCP is used, packets MUST be sent on the first available channel that is higher than the RTP channel. The channels are bidirectional, using the same Channel ID in both directions; therefore, RTCP traffic is sent on the second channel in both directions.

RTCP is sometimes 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 RTSP connection (TCP or TCP/TLS) when required by the network configuration and to transfer them onto UDP when possible.

```
C->S: SETUP rtsp://example.com/bar.file RTSP/2.0
      CSeq: 2
      Transport: RTP/AVP/TCP;unicast;interleaved=0-1
      Accept-Ranges: npt, smpte, clock
      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: OccldOFFq23KwjYpAnBbUr
      Accept-Ranges: npt
      Media-Properties: Random-Access=0.2, Immutable, Unlimited

C->S: PLAY rtsp://example.com/bar.file RTSP/2.0
      CSeq: 3
      Session: OccldOFFq23KwjYpAnBbUr
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 3
      Session: OccldOFFq23KwjYpAnBbUr
      Date: Thu, 05 Jun 1997 18:57:19 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 octet length}{"length" octets data, w/RTP header}
S->C: $005{2 octet length}{"length" octets data, w/RTP header}
S->C: $006{2 octet length}{"length" octets  RTCP packet}
```

## 15. Proxies

RTSP Proxies are RTSP agents that are located in between a client and a server. A proxy can take on the roles of both client and server depending on what it tries to accomplish. RTSP proxies use two transport-layer connections: one from the RTSP client to the RTSP proxy and a second from the RTSP proxy to the RTSP server. Proxies are introduced for several different reasons; those listed 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 Section 16. This type of proxy is also expected to understand RTSP endpoint 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 gets access to servers and content on an external network or gets access by 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 also to understand RTSP endpoint 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 an RTSP client gets 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 proxy is required to redirect the media to itself or a controlled gateway that performs 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, in the case of a firewalled network, the security proxy requests that the necessary pinholes in the firewall are opened when a client in the protected network wants to access media streams on the external side. 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. The functionality of this proxy is usually closely tied into understanding all aspects of the media transport.

Auditing Proxy: RTSP proxies can also provide network owners with a logging and auditing point for RTSP sessions, e.g., for corporations that track 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 types of proxies can also be used 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 types of deployment. In those cases, the secured sessions do bypass 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 [RFC7825]. 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. This would impede RTSP's future development and usage.

## 15.1. Proxies and Protocol Extensions

The existence of proxies must always be considered when developing new RTSP extensions. Most types of proxies will need to implement any new method to operate correctly in the presence of that extension. New headers can be introduced and will not be blocked by older proxies. However, it is important to consider if this header and its function are required to be understood by the proxy or if it can be simply forwarded. If the header needs to be understood, a feature tag representing the functionality MUST be included in the Proxy-Require header. Below are guidelines for analysis whether the header needs to be understood. The Transport header and its parameters are extensible, which requires handling rules for a proxy in order to ensure a correct interpretation.

Whether or not a proxy needs to understand a header is not easy to determine as they serve a broad variety of functions. When evaluating if a header needs to be understood, one can divide the functionality into three main categories:

Media modifying: The caching and translator proxies modify the actual media and therefore need also to understand the request directed to the server that affects how the media is rendered. Thus, this type of proxy also needs to understand the server-side functionality.

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

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

An extension has to be classified as mandatory to be implemented for a proxy, if an extension has to be understood by a "Transport modifying" type of proxy.

## 15.2. Multiplexing and Demultiplexing of Messages

RTSP proxies may have to multiplex several RTSP sessions from their clients towards RTSP servers. This requires that RTSP requests from multiple clients be multiplexed onto a common connection for requests outgoing to an RTSP server, and, on the way back, the responses be demultiplexed from the server to per-client responses. On the protocol level, this requires that request and response messages be handled in both directions, requiring that there be a mechanism to correlate which request/response pair exchanged between proxy and server is mapped to which client (or client request).

This multiplexing of requests and demultiplexing of responses is done by using the CSeq header field. The proxy has to rewrite the CSeq in requests to the server and responses from the server and remember which CSeq is mapped to which client. The proxy also needs to ensure that the order of the message related to each client is maintained. Section 18.20 defines the handling of how requests and responses are rewritten.

## 16. Caching

In HTTP, request/response 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 allowed behavior of the cache is given by the cache-response directives described in Section 18.11. 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 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, an RTSP proxy cache appears 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 (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.

## 16.1. Validation Model

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. This is called "validating" the cache entry. To avoid having to pay the overhead of retransmitting the full response if the cached entry is good, and at the same time avoiding having to pay the overhead of an extra round trip if the cached entry is invalid, RTSP 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 requested resource, and, if they match (see Section 16.1.3), 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, avoiding transmitting the full response if the validator matches and avoiding 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 or SETUP) into a conditional.

The protocol includes both positive and negative senses of cache-validating conditions. That is, it is possible to request that a method be performed either 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 directive (see Section 18.11). However, a cache cannot perform a conditional retrieval if it does not have a validator for the resource, which means it will not be refreshable after it expires.

Media streams that are being adapted based on the transport capacity between the server and the cache make caching more difficult. A server needs to consider how it views the caching of media streams that it adapts and potentially instruct any caches not to cache such streams.

### 16.1.1. Last-Modified Dates

The Last-Modified header (Section 18.27) value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the cache entry was created after the Last-Modified time.

### 16.1.2. 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 4.6

### 16.1.3. 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 this validator is a "strong validator". The Message body (i.e., the presentation description) or any associated message body headers is named 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 a 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.

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 subrange retrieval, since otherwise the client might end up with an internally inconsistent entity.

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

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

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

- o 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.

A 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 more than once 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 subrange 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.

#### 16.1.4. Rules for When to Use Message Body Tags and Last-Modified Dates

This document adopts 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:

- o SHOULD send a message body tag validator unless it is not feasible to generate one.

- o 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.

- o 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:

- o If a 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).

- o 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 a 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.

### 16.1.5. 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 octet equality would open up a can of worms. Thus, comparisons of any other headers are never used for purposes of validating a cache entry.

## 16.2. Invalidation after Updates or Deletions

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 RTSP 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 mark these as "invalid" and in need of a mandatory revalidation before they can be returned in response to a subsequent request.

Some RTSP 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:

- o DESCRIBE

- o SETUP

In order to prevent DoS 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.

## 17. Status Code Definitions

Where applicable, HTTP status codes (see Section 6 of [RFC7231]) are reused. See Table 4 in Section 8.1 for a listing of which status codes may be returned by which requests. All error messages, 4xx and 5xx, MAY return a body containing further information about the error.

### 17.1. Informational 1xx

#### 17.1.1. 100 Continue

The requesting agent SHOULD continue with its request. This interim response is used to inform the requesting agent that the initial part of the request has been received and has not yet been rejected by the responding agent. The requesting agent SHOULD continue by sending the remainder of the request or, if the request has already been completed, continue to wait for a final response (see Section 10.4). The responding agent MUST send a final response after the request has been completed.

### 17.2. Success 2xx

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

#### 17.2.1. 200 OK

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

### 17.3. Redirection 3xx

The notation "3xx" indicates response codes from 300 to 399 inclusive that are meant for redirection. We use the notation "3rr" to indicate all 3xx codes used for redirection, i.e., excluding 304. The 304 response code appears here, rather than a 2xx response code, which would have been appropriate; 304 has also been used in RTSP 1.0 [RFC2326].

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

A 3rr code MAY be used to respond to any request. The Location header MUST be included in any 3rr response. 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 agent, the server MAY need to resort to using 3rr responses to inform an agent with an established session about the need for redirecting the session. If a 3rr response is received for a request in relation to an established session, the agent 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 agent is redirected to; the URI MUST NOT only contain the hostname.

In the event that an unknown 3rr status code is received, the agent SHOULD behave as if a 302 response code had been received (Section 17.3.3).

### 17.3.1. 300

The 300 response code is not used in RTSP 2.0.

### 17.3.2. 301 Moved Permanently

The requested resource is moved permanently and resides now at the URI given by the Location header. The user agent SHOULD redirect automatically to the given URI. This response MUST NOT contain a message body. The Location header MUST be included in the response.

### 17.3.3. 302 Found

The requested resource resides temporarily at the URI given by the Location header. 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 Location header MUST be included in the response. The user agent 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, clock
      User-Agent: PhonyClient/1.2

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

### 17.3.4. 303 See Other

This status code MUST NOT be used in RTSP 2.0. However, it was allowed in RTSP 1.0 [RFC2326].

### 17.3.5. 304 Not Modified

If the agent has performed a conditional DESCRIBE or SETUP (see Sections 18.25 and 18.26) and the requested resource has not been modified, the server SHOULD send a 304 response. This response MUST NOT contain a message body.

The response MUST include the following header fields:

- o Date

- o MTag or Content-Location, if the headers would have been sent in a 200 response to the same request.

- o Expires and Cache-Control 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 header (Section 18.24) may be used to link the DESCRIBE and SETUP in this manner.

### 17.3.6. 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Location header that MUST be included. The Location header field-value 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.

### 17.4. Client Error 4xx

#### 17.4.1. 400 Bad Request

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

#### 17.4.2. 401 Unauthorized

The request requires user authentication using the HTTP authentication mechanism [RFC7235]. The usage of the error code is defined in [RFC7235] and any applicable HTTP authentication scheme, such as Digest [RFC7616]. The response is to include a WWW-Authenticate header (Section 18.58) field containing a challenge applicable to the requested resource. The agent can 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 message body that was given in the response, since that message body might include relevant diagnostic information.

#### 17.4.3. 402 Payment Required

This code is reserved for future use.

#### 17.4.4. 403 Forbidden

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

#### 17.4.5. 404 Not Found

The agent 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 agent 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 agent does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

### 17.4.6. 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.

### 17.4.7. 406 Not Acceptable

The resource identified by the request is only capable of generating response message bodies that have content characteristics not acceptable according to the Accept headers sent in the request.

The response SHOULD include a message body containing a list of available message body characteristics and location(s) from which the user or user agent can choose the one most appropriate. The message body 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.

### 17.4.8. 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized) (Section 17.4.2), but it indicates that the client must first authenticate itself with the proxy. The usage of this error code is defined in [RFC7235] and any applicable HTTP authentication scheme, such as Digest [RFC7616]. The proxy MUST return a Proxy-Authenticate header field (Section 18.34) containing a challenge applicable to the proxy for the requested resource.

### 17.4.9. 408 Request Timeout

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

### 17.4.10. 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.

### 17.4.11. 412 Precondition Failed

The precondition given in one or more of the 'if-' request-header fields evaluated to false when it was tested on the agent. See these sections for the 'if-' headers: If-Match Section 18.24, If-Modified-Since Section 18.25, and If-None-Match Section 18.26. This response code allows the agent 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.

### 17.4.12. 413 Request Message Body Too Large

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

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

### 17.4.13. 414 Request-URI Too Long

The responding agent is refusing to service the request because the Request-URI is longer than the agent is willing to interpret. This rare condition is only likely to occur when an agent has used a request with long query information, when the agent 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 agent is under attack by an agent attempting to exploit security holes present in some agents using fixed-length buffers for reading or manipulating the Request-URI.

#### 17.4.14. 415 Unsupported Media Type

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

#### 17.4.15. 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 agent SHOULD return a message body containing the offending parameter(s).

#### 17.4.16. 452 Illegal Conference Identifier

This status code MUST NOT be used in RTSP 2.0. However, it was allowed in RTSP 1.0 [RFC2326].

#### 17.4.17. 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.

#### 17.4.18. 454 Session Not Found

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

#### 17.4.19. 455 Method Not Valid in This State

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

#### 17.4.20. 456 Header Field Not Valid for Resource

The targeted agent could not act on a required request-header. For example, if PLAY request 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 agent of which formats are allowed.

### 17.4.21. 457 Invalid Range

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

### 17.4.22. 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).

### 17.4.23. 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.

### 17.4.24. 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.

### 17.4.25. 461 Unsupported Transport

The Transport field did not contain a supported transport specification.

### 17.4.26. 462 Destination Unreachable

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

### 17.4.27. 463 Destination Prohibited

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

### 17.4.28. 464 Data Transport Not Ready Yet

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

An example of when this error may occur is in the case in which a client sends a PLAY request to a server prior to ensuring that the TCP connections negotiated for carrying media data were successfully 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.

### 17.4.29. 465 Notification Reason Unknown

This indicates that the client has received a PLAY_NOTIFY (Section 13.5) with a Notify-Reason header (Section 18.32) unknown to the client.

### 17.4.30. 466 Key Management Error

This indicates that there has been an error in a Key Management function used in conjunction with a request. For example, usage of Multimedia Internet KEYing (MIKEY) [RFC3830] according to Appendix C.1.4.1 may result in this error.

### 17.4.31. 470 Connection Authorization Required

The secured connection attempt needs user or client authorization before proceeding. The next hop's certificate is included in this response in the Accept-Credentials header.

### 17.4.32. 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.

### 17.4.33. 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 the agent not accepting any cipher suites.

## 17.5. 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 a message body containing an explanation of the error situation and whether it is a temporary or permanent condition. User agents SHOULD display any included message body to the user. These response codes are applicable to any request method.

### 17.5.1. 500 Internal Server Error

The agent encountered an unexpected condition that prevented it from fulfilling the request.

### 17.5.2. 501 Not Implemented

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

### 17.5.3. 502 Bad Gateway

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

### 17.5.4. 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 that 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 agent SHOULD handle the response as it would for a 500 response. The agent MUST honor the length, if given, in the Retry-After header.

> 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 transport connection.

The response scope is dependent on the request. If the request was in relation to an existing RTSP session, the scope of the overload response is to this individual RTSP session. If the request was not session specific or intended to form an RTSP session, it applies to the RTSP server identified by the hostname in the Request-URI.

### 17.5.5. 504 Gateway Timeout

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

### 17.5.6. 505 RTSP Version Not Supported

The agent does not support, or refuses to support, the RTSP version that was used in the request message. The agent is indicating that it is unable or unwilling to complete the request using the same major version as the agent 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 agent.

### 17.5.7. 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.

### 17.5.8. 553 Proxy Unavailable

The proxy is currently unable to handle the request due to a temporary overloading or maintenance of the proxy. The implication is that this is a temporary condition that 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 agent SHOULD handle the response as it would for a 500 response. The agent MUST honor the length, if given in the Retry-After header.

> Note: The existence of the 553 status code does not imply that a proxy must use it when becoming overloaded. Some proxies may wish to simply refuse the connection.

The response scope is dependent on the Request. If the request was in relation to an existing RTSP session, the scope of the overload response is to this individual RTSP session. If the request was non-session specific or intended to form an RTSP session, it applies to all such requests to this proxy.

## 18. Header Field Definitions

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

This table is an overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on. "Body" denotes if a method is allowed to carry body and in which direction; R = request, r=response. Note: All error messages for statuses 4xx and 5xx are allowed to carry a body.

Table 8: Overview of RTSP Methods

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 definitions for header fields are present in Section 20.2.3. Examples of each header field are given.

Information about header fields in relation to methods and proxy processing is summarized in Figures 2, 3, 4, and 5.

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

R: header field may only appear in requests;

r: header field may only appear in responses;

2xx, 4xx, etc.: 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.

G: header field is a general-header and may be present in both requests and responses.

Note: General headers do not always use the "G" value in the "where" column. This is due to differences when the header may be applied in requests compared to responses. When such differences exist, they are expressed using two different rows: one with "where" being "R" and one with it being "r".

The "proxy" column describes the operations a proxy may perform on a header field. An empty proxy column indicates that the proxy MUST NOT make 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; 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 agents 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 Sections 18.17, 18.19 and 5.3 for details.

-: The header field is not applicable.

"Optional" means that an agent MAY include the header field in a request or response. The agent behavior when receiving such headers varies; for some, it may ignore the header field. In other cases, it is a request to process the header. This is regulated by the method and header descriptions. Examples of headers that require processing are the Require and Proxy-Require header fields discussed in Sections 18.43 and 18.37. A "mandatory" header field MUST be present in a request, and it MUST be understood by the agent receiving the request. A mandatory response-header field MUST be present in the response, and the header field MUST be understood by the 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 agent receiving the request. Similarly, a header field labeled "not applicable" for a response means that the agent MUST NOT place the header field in the response, and the agent 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 a 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 quotes, any semicolon-delimited parameters are header-parameters, not URI parameters.

| Header | Where | Proxy | DES | OPT | STP | PLY | PSE | TRD |
|---|---|---|---|---|---|---|---|---|
| Accept | R | | o | - | - | - | - | - |
| Accept- Credentials | R | rm | o | o | o | o | o | o |
| Accept-Encoding | R | r | o | - | - | - | - | - |
| Accept-Language | R | r | o | - | - | - | - | - |
| Accept-Ranges | G | r | - | - | m | - | - | - |
| Accept-Ranges | 456 | r | - | - | - | m | - | - |
| Allow | r | am | c | c | c | - | - | - |
| Allow | 405 | am | m | m | m | m | m | m |
| Authentication- Info | r | | o | o | o | o | o | o/- |
| Authorization | R | | o | o | o | o | o | o/- |
| Bandwidth | R | | o | o | o | o | - | - |
| Blocksize | R | | o | - | o | o | - | - |
| Cache-Control | G | r | o | - | o | - | - | - |
| Connection | G | ad | o | o | o | o | o | o |
| Connection- Credentials | 470, 407 | ar | o | o | o | o | o | o |
| Content-Base | r | | o | - | - | - | - | - |
| Content-Base | 4xx, 5xx | | o | o | o | o | o | o |
| Content-Encoding | R | r | - | - | - | - | - | - |
| Content-Encoding | r | r | o | - | - | - | - | - |
| Content-Encoding | 4xx, 5xx | r | o | o | o | o | o | o |
| Content-Language | R | r | - | - | - | - | - | - |
| Content-Language | r | r | o | - | - | - | - | - |
| Content-Language | 4xx, 5xx | r | o | o | o | o | o | o |
| Content-Length | r | r | * | - | - | - | - | - |
| Content-Length | 4xx, 5xx | r | * | * | * | * | * | * |
| Content-Location | r | r | o | - | - | - | - | - |
| Content-Location | 4xx, 5xx | r | o | o | o | o | o | o |
| Content-Type | r | r | * | - | - | - | - | - |
| Content-Type | 4xx, 5xx | ar | * | * | * | * | * | * |
| CSeq | Gc | rm | m | m | m | m | m | m |
| Date | G | am | o/* | o/* | o/* | o/* | o/* | o/* |
| Expires | r | r | o | - | o | - | - | - |
| From | R | r | o | o | o | o | o | o |
| If-Match | R | r | - | - | o | - | - | - |
| If-Modified-Since | R | r | o | - | o | - | - | - |
| If-None-Match | R | r | o | - | o | - | - | - |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Last-Modified | r | r | o | - | o | - | - | - |
| Location | 3rr | | m | m | m | m | m | m |
| Header | Where | Proxy | DES | OPT | STP | PLY | PSE | TRD |

Figure 2: Overview of RTSP Header Fields (A-L) Related to Methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN

| Header | Where | Proxy | DES | OPT | STP | PLY | PSE | TRD |
|---|---|---|---|---|---|---|---|---|
| Media-Properties | r | | - | - | m | o | o | - |
| Media-Range | r | | - | - | c | c | c | - |
| MTag | r | r | o | - | o | - | - | - |
| Pipelined- | G | amd | - | o | o | o | o | o |
| Requests | | r | | | | | | |
| Proxy- | 407 | amr | m | m | m | m | m | m |
| Authenticate | | | | | | | | |
| Proxy- | r | amd | o | o | o | o | o | o/- |
| Authentication- | | r | | | | | | |
| Info | | | | | | | | |
| Proxy- | R | rd | o | o | o | o | o | o |
| Authorization | | | | | | | | |
| Proxy-Require | R | ar | o | o | o | o | o | o |
| Proxy-Require | r | r | c | c | c | c | c | c |
| Proxy-Supported | R | amr | c | c | c | c | c | c |
| Proxy-Supported | r | | c | c | c | c | c | c |
| Public | r | amr | - | m | - | - | - | - |
| Public | 501 | amr | m | m | m | m | m | m |
| Range | R | | - | - | - | o | - | - |
| Range | r | | - | - | c | m | m | - |
| Referrer | R | | o | o | o | o | o | o |
| Request-Status | R | | - | - | - | - | - | - |
| Require | R | | o | o | o | o | o | o |
| Retry-After | 3rr,503 | | o | o | o | o | o | - |
| | ,553 | | | | | | | |
| Retry-After | 413 | | o | - | - | - | - | - |
| RTP-Info | r | | - | - | c | c | - | - |
| Scale | R | r | - | - | - | o | - | - |
| Scale | r | amr | - | - | c | c | c | - |
| Seek-Style | R | | - | - | - | o | - | - |
| Seek-Style | r | | - | - | - | m | - | - |
| Server | R | r | - | o | - | - | - | o |
| Server | r | r | o | o | o | o | o | o |
| Session | R | r | - | o | o | m | m | m |
| Session | r | r | - | c | m | m | m | o |
| Speed | R | admr | - | - | - | o | - | - |
| Speed | r | admr | - | - | - | c | - | - |
| Supported | R | r | o | o | o | o | o | o |
| Supported | r | r | c | c | c | c | c | c |
| Terminate-Reason | R | r | - | - | - | - | - | -/o |
| Timestamp | R | admr | o | o | o | o | o | o |
| Timestamp | c | admr | m | m | m | m | m | m |
| Transport | G | mr | - | - | m | - | - | - |
| Unsupported | r | | c | c | c | c | c | c |
| User-Agent | R | | m* | m* | m* | m* | m* | m* |

| Header | Where | Proxy | DES | OPT | STP | PLY | PSE | TRD |
|---|---|---|---|---|---|---|---|---|
| Via | R | amr | c | c | c | c | c | c |
| Via | r | amr | c | c | c | c | c | c |
| WWW-Authenticate | 401 | | m | m | m | m | m | m |
| Header | Where | Proxy | DES | OPT | STP | PLY | PSE | TRD |

Figure 3: Overview of RTSP Header Fields (M-W) Related to Methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN

| Header | Where | Proxy | GPR | SPR | RDR | PNY |
|---|---|---|---|---|---|---|
| Accept-Credentials | R | rm | o | o | o | - |
| Accept-Encoding | R | r | o | o | o | - |
| Accept-Language | R | r | o | o | o | - |
| Accept-Ranges | G | rm | o | - | - | - |
| Allow | 405 | amr | m | m | m | m |
| Authentication-Info | r | | o/- | o/- | - | - |
| Authorization | R | | o | o | o | - |
| Bandwidth | R | | - | o | - | - |
| Blocksize | R | | - | o | - | - |
| Cache-Control | G | r | o | o | - | - |
| Connection | G | | o | o | o | o |
| Connection-Credentials | 470, 407 | ar | o | o | o | - |
| Content-Base | R | | o | o | - | o |
| Content-Base | r | | o | o | - | - |
| Content-Base | 4xx, 5xx | | o | o | o | o |
| Content-Encoding | R | r | o | o | - | o |
| Content-Encoding | r | r | o | o | - | - |
| Content-Encoding | 4xx, 5xx | r | o | o | o | o |
| Content-Language | R | r | o | o | - | o |
| Content-Language | r | r | o | o | - | - |
| Content-Language | 4xx, 5xx | r | o | o | o | o |
| Content-Length | R | r | * | * | - | * |
| Content-Length | r | r | * | * | - | - |
| Content-Length | 4xx, 5xx | r | * | * | * | * |
| Content-Location | R | | o | o | - | o |
| Content-Location | r | | o | o | - | - |
| Content-Location | 4xx, 5xx | | o | o | o | o |
| Content-Type | R | | * | * | - | * |
| Content-Type | r | | * | * | - | - |
| Content-Type | 4xx, 5xx | | * | * | * | * |
| CSeq | R,c | mr | m | m | m | m |
| Date | R | a | o/* | o/* | m | o/* |
| Date | r | am | o/* | o/* | o/* | o/* |
| Expires | r | r | - | - | - | - |
| From | R | r | o | o | o | - |
| If-Match | R | r | - | - | - | - |
| If-Modified-Since | R | am | o | - | - | - |
| If-None-Match | R | am | o | - | - | - |

| Header | Where | Proxy | GPR | SPR | RDR | PNY |
|---|---|---|---|---|---|---|
| Last-Modified | R | r | - | - | - | - |
| Last-Modified | r | r | o | - | - | - |
| Location | 3rr | | m | m | m | - |
| Location | R | | - | - | m | - |

Figure 4: Overview of RTSP Header Fields (A-L) Related to Methods GET_PARAMETER, SET_PARAMETER, REDIRECT, and PLAY_NOTIFY

| Header | Where | Proxy | GPR | SPR | RDR | PNY |
|---|---|---|---|---|---|---|
| Media-Properties | R | amr | o | - | - | c |
| Media-Properties | r | mr | c | - | - | - |
| Media-Range | R | | o | - | - | c |
| Media-Range | r | | c | - | - | - |
| MTag | r | r | o | - | - | - |
| Notify-Reason | R | | - | - | - | m |
| Pipelined-Requests | R | amdr | o | o | - | - |
| Proxy-Authenticate | 407 | amdr | m | m | m | - |
| Proxy-Authentication-Info | r | amdr | o/- | o/- | - | - |
| Proxy-Authorization | R | amdr | o | o | o | - |
| Proxy-Require | R | ar | o | o | o | - |
| Proxy-Supported | R | amr | c | c | c | - |
| Proxy-Supported | r | | c | c | c | - |
| Public | 501 | admr | m | m | m | - |
| Range | R | | o | - | - | m |
| Range | r | | c | - | - | - |
| Referrer | R | | o | o | o | - |
| Request-Status | R | mr | - | - | - | c |
| Require | R | r | o | o | o | o |
| Retry-After | 3rr,503,<br>553 | | o | o | - | - |
| Retry-After | 413 | | o | o | - | - |
| RTP-Info | R | r | o | - | - | C |
| RTP-Info | r | r | c | - | - | - |
| Scale | G | | c | - | c | c |
| Seek-Style | G | | - | - | - | - |
| Server | R | r | o | o | o | o |
| Server | r | r | o | o | - | - |
| Session | R | r | o | o | o | m |
| Session | r | r | c | c | o | m |
| Speed | G | | - | - | - | - |
| Supported | R | r | o | o | o | - |
| Supported | r | r | c | c | c | - |
| Terminate-Reason | R | r | - | - | m | - |
| Timestamp | R | adrm | o | o | o | o |
| Timestamp | c | adrm | m | m | m | m |
| Transport | G | mr | - | - | - | - |
| Unsupported | r | arm | c | c | c | c |
| User-Agent | R | r | m* | m* | - | - |
| User-Agent | r | r | m* | m* | m* | m* |
| Via | R | amr | c | c | c | c |

| Header | Where | Proxy | GPR | SPR | RDR | PNY |
|---|---|---|---|---|---|---|
| Via | r | amr | c | c | c | c |
| WWW-Authenticate | 401 | | m | m | m | - |

Figure 5: Overview of RTSP Header Fields (M-W) Related to Methods GET_PARAMETER, SET_PARAMETER, REDIRECT, and PLAY_NOTIFY

## 18.1. Accept

The Accept request-header field can be used to specify certain presentation description and parameter media types [RFC6838] that are acceptable for the response to the DESCRIBE request.

See Section 20.2.3 for the syntax.

The asterisk "*" character is used to group media types into ranges, with "*/*" indicating all media types and "type/*" indicating all subtypes of that type. The range MAY include media type parameters that are generally applicable to that range.

Each media type or range MAY be followed by one or more accept-params, beginning with the "q" parameter to indicate a relative quality factor. The first "q" parameter (if any) separates the media type or range's parameters from the accept-params. Quality factors allow the user or user agent to indicate the relative degree of preference for that media type, using the qvalue scale from 0 to 1 (Section 5.3.1 of [RFC7231]). The default value is q=1.

Example of use:

```
Accept: application/example ;q=0.7, application/sdp
```

Indicates that the requesting agent prefers the media type application/sdp through the default 1.0 rating but also accepts the application/example media type with a 0.7 quality rating.

If no Accept header field is present, then it is assumed that the client accepts all media types. If an Accept header field is present, and if the server cannot send a response that is acceptable according to the combined Accept field-value, then the server SHOULD send a 406 (Not Acceptable) response.

## 18.2. 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. It MUST NOT be included in server-to-client requests. See Section 19 for the usage of this header

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 the credentials that the client accepts. The header may contain zero credentials in the first RTSP request to an RTSP server via a proxy when using the "User" method. This is because 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 agent's ASN.1 DER-encoded certificate [RFC5280] in Base64, according to Section 4 of [RFC4648] and where the padding bits are set to zero. All RTSP clients and proxies MUST implement the SHA-256 [FIPS180-4] algorithm for computation of the hash of the DER-encoded certificate. The SHA-256 algorithm is identified by the token "sha-256".

The intention of allowing for other hash algorithms is to enable the future retirement of algorithms that are not implemented somewhere other 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 exists.

Example:

```
Accept-Credentials:User
  "rtsps://proxy2.example.com/";sha-256;exaIl9VMbQMOFGClx5rXnPJKVNI=,
  "rtsps://server.example.com/";sha-256;lurbjj5khhB0NhIuOXtt4bBRH1M=
```

## 18.3. Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but it restricts the content-codings (see Section 18.15), i.e., transformation codings of the message body, such as gzip compression, 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 5.3.1 of [RFC7231], 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, i.e., no transformation at all, 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 that 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.

## 18.4. 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 (response message body) 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 are the same as those defined by [RFC5646].

Each language-range MAY be given an associated quality value that 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 full tag or if it exactly equals a prefix of the tag, i.e., the primary-tag in the ABNF, such that the character following primary-tag 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.

In the process of selecting a language, each language tag is assigned a qualification factor, i.e., if a language being supported by the client is actually supported by the server and what "preference" level the language achieves. The quality value (q-value) of the longest language-range in the field that matches the language tag is assigned as the qualification factor for a particular language tag. If no language-range in the field matches the tag, the language qualification 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 that are assigned a qualification factor greater than 0 are acceptable.

## 18.5. Accept-Ranges

The Accept-Ranges general-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 are acceptable when received in PLAY and PAUSE responses and REDIRECT requests. The server MUST include the header in SETUP responses and 456 (Header Field Not Valid for Resource) 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 is related to. The requester and responder will indicate their capabilities regarding Range formats respectively.

```
Accept-Ranges: npt, smpte, clock
```

The syntax is defined in Section 20.2.3.

## 18.6. Allow

The Allow message body header field lists the methods supported by the resource identified by the Request-URI. The purpose of this field is to inform the recipient of the complete set 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 message body header MUST also be included in SETUP and DESCRIBE responses, if the methods allowed for the resource are different from the complete set of methods defined in this memo.

Example of use:

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

## 18.7. Authentication-Info

The Authentication-Info response-header is used by the server to communicate some information regarding the successful HTTP authentication [RFC7235] in the response message. The definition of the header is in [RFC7615], and any applicable HTTP authentication schemes appear in other RFCs, such as Digest [RFC7616]. This header MUST only be used in response messages related to client to server requests.

## 18.8. Authorization

An RTSP client that wishes to authenticate itself with a server using the authentication mechanism from HTTP [RFC7235], 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. The definition of the header is in

[RFC7235], and any applicable HTTP authentication schemes appear in other RFCs, such as Digest [RFC7616] and Basic [RFC7617]. This header MUST only be used in client-to-server requests.

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). Each client-to-server request MUST be individually authorized by including the Authorization header with the information.

When a shared cache (see Section 16) 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 "max-age" cache 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 max-age.) If the response includes "max-age=0", the proxy MUST always revalidate it before reusing 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 directive, it MAY be returned in reply to any subsequent request.

## 18.9. Bandwidth

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

Clients may not be able to accurately determine the available bandwidth, for example, because the first hop is not a bottleneck. Such a case is when the local area network (LAN) is not the bottleneck, instead the LAN's Internet access link is, if the server is not in the same LAN. Thus, link speeds of WLAN or Ethernet networks are normally not a basis for estimating the available bandwidth. Cellular devices or other devices directly connected to a modem or connection-enabling device may more accurately estimate the bottleneck bandwidth and what is a reasonable share of it for RTSP-controlled media. The client will also need to take into account other traffic sharing the bottleneck. For example, by only assigning a certain fraction to RTSP and its media streams. It is RECOMMENDED that only clients that have accurate and explicit information about bandwidth bottlenecks use this header.

This header is not a substitute for proper congestion control. It is only a method providing an initial estimate and coarsely determines if the selected content can be delivered at all.

Example:

```
Bandwidth: 62360
```

## 18.10. 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 that 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.

## 18.11. 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 DESCRIBE, GET_PARAMETER, SET_PARAMETER, and SETUP request and its response. Note: Cache-Control does not govern only the caching of responses for the RTSP messages, instead it also applies to the media stream identified by the SETUP request. The RTSP requests are generally not cacheable; for further information, see Section 16. Below are the descriptions of the cache directives that can be included in the Cache-Control header.

no-cache: Indicates that the media stream or RTSP response 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 preventing the caching of content.

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

private: Indicates that the media stream or RTSP response 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 or RTSP response. Therefore, if a response includes the no-transform directive, an intermediate cache or proxy MUST NOT change the encoding of the stream or response. 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 or RTSP responses 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 the cache receives this directive, it SHOULD either respond using a cached media stream or response 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 or RTSP response 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 or RTSP response 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 cache 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 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 message body 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 message body with a 200 (OK) response.

## 18.12. Connection

The Connection general-header field allows the sender to specify options that are desired for that particular connection. It 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.

RTSP 2.0 defines the "close" connection option for the sender to signal that the connection will be closed after completion of the response. For example, "Connection: close in either the request or the response-header fields" indicates that the connection SHOULD NOT be considered "persistent" (Section 10.2) after the current request/response is complete.

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 sees it necessary to close the connection. The reason being that the client has the choice of continuing using a connection indefinitely, as long as it sends valid messages.

## 18.13. Connection-Credentials

The Connection-Credentials response-header is used to carry the chain of credentials for any next hop that needs 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 the form of a list of certificates providing the chain of certification) of the next hop to which an intermediary needs to securely connect. The header MUST include the URI of the next hop (proxy or server) and a Base64-encoded (according to Section 4 of [RFC4648] and where the padding bits are set to zero) 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 an NR_CERTS number of 16-bit unsigned integers providing the size, in octets, of each DER-encoded certificate. This is followed by an NR_CERTS number of DER-encoded X.509v3 certificates in a sequence (chain). This format is exemplified in Figure 6. The certificate of the proxy or server 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 that specifies the root certificate authority may optionally be omitted from the chain, under the assumption that the remote end must already possess it in order to validate it in any case.

Example:

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

Where MIIDNTCC... is a Base64 encoding of the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Number of certificates       | Size of certificate #1        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Size of certificate #2        | Size of certificate #3        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
: DER Encoding of Certificate #1                                :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
: DER Encoding of Certificate #2                                :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
: DER Encoding of Certificate #3                                :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Format Example of Connection-Credentials Header Certificate

### 18.14. Content-Base

The Content-Base message body 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 a message body is defined by either 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.

## 18.15. Content-Encoding

The Content-Encoding message body header field is used as a modifier of 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 message body identified by the Request-URI. Typically, the message body is stored with this encoding and is only decoded before rendering or analogous usage. However, an RTSP proxy MAY modify the content-coding if the new coding is known to be acceptable to the recipient, unless the "no-transform" cache directive is present in the message.

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

If the content-coding of a 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, first to last from left to right. Additional information about the encoding parameters MAY be provided by other header fields not defined by this specification.

## 18.16. Content-Language

The Content-Language message body 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 18.4. 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 a message body 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.

## 18.17. Content-Length

The Content-Length message body header field contains the length of the message body of the RTSP message (i.e., after the double CRLF following the last header) in octets of bits. 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.

## 18.18. Content-Location

The Content-Location message body header field MAY be used to supply the resource location for the message body enclosed in the message when that body 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 message body; especially in the case where a resource has multiple variants 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 that is returned.

As an example, if an RTSP client performs a DESCRIBE request on a given resource, e.g., "rtsp://a.example.com/movie/Plan9FromOuterSpace", then the server may use additional information, such as the User-Agent header, to determine the capabilities of the agent. The server will then return a media description tailored to that class of RTSP agents. To indicate which specific description the agent receives, the resource identifier ("rtsp://a.example.com/movie/Plan9FromOuterSpace/FullHD.sdp") is provided in Content-Location, while the description is still a valid response for the generic resource identifier, thus enabling both debugging and cache operation as discussed below.

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 variant 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 variant. This is useful if the RTSP agent desires to verify if the resource variant is current through a conditional request.

A cache cannot assume that a message body 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 variants retrieved from a single requested resource.

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

Note that Content-Location can be used in some cases to derive the base-URI for relative URI(s) present in session description formats. This needs to be taken into account when Content-Location is used. The easiest way to avoid needing to consider that issue is to include the Content-Base whenever the Content-Location is included.

Note also, when using Media Tags in conjunction with Content-Location, it is important that the different versions have different MTags, even if provided under different Content-Location URIs. This is because the different content variants still have been provided in response to the same request URI.

Note also, as in most cases, the URIs used in the DESCRIBE and the SETUP requests are different: the URI provided in a DESCRIBE Content-Location response can't directly be used in a SETUP request. Instead, the steps of deriving the media resource URIs are necessary. This commonly involves combing the media description's relative URIs, e.g., from the SDP's a=control attribute, with the base-URI to create the absolute URIs needed in the SETUP request.

## 18.19. Content-Type

The Content-Type message body 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.

## 18.20. CSeq

The CSeq general-header field specifies the sequence number (integer) for an RTSP request/response pair. This field MUST be present in all requests and responses. RTSP agents maintain a sequence number series for each responder to which they have an open message transport channel. For each new RTSP request an agent originates on a particular RTSP message transport, the CSeq value MUST be incremented by one. The initial sequence number can 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 requests to a server and the server has another when sending requests to the client. Each requester and responder is identified by its socket address (IP address and port number), i.e., per direction of a TCP connection. 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. The RTSP agent receiving requests MUST process the requests arriving on a particular transport in the order of the sequence numbers. Responses are sent in the order that they are generated. The RTSP response MUST have the same sequence number as was present in the corresponding request. An RTSP agent receiving a response MAY receive the responses out of order compared to the order of the requests it sent. Thus, the agent MUST use the sequence number in the response to pair it with the corresponding request.

> The main purpose of the sequence number is to map responses to requests.
>
> The requirement to use a sequence-number increment of one for each new request is to support any future specification of RTSP message transport over a protocol that does not provide in-order delivery or is unreliable.

The above rules relating to the initial sequence number may appear unnecessarily loose. The reason for this is to cater to some common behavior of existing implementations: when using multiple reliable connections in sequence, it may still be easiest to use a single sequence-number series for a client connecting with a particular server. Thus, the initial sequence number may be arbitrary depending on the number of previous requests. For any unreliable transport, a stricter definition or other solution will be required to enable detection of any loss of the first request.

When using multiple sequential transport connections, there is no protocol mechanism to ensure in-order processing as the sequence number is scoped on the individual transport connection and its five tuple. Thus, there are potential issues with opening a new transport connection to the same host for which there already exists a transport connection with outstanding requests and previously dispatched requests related to the same RTSP session.

RTSP Proxies also need to follow the above rules. This implies that proxies that aggregate requests from multiple clients onto a single transport towards a server or a next-hop proxy need to renumber these requests to form a unified sequence on that transport, fulfilling the above rules. A proxy capable of fulfilling some agent's request without emitting its own request (e.g., a caching proxy that fulfills a request from its cache) also causes a need to renumber as the number of received requests with a particular target may not be the same as the number of emitted requests towards that target agent. A proxy that needs to renumber needs to perform the corresponding renumbering back to the original sequence number for any received response before forwarding it back to the originator of the request.

A client connected to a proxy, and using that transport to send requests to multiple servers, creates a situation where it is quite likely to receive the responses out of order. This is because the proxy will establish separate transports from the proxy to the servers on which to forward the client's requests. When the responses arrive from the different servers, they will be forwarded to the client in the order they arrive at the proxy and can be processed, not the order of the client's original sequence numbers. This is intentional to avoid some session's requests being blocked by another server's slow processing of requests.

## 18.21. Date

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

- An RTSP message, sent by either 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 the Network Time Protocol (NTP) [RFC5905], to synchronize its clock with a reliable external standard.

The RTSP-date, a full date as specified by Section 3.3 of [RFC5322], 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 message body is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

> Note: The RTSP 2.0 date format is defined to be the full-date format in RFC 5322. This format is more flexible than the date format in RFC 1123 used by RTSP 1.0. Thus, implementations should use single spaces as separators, as recommended by RFC 5322, and support receiving the obsolete format.

Further, note that the syntax allows for a comment to be added at the end of the date.

## 18.22. Expires

The Expires message body 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 indicates a date and time after which the media stream SHOULD be considered stale.

A stale cache entry should not be returned by a cache (either a proxy cache or a 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 16 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: Wed, 23 Jan 2013 15:36:52 +0000
```

RTSP 2.0 clients and caches MUST treat other invalid date formats, especially those 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 that are more than one year in the future.

## 18.23. From

The From request-header field, if given, SHOULD contain an Internet email 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 email address in this field MAY be separate from the Internet host that 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.

## 18.24. If-Match

The If-Match request-header field is especially useful for ensuring the integrity of the presentation description, independent of how the presentation description was received. The presentation description can be fetched via means external to RTSP (such as HTTP) or via the DESCRIBE message. In the case of retrieving the presentation description via RTSP, 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 an If-Match header part of the SETUP request, the client ensures that resources set up are matching the description. A SETUP request with the If-Match header for which the MTag validation check fails MUST generate a response using 412 (Precondition Failed).

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

## 18.25. If-Modified-Since

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

An example of the field is:

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

## 18.26. 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 16.1.3 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 16.1.4 for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)

The result of a request having both an If-None-Match header field and an If-Match header field is unspecified and MUST be considered an illegal request.

## 18.27. Last-Modified

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

An origin server MUST NOT send a Last-Modified date that 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 message body 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 message body's modification time, especially if the message body changes near the time that the response is generated.

RTSP servers SHOULD send Last-Modified whenever feasible.

## 18.28. 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 3rr 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 18.18) differs from Location in that the Content-Location identifies the original location of the message body enclosed in the request. Therefore, it is possible for a response to contain header fields for both Location and Content-Location. Also, see Section 16.2 for cache requirements of some methods.

## 18.29. Media-Properties

This general-header is used in SETUP responses 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 taking effect. 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 are the ones applicable to all media in the RTSP session. 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 is enforced within the header. Property values should be placed 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 four 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 lifetime of the RTSP session.

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

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

Retention:

Unlimited: Content will be retained for the duration of the lifetime 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 4.4.3.

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. The value 0.0 is a valid 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 in arbitrary order. A range has a start and stop value separated by a colon. A range indicates that the content supports a fine-grained selection of scale values. Fine-graining allows for steps at least as small as one tenth of a scale value. Content is considered to support fine-grained selection when the server in response to a given scale value can produce content with an actual scale that is less than one tenth of scale unit, i.e., 0.1, from the requested value. Negative values are supported. The value 0 has no meaning and MUST NOT be used.

Examples of this header for on-demand content and a live stream without recording are:

```
On-demand:
Media-Properties: Random-Access=2.5, Unlimited, Immutable,
     Scales="-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
```

## 18.30. 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 the SETUP response, PLAY and PAUSE responses for media that are time-progressing, PLAY and PAUSE responses after any change for media that are Dynamic, and in PLAY_NOTIFY requests that are sent due to Media-Property-Update. A Media-Range header without any range specifications MAY be included in GET_PARAMETER requests to the server to request the current range. In this case, the server MUST 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 18.5) 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. The range specifications SHALL be ordered with the range with the lowest value or earliest start time first, followed by ranges with increasingly higher values or later start time.

For media that has the time-progressing property, the Media-Range header values will only be valid for the particular point in time when it was issued. As the wallclock progresses, so will the media range. However, it shall be assumed that media time progresses 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.

## 18.31. MTag

The MTag response-header MAY be included in DESCRIBE, GET_PARAMETER, or SETUP responses. The message body tags (Section 4.6) returned in a DESCRIBE response and the one in SETUP refer 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 be 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 18.24) and If-None-Match (Section 18.26).

## 18.32. Notify-Reason

The Notify-Reason response-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).

## 18.33. Pipelined-Requests

The Pipelined-Requests general-header is used to indicate that a request is to be executed in the context created by a previous request(s). 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 agent. 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 agent MUST look up if there exists a binding between this Pipelined-Requests identifier for the current persistent connection and an RTSP session ID. If the binding exists, then the received request is processed the same way as if it contained the Session header with the found session ID. If there does not exist a mapping and no Session header is included in the request, the responding agent MUST create a binding upon the successful completion of a session creating request, i.e., SETUP. A binding MUST NOT be created, if the request failed to create an RTSP session. In case the request contains both a Session header and the Pipelined-Requests header, the Pipelined-Requests header MUST be ignored.

Note: Based on the above definition, at least the first request containing a new unique Pipelined-Requests header 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 requests of any type using the RTSP session context may include the Pipelined-Requests header.

When responding to any request that contained the Pipelined-Requests header, the server MUST also include the Session header when a binding to a session context exists. An RTSP agent that knows the session identifier SHOULD NOT use the Pipelined-Requests header in any request and only use the Session header. This as the Session identifier is persistent across transport contexts, like TCP connections, which the Pipelined-Requests identifier is not.

The RTSP agent sending the request with a Pipelined-Requests header has the responsibility for using a unique and previously unused identifier within the transport context. Currently, only a TCP connection is defined as such a transport context. A server MUST delete the Pipelined-Requests identifier and its binding to a session upon the termination of that session. Despite the previous mandate, RTSP agents are RECOMMENDED not to reuse identifiers to allow for better error handling and logging.

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

### 18.34. 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 definition of the header is in [RFC7235], and any applicable HTTP authentication schemes appear in other RFCs, such as Digest [RFC7616] and Basic [RFC7617].

The HTTP access authentication process is described in [RFC7235]. This header MUST only be used in response messages related to client-to-server requests.

### 18.35. Proxy-Authentication-Info

The Proxy-Authentication-Info response-header is used by the proxy to communicate some information regarding the successful authentication to the proxy in the message response in some authentication schemes, such as the Digest scheme [RFC7616]. The definition of the header is in [RFC7615], and any applicable HTTP authentication schemes appear in other RFCs. This header MUST only be used in response messages related to client-to-server requests. This header has hop-by-hop scope.

### 18.36. Proxy-Authorization

The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy that requires authentication. The Proxy-Authorization field-value consists of credentials containing the authentication information of the user agent for the proxy or realm of the resource being requested. The definition of the header is in [RFC7235], and any applicable HTTP authentication schemes appear in other RFCs, such as Digest [RFC7616] and Basic [RFC7617].

The HTTP access authentication process is described in [RFC7235]. Unlike Authorization, the Proxy-Authorization header field applies only to the next-hop proxy. This header MUST only be used in client-to-server requests.

## 18.37. 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 endpoint (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 18.43 for more details on the mechanics of this message and a usage example. See discussion in the proxies section (Section 15.1) about when to consider that a feature requires proxy support.

Example of use:

```
Proxy-Require: play.basic
```

## 18.38. Proxy-Supported

The Proxy-Supported general-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 copy the received Proxy-Supported header in the response.

The Proxy-Supported header field contains a list of feature tags applicable to proxies, as described in Section 4.5. The list is 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 MUST check the list and remove any feature tag(s) it does not support. A Proxy-Supported header present in the response MUST NOT be modified by the proxies. These feature tags are the ones the proxy chains support in general and are not specific to the request resource.

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
```

## 18.39. 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 18.6) 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.

## 18.40. Range

The Range general-header specifies a time range in PLAY (Section 13.4), PAUSE (Section 13.6), SETUP (Section 13.3), and PLAY_NOTIFY (Section 13.5) requests and responses. It MAY be included in GET_PARAMETER requests from the client to the server with only a Range format and no value to request the current media position, whether the session is in Play or Ready state in the included format. The server SHALL, if supporting the 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 the Range header, like scale value other than 1, that fact is also required to be included in any GET_PARAMETER response by including the Scale header to provide complete information.

The range can be specified in a number of units. This specification defines smpte (Section 4.4.1), npt (Section 4.4.2), and clock (Section 4.4.3) range units. While octet ranges (Byte Ranges) (see Section 2.1 of [RFC7233]) 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 the 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 points is not allowed in the same Range header; however, the omission of the Range header has that meaning, i.e., from pause point (or start) until end of content.

As noted, Range headers define 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 timescales' "now" and an implicit start value. Implicit values reference the current pause-point, while "now" is the current 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 18.46) 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 is 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 value in the Scale header is not valid.

## 18.41. Referrer

The Referrer 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 URI refers to that of the presentation description, typically retrieved via HTTP. The Referrer 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 Referrer 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 identifier.

Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referrer field is sent. For example, a streaming client could have a toggle switch for openly/anonymously, which would respectively enable/disable the sending of Referrer and From information.

Clients SHOULD NOT include a Referrer header field in an (non-secure) RTSP request if the referring page was transferred with a secure protocol.

## 18.42. Request-Status

This request-header is used to indicate the end result for requests that take time to complete, such as 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 it reports on using the CSeq number and the Session ID used in the request reported on. This is not ensured to be unambiguous due to the fact that the CSeq number is scoped by the transport connection. Agents originating requests can reduce the issue by using a monotonically increasing counter across all sequential transports used. The header 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"

Proxies that renumber the CSeq header need to perform corresponding remapping of the cseq parameter in this header when forwarding the request to the next-hop agent.

## 18.43. Require

The Require request-header field is used by agents to ensure that the other endpoint supports features that are required in respect to this request. It can also be used to query if the other endpoint supports certain features; however, the use of the Supported general-header (Section 18.51) is much more effective in this purpose. In case any of the feature tags listed by the Require header are not supported by the server or client receiving the request, it MUST respond to the request using the error code 551 (Option Not Supported) and include the Unsupported header listing those feature tags that are NOT supported. This header does not apply to proxies; for the same functionality with respect to proxies, see the Proxy-Require header (Section 18.37) 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 a server, do need to understand also the server's 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 that 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 18.37). See discussion in the proxies section (Section 15.1) about when to consider that a feature requires proxy support.

## 18.44. Retry-After

The Retry-After response-header field can be used with a 503 (Service Unavailable) or 553 (Proxy 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 3rr (Redirection) response to indicate the minimum time the user agent is asked to wait before issuing the redirected request. A response using 413 (Request Message Body Too Large) when the restriction is temporary MAY also include the Retry-After header. 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: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120
```

In the latter example, the delay is 2 minutes.

## 18.45. RTP-Info

The RTP-Info general-header field is used to set RTP-specific parameters in the PLAY and GET_PARAMETER responses or PLAY_NOTIFY and GET_PARAMETER requests. 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 a client needing 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 its arrival. Also, functionality that informs 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 and the Range header MAY be included in a GET_PARAMETER request from client to server without any values to request the current playback point and corresponding RTP synchronization information. When the RTP-Info header is included in a Request, the Range header MUST also be included. The server response SHALL include both the Range header and the RTP-Info header. If the session is in Play state, then the value of the Range header SHALL be filled in with the current playback point and with the corresponding RTP-Info values. If the server is in another state, no values are included in the RTP-Info header. The header is included in PLAY_NOTIFY requests with the Notify-Reason of the end of stream to provide RTP information about the end of the stream.

The header can carry the following parameters:

url: Indicates the stream URI for which the following RTP parameters correspond; this URI MUST be the same as 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 SSRC to which the RTP timestamp and sequence number provided applies. 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 may receive 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 exists 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 format 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 16 kHz RTP timestamp clock and Video a 90 kHz 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

## 18.46. Scale

The Scale general-header indicates the requested or used view rate for the media resource being played back. 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 value of 2 indicates twice the normal viewing rate ("fast forward") and a value of 0.5 indicates half the normal viewing rate. In other words, a value of 2 has content time increase at twice the playback time. For every second of elapsed (wallclock) time, 2 seconds of content time will be delivered. A negative value indicates reverse direction. For certain media transports, this may require certain considerations to work consistently; 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 bitrate should be reasonably close to the nominal bitrate 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 frames. For audio, 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 18.29). The client SHOULD only indicate request values 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, a non-supported value in a request does not generate an error, it 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 18.40.

Example of playing in reverse at 3.5 times normal rate:

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

### 18.47. 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 the 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 general-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 the 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. A client receiving an unknown policy MUST ignore it and use the Range header and any media synchronization information as basis to determine what the server did.

This specification defines the following seek policies that may be requested (see also Section 4.7.1):

RAP: Random Access Point (RAP) is the behavior of requesting the server to locate the closest previous random access point that exists 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.

CoRAP: Conditional Random Access Point (CoRAP) is a variant of the above RAP behavior. This policy is primarily intended for cases where there is larger distance between the random access points in the media. CoRAP uses the RAP policy if the condition that there is a Random Access Point closer to the requested start point than to the current pause point is fulfilled. Otherwise, no seeking is performed and playback will continue from the current pause point. This policy assumes that the media state existing prior to the pause is usable if delivery is continued. If the client or server knows that this is not the fact, the RAP policy should be used. In other words, in most cases when the client requests a start point prior to the current pause point, a valid decoding dependency chain from the media delivered prior to the pause and to the requested media unit will not exist. If the server searched to a random access point, the server MUST return the CoRAP policy in the Seek-Style header and adjust the Range header to reflect the position of the selected RAP. In case the random access point is farther away and the server chooses to continue from the current pause point, it MUST include the "Next" policy in the Seek-Style header and adjust the Range header start point to the current pause point.

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 rendered 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 and for discrete media, the first unit that is to be rendered after the provided time. The main usage 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 uninterrupted media playback can be achieved. An example of such a scenario would be 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.

## 18.48. Server

The Server general-header field contains information about the software used by the origin server to create or handle the request. This 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 18.57). If the response is generated by the proxy, the proxy application MUST return the Server response-header as previously returned by the server.

## 18.49. Session

The Session general-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 exists until destroyed by a TEARDOWN or a REDIRECT or is 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, PLAY_NOTIFY and TEARDOWN. It MAY be included in SETUP, OPTIONS, SET_PARAMETER, GET_PARAMETER, and REDIRECT. It MUST NOT be included in DESCRIBE. The Session header MUST NOT be included in the following methods, if these requests are pipelined and if the session identifier is not yet known: PLAY, PAUSE, TEARDOWN, SETUP, OPTIONS SET_PARAMETER, and GET_PARAMETER.

In an RTSP response, the session header MUST be included in methods, SETUP, PLAY, PAUSE, and PLAY_NOTIFY, and it MAY be included in methods TEARDOWN and REDIRECT. If included in the request of the following methods it MUST also be included in the response: OPTIONS, GET_PARAMETER, and SET_PARAMETER. It MUST NOT be included in DESCRIBE responses.

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 as to which URIs 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 a parameter for 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.

> The 60-second value was chosen as the session timeout value as it results in keep-alive messages that are not too frequent and 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. The 60-second value also allows for reasonably rapid recovery of committed server resources in case of client failure.

## 18.50. Speed

The Speed general-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 a lower and upper 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.speed. 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. However, note 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, and 2.5-2.5 all have their usages.

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. Implementers should keep in mind that bandwidth for the session may be negotiated beforehand (by means other than RTSP) and, therefore, renegotiation may be necessary. To perform Speed operations, the server needs to ensure that the network path can support the resulting bitrate. Thus, the media transport needs to support feedback so that the server can react and adapt to the available bitrate.

## 18.51. Supported

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

The Supported header contains a list of feature tags, described in Section 4.5, that are understood by the client or server. These feature tags are the ones the server or client supports in general and are not specific to the request resource.

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
```

## 18.52. Terminate-Reason

The Terminate-Reason request-header allows the server, when sending a REDIRECT or TEARDOWN 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 shut down for some administrative reason.

Session-Timeout: A client's session has been 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 providing any service.

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

## 18.53. 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 can be used by the agent to compute the round-trip time to the responding agent so that it can adjust the timeout value for retransmissions when running over an unreliable protocol. It also resolves retransmission ambiguities for unreliable transport of RTSP.

Note that the present specification provides only for reliable transport of RTSP messages. The Timestamp general-header is specified in case the protocol is extended in the future to use unreliable transport.

## 18.54. Transport

The Transport general-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 and listed in decreasing order of preference. Each transport specification consists of a transport protocol identifier, followed by any number of parameters separated by semicolons. A Transport request-header MAY contain multiple transport specifications using the same transport protocol identifier. The server MUST return a Transport response-header in the response to indicate the values actually chosen, if any. If no transport specification is supported, no transport header is returned and the response MUST use the status code 461 (Unsupported Transport) (Section 17.4.25). In case more than one transport specification was present in the request, the server MUST return the single transport specification (transport-spec) that was actually chosen, if any. The number of transport-spec entries is expected to be limited as the client will receive guidance on what configurations 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.

The transport protocol identifier defines, for each transport specification, which transport protocol to use and any related rules. Each transport protocol identifier defines the parameters that are required to occur; additional optional parameters MAY occur. This flexibility is provided as parameters may be different and provide different options to the RTSP agent. A transport specification may only contain one of any given parameter within it. A parameter consists of a name and optionally a value string. Parameters MAY be given in any order. Additionally, a transport specification may only contain either the unicast or the multicast transport type parameter. The transport protocol identifier, and all parameters, need to be understood in a transport specification; if not, the transport specification MUST be ignored. An RTSP proxy 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 results in no remaining transport specification, the proxy SHALL send a 461 (Unsupported Transport) (Section 17.4.25) 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 protocol identifier is a list of slash-separated tokens:

Value1/Value2/Value3...

Which, for RTP transports, takes 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 the same address from which the RTSP messages originates.

The choice of method for indicating where the media is to be delivered depends on the use case. In some cases, 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 are 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 need to be addressed because an RTSP server can be used as a DoS attacker on an 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=" lines. 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 need 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 parameter per transport spec is intended. The usage of multiple destinations 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 as 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 DoS attack, a server MUST perform security checks (see Section 21.2.1) 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 a 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 it is desired to specify only the destination port. Responses to requests 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 that is already present 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 an address other than the one RTSP messages are sent to. This will allow the client to verify the 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 the sending and receiving of the media stream's data packets. The main reasons are threefold: First, indicating the port and source address(s) lets the receiver know where from the packets is expected to originate. Second, traversal of NATs is greatly simplified when traffic is flowing symmetrically over a NAT binding. Third, certain NAT traversal mechanisms need to know to which address and port to send so-called "binding packets" from the receiver to the sender, thus creating an 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. The currently defined valid value is "PLAY". If not provided, the default is "PLAY". The "RECORD" value was defined in RFC 2326; in this specification, it is unspecified but reserved. RECORD and other values may be specified in the future.

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 $ block (see Section 14) and MUST be present. This parameter MAY be specified as an 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 is 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 channels are bidirectional, 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 send 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.

MIKEY: This parameter is used in conjunction with transport specifications that can utilize MIKEY [RFC3830] for security context establishment. So far, only the SRTP-based RTP profiles SAVP and SAVPF can utilize MIKEY, and this is defined in Appendix C.1.4.1. This parameter can be included both in request and response messages. The binary MIKEY message SHALL be Base64-encoded [RFC4648] before being included in the value part of the parameter, where the encoding adheres to the definition in Section 4 of RFC 4648 and where the padding bits are set to zero.

Multicast-specific:

ttl: multicast time-to-live for IPv4. When included in requests, the value indicates the TTL value that the client requests 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 functions are not needed for IPv6 as the scoping is part of the IPv6 multicast address [RFC4291].

RTP-specific:

These parameters 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. The values are expressed as a slash-separated sequence of SSRC values, each SSRC 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 [RFC3550]. If the parameter is included in the Transport header of a SETUP request, the server SHOULD ignore it, and choose appropriate SSRCs for the stream. The server SHOULD set the ssrc parameter in the Transport header of the response.

RTCP-mux: Used to negotiate the usage of RTP and RTCP multiplexing [RFC5761] on a single underlying transport stream/flow. The presence of this parameter in a SETUP request indicates the client's support and requires the server to use RTP and RTCP multiplexing. The client SHALL only include one transport stream in the Transport header specification. To provide the server with a choice between using RTP/RTCP multiplexing or not, two different transport header specifications must be included.

The parameter setup and connection 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 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]. The use of this parameter in RTP/AVP/TCP non-interleaved transport is discussed 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 connection parameter in a transport specification part of the Transport header in a SETUP request to indicate the client's preference for either reusing an existing connection between client and server (in which case the client sets the "connection" parameter to "existing") or requesting the creation of a new connection between client and server (in which cast the client sets the "connection" parameter to "new"). Typically, clients use the "new" value for the first SETUP request for a URL, and "existing" for subsequent SETUP requests for a URL.

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

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

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

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

```
  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, clock
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 302
        Date: Fri, 20 Dec 2013 10:20:32 +0000
        Session: rQi1hBrGlFdiYld241FxUO
        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
```

## 18.55. Unsupported

The Unsupported response-header lists the features not supported by the responding RTSP agent. In the case where the feature was specified via the Proxy-Require field (Section 18.37), 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 18.43 for a usage example.

## 18.56. User-Agent

The User-Agent general-header field contains information about the user agent originating the request or producing a response. 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
```

## 18.57. 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.

Each of multiple values in the Via field 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. So messages originating with the client or server do not include the Via header. The first proxy or other intermediate adds the header and its information into the field. Any additional intermediate adds additional field-values. Resulting in the server seeing the chains of intermediates in a client-to-server request and the client seeing the full chain in the response message.

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 that have different received-protocol values.

### 18.58. WWW-Authenticate

The WWW-Authenticate header is specified in [RFC7235]; its usage depends on the used authentication schemes, such as Digest [RFC7616] and Basic [RFC7617]. 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. This header MUST only be used in response messages related to client to server requests.

The HTTP access authentication process is described in [RFC7235] with some clarification in Section 19.1. 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.

## 19. Security Framework

The RTSP security framework consists of two high-level components: the pure authentication mechanisms based on HTTP authentication and the message 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 reused to a large extent.

### 19.1. RTSP and HTTP Authentication

RTSP and HTTP share common authentication schemes; thus, they follow the same framework as specified in [RFC7235]. RTSP uses the corresponding RTSP error codes (401 and 407) and headers (WWW-Authenticate, Authorization, Proxy-Authenticate, Proxy-Authorization) by importing the definitions from [RFC7235]. Servers SHOULD implement both the Basic [RFC7617] and the Digest [RFC7616] authentication schemes. Clients MUST implement both the Basic and the Digest authentication schemes so that a server that requires the client to authenticate can trust that the capability is present. If implementing the Digest authentication scheme, then the additional considerations specified below in Section 19.1.1 MUST be followed.

It should be stressed that using the HTTP authentication alone does not provide full RTSP message security. Therefore, TLS SHOULD be used; see Section 19.2. Any RTSP message containing an Authorization header using the Basic authentication scheme MUST be using a TLS connection with confidentiality protection enabled, i.e., no NULL encryption.

In cases where there is a chain of proxies between the client and the server, each proxy may individually request the client or previous proxy to authenticate itself. This is done using the Proxy-Authenticate (Section 18.34), the Proxy-Authorization (Section 18.36), and the Proxy-Authentication-Info (Section 18.35) headers. These headers are hop-by-hop headers and are only scoped to the current connection and hop. Thus, if a proxy chain exists, a proxy connecting to another proxy will have to act as a client to authorize itself towards the next proxy. The WWW-Authenticate (Section 18.58), Authorization (Section 18.8), and Authentication-Info (Section 18.7) headers are end-to-end and MUST NOT be modified by proxies.

This authentication mechanism works only for client-to-server requests as currently defined. This leaves server-to-client request outside of the context of TLS-based communication more vulnerable to message-injection attacks on the client. Based on the server-to-client methods that exist, the potential risks are various: hijacking (REDIRECT), denial of service (TEARDOWN and PLAY_NOTIFY), or attacks with uncertain results (SET_PARAMETER).

### 19.1.1. Digest Authentication

This section describes the modifications and clarifications required to apply the HTTP Digest authentication scheme to RTSP. The RTSP scheme usage is almost completely identical to that for HTTP [RFC7616]. These modifications are based on the procedures defined for SIP 2.0 [RFC3261] (in Section 22.4) but updated to use RFC 7235, RFC 7616 and RFC 7615 instead of RFC 2617.

Digest authentication uses two additional headers, Authentication-Info and Proxy-Authentication-Info, that are defined as in [RFC7615]. The rules for Digest authentication follow those defined in [RFC7616], with "HTTP/1.1" replaced by "RTSP/2.0" in addition to the following differences:

1. Use the ABNF specified in the referenced documents, with the difference that the URI parameter uses the request URI format for RTSP, i.e. the ABNF element: Request-URI (see Section 20.2.1). The domain parameter uses the RTSP-URI-Ref element for absolute and relative URIs.

2. If MTags are used, then the example procedure for choosing a nonce based on ETag can work, based on replacing the ETag with the MTag.

3. As a clarification to the calculation of the A2 value for message integrity assurance in the Digest authentication scheme, implementers should assume, when the entity-body is empty (that is, when the RTSP messages have no message body) that the hash of the message body resolves to the hash of an empty string, or: H(entity-body), example MD5("") = "d41d8cd98f00b204e9800998ecf8427e".

### 19.2. RTSP over TLS

RTSP agents MUST implement RTSP over TLS as defined in this section and the next Section 19.3. RTSP MUST follow the same guidelines with regard to TLS [RFC5246] usage as specified for HTTP; see [RFC2818]. RTSP over TLS is separated from unsecured RTSP both on the URI level and the 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 set up 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 ways TLS is used is to provide only 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.

There exists a potential security vulnerability when reusing TCP and TLS state for different resources (URIs). If two different hostnames point at the same IP address, it can be desirable to reuse the TCP/TLS connection to that server. In that case, the RTSP agent having the TCP/TLS connection MUST verify that the server certificate associated with the connection has a SubjectAltName matching the hostname present in the URI for the resource an RTSP request is to be issued.

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 one. 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 cannot be identified based on certificates. The client needs to select between using the procedure specified below or using a TLS connection directly (bypassing any proxies) to the server. The choice may be dependent on policies.

In general, there are 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). This memo specifies only non-transparent RTSP proxies, i.e., proxies visible to the RTSP client and RTSP server. An infrastructure based on proxies requires that the trust model be such that both client and server can trust the proxies to handle the RTSP messages correctly. To be able to trust a proxy, the client and server also need 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 the transport layer). In the rest of this section, any reference to "proxy" will be to a non-transparent proxy, which inspects or manipulates 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 should have the possibility to select the desired behavior. To handle this case, the Accept-Credentials header (see Section 18.2) is used, where the client can request the proxy or proxies to relay back the chain of certificates used to authenticate any intermediate proxies as well as the server. The assumption that the proxies are viewed as trusted gives the user a possibility to enforce policies on each trusted proxy of whether it should accept the next agent in the chain. However, it should be noted that not all deployments will return the chain of certificates used to authenticate any intermediate proxies as well as the server. An operator of such a deployment may want to hide its topology from the client. It should be noted well that the client does not have any insight into the proxy's operation. Even if the proxy is trusted, it can still return an incomplete chain of certificates.

A proxy MUST use TLS for the next hop if the RTSP request includes an "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 are not required to use it. The chain of proxies used by a client to reach a server and its TLS sessions MUST have commensurate security. Therefore, a proxy MUST, when initiating the next-hop TLS connection, use the incoming TLS connections cipher-suite list, only modified by removing any cipher suites 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: With "any", the proxy (or proxies) MUST accept whatever certificate is presented. Of course, this is not a recommended option to use, but it may be useful in certain circumstances (such as testing).

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

User: For the "user" method, the proxy (or proxies) MUST send credential information about the next hop to the client for authorization. The client can then decide whether or not the proxy should accept the certificate. 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 a method other than the "Proxy" is to be used, then the Accept-Credentials header MUST be included in all of the RTSP requests from the client. This is because it cannot be assumed that the proxy always keeps the TLS state or the user's 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 each method. If the policy does not accept the credentials of the next hop, the proxy 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-Credentials 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 exists at the time when the server desires to send the request, the server MUST discard the message.

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

### 19.3.2. User-Approved TLS Procedure

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

- o Set up 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 are 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 18.13, with the next hop's identity and certificate.

The client MUST upon receiving a 470 (Connection Authorization Required) or 407 (Proxy Authentication Required) response with Connection-Credentials header take the decision on whether or not to accept the certificate (if it cannot do so, the user SHOULD be consulted). Using IP addresses in the next-hop URI and certificates rather than domain names makes it very difficult for a user to determine whether or not it should approve the next hop. Proxies are RECOMMENDED to use domain names to identify themselves in URIs and in the certificates. 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, clock
      Accept-Credentials: User

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

C->P: SETUP rtsps://test.example.org/secret/audio RTSP/2.0
      CSeq: 3
      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;
      dPYD7txpoGTbAqZZQJ+vaeOkyH4=
      Accept-Ranges: npt, smpte, clock

P->S: SETUP rtsps://test.example.org/secret/audio RTSP/2.0
      CSeq: 3
      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;
      dPYD7txpoGTbAqZZQJ+vaeOkyH4=
      Accept-Ranges: npt, smpte, clock
```

One implication of this process is that the connection for secured RTSP messages may take significantly more round-trip times for the first message. A 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, if each message contains the chain of proxies that the requester accepts, the remaining message exchange should not be delayed. 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.

The RTSP syntax makes use of the ISO 10646 character set in UTF-8 encoding [RFC3629].

## 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 whitespace, including folding, has the same semantics as SP. A recipient MAY replace any linear whitespace with a single SP before interpreting the field-value or forwarding the message downstream. The SWS construct is used when linear whitespace 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)
BACKSLASH       =  %x5C  ; US-ASCII backslash (92)
CRLF            =  CR LF
LWS             =  [CRLF] 1*( SP / HT ) ; Line-breaking whitespace
SWS             =  [LWS] ; Separating whitespace
HCOLON          =  *( SP / HT ) ":" SWS
TEXT            =  %x20-7E / %x80-FF  ; any OCTET except CTLs
tspecials       =  "(" / ")" / "<" / ">" / "@"
                /  "," / ";" / ":" / BACKSLASH / DQUOTE
                /  "/" / "[" / "]" / "?" / "="
                /  "{" / "}" / 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 tspecials>
quoted-string   =  ( DQUOTE *qdtext DQUOTE )
```

```
   qdtext          = %x20-21 / %x23-5B / %x5D-7E / quoted-pair
                   / UTF8-NONASCII
                   ; No DQUOTE and no "\"
   quoted-pair     = "\\" / ( "\" DQUOTE )
   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 /  "a" / "b" / "c" / "d" / "e" / "f"
                      ; lowercase "a-f" Hex
   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 / "+" / "/"
   SLASH     =  SWS "/" SWS ; slash
   EQUAL     =  SWS "=" SWS ; equal
   LPAREN    =  SWS "(" SWS ; left parenthesis
   RPAREN    =  SWS ")" SWS ; right parenthesis
   COMMA     =  SWS "," SWS ; comma
   SEMI      =  SWS ";" SWS ; semicolon
   COLON     =  SWS ":" SWS ; colon
   MINUS     =  SWS "-" SWS ; minus/dash
   LDQUOT    =  SWS DQUOTE ; open double quotation mark
   RDQUOT    =  DQUOTE SWS ; close double quotation mark
   RAQUOT    =  ">" SWS ; right angle quote
   LAQUOT    =  SWS "<" ; left angle quote

   TEXT-UTF8char    =  %x21-7E / UTF8-NONASCII
   UTF8-NONASCII    = UTF8-2 / UTF8-3 / UTF8-4
   UTF8-1           = <As defined in RFC 3629>
   UTF8-2           = <As defined in RFC 3629>
   UTF8-3           = <As defined in RFC 3629>
   UTF8-4           = <As defined in RFC 3629>
   UTF8-tail        = <As defined in RFC 3629>
```

```
   POS-FLOAT         = 1*12DIGIT ["." 1*9DIGIT]
   FLOAT             = ["-"] POS-FLOAT
```

## 20.2. RTSP Protocol Definition

### 20.2.1. Generic Protocol Elements

```
   RTSP-IRI        =  schemes ":" IRI-rest
   IRI-rest        =  ihier-part [ "?" iquery ]
   ihier-part      =  "//" iauthority ipath-abempty
   RTSP-IRI-ref    =  RTSP-IRI / irelative-ref
   irelative-ref   =  irelative-part [ "?" iquery ]
   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 ":"
                       ; No parameter (; delimited) inside path.

   ipchar          =  iunreserved / pct-encoded / sub-delims / ":" / "@"
   ipchar-nc       =  iunreserved / pct-encoded / sub-delims / "@"
                      ; sub-delims is different from RFC 3987
                      ; not including ";"

   iquery          =  < 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 ]
URI-req-rest   =  hier-part [ "?" query ]
                  ; Note fragment part not allowed in requests
hier-part      =  "//" authority path-abempty

RTSP-Relative  =  relative-part [ "?" query ]
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>

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 ":"
                  ; No parameter (; delimited) inside path.

pchar          =  unreserved / pct-encoded / sub-delims / ":" / "@"
pchar-nc       =  unreserved / pct-encoded / sub-delims / "@"

sub-delims     =  "!" / "$" / "&" / "'" / "(" / ")"
                  / "*" / "+" / "," / "="
                  ; sub-delims is different from RFC 3986/3987
                  ; not including ";"
```

```
   smpte-range        =  smpte-type [EQUAL smpte-range-spec]
                         ; See section 4.4
   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" [EQUAL npt-range-spec]
   npt-range-spec   =  ( npt-time "-" [ npt-time ] ) / ( "-" npt-time )
   npt-time         =  "now" / npt-sec / npt-hhmmss / npt-hhmmss-comp
   npt-sec          =  1*19DIGIT [ "." 1*9DIGIT ]
   npt-hhmmss       =  npt-hh ":" npt-mm ":" npt-ss [ "." 1*9DIGIT ]
   npt-hh           =  2*19DIGIT   ; any positive number
   npt-mm           =  2*2DIGIT  ; 0-59
   npt-ss           =  2*2DIGIT  ; 0-59
   npt-hhmmss-comp  =  npt-hh-comp ":" npt-mm-comp ":" npt-ss-comp
                       [ "." 1*9DIGIT ] ; Compatibility format
   npt-hh-comp      =  1*19DIGIT   ; any positive number
   npt-mm-comp      =  1*2DIGIT  ; 0-59
   npt-ss-comp      =  1*2DIGIT  ; 0-59

   utc-range        =  "clock" [EQUAL 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 [ "." 1*9DIGIT ]

   feature-tag       =  token

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

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

### 20.2.2. Message Syntax

```
   RTSP-message  = Request / Response  ; RTSP/2.0 messages

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

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

   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-data = 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
                 /   "403"  ; Forbidden
                 /   "404"  ; Not Found
                 /   "405"  ; Method Not Allowed
                 /   "406"  ; Not Acceptable
                 /   "407"  ; Proxy Authentication Required
                 /   "408"  ; Request Timeout
                 /   "410"  ; Gone
                 /   "412"  ; Precondition Failed
                 /   "413"  ; Request Message Body Too Large
                 /   "414"  ; Request-URI Too Long
                 /   "415"  ; Unsupported Media Type
                 /   "451"  ; Parameter Not Understood
                 /   "452"  ; reserved
                 /   "453"  ; Not Enough Bandwidth
                 /   "454"  ; Session Not Found
                 /   "455"  ; Method Not Valid In This State
                 /   "456"  ; Header Field Not Valid for Resource
                 /   "457"  ; Invalid Range
                 /   "458"  ; Parameter Is Read-Only
                 /   "459"  ; Aggregate Operation Not Allowed
                 /   "460"  ; Only Aggregate Operation Allowed
                 /   "461"  ; Unsupported Transport
                 /   "462"  ; Destination Unreachable
                 /   "463"  ; Destination Prohibited
                 /   "464"  ; Data Transport Not Ready Yet
                 /   "465"  ; Notification Reason Unknown
                 /   "466"  ; Key Management Error
                 /   "470"  ; Connection Authorization Required
                 /   "471"  ; Connection Credentials Not Accepted
                 /   "472"  ; Failure to Establish Secure Connection
                 /   "500"  ; Internal Server Error
                 /   "501"  ; Not Implemented
                 /   "502"  ; Bad Gateway
                 /   "503"  ; Service Unavailable
                 /   "504"  ; Gateway Timeout
                 /   "505"  ; RTSP Version Not Supported
                 /   "551"  ; Option Not Supported
                 /   "553"  ; Proxy Unavailable
                 /   extension-code

   extension-code  =  3DIGIT

   Reason-Phrase   =  1*(TEXT-UTF8char / HT / SP)
```

```
   rtsp-header      = general-header
                    / request-header
                    / response-header
                    / message-body-header

   general-header   =  Accept-Ranges
                    /  Cache-Control
                    /  Connection
                    /  CSeq
                    /  Date
                    /  Media-Properties
                    /  Media-Range
                    /  Pipelined-Requests
                    /  Proxy-Supported
                    /  Range
                    /  RTP-Info
                    /  Scale
                    /  Seek-Style
                    /  Server
                    /  Session
                    /  Speed
                    /  Supported
                    /  Timestamp
                    /  Transport
                    /  User-Agent
                    /  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-Authorization
                    /  Proxy-Require
                    /  Referrer
                    /  Request-Status
                    /  Require
                    /  Terminate-Reason
                    /  extension-header
```

```
   response-header  =  Authentication-Info
                    /  Connection-Credentials
                    /  Location
                    /  MTag
                    /  Proxy-Authenticate
                    /  Proxy-Authentication-Info
                    /  Public
                    /  Retry-After
                    /  Unsupported
                    /  WWW-Authenticate
                    /  extension-header

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

### 20.2.3.  Header Syntax

```
   Accept             =  "Accept" HCOLON
                         [ accept-range *(COMMA accept-range) ]
   accept-range       =  media-type-range [SEMI accept-params]
   media-type-range   =  ( "*/*"
                         / ( m-type SLASH "*" )
                         / ( m-type SLASH m-subtype )
                        ) *( SEMI m-parameter )
   accept-params      =  "q" EQUAL qvalue *(SEMI generic-param )
   qvalue             =  ( "0" [ "." *3DIGIT ] )
                      /  ( "1" [ "." *3("0") ] )
   Accept-Credentials =  "Accept-Credentials" HCOLON cred-decision
   cred-decision      =  ("User" [LWS cred-info])
                      /  "Proxy"
                      /  "Any"
                      /  (token [LWS 1*header-value])
                                      ; For future extensions
   cred-info          =  cred-info-data *(COMMA cred-info-data)

   cred-info-data     =  DQUOTE RTSP-REQ-URI DQUOTE 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-params]
   codings              =  content-coding / "*"
   content-coding       =  "identity" / token
   Accept-Language      =  "Accept-Language" HCOLON
                           language *(COMMA language)
   language             =  language-range [SEMI accept-params]
   language-range       =  language-tag / "*"
   language-tag         =  primary-tag *( "-" subtag )
   primary-tag          =  1*8ALPHA
   subtag               =  1*8ALPHA
   Accept-Ranges        =  "Accept-Ranges" HCOLON acceptable-ranges
   acceptable-ranges =  (range-unit *(COMMA range-unit))
   range-unit           =  "npt" / "smpte" / "smpte-30-drop" / "smpte-25"
                           / "clock" / extension-format
   extension-format  =  token
   Allow                =  "Allow" HCOLON Method *(COMMA Method)
   Authentication-Info = "Authentication-Info" HCOLON auth-param-list
   auth-param-list   =  <As the Authentication-Info element in RFC 7615>
   Authorization        =  "Authorization" HCOLON credentials
   credentials          =  <As defined by RFC 7235>

   Bandwidth            =  "Bandwidth" HCOLON 1*19DIGIT

   Blocksize            =  "Blocksize" HCOLON 1*9DIGIT

   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*19DIGIT
```

```
   Connection          =  "Connection" HCOLON connection-token
                          *(COMMA connection-token)
   connection-token    =  "close" / token

   Connection-Credentials = "Connection-Credentials" HCOLON cred-chain
   cred-chain          =  DQUOTE RTSP-REQ-URI DQUOTE SEMI base64

   Content-Base        =  "Content-Base" HCOLON RTSP-URI
   Content-Encoding    =  "Content-Encoding" HCOLON
                          content-coding *(COMMA content-coding)
   Content-Language    =  "Content-Language" HCOLON
                          language-tag *(COMMA language-tag)
   Content-Length      =  "Content-Length" HCOLON 1*19DIGIT
   Content-Location    =  "Content-Location" HCOLON RTSP-REQ-Ref
   Content-Type        =  "Content-Type" 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       =  date-time ;
   date-time       =  <As defined in RFC 5322>
   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-time)
                 / ("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 = DQUOTE scale-entry *(COMMA scale-entry) DQUOTE
scale-entry      = scale-value / (scale-value COLON scale-value)
scale-value      = FLOAT
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     = <As defined by the WWW-Authenticate in RFC 7235>
Proxy-Authentication-Info = "Proxy-Authentication-Info" HCOLON
                      auth-param-list
Proxy-Authorization = "Proxy-Authorization" HCOLON credentials
Proxy-Require      =  "Proxy-Require" HCOLON feature-tag-list
feature-tag-list   =  feature-tag *(COMMA feature-tag)
Proxy-Supported    =  "Proxy-Supported" HCOLON [feature-tag-list]

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 [EQUAL range-value]
   range-value         =  1*(rtsp-unreserved / quoted-string / ":" )

   Referrer            =  "Referrer" HCOLON (absolute-URI / RTSP-URI-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 DQUOTE Reason-Phrase DQUOTE
   Require             =  "Require" HCOLON feature-tag-list
```

```
   RTP-Info          =  "RTP-Info" HCOLON [rtsp-info-spec
                        *(COMMA rtsp-info-spec)]
   rtsp-info-spec    =  stream-url 1*ssrc-parameter
   stream-url        =  "url" EQUAL DQUOTE RTSP-REQ-Ref DQUOTE
   ssrc-parameter    =  LWS "ssrc" EQUAL ssrc HCOLON
                        ri-parameter *(SEMI ri-parameter)
   ri-parameter      =  ("seq" EQUAL 1*5DIGIT)
                     /  ("rtptime" EQUAL 1*10DIGIT)
                     /  generic-param

   Retry-After       =  "Retry-After" HCOLON (RTSP-date / delta-seconds)
   Scale             =  "Scale" HCOLON scale-value
   Seek-Style        =  "Seek-Style" HCOLON Seek-S-values
   Seek-S-values     =  "RAP"
                     /  "CoRAP"
                     /  "First-Prior"
                     /  "Next"
                     /  Seek-S-value-ext
   Seek-S-value-ext =  token

   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 ]

   Speed             =  "Speed" HCOLON lower-bound MINUS upper-bound
   lower-bound       =  POS-FLOAT
   upper-bound       =  POS-FLOAT

   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  =  *19DIGIT [ "." *9DIGIT ]
   delay            =  *9DIGIT [ "." *9DIGIT ]

   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" / "SAVPF" / 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 "setup" EQUAL contrans-setup)
                     / (SEMI "connection" EQUAL contrans-con)
                     / (SEMI "RTCP-mux")
                     / (SEMI "MIKEY" EQUAL MIKEY-Value)
                     / (SEMI trn-param-ext)
   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 ; 0 to 255
   MIKEY-Value       = base64
   mode-spec         = ( DQUOTE mode *(COMMA mode) DQUOTE )
   mode              = "PLAY" / token
   addr-list         = quoted-addr *(SLASH quoted-addr)
   quoted-addr       = DQUOTE (host-port / extension-addr) DQUOTE
   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 )
                  *(LWS (product / comment))
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-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-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 CRLF

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

a-mtag-def          =  "a=mtag:" message-tag CRLF
```

## 21. Security Considerations

The security considerations and threats around RTSP and its usage can be divided into considerations around the signaling protocol itself and the issues related to the media-stream delivery.  However, when it comes to mitigation of security threats, a threat depending on the media-stream delivery may in fact be mitigated by a mechanism in the signaling protocol.

There are several chapters and an appendix in this document that define security solutions for the protocol. These sections will be referenced when discussing the threats below. However, the reader should take special notice of the Security Framework (Section 19) and the specification of how to use SRTP and its key-management (Appendix C.1.4) to achieve certain aspects of the media security.

## 21.1. Signaling Protocol Threats

This section focuses on issues related to the signaling protocol. Because of the similarity in syntax and usage between RTSP servers and HTTP servers, the security considerations outlined in [RFC7230], [RFC7231], [RFC7232], [RFC7233], [RFC7234], and [RFC7235] apply as well.

Specifically, please note the following:

Abuse of Server Log Information: A server is in the position to save personal data about a user's requests that might identify their media consumption patterns or subjects of interest. This information is clearly confidential in nature, and its handling can be constrained by law in certain countries. Log information needs to be securely stored and appropriate guidelines followed for its analysis. See Section 9.8 of [RFC7230] for additional guidelines.

Transfer of Sensitive Information: There is no reason to believe that information transferred in RTSP message, such as the URI and the content of headers, especially the Server, Via, Referrer, and From headers, may be any less sensitive than when used in HTTP. Therefore, all of the precautions regarding the protection of data privacy and user privacy apply to implementers of RTSP clients, servers, and proxies. See Sections 9.3-9.6 of [RFC7231] for further details.

The RTSP methods defined in this document are primarily used to establish and control the delivery of the media data represented by the URI; thus, the RTSP message bodies are generally less sensitive than the ones in HTTP. Where HTTP bodies could contain, for example, your medical records, in RTSP, the sensitive video of your medical operation would be in the media stream over the media-transport protocol, not in the RTSP message. Still, one has to take note of what potential sensitive information is included in RTSP. The protection of the media data is separate, can be applied directly between client and server, and is dependent on the media-transport protocol in use. See Section 21.2 for further discussion. This possibility for separation of security between media-resource content and the signaling protocol mitigates the risk of exposing the media content when using hop-by-hop security for RTSP signaling using proxies (Section 19.3).

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 Section 9.1 of [RFC7231], such as checking for ".." in path components.

Personal Information: RTSP clients are often privy to the same information that HTTP clients are (username, location, etc.) and thus should be equally sensitive. See Section 9.8 of [RFC7230], Sections 9.3-9.7 of [RFC7231], and Section 8 of [RFC7234] for further recommendations.

Privacy Issues Connected to Accept Headers: Since similar usages of the "Accept" headers exist in RTSP as in HTTP, the same caveats outlined in Section 9.4 of [RFC7231] with regard to their use should be followed.

Establishing Authority: RTSP shares with HTTP the question of how a client communicates with the authoritative source for media streams (Section 9.1 of [RFC7230]). The used DNS servers, the security of the communication, and any possibility of a man in the middle, and the trust in any RTSP proxies all affect the possibility that a client has received a non-authoritative response to a request. Ensuring that a client receives an authoritative response is challenging, although using the secure communication for RTSP signaling (rtsps) simplifies it significantly as the server can provide a hostname identity assertion in the TLS handshake.

Location Headers and Spoofing: If a single server supports multiple organizations that do not trust each another, then it MUST check the values of the 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 (see Section 15.4 of [RFC2616]).

In addition to the recommendations in the current HTTP specifications ([RFC7230], [RFC7231], [RFC7232], [RFC7233], [RFC7234], and [RFC7235] as of this writing) and also those of the previous relevant RFCs [RFC2068] [RFC2616], future HTTP specifications may provide additional guidance on security issues.

The following are added considerations for RTSP implementations.

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 that could affect other clients of an unsuspecting server. To mitigate this, the server SHALL use a large, random and non-sequential session identifier to minimize the possibility of this kind of attack. However, unless the RTSP signaling is always confidentiality protected, e.g., using TLS, an on-path attacker will be able to hijack a session. Another choice for preventing session hijacking is to use client authentication and only allow the authenticated client creating the session to access that session.

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.

Suspicious Behavior: Upon detecting instances of behavior that is deemed a security risk, RTSP servers SHOULD return error code 403 (Forbidden). 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 from clients that are deemed to be in violation of local security policy.

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. This procedure requires trust in all proxies part of the path to the server. The proxies one connects through are identified, assuming the proxies so far connected through are well behaved and fulfilling the trust. The accepted proxies are men in the middle and have 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. Further discussion of the actual trust model is in Section 19.3. It is important to note what difference in security properties, if any, may exist with the used media-transport protocol and its security mechanism. Using SRTP and the MIKEY-based key-establishment defined in Appendix C.1.4.1 enables media key-establishment to be done end-to-end without revealing the keys to the proxies.

Resource Exhaustion: As RTSP is a stateful protocol and establishes resource usage on the server, there is a clear possibility to attack the server by trying to overbook these resources to perform a DoS attack. This attack can be both against ongoing sessions and to prevent others from establishing sessions. RTSP agents will need to have mechanisms to prevent single peers from consuming extensive amounts of resources. The methods for guarding against this are varied and depend on the agent's role and capabilities and policies. Each implementation has to carefully consider its methods and policies to mitigate this threat. There are recommendations regarding the handling of connections in Section 10.7.

The above threats and considerations have resulted in a set of security functions and mechanisms built into or used by the protocol. The signaling protocol relies on two security features defined in the Security Framework (Section 19): namely client authentication using HTTP authentication and TLS-based transport protection of the signaling messages. Both of these mechanisms are required to be implemented by any RTSP agent.

A number of different security mitigations have been designed into the protocol and will be instantiated if the specification is implemented as written, for example, by ensuring sufficient amounts of entropy in the randomly generated session identifiers when not using client authentication to minimize the risk of session hijacking. When client authentication is used, protection against hijacking will be greatly improved by scoping the accessible sessions to the one this client identity has created. Some of the above threats are such that the implementation of the RTSP functionality itself needs to consider which policy and strategy it uses to mitigate them.

## 21.2. Media Stream Delivery Threats

The fact that RTSP establishes and controls a media-stream delivery results in a set of security issues related to the media streams. This section will attempt to analyze general threats; however, the choice of media-stream transport protocol, such as RTP, will result in some differences in threats and what mechanisms exist to mitigate them. Thus, it becomes important that each specification of a new media-stream transport and delivery protocol usable by RTSP requires its own security analysis. This section includes one for RTP.

The set of general threats from or by the media-stream delivery itself are:

Concentrated Denial-of-Service Attack: The protocol offers the opportunity for a remote-controlled DoS attack, where the media stream is the hammer in that DoS attack. See Section 21.2.1.

Media Confidentiality: The media delivery may contain content of any type, and it is not possible, in general, to determine how sensitive this content is from a confidentiality point. Thus, it is a strong requirement that any media delivery protocol supply a method for providing confidentiality of the actual media content. In addition to the media-level confidentiality, it becomes critical that no resource identifiers used in the signaling be exposed to an attacker as they may have human-understandable names or may be available to the attacker, allowing it to determine the content the user received. Thus, the signaling protocol must also provide confidentiality protection of any information related to the media resource.

Media Integrity and Authentication: There are several reasons why an attacker will be interested in substituting the media stream sent out from the RTSP server with one of the attacker's creation or selection, such as discrediting the target and misinformation about the target. Therefore, it is important that the media protocol provide mechanisms to verify the source authentication and integrity and to prevent replay attacks on the media stream.

Scope of Multicast: If RTSP is used to control the transmission of media onto a multicast network, the scope of the delivery must be considered. RTSP supports the TTL Transport header parameter to indicate this scope for IPv4. IPv6 has a different mechanism for the scope boundary. However, such scope control has risks, as it may be set too large and distribute media beyond the intended scope.

Below (Section 21.2.2) a protocol-specific analysis of security considerations for RTP-based media transport is included. In that section, the requirements on implementing security functions for RTSP agents supporting media delivery over RTP are made clear.

### 21.2.1. Remote DoS 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 the prevention of more attacks or ascertaining the attacker's 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 order of increasing generality are:

- o Verification of the client's identity against a database of known users using RTSP authentication mechanisms (preferably Digest authentication or stronger)

- o A list of addresses that have consented to be media destinations, especially considering user identity

- o Verification based on media path

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 be passed along to the server. However, in certain cases, such as when the 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 NAT traversal method based on Interactive Connectivity Establishment (ICE) [RFC5245] described in [RFC7825]. This mechanism uses random passwords and a username so that the probability of unintended indication as a valid media destination is very low. In addition, the server includes in its Session Traversal Utilities for NAT (STUN) [RFC5389] requests a cookie (consisting of random material) that the destination echoes back; thus, the solution also safeguards against having an off-path attacker being able to spoof the STUN checks. This leaves this solution vulnerable only to on-path attackers that can see the STUN requests go to the target of attack and thus forge a response.

For delivery to multicast addresses, there is a need for another solution that is not specified in this memo.

### 21.2.2. RTP Security Analysis

RTP is a commonly used media-transport protocol and has been the most common choice for RTSP 1.0 implementations. The core RTP protocol has been in use for a long time, and it has well-known security properties and the RTP security consideration (Section 9 of [RFC3550]) needs to be reviewed. In perspective of the usage of RTP in the context of RTSP, the following properties should be noted:

Stream Additions: RTP has support for multiple simultaneous media streams in each RTP session. As some use cases require support for non-synchronized adding and removal of media streams and their identifiers, an attacker can easily insert additional media streams into a session context that, according to protocol design, is intended to be played out. Another threat vector is one of DoS by exhausting the resources of the RTP session receiver, for example, by using a large number of SSRC identifiers simultaneously. The strong mitigation of this is to ensure that one cryptographically authenticates any incoming packet flow to the RTP session. Weak mitigations like blocking additional media streams in session contexts easily lead to a DoS vulnerability in addition to preventing certain RTP extensions or use cases that rely on multiple media streams, such as RTP retransmission [RFC4588] to function.

Forged Feedback: The built-in RTCP also offers a large attack surface for a couple of different types of attacks. One venue is to send RTCP feedback to the media sender indicating large amounts of packet loss and thus trigger a media bitrate adaptation response from the sender resulting in lowered media quality and potentially a shutdown of the media stream. Another attack is to perform a resource-exhaustion attack on the receiver by using many SSRC identifiers to create large state tables and increase the RTCP-related processing demands.

RTP/RTCP Extensions: RTP and RTCP extensions generally provide additional and sometimes extremely powerful tools for DoS attacks or service disruption. For example, the Code Control Message [RFC5104] RTCP extensions enables both the lock down of the bitrate to low values and disruption of video quality by requesting intra-frames.

Taking into account the above general discussion in Section 21.2 and the RTP-specific discussion in this section, it is clear that it is necessary that a strong security mechanism be supported to protect

RTP. Therefore, this specification has the following requirements on RTP security functions for all RTSP agents that handle media streams and where media-stream transport is completed using RTP.

RTSP agents supporting RTP MUST implement Secure RTP (SRTP) [RFC3711] and, thus, SAVP. In addition, SAVPF [RFC5124] MUST also be supported if AVPF is implemented. This specification requires no additional cryptographic transforms or configuration values beyond those specified as mandatory to implement in RFC 3711, i.e., AES-CM and HMAC-SHA1. The default key-management mechanism that MUST be implemented is the one defined in MIKEY Key Establishment (Appendix C.1.4.1). The MIKEY implementation MUST implement the necessary functions for MIKEY-RSA-R mode [RFC4738] and the SRTP parameter negotiation necessary to negotiate the supported SRTP transforms and parameters.

## 22. IANA Considerations

This section describes a number of registries for RTSP 2.0 that have been established and are maintained by IANA. These registries are separate from any registries existing for RTSP 1.0. For each registry, there is a description of the required content, the registration procedures, and the entries that this document registers. For more information on extending RTSP, see Section 2.7. In addition, this document registers three SDP attributes.

Registries or entries in registries that have been made for RTSP 1.0 are not moved to RTSP 2.0: the registries and entries of RTSP 1.0 and RTSP 2.0 are independent. If any registry or entry in a registry is also required in RTSP 2.0, it MUST follow the procedure defined below to allocate the registry or entry in a registry.

The sections describing how to register an item use some of the registration policies described in [RFC5226] -- namely, "First Come First Served", "Expert Review", "Specification Required", and "Standards Action".

In case a registry requires a contact person, the authors (with Magnus Westerlund <magnus.westerlund@ericsson.com> as primary) are the contact persons for any entries created by this document.

IANA will request the following information for any registration request:

- 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, the 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 [RFC5226] basis.

The registry entry for a feature tag has the following information:

- The name of the feature tag

    * If the registrant indicates that the feature is proprietary, IANA should request a vendor "prefix" portion of the name. The name will then be the vendor prefix followed by a "." followed by the rest of the provided feature name.

    * If the feature is not proprietary, then IANA need not collect a prefix for the name.

- A one-paragraph description of what the feature tag represents

- The applicability (server, client, proxy, or some combination)

- A reference to a specification, if applicable

Feature tag names (including the vendor prefix) may contain any non-space and non-control characters. There is no length limit on feature tags.

Examples for a vendor tag describing a proprietary feature are:

vendorA.specfeat01

vendorA.specfeat02

#### 22.1.3. Registered Entries

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

play.basic: The implementation for delivery and playback operations according to the core RTSP specification, as defined in this memo. Applies for clients, servers, and proxies. See Section 11.1.

play.scale: Support of scale operations for media playback. Applies only for servers. See Section 18.46.

play.speed: Support of the speed functionality for media delivery. Applies only for servers. See Section 18.50.

setup.rtp.rtcp.mux: Support of the RTP and RTCP multiplexing as discussed in Appendix C.1.6.4. Applies for both client and servers and any media caching proxy.

The IANA registry is a table with the name, description, and reference for each feature tag.

### 22.2. RTSP Methods

#### 22.2.1. Description

Methods are described in Section 13. Extending the protocol with new methods allows for totally new functionality.

#### 22.2.2. Registering New Methods with IANA

A new method is registered through a Standards Action [RFC5226] because new methods may radically change the protocol’s behavior and purpose.

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

- o A method name that follows the ABNF rules for methods.

- o A clear specification of what a request using the method does and what responses are expected. In which directions the method is used: C->S, S->C, or both. How the use of headers, if any, modifies the behavior and effect of the method.

- o A list or table specifying which of the IANA-registered headers that are allowed to be used with the method in the request or/and response. The list or table SHOULD follow the format of tables in Section 18.

- o Describe how the method relates to network proxies.

#### 22.2.3. Registered Entries

This specification, RFC 7826, registers 10 methods: DESCRIBE, GET_PARAMETER, OPTIONS, PAUSE, PLAY, PLAY_NOTIFY, REDIRECT, SETUP, SET_PARAMETER, and TEARDOWN. The initial table of the registry is provided below.

| Method | Directionality | Reference |
|---|---|---|
| DESCRIBE | C->S | RFC 7826 |
| GET_PARAMETER | C->S, S->C | RFC 7826 |
| OPTIONS | C->S, S->C | RFC 7826 |
| PAUSE | C->S | RFC 7826 |
| PLAY | C->S | RFC 7826 |
| PLAY_NOTIFY | S->C | RFC 7826 |
| REDIRECT | S->C | RFC 7826 |
| SETUP | C->S | RFC 7826 |
| SET_PARAMETER | C->S, S->C | RFC 7826 |
| TEARDOWN | C->S, S->C | RFC 7826 |

### 22.3. RTSP Status Codes

#### 22.3.1. Description

A status code is the three-digit number 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 registration follows the policy of IETF Review [RFC5226]. New RTSP functionality requiring Status Codes should first be registered in the range of x50-x99. Only when the range is full should registrations be made in the x00-x49 range, unless it is to adopt an HTTP extension to RTSP. This is done to enable any HTTP extension to be adopted to RTSP without needing to renumber any related status codes. A specification for a new status code must include the following:

- o The registered number.

- o A description of what the status code means and the expected behavior of the sender and receiver of the code.

### 22.3.3. Registered Entries

RFC 7826 (this document) registers the numbered status code defined in the ABNF entry "Status-Code", except "extension-code" (that defines the syntax allowed for future extensions) in Section 20.2.2.

## 22.4. RTSP Headers

### 22.4.1. Description

By specifying new headers, one or more methods can be enhanced in many different ways. An unknown header will be ignored by the receiving agent. If the new header is vital for certain functionality, a feature tag for the functionality can be created and demanded to be used by the counterpart with the inclusion of a Require header carrying the feature tag.

### 22.4.2. Registering New Headers with IANA

Registrations can be made following the Expert Review policy [RFC5226]. A specification is recommended to be provided, preferably an RFC or other specification from a Standards Developing Organization. The minimal information in a registration request is the header name and the contact information.

The expert reviewer verifies that the registration request contains the following information:

- o The name of the header.

- o 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, and 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 18 in RFC 7826 have been registered. The registry includes the header name and reference.

Furthermore, the following legacy RTSP headers defined in other specifications are registered with header name, and reference according to below list. Note: these references may not fulfill all of the above rules for registrations due to their legacy status.

- x-wap-profile defined in [TS-26234]. The x-wap-profile request-header contains one or more absolute URLs to the requesting agent's device-capability profile.

- x-wap-profile-diff defined in [TS-26234]. The x-wap-profile-diff request-header contains a subset of a device-capability profile.

- x-wap-profile-warning defined in [TS-26234]. The x-wap-profile-warning is a response-header that contains error codes explaining to what extent the server has been able to match the terminal request in regard to device-capability profiles, as described using x-wap-profile and x-wap-profile-diff headers.

- x-predecbufsize defined in [TS-26234]. This response-header provides an RTSP agent with the TS 26.234 Annex G hypothetical pre-decoder buffer size.

- x-initpredecbufperiod defined in [TS-26234]. This response-header provides an RTSP agent with the TS 26.234 Annex G hypothetical pre-decoder buffering period.

- x-initpostdecbufperiod defined in [TS-26234]. This response-header provides an RTSP agent with the TS 26.234 Annex G post-decoder buffering period.

- 3gpp-videopostdecbufsize defined in [TS-26234]. This response-header provides an RTSP agent with the TS 26.234 defined post-decoder buffer size usable for H.264 (AVC) video streams.

o 3GPP-Link-Char defined in [TS-26234]. This request-header provides the RTSP server with the RTSP client's link characteristics as determined from the radio interface. The information that can be provided are guaranteed bitrate, maximum bitrate and maximum transfer delay.

o 3GPP-Adaptation defined in [TS-26234]. This general-header is part of the bitrate adaptation solution specified for the Packet-switched Streaming Service (PSS). It provides the RTSP client's buffer sizes and target buffer levels to the server, and responses are used to acknowledge the support and values.

o 3GPP-QoE-Metrics defined in [TS-26234]. This general-header is used by PSS RTSP agents to negotiate the quality of experience metrics that a client should gather and report to the server.

o 3GPP-QoE-Feedback defined in [TS-26234]. This request-header is used by RTSP clients supporting PSS to report the actual values of the metrics gathered in its quality of experience metering.

The use of "x-" is NOT RECOMMENDED, but the above headers in the list were defined prior to the clarification.

## 22.5. Accept-Credentials

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

### 22.5.1. Accept-Credentials Policies

This registry is for policies for an RTSP proxy's handling and verification of TLS certificates when establishing an outbound TLS connection on behalf of a client. In Section 19.3.1, three policies for how to handle certificates are specified. Further policies may be defined; registration is made through Standards Action [RFC5226]. A registration request is required to contain the following information:

o Name of the policy.

o Text that describes how the policy works for handling the certificates.

o A contact person.

This specification registers the following values:

Any: A policy requiring the proxy to accept any received certificate.

Proxy: A policy where the proxy applies its own policies to determine which certificates are accepted.

User: A policy where the certificate is required to be forwarded down the proxy chain to the client, thus allowing the user to decided to accept or refuse a certificate.

### 22.5.2. Accept-Credentials Hash Algorithms

The Accept-Credentials header (see Section 18.2) 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 requires Standards Action [RFC5226]. The registration needs to fulfill the following requirement:

- o The algorithms identifier meeting the "token" ABNF requirement.
- o Provide a definition of the algorithm.

The registered value is:

| Hash Alg. ID | Reference |
|---|---|
| sha-256 | RFC 7826 |

## 22.6. Cache-Control Cache Directive Extensions

There exist a number of cache directives that can be sent in the Cache-Control header. A registry for these cache directives has been established by IANA. New registrations in this registry require Standards Action or IESG Approval [RFC5226]. A registration request needs to contain the following information.

- o The name of the cache directive.
- o A definition of the parameter value, if any is allowed.
- o The specification if it is a request or response directive.

- Text that explains how the cache directive is used for RTSP-controlled media streams.

- A contact person.

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:

The registry contains the name of the directive and the reference.

## 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 were 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 a new media property is done following the Specification Required policy [RFC5226]. The expert reviewer verifies that a registration request fulfills the following requirements.

- o An ABNF definition of the media property value name that meets "media-prop-ext" definition is included.
- o A definition of which media property group it belongs to or define a new group is included.
- o A description of all changes to the behavior of RTSP as result of these changes is included.
- o A contact person for the registration is listed.

### 22.7.3. Registered Values

This specification registers the ten values listed in Section 18.29. The registry contains the property group, the name of the media property, and the reference.

## 22.8. Notify-Reason Values

### 22.8.1. Description

Notify-Reason values are used to indicate the reason the notification was sent. Each reason has its associated rules on what headers and information may or must be included in the notification. New notification behaviors need to be specified to enable interoperable usage; thus, a specification of each new value is required.

### 22.8.2. Registration Rules

Registrations for new Notify-Reason values follow the Specification Required policy [RFC5226]. The expert reviewer verifies that the request fulfills the following requirements:

- o An ABNF definition of the Notify-Reason value name that meets "Notify-Reason-extension" definition is included.
- o A 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 is made clear.
- o A contact person for the registration is listed.

### 22.8.3. Registered Values

This specification registers three values defined in the Notify-Reas-val ABNF, Section 20.2.3:

end-of-stream: This Notify-Reason value indicates the end of a media stream.

media-properties-update: This Notify-Reason value allows the server to indicate that the properties of the media have changed during the playout.

scale-change: This Notify-Reason value allows the server to notify the client about a change in the scale of the media.

The registry contains the name, description, and reference.

## 22.9. Range Header Formats

### 22.9.1. Description

The Range header (Section 18.40) allows for different range formats. These range formats also need an identifier to be used in the Accept-Ranges header (Section 18.5). New range formats may be registered, but moderation should be applied as it makes interoperability more difficult.

### 22.9.2. Registration Rules

A registration follows the Specification Required policy [RFC5226]. The expert reviewer verifies that the request fulfills the following requirements:

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

- The range format identifier used in Accept-Ranges header according to the "extension-format" definition is defined.

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

- A contact person for the registration is listed.

### 22.9.3. Registered Values

The registry contains the Range header format identifier, the name of the range format, and the reference. This specification registers the following values.

npt: Normal Play Time

clock: UTC Absolute Time format

smpte: SMPTE Timestamps

smpte-30-drop: SMPTE Timestamps 29.97 Frames/sec (30 Hz with Drop)

smpte-25: SMPTE Timestamps 25 Frames/sec

## 22.10. Terminate-Reason Header

The Terminate-Reason header (Section 18.52) has two registries for extensions.

### 22.10.1. Redirect Reasons

This registry contains reasons for session termination that can be included in a Terminate-Reason header (Section 18.52). Registrations follow the Expert Review policy [RFC5226]. The expert reviewer verifies that the registration request contains the following information:

- o That the value follows the Terminate-Reason ABNF, i.e., be a token.

- o That the specification provide a definition of what procedures are to be followed when a client receives this redirect reason.

- o A contact person

This specification registers three values:

- o Session-Timeout

- o Server-Admin

- o Internal-Error

The registry contains the name of the Redirect Reason and the reference.

### 22.10.2. Terminate-Reason Header Parameters

This registry contains parameters that may be included in the Terminate-Reason header (Section 18.52) in addition to a reason. Registrations are made under the Specification Required policy [RFC5226]. The expert reviewer verifies that the registration request contains the following:

- A parameter name.
- A parameter following the syntax allowed by the RTSP 2.0 specification.
- A reference to the specification.
- A contact person.

This specification registers:

- time
- user-msg

The registry contains the name of the Terminate Reason and the reference.

## 22.11. RTP-Info Header Parameters

### 22.11.1. Description

The RTP-Info header (Section 18.45) 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, new registration is made under the Specification Required policy.

### 22.11.2. Registration Rules

Registrations for new RTP-Info values follow the policy of Specification Required [RFC5226]. The expert reviewer verifies that the registration request contains the following information.

- An ABNF definition that meets the "generic-param" definition.
- A reference to the specification.
- A contact person for the registration.

### 22.11.3. Registered Values

This specification registers the following parameter value pairs:

- url
- ssrc
- seq
- rtptime

The registry contains the name of the parameter and the reference.

## 22.12. Seek-Style Policies

### 22.12.1. Description

Seek-Style policy defines how the RTSP agent seeks in media content when given a position within the media content. 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 will use the server default policy instead.

### 22.12.2. Registration Rules

Registrations of new Seek-Style policies follow the Specification Required policy [RFC5226]. The expert reviewer verifies that the registration request fulfills the following requirements:

- Has an ABNF definition of the Seek-Style policy name that meets "Seek-S-value-ext" definition.
- Includes a short description.
- Lists a contact person for the registration.
- Includes a 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 four values (Name - Short Description):

- RAP - Using the closest Random Access Point prior to or at the requested start position.

- o CoRAP - Conditional Random Access Point is like RAP, but only if the RAP is closer prior to the requested start position than current pause point.

- o First-Prior - The first-prior policy will start delivery with the media unit that has a playout time first prior to the requested start position.

- o Next - The next media units after the provided start position.

The registry contains the name of the Seek-Style policy, the description, and the reference.

### 22.13. Transport Header Registries

The transport header (Section 18.54) contains a number of parameters that have possibilities for future extensions. Therefore, registries for these are defined below.

#### 22.13.1. Transport Protocol Identifier

A Transport Protocol specification consists of a transport protocol identifier, representing some combination of transport protocols, and any number of transport header parameters required or optional to use with the identified protocol specification. This registry contains the identifiers used by registered transport protocol identifiers.

A registration for the parameter transport protocol identifier follows the Specification Required policy [RFC5226]. The expert reviewer verifies that the registration request fulfills the following requirements:

- o A contact person or organization with address and email.

- o A value definition that follows the ABNF syntax definition of "transport-id" Section 20.2.3.

- o A descriptive text that explains how the registered values are used in RTSP, which underlying transport protocols are used, and any required Transport header parameters.

The registry contains the protocol ID string and the reference.

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 7826, Appendix C.1.

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

RTP/AVPF: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended RTP Profile for RTCP-based Feedback (RTP/AVPF)" [RFC4585] over UDP. The usage is explained in RFC 7826, Appendix C.1.

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

RTP/SAVP: Use of the RTP [RFC3550] protocol for media transport in combination with the "The Secure Real-time Transport Protocol (SRTP)" [RFC3711] over UDP. The usage is explained in RFC 7826, Appendix C.1.

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

RTP/SAVPF: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] over UDP. The usage is explained in RFC 7826, 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 7826, Appendix C.2.2.

RTP/AVPF/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)" [RFC4585] over TCP. The usage is explained in RFC 7826, Appendix C.2.2.

RTP/SAVP/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "The Secure Real-time Transport Protocol (SRTP)" [RFC3711] over TCP. The usage is explained in RFC 7826, Appendix C.2.2.

RTP/SAVPF/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] over TCP. The usage is explained in RFC 7826, Appendix C.2.2.

### 22.13.2. Transport Modes

The Transport Mode is a Transport header (Section 18.54) parameter. It is used to identify the general mode of media transport. The PLAY value registered defines a PLAYBACK mode, where media flows from server to client.

A registration for the transport parameter mode follows the Standards Action policy [RFC5226]. The registration request needs to meet the following requirements:

- o A value definition that follows the ABNF "token" definition Section 20.2.3.

- o Text that explains how the registered value is used in RTSP.

This specification registers one value:

PLAY: See RFC 7826.

The registry contains the transport mode value and the reference.

### 22.13.3. Transport Parameters

Transport Parameters are different parameters used in a Transport header's transport specification (Section 18.54) to provide additional information required beyond the transport protocol identifier to establish a functioning transport.

A registration for parameters that may be included in the Transport header follows the Specification Required policy [RFC5226]. The expert reviewer verifies that the registration request fulfills the following requirements:

- o A Transport Parameter Name following the "token" ABNF definition.

- o A value definition, if the parameter takes a value, that follows the ABNF definition of "trn-par-value" Section 20.2.3.

- o Text that explains how the registered value is used in RTSP.

This specification registers all the transport parameters defined in Section 18.54. This is a copy of that list:

- o unicast
- o multicast
- o interleaved
- o ttl
- o layers
- o ssrc
- o mode
- o dest_addr
- o src_addr
- o setup
- o connection
- o RTCP-mux
- o MIKEY

The registry contains the transport parameter name and the reference.

### 22.14. URI Schemes

This specification updates two URI schemes: one previously registered, "rtsp", and one missing in the registry, "rtspu" (previously only defined in RTSP 1.0 [RFC2326]). One new URI scheme, "rtsps", is also registered. These URI schemes are registered in an existing registry ("Uniform Resource Identifier (URI) Schemes") not created by this memo. Registrations follow [RFC7595].

#### 22.14.1. The "rtsp" URI Scheme

URI scheme name: rtsp

Status: Permanent

URI scheme syntax: See Section 20.2.1 of RFC 7826.

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 DESCRIBE, GET_PARAMETER, OPTIONS, PLAY, PLAY_NOTIFY, PAUSE, REDIRECT, SETUP, SET_PARAMETER, and TEARDOWN.

Encoding considerations: IRIs in this scheme are defined and need to be encoded as RTSP URIs when used within RTSP. 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 7826).

Interoperability considerations: The extensions in the URI syntax performed between RTSP 1.0 and 2.0 can create interoperability issues. The changes are:

Support for IPv6 literals in the host part and future IP literals through a mechanism as defined in RFC 3986.

A new relative format to use in RTSP elements that is not required to start with "/".

The above changes should have no impact on interoperability as discussed in detail in Section 4.2 of RFC 7826.

Security considerations: All the security threats identified in Section 7 of RFC 3986 also apply 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, and RFC 7826

### 22.14.2. The "rtsps" URI Scheme

URI scheme name: rtsps

Status: Permanent

URI scheme syntax: See Section 20.2.1 of RFC 7826.

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 DESCRIBE, GET_PARAMETER, OPTIONS, PLAY, PLAY_NOTIFY, PAUSE, REDIRECT, SETUP, SET_PARAMETER, and TEARDOWN.

Encoding considerations: IRIs in this scheme are defined and need to be encoded as RTSP URIs when used within RTSP. 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 7826).

Interoperability considerations: The "rtsps" scheme was never officially defined for RTSP 1.0; however, it has seen widespread use in actual deployments of RTSP 1.0. Therefore, this section discusses the believed changes between the unspecified RTSP 1.0 "rtsps" scheme and RTSP 2.0 definition. The extensions in the URI syntax performed between RTSP 1.0 and 2.0 can create interoperability issues. The changes are:

Support for IPv6 literals in the host part and future IP literals through a mechanism as defined by RFC 3986.

A new relative format to use in RTSP elements that is not required to start with "/".

The above changes should have no impact on interoperability as discussed in detail in Section 4.2 of RFC 7826.

Security considerations: All the security threats identified in Section 7 of RFC 3986 also apply 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, and RFC 7826

### 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 DESCRIBE, GET_PARAMETER, OPTIONS, REDIRECT,PLAY, PLAY_NOTIFY, PAUSE, SETUP, SET_PARAMETER, and TEARDOWN.

Encoding considerations: This scheme is not intended to be used with characters outside the US-ASCII repertoire.

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 also apply 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

## 22.15. SDP Attributes

This specification defines three SDP [RFC4566] attributes that have been registered by IANA.

SDP Attribute ("att-field"):

```
Attribute name:     range
Long form:          Media Range Attribute
Type of name:       att-field
Type of attribute:  both session and media level
Subject to charset: No
Purpose:            RFC 7826
Reference:          RFC 2326, RFC 7826
Values:             See ABNF definition.

Attribute name:     control
Long form:          RTSP control URI
Type of name:       att-field
Type of attribute:  both session and media level
Subject to charset: No
Purpose:            RFC 7826
Reference:          RFC 2326, RFC 7826
Values:             Absolute or Relative URIs.

Attribute name:     mtag
Long form:          Message Tag
Type of name:       att-field
Type of attribute:  both session and media level
Subject to charset: No
Purpose:            RFC 7826
Reference:          RFC 7826
Values:             See ABNF definition
```

## 22.16. Media Type Registration for text/parameters

Type name: text

Subtype name: parameters

Required parameters:

Optional parameters: charset: The charset parameter is applicable to the encoding of the parameter values. The default charset is UTF-8, if the 'charset' parameter is not present.

Encoding considerations: 8bit

Security considerations: This format may carry any type of parameters. Some can have security requirements, like privacy, confidentiality, or integrity requirements. The format has no built-in security protection. For the usage, the transport can be protected between server and client using TLS. However, care must be taken to consider if the proxies are also trusted with the parameters in case hop-by-hop security is used. If stored as a file in a file system, the necessary precautions need to be taken in relation to the parameter requirements including object security such as S/MIME [RFC5751].

Interoperability considerations: This media type was mentioned as a fictional example in [RFC2326], but was not formally specified. This has resulted in usage of this media type that may not match its formal definition.

Published specification: RFC 7826, 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): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

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

[FIPS180-4]
National Institute of Standards and Technology (NIST), "Federal Information Processing Standards Publication: Secure Hash Standard (SHS)", DOI 10.6028/NIST.FIPS.180-4, August 2015, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <http://www.rfc-editor.org/info/rfc2617>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, DOI 10.17487/RFC3551, July 2003, <http://www.rfc-editor.org/info/rfc3551>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <http://www.rfc-editor.org/info/rfc3711>.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, DOI 10.17487/RFC3830, August 2004, <http://www.rfc-editor.org/info/rfc3830>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <http://www.rfc-editor.org/info/rfc3987>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <http://www.rfc-editor.org/info/rfc7595>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <http://www.rfc-editor.org/info/rfc4566>.

[RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, DOI 10.17487/RFC4571, July 2006, <http://www.rfc-editor.org/info/rfc4571>.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <http://www.rfc-editor.org/info/rfc4585>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC4738] Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, DOI 10.17487/RFC4738, November 2006, <http://www.rfc-editor.org/info/rfc4738>.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, DOI 10.17487/RFC5124, February 2008, <http://www.rfc-editor.org/info/rfc5124>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <http://www.rfc-editor.org/info/rfc5322>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <http://www.rfc-editor.org/info/rfc5751>.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <http://www.rfc-editor.org/info/rfc5761>.

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, DOI 10.17487/RFC5888, June 2010, <http://www.rfc-editor.org/info/rfc5888>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <http://www.rfc-editor.org/info/rfc7615>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <http://www.rfc-editor.org/info/rfc7616>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <http://www.rfc-editor.org/info/rfc7617>.

[RFC7825] Goldberg, J., Westerlund, M., and T. Zeng, "A Network Address Translator (NAT) Traversal Mechanism for Media Controlled by Real-Time Streaming Protocol (RTSP)", RFC 7825, DOI 10.17487/RFC7825, December 2016, <http://www.rfc-editor.org/info/rfc7825>.

[RTP-CIRCUIT-BREAKERS]
Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", Work in Progress, draft-ietf-avtcore-rtp-circuit-breakers-13, February 2016.

[SMPTE-TC] Society of Motion Picture and Television Engineers, "ST 12-1:2008 For Television -- Time and Control Code", DOI 10.5594/SMPTE.ST12-1.2008, February 2008, <http://ieeexplore.ieee.org/servlet/opac?punumber=7289818>.

[TS-26234] 3rd Generation Partnership Project (3GPP), "Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs", Technical Specification 26.234, Release 13, September 2015, <http://www.3gpp.org/DynaReport/26234.htm>.

## 23.2. Informative References

[ISO.13818-6.1995]
International Organization for Standardization, "Information technology -- Generic coding of moving pictures and associated audio information - part 6: Extension for DSM-CC", ISO Draft Standard 13818-6:1998, October 1998, <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=25039>.

[ISO.8601.2000]
International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO/IEC Standard 8601, December 2000.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <http://www.rfc-editor.org/info/rfc791>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <http://www.rfc-editor.org/info/rfc1123>.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, DOI 10.17487/RFC2068, January 1997, <http://www.rfc-editor.org/info/rfc2068>.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, DOI 10.17487/RFC2326, April 1998, <http://www.rfc-editor.org/info/rfc2326>.

[RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, DOI 10.17487/RFC2663, August 1999, <http://www.rfc-editor.org/info/rfc2663>.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, DOI 10.17487/RFC2974, October 2000, <http://www.rfc-editor.org/info/rfc2974>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <http://www.rfc-editor.org/info/rfc3261>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <http://www.rfc-editor.org/info/rfc3264>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, DOI 10.17487/RFC4145, September 2005, <http://www.rfc-editor.org/info/rfc4145>.

[RFC4567] Arkko, J., Lindholm, F., Naslund, M., Norrman, K., and E. Carrara, "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)", RFC 4567, DOI 10.17487/RFC4567, July 2006, <http://www.rfc-editor.org/info/rfc4567>.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <http://www.rfc-editor.org/info/rfc4588>.

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, DOI 10.17487/RFC4855, February 2007, <http://www.rfc-editor.org/info/rfc4855>.

[RFC4856] Casner, S., "Media Type Registration of Payload Formats in the RTP Profile for Audio and Video Conferences", RFC 4856, DOI 10.17487/RFC4856, February 2007, <http://www.rfc-editor.org/info/rfc4856>.

[RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, DOI 10.17487/RFC5104, February 2008, <http://www.rfc-editor.org/info/rfc5104>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <http://www.rfc-editor.org/info/rfc5245>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <http://www.rfc-editor.org/info/rfc5389>.

[RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, DOI 10.17487/RFC5583, July 2009, <http://www.rfc-editor.org/info/rfc5583>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <http://www.rfc-editor.org/info/rfc6298>.

[Stevens98]
Stevens, W., Fenner, B., and A. Rudoff, "Unix Networking Programming, Volume 1: The Sockets Networking API (3rd Edition)", 1998.

## 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 these are examples and the normative and syntax descriptions in the other sections take precedence. Please also note that many of the examples have been broken into several lines, where following lines start with whitespace as allowed by the syntax.

### A.1. Media on Demand (Unicast)

This is an example of media-on-demand streaming of media stored in a container file. For the 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 not to write any URI parts that are 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.

Also, this presentation description (SDP) is not cacheable, as the Expires header is set to an equal value with date indicating immediate expiration of its validity.

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: Fri, 20 Dec 2013 10:20:32 +0000
      Content-Type: application/sdp
      Content-Length: 271
      Content-Base: rtsp://example.com/twister.3gp/
      Expires: Fri, 20 Dec 2013 12:20:32 +0000

      v=0
      o=- 2890844256 2890842807 IN IP4 198.51.100.5
      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=00:00:00-00:10:34.10
      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, clock

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="198.51.100.5:9000"/"198.51.100.5:9001"
      Session: OccldOFFq23KwjYpAnBbUr
      Expires: Fri, 20 Dec 2013 12:20:33 +0000
      Date: Fri, 20 Dec 2013 10:20:33 +0000
      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: OccldOFFq23KwjYpAnBbUr
         Accept-Ranges: npt, smpte, clock

   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="198.51.100.5:9002"/"198.51.100.5:9003";

         Session: OccldOFFq23KwjYpAnBbUr
         Expires: Fri, 20 Dec 2013 12:20:33 +0000
         Date: Fri, 20 Dec 2013 10:20:33 +0000
         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: OccldOFFq23KwjYpAnBbUr

   M->C: RTSP/2.0 200 OK
         CSeq: 4
         Server: PhonyServer/1.0
         Date: Fri, 20 Dec 2013 10:20:34 +0000
         Session: OccldOFFq23KwjYpAnBbUr
         Range: npt=30-634.10
         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: OccldOFFq23KwjYpAnBbUr

   # Pause happens 0.87 seconds after starting to play
```

```
M->C: RTSP/2.0 200 OK
      CSeq: 5
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:20:35 +0000
      Session: OccldOFFq23KwjYpAnBbUr
      Range: npt=30.87-634.10

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
      CSeq: 6
      User-Agent: PhonyClient/1.2
      Range: npt=30.87-634.10
      Seek-Style: Next
      Session: OccldOFFq23KwjYpAnBbUr

M->C: RTSP/2.0 200 OK
      CSeq: 6
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:22:13 +0000
      Session: OccldOFFq23KwjYpAnBbUr
      Range: npt=30.87-634.10
      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

C->M: TEARDOWN rtsp://example.com/twister.3gp/ RTSP/2.0
      CSeq: 7
      User-Agent: PhonyClient/1.2
      Session: OccldOFFq23KwjYpAnBbUr

M->C: RTSP/2.0 200 OK
      CSeq: 7
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:31:53 +0000
```

## 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 set up. 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: Fri, 20 Dec 2013 10:20:32 +0000
      Content-Type: application/sdp
      Content-Length: 271
      Content-Base: rtsp://example.com/twister.3gp/
      Expires: Fri, 20 Dec 2013 12:20:32 +0000

      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=00:00:00-00:10:34.10
      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, clock
      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, clock
      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
      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="198.51.100.5:9000"/"198.51.100.5:9001";
                 ssrc=93CB001E
      Session: OccldOFFq23KwjYpAnBbUr
      Expires: Fri, 20 Dec 2013 12:20:32 +0000
      Date: Fri, 20 Dec 2013 10:20:32 +0000
      Accept-Ranges: npt
      Pipelined-Requests: 7654
      Media-Properties: Random-Access=0.2, Immutable, 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="198.51.100.5:9002"/"198.51.100.5:9003";
                 ssrc=A813FC13
      Session: OccldOFFq23KwjYpAnBbUr
      Expires: Sat, 21 Dec 2013 10:20:32 +0000
      Date: Fri, 20 Dec 2013 10:20:32 +0000
      Accept-Range: NPT
      Pipelined-Requests: 7654
      Media-Properties: Random-Access=0.8, Immutable, Unlimited
```

```
M->C: RTSP/2.0 200 OK
      CSeq: 4
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:20:32 +0000
      Session: OccldOFFq23KwjYpAnBbUr
      Range: npt=0-623.10
      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
      Pipelined-Requests: 7654
```

### A.3. Secured Media Session for On-Demand Content

This example is basically the above example (Appendix A.2), but now including establishment of SRTP crypto contexts to get a secured media delivery. First of all, the client attempts to fetch this insecurely, but the server redirects to a URI indicating a requirement on using a secure connection for the RTSP messages. The client establishes a TCP/TLS connection, and the session description is retrieved to determine what media resources need to be set up. In the this session description, secure media (SRTP) is indicated. In the next step, the client sends the necessary SETUP requests including MIKEY messages. This is pipelined with a 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.

Note: The MIKEY messages below are not valid MIKEY messages and are Base64-encoded random data to represent where the MIKEY messages would go.

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

M->C: RTSP/2.0 301 Moved Permanently
      CSeq: 1
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:25:32 +0000
      Location: rtsps://example.com/twister.3gp

C->M: Establish TCP/TLS connection and verify server's
      certificate that represents example.com.
      Used for all below RTSP messages.
```

```
C->M: DESCRIBE rtsps://example.com/twister.3gp RTSP/2.0
      CSeq: 2
      User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
      CSeq: 2
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:25:33 +0000
      Content-Type: application/sdp
      Content-Length: 271
      Content-Base: rtsps://example.com/twister.3gp/
      Expires: Fri, 20 Dec 2013 12:25:33 +0000

      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=00:00:00-00:10:34.10
      t=0 0
      m=audio 0 RTP/SAVP 0
      a=control: trackID=1
      m=video 0 RTP/SAVP 26
      a=control: trackID=4

C->M: SETUP rtsps://example.com/twister.3gp/trackID=1 RTSP/2.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Require: play.basic
      Transport: RTP/SAVP;unicast;dest_addr=":8000"/":8001";
         MIKEY=VGhpcyBpcyB0aGUgZmlyc3Qgc3RyZWFtcyBNSUtFWSBtZXNzYWdl
      Accept-Ranges: npt, smpte, clock
      Pipelined-Requests: 7654

C->M: SETUP rtsps://example.com/twister.3gp/trackID=4 RTSP/2.0
      CSeq: 4
      User-Agent: PhonyClient/1.2
      Require: play.basic
      Transport: RTP/SAVP;unicast;dest_addr=":8002"/":8003";
         MIKEY=TUlLRVkgZm9yIHN0cmVhbSB0d2lzdGVyLjNncC90cmFja0lEPTQ=
      Accept-Ranges: npt, smpte, clock
      Pipelined-Requests: 7654
```

```
   C->M: PLAY rtsps://example.com/twister.3gp/ RTSP/2.0
         CSeq: 5
         User-Agent: PhonyClient/1.2
         Range: npt=0-
         Seek-Style: RAP
         Pipelined-Requests: 7654

   M->C: RTSP/2.0 200 OK
         CSeq: 3
         Server: PhonyServer/1.0
         Transport: RTP/SAVP;unicast;
            dest_addr="192.0.2.53:8000"/"192.0.2.53:8001";
            src_addr="198.51.100.5:9000"/"198.51.100.5:9001";
            ssrc=93CB001E;
            MIKEY=TUlLRVkgUmVzcG9uc2UgdHdpc3Rlci4zZ3AvdHJhY2tJRD0x
         Session: OccldOFFq23KwjYpAnBbUr
         Expires: Fri, 20 Dec 2013 12:25:34 +0000
         Date: Fri, 20 Dec 2013 10:25:34 +0000
         Accept-Ranges: npt
         Pipelined-Requests: 7654
         Media-Properties: Random-Access=0.2, Immutable, Unlimited

   M->C: RTSP/2.0 200 OK
         CSeq: 4
         Server: PhonyServer/1.0
         Transport: RTP/SAVP;unicast;
            dest_addr="192.0.2.53:8002"/"192.0.2.53:8003;
            src_addr="198.51.100.5:9002"/"198.51.100.5:9003";
            ssrc=A813FC13;
            MIKEY=TUlLRVkgUmVzcG9uc2UgdHdpc3Rlci4zZ3AvdHJhY2tJRD00
         Session: OccldOFFq23KwjYpAnBbUr
         Expires: Fri, 20 Dec 2013 12:25:34 +0000
         Date: Fri, 20 Dec 2013 10:25:34 +0000
         Accept-Range: NPT
         Pipelined-Requests: 7654
         Media-Properties: Random-Access=0.8, Immutable, Unlimited
```

```
M->C: RTSP/2.0 200 OK
      CSeq: 5
      Server: PhonyServer/1.0
      Date: Fri, 20 Dec 2013 10:25:34 +0000
      Session: OccldOFFq23KwjYpAnBbUr
      Range: npt=0-623.10
      Seek-Style: RAP
      RTP-Info: url="rtsps://example.com/twister.3gp/trackID=4"
         ssrc=0D12F123:seq=12345;rtptime=3450012,
        url="rtsps://example.com/twister.3gp/trackID=1"
         ssrc=4F312DD8:seq=54321;rtptime=2876889;
      Pipelined-Requests: 7654
```

## A.4. Media on Demand (Unicast)

An alternative example of media on demand with a few 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 and the protocol stack.

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/1.1 200 OK
      Date: Wed, 23 Jan 2013 15:35:06 GMT
      Content-Type: application/sdp
      Content-Length: 278
      Expires: Thu, 24 Jan 2013 15:35:06 GMT

      v=0
      o=- 2890844526 2890842807 IN IP4 198.51.100.5
      s=RTSP Session
      e=adm@example.com
      c=IN IP4 0.0.0.0
      a=range:npt=00:00:00-01:49:34
      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, clock

A->C: RTSP/2.0 200 OK
      CSeq: 1
      Session: OccldOFFq23KwjYpAnBbUr
      Transport: RTP/AVP/UDP;unicast;
                 dest_addr="192.0.2.53:3056"/"192.0.2.53:3057";
                 src_addr="198.51.100.5:5000"/"198.51.100.5:5001"
      Date: Wed, 23 Jan 2013 15:35:12 +0000
      Server: PhonyServer/1.0
      Expires: Thu, 24 Jan 2013 15:35:12 +0000
      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, clock
```

```
   V->C: RTSP/2.0 200 OK
         CSeq: 1
         Session: P5it3pMo6xHkjUcDrNkBjf
         Transport: RTP/AVP/UDP;unicast;
            dest_addr="192.0.2.53:3058"/"192.0.2.53:3059";
            src_addr="198.51.100.5:5002"/"198.51.100.5:5003"
         Date: Wed, 23 Jan 2013 15:35:12 +0000
         Server: PhonyServer/1.0
         Cache-Control: public
         Expires: Thu, 24 Jan 2013 15:35:12 +0000
         Accept-Ranges: npt, smpte
         Media-Properties: Random-Access=1.2, Immutable, Unlimited

   C->V: PLAY rtsp://video.example.com/twister/video RTSP/2.0
         CSeq: 2
         User-Agent: PhonyClient/1.2
         Session: P5it3pMo6xHkjUcDrNkBjf
         Range: smpte=0:10:00-

   V->C: RTSP/2.0 200 OK
         CSeq: 2
         Session: P5it3pMo6xHkjUcDrNkBjf
         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: Wed, 23 Jan 2013 15:35:13 +0000

   C->A: PLAY rtsp://audio.example.com/twister/audio.en RTSP/2.0
         CSeq: 2
         User-Agent: PhonyClient/1.2
         Session: OccldOFFq23KwjYpAnBbUr
         Range: smpte=0:10:00-

   A->C: RTSP/2.0 200 OK
         CSeq: 2
         Session: OccldOFFq23KwjYpAnBbUr
         Range: smpte=0:10:00-1:49:23
         Seek-Style: First-Prior
         RTP-Info: url="rtsp://audio.example.com/twister/audio.en"
                   ssrc=3D124F01:seq=876655;rtptime=1032181
         Server: PhonyServer/1.0
         Date: Wed, 23 Jan 2013 15:35:13 +0000
```

```
C->A: TEARDOWN rtsp://audio.example.com/twister/audio.en RTSP/2.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Session: OccldOFFq23KwjYpAnBbUr

A->C: RTSP/2.0 200 OK
      CSeq: 3
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000

C->V: TEARDOWN rtsp://video.example.com/twister/video RTSP/2.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Session: P5it3pMo6xHkjUcDrNkBjf

V->C: RTSP/2.0 200 OK
      CSeq: 3
      Server: PhonyServer/2.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000
```

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.5. Single-Stream Container Files

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

```
C->S: DESCRIBE rtsp://foo.example.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.example.com/test.wav/
      Content-type: application/sdp
      Content-length: 163
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000
      Expires: Thu, 24 Jan 2013 15:36:52 +0000

      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.example.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, clock

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="198.51.100.5:6970"/"198.51.100.5:6971";
          mode="PLAY";ssrc=EAB98712
      CSeq: 2
      Session: NYkqQYKk0bb12BY3goyoyO
      Expires: Thu, 24 Jan 2013 15:36:52 +0000
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000
      Accept-Ranges: npt
      Media-Properties: Random-Access=0.5, Immutable, Unlimited
```

```
C->S: PLAY rtsp://foo.example.com/test.wav/ RTSP/2.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Session: NYkqQYKk0bb12BY3goyoyO

S->C: RTSP/2.0 200 OK
      CSeq: 3
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000
      Session: NYkqQYKk0bb12BY3goyoyO
      Range: npt=0-600
      Seek-Style: RAP
      RTP-Info: url="rtsp://foo.example.com/test.wav/streamid=0"
         ssrc=0D12F123:seq=981888;rtptime=3781123
```

Note the different URI in the SETUP command and then the switch back to the aggregate URI in the PLAY command. This makes complete sense when there are multiple streams with aggregate control, but it is less than intuitive in the special case where the number of streams is one. However, the server has declared the aggregated control URI in the SDP; therefore, this is legal.

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

```
C->S: PLAY rtsp://example.com/test.wav/streamid=0 RTSP/2.0
      CSeq: 3
      User-Agent: PhonyClient/1.2
      Session: NYkqQYKk0bb12BY3goyoyO
```

## A.6. 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>
        ...
        <a href "rtsp://live.example.com/concert/audio">
           Streamed Live Music performance </a>
        ...
      </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: Wed, 23 Jan 2013 15:36:52 +0000
      Supported: play.basic

      v=0
      o=- 2890844526 2890842807 IN IP4 192.0.2.5
      s=RTSP Session
      t=0 0
      m=audio 3456 RTP/AVP 0
      c=IN IP4 233.252.0.54/16
      a=control: rtsp://live.example.com/concert/audio
      a=range:npt=0-
```

```
C->M: SETUP rtsp://live.example.com/concert/audio RTSP/2.0
      CSeq: 2
      Transport: RTP/AVP;multicast;
           dest_addr="233.252.0.54:3456"/"233.252.0.54:3457";ttl=16
      Accept-Ranges: npt, smpte, clock
      User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
      CSeq: 2
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000
      Transport: RTP/AVP;multicast;
           dest_addr="233.252.0.54:3456"/"233.252.0.54:3457";ttl=16
           ;ssrc=4D12AB92/0DF876A3
      Session: qHj4jidpmF6zy9v9tNbtxr
      Accept-Ranges: npt, clock
      Media-Properties: No-Seeking, Time-Progressing, Time-Duration=0


C->M: PLAY rtsp://live.example.com/concert/audio RTSP/2.0
      CSeq: 3
      Session: qHj4jidpmF6zy9v9tNbtxr
      User-Agent: PhonyClient/1.2

M->C: RTSP/2.0 200 OK
      CSeq: 3
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:52 +0000
      Session: qHj4jidpmF6zy9v9tNbtxr
      Seek-Style: Next
      Range:npt=1256-
      RTP-Info: url="rtsp://live.example.com/concert/audio"
                ssrc=0D12F123:seq=1473; rtptime=80000
```

## A.7. Capability Negotiation

This example illustrates how the client and server determine their capability to support a special feature, in this case, "play.scale". The server, through the client request and the included Supported header, learns that 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 media delivery functions (play.basic), the extended functionality of time scaling of content (play.scale), and one "example.com" proprietary feature (com.example.flight). The client also learns the methods supported (Public header) by the server for the indicated resource.

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

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

When the client sends its SETUP request, it tells the server that it 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, clock
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 3
      Session: OccldOFFq23KwjYpAnBbUr
      Transport: RTP/AVP/UDP;unicast;
         dest_addr="192.0.2.53:3056"/"192.0.2.53:3057";
         src_addr="198.51.100.5:5000"/"198.51.100.5:5001"
      Server: PhonyServer/2.0
      Accept-Ranges: npt, smpte
      Media-Properties: Random-Access=0.8, Immutable, 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. It is probably easiest to think of this as the server's state and then view the client as needing to track what it believes the server's state will be based on sent or received RTSP messages. Thus, in most cases, the state tables below can be read as: if the client does X, and assuming it fulfills any prerequisite(s), the (server) state will move to the new state and the indicated response will returned. However, there are also server-to-client notifications or requests, where the action describes what notification or request will occur, its requisites, what new state will result after the server has received the response, as well as describing the client's response to the action.

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 are part of the session, it is in aggregated control.

The below state machine is an informative description of the protocol's behavior. In case of ambiguity with the earlier parts of this specification, the description in the earlier parts take precedence.

## B.1. States

The state machine contains three states, described below. For each state, there exists a table that shows which requests and events are allowed and whether they will result in a state change.

Init: Initial state, no session exists.

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 they are explained below.

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

RP: Resume point, the point in the presentation time line at which a request to continue playing 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 timeline at which the presentation was paused.

Prs: Presentation, the complete multimedia presentation.

RedP: Redirect Point, the point in the presentation timeline 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 on which this state is allowed to act. The events that are method names are, unless noted, requests with the given method in the direction client to server (C->S). In some cases, there exists one or more requisites. The response column tells what type of response actions should be performed. Possible actions that are requested for an event include: response codes, e.g., 200, headers that need to be included in the response, setting of state variables, or settings of other session-related parameters. The new state column tells which state the state machine changes to.

The response to a valid request meeting the requisites is normally a 2xx (SUCCESS) unless otherwise 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 occurs, the appropriate response code is to 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 being ended and its state changed to Init. A response code of 304 results in no state change. However, there are restrictions to when a 3rr response may be used. A 5xx response does not result in any change of the session state, except if the error is not possible to recover from. An unrecoverable error results in the ending of the session. In the general case, if it can't be determined whether or not it was an unrecoverable error, the client will be required to test. In the case that the next request after a 5xx is responded to with a 454 (Session Not Found), the client knows that the session has ended. For any request message that cannot be responded to within the time defined in Section 10.4, a 100 response must be sent.

The server will time out the session after the period of time specified in the SETUP response, if no activity from the client is detected. Therefore, there exists 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 needs to be other than any of the media that are part of the session. This applies to all states.

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

Table 9: Non-State-Machine Changing Events

The methods in Table 9 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 allow the Session header will also update the keep-alive timer for the session.

| Action | Requisite | New State | Response |
|---|---|---|---|
| SETUP | | Ready | NRM=1, RP=0.0 |
| SETUP | Needs Redirect | Init | 3rr Redirect |
| S -> C: REDIRECT | No Session hdr | Init | Terminate all SES |

Table 10: State: Init

The initial state of the state machine (Table 10) 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 or server by a 3rr response.

| Action | Requisite | New State | Response |
|---|---|---|---|
| SETUP | New URI | Ready | NRM +=1 |
| SETUP | URI Setup prior | Ready | Change transport param |
| TEARDOWN | Prs URI, | Init | No session hdr, NRM = 0 |
| TEARDOWN | md URI,NRM=1 | Init | No Session hdr, NRM = 0 |
| TEARDOWN | md URI,NRM>1 | Ready | Session hdr, NRM -= 1 |
| PLAY | Prs URI, No range | Play | Play from RP |
| PLAY | Prs URI, Range | Play | According to range |
| PLAY | md URI, NRM=1, Range | Play | According to range |
| PLAY | md URI, NRM=1 | Play | Play from RP |
| PAUSE | Prs URI | Ready | Return PP |
| SC:REDIRECT | Terminate-Reason | Ready | Set RedP |
| SC:REDIRECT | No Terminate-Reason time parameter | Init | Session is removed |
| Timeout | | Init | |
| RedP reached | | Init | TEARDOWN of session |

Table 11: State: Ready

In the Ready state (Table 11), some of the actions depend 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 depends on both the Request-URI and the number of media streams within the session. If the Request-URI is the presentation URI, the whole session is torn down. If a media URI is used in the TEARDOWN request and more than one media exists in the session, the session will remain and a session header is 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.

| Action | Requisite | New State | Response |
|---|---|---|---|
| PAUSE | Prs URI | Ready | Set RP to present point |
| End of media | All media | Play | Set RP = End of media |
| End of range | | Play | Set RP = End of range |
| PLAY | Prs URI, No range | Play | Play from present point |
| PLAY | Prs URI, Range | Play | According to range |
| SC:PLAY_NOTIFY | | Play | 200 |
| SETUP | New URI | Play | 455 |
| SETUP | md URI | Play | 455 |
| SETUP | md URI, IFI | Play | Change transport param. |
| TEARDOWN | Prs URI | Init | No session hdr |
| TEARDOWN | md URI,NRM=1 | Init | No Session hdr, NRM=0 |
| TEARDOWN | md URI | Play | 455 |
| SC:REDIRECT | Terminate Reason with Time parameter | Play | Set RedP |
| SC:REDIRECT | | Init | Session is removed |
| RedP reached | | Init | TEARDOWN of session |
| Timeout | | Init | Stop Media playout |

Table 12: State: Play

The Play state table (Table 12) contains a number of requests that need a presentation URI (labeled as Prs URI) to work on (i.e., the presentation URI has to be used as 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, an "End of media" event when all media has finished playing but the session still remains in Play state. An explicit PAUSE request needs to be sent to change the state to Ready. It may appear that there exist automatic transitions in "RedP reached" and "PP reached". However, they are requested and acknowledged before they take place. The time at which the transition will happen is known by looking at the Terminate-Reason header's time parameter and Range header, respectively. If the client sends a request close in time to these transitions, it needs to be prepared for receiving error messages, 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 signaling with regard to RTP.

The available RTP profiles and lower-layer transports are described below along with rules on signaling 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 regard 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 including the "interleaved" parameter.

When using embedded binary data, "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 the sending of RTP [RFC3550] over lower-transport-layer UDP [RFC768] according to the profile "RTP Profile for Audio and Video Conferences with Minimal Control" defined in [RFC3551]. Implementations of RTP/AVP/UDP MUST implement RTCP (Appendix C.1.6). This profile requires one or two unidirectional or bidirectional UDP flows per media stream. The first UDP flow is for RTP and the second is for RTCP. Multiplexing of RTP and RTCP (Appendix C.1.6.4) MAY be used, in which case, a single UDP flow is used for both parts. 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 [RFC768].

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 regard to such RTP packets MAY be defined in future.

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

o The "src_addr" and "dest_addr" parameters MUST contain either 1 or 2 address specifications. Note that two address specifications MAY be provided even if RTP and RTCP multiplexing is negotiated.

o 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.

o The first address specification given in either of the parameters applies to the RTP stream. The second specification, if present, applies to the RTCP stream, unless in the case RTP and RTCP multiplexing is negotiated where both RTP and RTCP will use the first specification.

- The RTP/UDP packets from the server to the client MUST be sent to the address and port given by the first address specification 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 specification of the "dest_addr" parameter, unless RTP and RTCP multiplexing has been negotiated, in which case RTCP MUST be sent to the first address specification. If no second pair is specified and RTP and RTCP multiplexing has not been negotiated, 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 specification of the "src_addr" parameter, unless RTP and RTCP multiplexing has been negotiated, in which case RTCP MUST be sent to the first address specification. If no second pair is specified and RTP and RTCP multiplexing has not been negotiated, 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 specification of the "src_addr" parameter.

- RTP and RTCP packets SHOULD be sent from the corresponding receiver port, i.e., RTCP packets from the server should be sent from the "src_addr" parameters second address port pair, unless RTP and RTCP multiplexing has been negotiated in which case the first address port pair is used.

### 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 sessions 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 to be used [RFC4585]. This configuration 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 context be established. The default key-management unless otherwise signaled SHALL be MIKEY in RSA-R mode as defined in Appendix C.1.4.1 and not according to the procedure defined in "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)" [RFC4567]. The reason is that RFC 4567 sends the initial MIKEY message in SDP, thus, both requiring the usage of the DESCRIBE method and forcing the server to keep state for clients performing DESCRIBE in anticipation that they might require key management.

MIKEY is selected as the default method for establishing SRTP cryptographic context within an RTSP session as it can be embedded in the RTSP messages while still ensuring confidentiality of content of the keying material, even when using hop-by-hop TLS security for the RTSP messages. This method also supports pipelining of the RTSP messages.

#### C.1.4.1. MIKEY Key Establishment

This method for using MIKEY [RFC3830] to establish the SRTP cryptographic context is initiated in the client's SETUP request, and the server's response to the SETUP carries the MIKEY response. This ensures that the crypto context establishment happens simultaneously with the establishment of the media stream being protected. By using MIKEY's RSA-R mode [RFC4738] the client can be the initiator and still allow the server to set the parameters in accordance with the actual media stream.

The SRTP cryptographic context establishment is done according to the following process:

1. The client determines that SAVP or SAVPF shall be used from the media-description format, e.g., SDP. If no other key-management method is explicitly signaled, then MIKEY SHALL be used as defined herein. The use of SRTP with RTSP is only defined with MIKEY with keys established as defined in this section. Future documents may define how an RTSP implementation treats SDP that indicates some other key mechanism to be used. The need for such specification includes [RFC4567], which is not defined for use in RTSP 2.0 within this document.

2. The client SHALL establish a TLS connection for RTSP messages, directly or hop-by-hop with the server. If hop-by-hop TLS security is used, the User method SHALL be indicated in the Accept-Credentials header. Note that using hop-by-hop does allow the proxy to insert itself as a man in the middle. This can also occur in the MIKEY exchange by the proxy providing one of its certificates rather than the server's in the Connection-Credentials header. Therefore, the client SHALL validate the server certificate.

3. The client retrieves the server's certificate from a direct TLS connection or hop-by-hop from a Connection-Credentials header. The client then checks that the server certificate is valid and belongs to the server.

4. The client forms the MIKEY Initiator message using RSA-R mode in unicast mode as specified in [RFC4738]. The client SHOULD use the same certificate for TLS and MIKEY to enable the server to bind the two together. The client's certificate SHALL be included in the MIKEY message. The client SHALL indicate its SRTP capabilities in the message.

5. The MIKEY message from the previous step is base64-encoded [RFC4648] and becomes the value of the MIKEY parameter that is included in the transport specification(s) that specifies an SRTP-based profile (SAVP, SAVPF) in the SETUP request.

6. Any proxy encountering the MIKEY parameter SHALL forward it without modification. A proxy that is required to understand the Transport specifications will need to understand SAVP/SAVPF with MIKEY to enable the default keying for SRTP-protected media streams. If such a proxy does not support SAVP/SAVPF with MIKEY, it will discard the whole transport specification. Most types of proxies can easily support SAVP and SAVPF with MIKEY. If a client encounters a proxy not supporting SAVP/SAVPF with MIKEY, the client should attempt bypassing that proxy.

7. The server, upon receiving the SETUP request, will need to decide upon the transport specification to use, if multiple are included by the client. In the determination of which transport specifications are supported and preferred, the server SHOULD decode the MIKEY message to take the embedded SRTP parameters into account. If all transport spec require SRTP but no MIKEY parameter or other supported keying method is included, the server SHALL respond with 403 (Forbidden).

8. Upon generating a response, the following outcomes can occur:

   * A transport spec not using SRTP and MIKEY is selected. Thus, the response will not contain any MIKEY parameters.

   * A transport spec using SRTP and MIKEY is selected but an error is encountered in the MIKEY processing. In this case, an RTSP error response code of 466 (Key Management Error) SHALL be used. A MIKEY message describing the error MAY be included.

   * A transport spec using SRTP and MIKEY is selected and a MIKEY response message can be created. The server SHOULD use the same certificate for TLS and in MIKEY to enable the client to bind the two together. If a different certificate is used, it SHALL be included in the MIKEY message. It is RECOMMENDED that the envelope key-cache type be set to 'Cache' and that a single envelope key is reused for all MIKEY messages to the client. That message is included in the MIKEY parameter part of the single selected transport specification in the SETUP response. The server will set the SRTP parameters as preferred for this media stream within the supported range by the client.

9. The server transmits the SETUP response back to the client.

10. The client receives the SETUP response and, if the response code indicates a successful request, it decodes the MIKEY message and establishes the SRTP cryptographic context from the parameters in the MIKEY response.

In the above method, the client's certificate may be self signed in cases where the client's identity is not necessary to authenticate and the security goal is only to ensure that the RTSP signaling client is the same as the one receiving the SRTP security context.

### C.1.5. SAVPF/UDP

The RTP profile "Extended Secure RTP Profile for Real-time Transport Control Protocol (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 AVPF MUST also apply for SAVPF.

The usage of SRTP requires that a cryptographic context be established. The default mechanism for establishing that security association is to use MIKEY[RFC3830] with RTSP as defined in Appendix C.1.4.1.

### C.1.6. RTCP Usage with RTSP

RTCP has several usages when RTP is implemented for media transport as explained below. Thus, 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. This 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. Bitrate Adaption

RTCP Receiver reports and any additional feedback from the client MUST be used to adapt the bitrate 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 terms (some seconds) the used bitrate and adapting it so that the RTP sender sends at a bitrate comparable to what a TCP sender would achieve on average over the same path.

To ensure that the implementation's adaptation mechanism has a well-defined outer envelope, all implementations using a non-congestion-controlled unicast transport protocol, like UDP, MUST implement "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions" [RTP-CIRCUIT-BREAKERS].

#### C.1.6.4. RTP and RTCP Multiplexing

RTSP can be used to negotiate the usage of RTP and RTCP multiplexing as described in [RFC5761]. 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 NATs [RFC2663].

Content must be prepared with some consideration for RTP and RTCP multiplexing, mainly ensuring that the RTP payload types used do 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 support RTP and RTCP multiplexing, 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". Servers and Clients SHOULD support RTP and RTCP multiplexing.

For capability exchange, an RTSP feature tag for RTP and RTCP multiplexing is defined: "setup.rtp.rtcp.mux".

To minimize the risk of negotiation failure while using RTP and RTCP multiplexing, some recommendations are here provided. If the session description includes explicit indication of support ("a=rtcp-mux" in SDP), then an RTSP agent can safely create a SETUP request with a transport specification with only a single "dest_addr" parameter address specification. If no such explicit indication is provided, then even if the feature tag "setup.rtp.rtcp.mux" is provided in a Supported header by the RTSP server or the feature tag included in the Required header in the SETUP request, the media resource may not support RTP and RTCP multiplexing. Thus, to maximize the probability of successful negotiation, the RTSP agent is recommended to include two "dest_addr" parameter address specifications in the first or first set (if pipelining is used) of SETUP request(s) for any media resource aggregate. That way, the RTSP server can accept RTP and RTCP multiplexing and only use the first address specification or, if not, use both specifications. The RTSP agent, after having received the response for a successful negotiation of the usage of RTP and RTCP multiplexing, can then release the resources associated with the second address specification.

## C.2. RTP over TCP

Transport of RTP over TCP can be done in two ways: over independent TCP connections using [RFC4571] or interleaved in the RTSP 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. If TLS is used, both RTSP messages and the binary data will be protected by TLS.

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

### C.2.2. RTP over Independent TCP

In this section, the sending of RTP [RFC3550] over lower-layer transport TCP [RFC793] according to "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport" [RFC4571] is described. This section 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 address specifications need to be included in the "dest_addr" parameter. If the client wishes to use RTP without RTCP, one address specification is included in the "dest_addr" parameter. If the client wishes to multiplex RTP and RTCP on a single transport flow (see Appendix C.1.6.4), one or two address specifications are included in the "dest_addr" parameter in addition to the "RTCP-mux" transport parameter. Two address specifications are allowed to facilitate successful negotiation when the server or content can't support RTP and RTCP multiplexing. 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 18.54) 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 in the address specifications in the "dest_addr" parameter 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" parameter's address specification port value for RTP MUST be set to the TCP port number on which the client is expecting to receive the TCP connection for RTP, and the "dest_addr" address specification port value for RTCP MUST be set to the TCP port number on which the client is expecting to receive the TCP connection for RTCP. In the case that the client wishes to multiplex RTP and RTCP on a single transport flow, the "RTCP-mux" parameter is included and one or two "dest_addr" parameter address specifications are included, as mentioned earlier in this section.

Upon receipt of a non-interleaved RTP/AVP/TCP SETUP request, if 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 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 on which the server is willing to receive a TCP connection for RTP and (if the client requested a TCP connection for RTCP by specifying two "dest_addr" address specifications) a TCP/RTCP connection. If a server sets setup to "active", the ports specified in "src_addr" address specifications MUST be set to 9. The server MAY use the "ssrc" parameter, following the guidance in Section 18.54. The server sets only one address specification in the case that the client has indicated only a single address specification or in case RTP and RTCP multiplexing was requested and accepted by the server. Port ordering for "src_addr" follows the rules for RTP/AVP/UDP.

Servers MUST support taking the passive role and MAY support taking the active role. Servers with a public IP address take the passive role, thus enabling clients behind NATs and firewalls a better chance of successful connect to the server by actively connecting outwards. Therefore, the clients are RECOMMENDED to take the active role.

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 17.4.28) 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. Unless RTP and RTCP multiplexing has been negotiated; in which case, RTP and RTCP will flow over a common TCP connection. As in the RTP/UDP case, client-to-server traffic on an RTP-only TCP session 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 connection defined in Section 14.

A successful PAUSE request for 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 by the RTSP client as soon as possible.

Subsequent SETUP requests using a URI already set up in an RTSP session using an RTP/AVP/TCP transport specification may be ambiguous in the following way: does the client wish to open up a new TCP connection for RTP or RTCP for the URI, or does the client wish to continue using the existing TCP connections? The client SHOULD use the "connection" parameter (defined in Section 18.54) on the Transport line to make its intention clear (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 preexisting connections, after waiting a suitable period for any stray RTP or RTCP packets to arrive.

The usage of SRTP, i.e., either SAVP or SAVPF profiles, requires that a security association be established. The default mechanism for establishing that security association is to use MIKEY[RFC3830] with RTSP as defined Appendix C.1.4.1.

Below, a rewritten version of the example "Media on Demand" (Appendix A.1) shows the use of 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: Wed, 23 Jan 2013 15:36:52 +0000
      Content-Type: application/sdp
      Content-Length: 227
      Content-Base: rtsp://example.com/twister.3gp/
      Expires: Thu, 24 Jan 2013 15:36:52 +0000

      v=0
      o=- 2890844256 2890842807 IN IP4 198.51.100.34
      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=00:00:00-00:10:34.10
      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, clock

M->C: RTSP/2.0 200 OK
      CSeq: 2
      Server: PhonyServer/1.0
      Transport: RTP/AVP/TCP;unicast;
                 dest_addr=":9"/":9";
                 src_addr="198.51.100.5:53478"/"198.51.100:54091";
                 setup=passive;connection=new;ssrc=93CB001E
      Session: OccldOFFq23KwjYpAnBbUr
      Expires: Thu, 24 Jan 2013 15:36:52 +0000
      Date: Wed, 23 Jan 2013 15:36:52 +0000
      Accept-Ranges: npt
      Media-Properties: Random-Access=0.8, Immutable, Unlimited
```

```
C->M: TCP Connection Establishment x2

C->M: PLAY rtsp://example.com/twister.3gp/ RTSP/2.0
      CSeq: 4
      User-Agent: PhonyClient/1.2
      Range: npt=30-
      Session: OccldOFFq23KwjYpAnBbUr

M->C: RTSP/2.0 200 OK
      CSeq: 4
      Server: PhonyServer/1.0
      Date: Wed, 23 Jan 2013 15:36:54 +0000
      Session: OccldOFFq23KwjYpAnBbUr
      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 continuous media stream at the RTP layer. A client may also immediately follow a completed PLAY request with 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 an ongoing PLAY request allows the media layer rendering the RTP stream to do so continuously without being affected by jumps in media-clock time. The RTP timestamps for the new media range are 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 signaling operations arriving after a completed PLAY operation, the request timing will result in that media becoming non-continuous. The server becomes unable to send the media so that it arrives timely and still carries timestamps to make the media stream continuous. In these situations, the server will produce RTP streams where there are gaps in the RTP timeline for the media. 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, such as live streams, any media-clock jump is commonly the result of a 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 already cache media or perform time shifting, the rewriting should impose only a minor burden.

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 that the delivery was halted. An 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 the 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 contain loss reports for all the packets that were a part of the discontinuity. A client cannot rely on the fact that a server will align when resuming play, even if it is RECOMMENDED. The RTP-Info header will provide information on how the server acts in each case.

> One 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 an 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: ymIqLXufHkMHGdtENdblWK
  Range: npt=10-15
  User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
  CSeq: 4
  Session: ymIqLXufHkMHGdtENdblWK
  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 delivery, the server sends a PLAY_NOTIFY.

```
S->C: PLAY_NOTIFY rtsp://example.com/fizzle RTSP/2.0
      CSeq: 5
      Notify-Reason: end-of-stream
      Request-Status: cseq=4 status=200 reason="OK"
      Range: npt=-15
      RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
         ssrc=0D12F123:seq=49;rtptime=39200
      Session: ymIqLXufHkMHGdtENdblWK

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

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: 6
      Session: ymIqLXufHkMHGdtENdblWK
      Range: npt=18-20
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 6
      Session: ymIqLXufHkMHGdtENdblWK
      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 are played, then the client requests 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 timestamps 0 through 39,200. The second segment consists of RTP packets with sequence numbers 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.

> RFC 3550 [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: ymIqLXufHkMHGdtENdblWK
      Range: npt=10-15

      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 4
      Session: ymIqLXufHkMHGdtENdblWK
      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: ymIqLXufHkMHGdtENdblWK
      User-Agent: PhonyClient/1.2

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

S->C: RTSP/2.0 200 OK
      CSeq: 6
      Session: ymIqLXufHkMHGdtENdblWK
      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 that takes 15 ms 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: it may be required to reset some of the state to ensure the correct media decoding and the usual jitter-buffer handling when issuing a PLAY request.

## C.5. RTSP/RTP Integration

For certain data types, 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 18.46), RTP timestamps should correspond to the rendering timing. For example, when playing video recorded at 30 frames per second at a scale of two and speed (Section 18.50) 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 by noting the RTP timestamp value of the first packet arriving after repositioning. The sequence parameter of the RTP-Info (Section 18.45) 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 an RTCP

BYE message until it has finished using a SSRC. A server SHOULD keep using an 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 normally force the media sender to change its SSRC in accordance with the RTP specification [RFC3550]. However, an RTSP server may wait and see if the client changes and thus resolve the conflict to minimize the impact. As media sender, SSRC change 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 not to select the same SSRC(s) as the server indicates in the ssrc Transport header parameter. Any SSRC signaled in the Transport header MUST be avoided. A client detecting a collision prior to sending any RTP or RTCP messages SHALL also select a new SSRC.

## C.11. Future Additions

It is the intention that any future protocol or profile regarding media delivery and lower transport should be easy to add to RTSP. This section provides the necessary steps that need to be met.

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

- o The protocol or profile needs to define a name tag representing it. This tag is required to be an ABNF "token" to be possible to use in the Transport header specification.

- o The useful combinations of protocol, profiles, and lower-layer transport for this extension need to be defined. For each combination, declare the necessary parameters to use in the Transport header.

- o For new media protocols, the interaction with RTSP needs to be addressed. One important factor will be the media synchronization. It may be necessary to have new headers similar to RTP info to carry this information.

- o Discussion needs to occur regarding congestion control for media, especially if transport without built-in congestion control is used.

See the IANA Considerations section (Section 22) for information on 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 a 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. Thus, it is worth pointing out that the interpretation of the SDP is done in the context of the SDP receiver, which is the one being configured. This is the same as in SAP [RFC2974]; this differs from SDP Offer/Answer [RFC3264] where each SDP is interpreted in the context of the agent providing it.

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). The SDP extension found in "The Session Description Protocol (SDP) Grouping Framework" [RFC5888] provides such functionality to some degree. Appendix D.4 describes the usage of SDP media line grouping for RTSP.

## 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 [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 the media level unless the presentation only contains a single media stream; in which case, the attribute MAY be present on the session level only and then also apply to that single media stream.

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; thus, it will inherit the entire base URI.

> Note: RFC 2326 was very unclear on the processing of relative URIs 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 URIs in the SDP.

The URI handling for SDPs from container files needs special consideration. For example, let's assume that a container file has the URI: "rtsp://example.com/container.mp4". Let's further assume this URI is the base URI and that there is an 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 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 the control URI including the trailing slash, i.e., "rtsp://example.com/container.mp4/".

> Note: the usage of TrackID in the above is not a 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 whether or not this is allowed. 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 are 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 discussed in Appendix C; further combinations MAY be specified.

Example:

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

### D.1.3. Payload Type(s)

The payload type or types 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 [RFC4856], a media type registered with IANA according to [RFC4855], or an experimental encoding as specified in SDP [RFC4566]). Codec-specific parameters are not specified in this field, but rather in the "fmtp" attribute described below.

The selection of the RTP payload type numbers used may be required to consider RTP and RTCP Multiplexing [RFC5761], if that is to be supported by the server.

### D.1.4. Format-Specific Parameters

Format-specific parameters are conveyed using the "fmtp" media attribute. The syntax of the "fmtp" attribute is specific to the encoding(s) to which the attribute refers. Note that some of the format-specific parameters may be specified outside of the "fmtp" parameters, for example, like the "ptime" attribute for most audio encodings.

### D.1.5. Directionality of Media Stream

The SDP attributes "a=sendrecv", "a=recvonly", and "a=sendonly" provide instructions about the 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, media streams delivered from the RTSP server to the client would be given the "a=recvonly" attribute.

"a=recvonly" in an SDP provided to the RTSP client indicates that media delivery will only occur in the direction from the RTSP server to the client. SDP provided to the RTSP client that lacks any of the directionality attributes ("a=recvonly", "a=sendonly", "a=sendrecv") would be interpreted as having "a=sendrecv". At the time of writing, there exists no RTSP mode suitable for media traffic in the direction from the RTSP client to the server. Thus, all RTSP SDP SHOULD have an "a=recvonly" attribute when using the PLAY mode defined in this document. If future modes are defined for media in the client-to-server direction, then usage of "a=sendonly" or "a=sendrecv" may become suitable to indicate intended media directions.

### D.1.6. Range of Presentation

The "a=range" attribute defines the total time range of the stored session or an individual media. Live sessions that are not seekable can be indicated as specified below; whereas 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 contain media streams of the same duration, the range attribute SHOULD only be used at the session level. In case of different lengths, the range attribute MUST be given at media level for all media and SHOULD NOT be given at the 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 accept PLAY requests within that range.

Servers MUST take care to provide RTSP Range (see Section 18.40) 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 content that is not seekable (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 4.4.1, Section 4.4.2, and Section 4.4.3 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 content that is not seekable 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 defines when the SDP is valid. For on-demand content, 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 not seekable, 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 being unnecessary far into the future.

### D.1.8. Connection Information

In SDP used with RTSP, the "c=" field contains the destination address for the media stream. If a multicast address is specified, the client SHOULD use this address in any SETUP request as destination address, including any additional parameters, such as TTL. 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 18.24) to allow session establishment only 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 an 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.example.com/movie.aud
m=audio 8004 RTP/AVP 3
a=control:rtsp://video.example.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.example.com and video.example.com.

It is recommended that an SDP file contain 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" attribute, which is 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: Wed, 23 Jan 2013 15:36:52 +0000
      Expires: Wed, 23 Jan 2013 16:36:52 +0000
      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 recommended to establish a single RTSP session to the server, and it 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 issue 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. Grouping of Media Lines in SDP

For some types of media, it is desirable to express a relationship between various media components, for instance, for lip synchronization or Scalable Video Codec (SVC) [RFC5583]. This relationship is expressed on the SDP level by grouping of media lines, as described in [RFC5888], and can be exposed to RTSP.

For RTSP, it is mainly important to know how to handle grouped media received by means of SDP, i.e., if the media are under aggregate control (see Appendix D.3) or if aggregate control is not available (see Appendix D.2).

It is RECOMMENDED that grouped media are handled by aggregate control, to give the client the ability to control either the whole presentation or single media.

### D.5. RTSP External SDP Delivery

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

First of all, the SDP needs to contain absolute URIs, since relative will, in most cases, not work as the delivery will not correctly forward the base URI.

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 is 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] in Section 4.2.

## 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 regard 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 delivery to an address other than the controlling agent's (client's).

### 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 the necessary description and synchronization information for the content.

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

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 streams 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 [RFC7230] 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 consists 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 presentations 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 agent controlling the traffic. However, allowing this without appropriate mechanisms for checking that the destination approves of this allows for Distributed DoS (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 actually happens "now"; i.e., very similar to broadcast TV. However, in this case, it is assumed that there exists 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 available for the duration of the event the content covers. An example of such live content could be a music concert that lasts two hours and starts at a predetermined time. Thus, there is a 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 DoS 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 specification has to rely on an 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 a multicast or a conference focus that redistributes 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:

DoS: To avoid an RTSP server from being an unknowing participant in a DoS 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.

## 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 a 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 form is in Section 22.16.

There is 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 implementing application needs to have out-of-band agreement or using feature tag anyway to determine if the endpoint supports the parameters.

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

```
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     = <as defined in Section 20.1>
WSP               = <See RFC 5234> ; Space or HTAB
VCHAR             = <See RFC 5234>
CRLF              = <See RFC 5234>
```

## Appendix G. Requirements for Unreliable Transport of RTSP

This appendix provides guidance for those who want to implement RTSP messages over unreliable transports as has been defined in RTSP 1.0 [RFC2326]. RFC 2326 defined the "rtspu" URI scheme and provided some basic information for the transport of RTSP messages over UDP. The information is being provided here as there has been at least one commercial implementation and compatibility with that should be maintained.

The following points should be considered for an interoperable implementation:

- o Requests 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.

- o The RTT can be estimated as in TCP (RFC 6298) [RFC6298], with an initial round-trip value of 500 ms. An implementation may cache the last RTT measurement as the initial value for future connections.

- o The Timestamp header (Section 18.53) is used to avoid the retransmission ambiguity problem [Stevens98].

- o The registered default port for RTSP over UDP for the server is 554.

- o RTSP messages can be carried over any lower-layer transport protocol that is 8-bit clean.

- o RTSP messages are vulnerable to bit errors and should not be subjected to them.

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

- o CSeq: See Section 18.20. It should be noted that the CSeq header is also required to match requests and responses independent whether a reliable or unreliable transport is used.

- o Timestamp: See Section 18.53

## 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 exists no requirement to implement RTSP 1.0; in fact, this document recommends against it as it is difficult to do in an interoperable way.

A server implementing RTSP 2.0 MUST include an RTSP-Version of "RTSP/2.0" in all responses to requests containing RTSP-Version value of "RTSP/2.0". If a server receives an RTSP 1.0 request, it MAY respond with an 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 an RTSP 1.0 request with an RTSP 2.0 response.

Clients implementing RTSP 2.0 MAY use an OPTIONS request with an RTSP-Version of "RTSP/2.0" to determine whether a server supports RTSP 2.0. If the server responds with either an RTSP-Version of "RTSP/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 State

The behavior in the server when a Play is received in Play state 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 takes 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 an RTSP 2.0 client also supporting 1.0 connects to a 1.0 server.

# Appendix I. Changes

This appendix briefly lists the differences between RTSP 1.0 [RFC2326] and RTSP 2.0 for an informational purpose. For implementers of RTSP 2.0, it is recommended to read carefully through this memo and not to rely on the list of changes below to adapt from RTSP 1.0 to RTSP 2.0, as RTSP 2.0 is not intended to be backwards compatible with RTSP 1.0 [RFC2326] other than the version negotiation mechanism.

### I.1. Brief Overview

The following protocol elements were removed in RTSP 2.0 compared to RTSP 1.0:

- o the RECORD and ANNOUNCE methods and all related functionality (including 201 (Created) and 250 (Low On Storage Space) status codes);

- o the use of UDP for RTSP message transport (due to missing interest and to broken specification);

- o the use of PLAY method for keep-alive in Play state.

The following protocol elements were added or changed in RTSP 2.0 compared to RTSP 1.0:

- o RTSP session TEARDOWN from the server to the client;

- o IPv6 support;

- o extended IANA registries (e.g., transport headers parameters, transport-protocol, profile, lower-transport, and mode);

- o request pipelining for quick session start-up;

- o fully reworked state machine;

- o RTSP messages now use URIs rather than URLs;

- o incorporated much of related HTTP text ([RFC2616]) in this memo, compared to just referencing the sections in HTTP, to avoid ambiguities;

- o the REDIRECT method was expanded and diversified for different situations;

- o Includes a new section about how to set up different media-transport alternatives and their profiles in addition to lower-layer protocols. This caused the appendix on RTP interaction to be moved to the new section instead of being in the part that describes RTP. The section also includes guidelines what to consider when writing usage guidelines for new protocols and profiles;

o Added an asynchronous notification method PLAY_NOTIFY. This method is used by the RTSP server to asynchronously notify clients about session changes while in Play state. To a limited extent, this is comparable with some implementations of ANNOUNCE in RTSP 1.0 not intended for Recording.

## I.2. Detailed List of Changes

The below changes have been made to RTSP 1.0 (RFC 2326) when defining RTSP 2.0. Note that this list does not reflect minor changes in wording or correction of typographical errors.

o The section on minimal implementation was deleted. Instead, the main part of the specification defines the core of RTSP 2.0.

o The Transport header has been changed in the following ways:

  * The ABNF has been changed to define that extensions are possible and that unknown parameters result in servers ignoring the transport specification.

  * To prevent backwards compatibility issues, any extension or new parameter requires the usage of a feature tag combined with the Require header.

  * Syntax ambiguities with the Mode parameter have been resolved.

  * Syntax error with ";" for multicast and unicast has been resolved.

  * Two new addressing parameters have been defined: src_addr and dest_addr. These replace the parameters "port", "client_port", "server_port", "destination", and "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.

  * IANA registries for the transport header parameters, transport-protocol, profile, lower-transport, and mode have been defined.

  * The Transport header's interleaved parameter's text was made more strict and uses 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 extended to allow multiple values for clients supporting this version.

* Clarified the text on the Transport header's "dest_addr" parameters regarding what security precautions the server is required to perform.

o The Range formats have 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" and also format only "Range: smpte" ranges for usage with GET_PARAMETER requests.

  * The npt-hhmmss notation now follows ISO 8601 more strictly.

o RTSP message handling has been changed in the following ways:

  * RTSP messages now use URIs rather than URLs.

  * It has been clarified that a 4xx message due to a missing CSeq header shall be returned without a CSeq header.

  * The 300 (Multiple Choices) response code has been removed.

  * Rules for how to handle the timing out RTSP messages have been added.

  * Extended Pipelining rules allowing for quick session startup.

  * Sequence numbering and proxy handling of sequence numbers have been defined, including cases when responses arrive out of order.

o The HTTP references have been updated to first RFCs 2616 and 2617 and then to RFC 7230-7235. Most of the text has been copied and then altered to fit RTSP into this specification. The Public and the Content-Base headers have also been imported from RFC 2068 so that they are 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 been completely rewritten. It now includes more details and is also more clear about the model used.

- An IANA section has been included that contains a number of registries and their rules. This will allow us to use IANA to keep track of RTSP extensions.

- The 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 are to be handled have 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 servers and clients should use persistent connections have 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 proxy's TLS connection to the next hop.

- The following header-related changes have been made:

  * Accept-Ranges response-header has been added. This header clarifies which range formats 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 0 has also been added.

  * Added a requirement that the Date header must be used for all messages with a message body and the Server should always include it.

* Removed the 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 a rule that lack of Scale header in a response indicates lack of support for the header. feature tags for scaled playback have been defined.

* The Speed header must now be responded to in order to indicate support and the actual speed going to be used. A feature tag is defined. Notes on congestion control were 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 an explanation on keep-alive and which methods to use. SET_PARAMETER is now recommended to use if only keep-alive within RTSP is desired.

* It has been clarified how the Range header formats are used to indicate pause points in the PAUSE response.

* Clarified that RTP-Info URIs that are relative use the Request-URI as base URI. Also clarified that the used URI must be the one that was used in the SETUP request. The URIs 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 structure borrowed from SIP. Those tables convey much more information and should provide a good overview of the available headers.

* It has been 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.

* ETag has changed its name to MTag.

* To resolve functionality around MTag, the MTag and If-None-Match header have been added from HTTP with necessary clarification in regard 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.

   * A new header Seek-Style has been defined to direct and inform how any seek operation should/have been performed.

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 have been gathered in a separate section (Section 20).

   * The ABNF for the User-Agent and Server headers have been corrected.

   * Some definitions in the introduction regarding the RTSP session have been changed.

   * The protocol has been made fully IPv6 capable.

   * The CHAR rule has been changed to exclude NULL.

o The Status codes have been changed in the following ways:

   * The use of status code 303 (See Other) has been deprecated as it does not make sense to use in RTSP.

   * The never-defined status code 411 "Length Required" has been completely removed.

   * When sending response 451 (Parameter Not Understood) and 458 (Parameter Is Read-Only), 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 a request within an 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.

   * Several new status codes have been defined: 464 (Data Transport Not Ready Yet), 465 (Notification Reason Unknown), 470 (Connection Authorization Required), 471 (Connection Credentials Not Accepted), and 472 (Failure to Establish Secure Connection).

o The following functionality has been deprecated from the protocol:

   * The use of Queued Play.

   * The use of PLAY method for keep-alive in Play state.

   * The RECORD and ANNOUNCE methods and all related functionality. Some of the syntax has been removed.

   * The possibility to use timed execution of methods with the time parameter in the Range header.

   * The description on how rtspu works is not part of the core specification and will require external description. Only that it exists is mentioned here and some requirements for the transport are provided.

o The following changes have been made in relation to methods:

   * The OPTIONS method has been clarified with regard to the use of the Public and Allow headers.

   * Added text clarifying the usage of SET_PARAMETER for keep-alive and usage without a 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.

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

- Wrote a new section about how to set up different media-transport alternatives and their profiles as well as lower-layer protocols. This caused the appendix on RTP interaction to be moved to the new section instead of being in the part that describes RTP. The new section also includes guidelines what to consider 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.

- Security features of RTSP have been clarified:

   * HTTP-based authorization has been clarified requiring both Basic and Digest support

   * TLS support has been mandated

   * If one implements RTP, then SRTP and defined MIKEY-based key-exchange must be supported

   * Various minor mitigations discussed or resulted in protocol changes.

## Acknowledgements

This memorandum defines RTSP version 2.0, which is a revision of the Proposed Standard RTSP version 1.0 defined in [RFC2326]. The authors of RFC 2326 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.

Robert Sparks and especially Elwyn Davies provided very valuable and detailed reviews in the IETF Last Call that greatly improved the document and resolved many issues, especially regarding consistency.

This document has benefited greatly from the comments of all those participating in the MMUSIC WG. In addition to those already mentioned, the following individuals have contributed to this specification:

Rahul Agarwal, Claudio Allocchio, Jeff Ayars, Milko Boic, Torsten Braun, Brent Browning, Bruce Butterfield, Steve Casner, Maureen Chesire, Jinhang Choi, Francisco Cortes, Elwyn Davies, Spencer Dawkins, Kelly Djahandari, Martin Dunsmuir, Adrian Farrel, Stephen Farrell, Ross Finlayson, Eric Fleischman, Jay Geagan, Andy Grignon, Christian Groves, V. Guruprasad, Peter Haight, Mark Handley, Brad Hefta-Gaub, Volker Hilt, John K. Ho, Patrick Hoffman, Go Hori, Philipp Hoschka, Anne Jones, Ingemar Johansson, Jae-Hwan Kim, Anders Klemets, Ruth Lang, Barry Leiba, Stephanie Leif, Jonathan Lennox, Eduardo F. Llach, Chris Lonvick, Xavier Marjou, Thomas Marshall, Rob McCool, Martti Mela, David Oran, Joerg Ott, Joe Pallas, Maria Papadopouli, Sujal Patel, Ema Patki, Alagu Periyannan, Colin Perkins, Pekka Pessi, Igor Plotnikov, Pete Resnick, Peter Saint-Andre, Holger Schmidt, Jonathan Sergent, Pinaki Shah, David Singer, Lior Sion, Jeff Smith, Alexander Sokolsky, Dale Stammen, John Francis Stracke, Geetha Srikantan, Scott Taylor, David Walker, Stephan Wenger, Dale R. Worley, and Byungjo Yoon, and especially Flemming Andreasen.

## 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 (Destination Prohibited) status code, and proposed an earlier version of the PLAY_NOTIFY method.
- Greg Sherwood proposed an earlier version of the PLAY_NOTIFY method.
- Fredrik Lindholm contributed text about the RTSP security framework.
- John Lazzaro contributed the text for RTP over Independent TCP.
- Aravind Narasimhan contributed by rewriting "Media-Transport Alternatives" (Appendix C) and making editorial improvements on a number of places in the specification.
- Torbjorn Einarsson has done some editorial improvements of the text.

## Authors' Addresses

Henning Schulzrinne
Columbia University
1214 Amsterdam Avenue
New York, NY  10027
United States of America

Email: schulzrinne@cs.columbia.edu

Anup Rao
Cisco
United States of America

Email: anrao@cisco.com

Rob Lanphier
San Francisco, CA
United States of America

Email: robla@robla.net

Magnus Westerlund
Ericsson
Faeroegatan 2
Stockholm  SE-164 80
Sweden

Email: magnus.westerlund@ericsson.com

Martin Stiemerling (editor)
University of Applied Sciences Darmstadt
Haardtring 100
64295 Darmstadt
Germany

Email: mls.ietf@gmail.com
URI:   http://www.stiemerling.org