[MS-RTSP]:
Real-Time Streaming Protocol (RTSP) Windows Media Extensions

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.1	Minor	Updated the technical content.
08/10/2007	2.0	Major	Added new protocol examples.

Date	Revision History	Revision Class	Comments
09/28/2007	2.0.1	Editorial	Revised and edited the technical content.
10/23/2007	2.0.2	Editorial	Revised and edited the technical content.
11/30/2007	2.1	Minor	Updated the technical content.
01/25/2008	2.1.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document defines Windows Media extensions to the Real-Time Streaming Protocol (RTSP), as specified in [RFC2326]. RTSP **streams** multimedia from Windows Media Services to Windows Media Player or other instances of Windows Media Services. RTSP Windows Media Extensions use TCP and the User Datagram Protocol (UDP).

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Advanced Systems Format (ASF)
Big-Endian
Globally Unique Identifier (GUID)
Unicode

The following terms are specific to this document:

Content: Multimedia data. **Content** is always in **ASF**, for example, a single **ASF** music file or a single **ASF** video file.

Playlist: One or more **content** items that are **streamed** sequentially.

Session: The state maintained by the server when it is **streaming content** to a client. If a server-side **playlist** is used, the same **session** is used for all **content** in the **playlist**.

Stream: A sequence of **ASF** data packets, as specified in [ASF], that can be selected individually. For example, if a movie has an English and a Spanish soundtrack, each may be encoded in the **ASF** file as a separate **stream**. The video data would also be a separate **stream**.

Streaming: The act of transferring **content** from a sender to a receiver.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[ASF] Microsoft Corporation, "Advanced Systems Format Specification", December 2004, http://download.microsoft.com/download/7/9/0/790fecaa-f64a-4a5e-a430-0bccdab3f1b4/ASF_Specification.doc

If you have any trouble finding [ASF], please check here.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-WMLOG] Microsoft Corporation, "Windows Media Log Data Structure", July 2007.

[MS-WMSP] Microsoft Corporation, "Windows Media HTTP Streaming Protocol Specification", March 2007.

[RFC1945] Berners-Lee, T., Fielding, R., and Frystyk, H., "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996, http://www.ietf.org/rfc/rfc1945.txt

[RFC2109] Kristol, D., and Montulli, L., "HTTP State Management Mechanism", RFC 2109, February 1997, http://www.ietf.org/rfc/rfc2109.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2326] Schulzrinne, H., Rao, A., and Lanphier, R., "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998, http://www.ietf.org/rfc/rfc2326.txt

[RFC2327] Handley, M. and Jacobson, V., "SDP: Session Description Protocol", RFC 2327, April 1998, http://www.ietf.org/rfc/rfc2327.txt

[RFC2397] Masinter, L., "The "data" URL Scheme", RFC 2397, August 1998, http://www.ietf.org/rfc/rfc2397.txt

[RFC2616] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.txt

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003, http://www.ietf.org/rfc/rfc3550.txt

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003, http://www.ietf.org/rfc/rfc3556.txt

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.txt

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005, http://www.ietf.org/rfc/rfc3986.txt

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

[RFC4559] Jaganathan, K., Zhu, L., and Brezak, J., "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006, http://www.ietf.org/rfc/rfc4559.txt

[RFC4566] Handley, M., Jacobson, V., and Perkins, C., "SDP: Session Description Protocol", RFC 4566, July 2006, http://www.ietf.org/rfc/rfc4566.txt

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and Rey, J., "Extended RTP Profile for Real-Time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006, http://www.ietf.org/rfc/rfc4585.txt

1.2.2 Informative References

[MS-MMSP] Microsoft Corporation, "Microsoft Media Server (MMS) Protocol Specification", June 2007.

[RFC2733] Rosenberg, J., and Schulzrinne, H., "An RTP Payload Format for Generic Forward Error Correction", RFC 2733, December 1999, http://www.ietf.org/rfc/rfc2733.txt

1.3 Protocol Overview (Synopsis)

RTSP, as specified in [RFC2326], is used for transferring real-time multimedia data (for example, audio and video) between a server and a client. It is a **streaming** protocol; this means RTSP attempts to facilitate scenarios in which the multimedia data is being simultaneously transferred and rendered (that is, video is displayed and audio is played).

RTSP typically uses a TCP connection for control of the streaming media **session**, although it is also possible to use UDP for this purpose.

The entity that sends the RTSP request that initiates the session is referred as the client, and the entity that responds to that request is referred to as the server. Typically, the multimedia data flows from the server to the client. RTSP also allows multimedia data to flow in the opposite direction. However, the extensions defined in this specification were not designed for such scenarios.

Clients can send RTSP requests to the server requesting information on **content** before a session is established. The information that the server returns is formatted by using a syntax called Session Description Protocol (SDP), as specified in [RFC4566].

Clients use RTSP requests to control the session and to request that the server perform actions, such as starting or stopping the flow of multimedia data. Each request has a corresponding RTSP response that is sent in the opposite direction. Servers can also send RTSP requests to clients, for example, to inform them that the session state has changed.

If TCP is used to exchange RTSP requests and responses, the multimedia data can also be transferred over the same TCP connection. Otherwise, the multimedia data is transferred over UDP.

The multimedia data is encapsulated in Real-time Transport Protocol (RTP) packets, as specified in [RFC3550]. For each RTP stream, the server and client can also exchange Real-Time Transport Control Protocol (RTCP) packets, as specified in [RFC3556].

This specification defines extensions to RTSP, SDP, RTP, and RTCP that enable the delivery of multimedia data that is encapsulated in **Advanced Systems Format (ASF)** packets, as specified in [ASF].

1.4 Relationship to Other Protocols

The RTSP relies on TCP for controlling the streaming media session. Although UDP is also allowed, it is rarely used for this purpose.

RTSP uses the SDP syntax to describe the properties of content.

RTSP uses the RTP for the delivery of multimedia data, and uses the RTCP for RTP feedback and statistics. RTP and RTCP packets are transmitted over either UDP or TCP. It is possible to transmit some RTP streams over UDP and other RTP streams over TCP.

RTSP with Windows Media extensions depends on ASF, which is used both in the SDP syntax and in the payload of the RTP packets.

RTSP is similar in functionality to the Microsoft Media Server (MMS) Protocol (for more information, see [MS-MMSP]). However, RTSP with Windows Media extensions provides additional functionality that is not available in MMS.

RTSP with Windows Media extensions is similar in functionality to the Windows Media HTTP Streaming Protocol, as specified in [MS-WMSP]. However, in that protocol, the delivery of ASF packets is limited to TCP only.

1.5 Prerequisites/Preconditions

The RTSP Windows Media Extensions do not provide a mechanism for a client to discover the URL to the server. Therefore, it is a prerequisite that the client obtain a URL to the server before this protocol can be used.

1.6 Applicability Statement

RTSP is suitable for streaming delivery of real-time multimedia data. The term streaming means that the data is transmitted at some fixed rate or at some rate that is related to the rate at which the data will be consumed (for example, displayed) by the receiver.

It is appropriate to use RTSP Windows Media Extensions when there is a need for a streaming protocol that can deliver multimedia data over either UDP or TCP.

Although the MMS Protocol also supports delivery of multimedia data over UDP and TCP, RTSP with the Windows Media extensions provides additional functionality that is not available in MMS. MMS is an older protocol that has been deprecated. For more information, see [MS-MMSP].

If the multimedia data is always transmitted over TCP, the Windows Media HTTP Streaming Protocol, as specified in [MS-WMSP], might be a suitable alternative. That protocol provides the same functionality as RTSP with the Windows Media extensions, except that the delivery of ASF packets is restricted to TCP.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

Supported Transports: RTSP Windows Media Extensions must be implemented on top of TCP. Also, implementations that require connectionless transmission of multimedia data over an unreliable network service must support UDP. For more information, see section 2.1.

Protocol Versions: RTSP version 1.0, as specified in [RFC2326], must be supported.

Security and Authentication Methods: RTSP Windows Media Extensions support HTTP access authentication, as specified in [RFC2616] section 11.

Localization: RTSP Windows Media Extensions do not specify any localization-dependent protocol behavior.

Capability Negotiation: RTSP Windows Media Extensions perform explicit capability negotiation by using the following mechanisms:

- The **type** attribute in SDP, as specified in section 2.2.5.2.6.
- The Supported (section 2.2.6.7) header.
- The X-Accept-Authentication (section 2.2.6.11) header.

1.8 Vendor-Extensible Fields

Vendor-extensible fields are as specified in [RFC2326].

1.9 Standards Assignments

The following port numbers have been assigned for use by RTP and RTSP.

Parameter	Value	Reference
Port number used by server RTSP requests (both UDP and TCP)	554	IANA
Destination UDP port for RTP packets	5004	IANA
Destination UDP port for RTCP packets	5005	IANA

2 Messages

The following sections specify how messages are encapsulated on the wire and common data types.

2.1 Transport

RTSP requests and responses are sent over either UDP or TCP.<1>

The default port that an RTSP server listens on for incoming requests is port 554, but the use of other port numbers is permitted.

The extensions defined in this specification use the access authentication functionality that was originally defined for HTTP. This is possible because when RTSP requests a response, the syntax of the request is, in many aspects, compatible with HTTP. The specific access authentication schemes supported by any one implementation are implementation-specific.<2>

HTTP access authentication is as specified in [RFC2616] section 11.

2.2 Message Syntax

This section is organized as follows:

Section 2.2.1 specifies the RTP payload format for ASF data packets.

Section 2.2.2 specifies the RTP payload format for forward error correction (FEC).

Section 2.2.3 specifies the RTP payload format for retransmitted RTP packets and packet-pair data.

Section 2.2.4 specifies the syntax of RTCP negative acknowledgement (NACK) packets.

Section 2.2.5 specifies extensions to the SDP.

Section 2.2.6 specifies the syntax of RTSP headers. Only newly defined or modified headers are listed in this specification.

Section 2.2.7 specifies logical request types and explains how each type of request is mapped to an RTSP method.

2.2.1 RTP Payload Format for ASF Data Packets

This section defines an RTP payload format for ASF packets. RTP and ASF are as specified in [RFC3550] and [ASF].

2.2.1.1 General Usage

The RTP payload format defined in this section is suitable for any kind of multimedia data that is encapsulated in ASF data packets. The RTP payload format is used for audio streams and video streams, as well as streams of any of the other types as specified in [ASF].

ASF data packets can contain multiple payloads from different streams of different types. Therefore, it is possible for a single RTP packet to contain both audio and video data because the ASF packet contained in the RTP packet can multiplex data from different streams.

This RTP payload format allows for multiple ASF data packets to be combined into a single RTP packet. It is also possible to split (fragment) an ASF data packet across several consecutive RTP packets.

Each ASF data packet (or fragment thereof) is preceded by an RTP payload format header, which is specified in section 2.2.1.3.

2.2.1.2 RTP Header Usage for ASF Data

The syntax of the RTP header is as specified in [RFC3550]. The fields of the fixed RTP header have their usual meaning (which is specified in [RFC3550] and by the RTP profile in use) with the following additional notes:

Marker (M): This bit MUST be set to 1 if the RTP packet contains the last fragment of an ASF data packet, or one or more complete ASF data packets. Otherwise, it MUST be set to 0.

Payload Type (PT): There is no predefined RTP payload type number for this RTP payload format. Instead, this 7-bit field MUST be assigned to a number that is established through some mechanism outside of RTP. For example, the payload type can be assigned by using SDP, as specified in section 2.2.5.

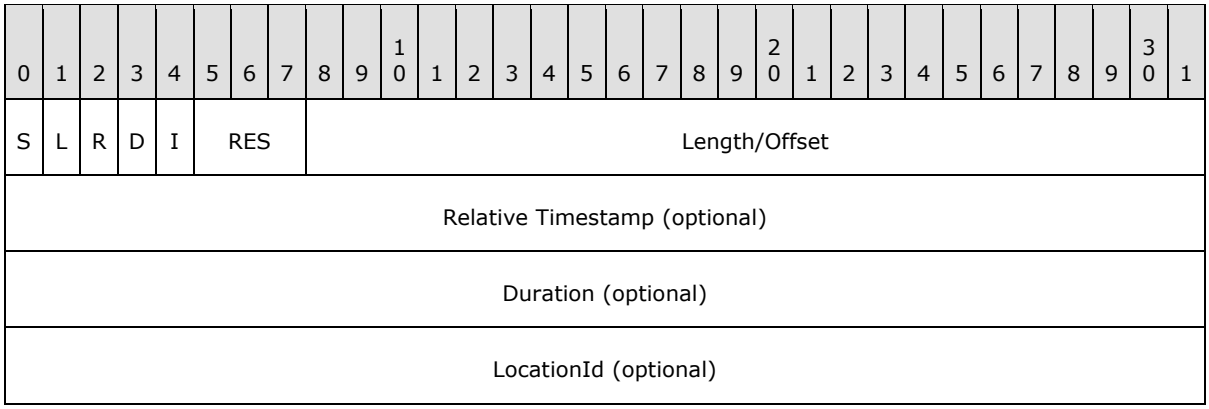
Sequence Number: This 16-bit field MUST increment by one for each RTP packet that is transmitted in the RTP session.

Timestamp: This 32-bit field MUST be set to the value of the **Send Time** field of the first ASF data packet contained in the RTP packet. To find the **Send Time** field of an ASF data packet, see [ASF] section 5.2.2. The time is expressed in milliseconds, unless otherwise specified (for example, through SDP).

2.2.1.3 RTP Payload Format Header

The RTP payload format header is inserted in front of each ASF data packet, or fragment thereof. Therefore, if the RTP packet contains multiple ASF data packets, the RTP payload format header will also be present multiple times.

The fields in the RTP payload format header are transmitted in **big-endian** byte order, also called network byte order. The syntax of the RTP payload format header is as follows:



S (1 bit): This field MUST be set to 1one if the ASF data packet contains a payload that is a key-frame. Otherwise, this field MUST be set to zero. In all RTP payload format headers that

precede fragments of the same ASF data packet, the **S** field MUST be set to the same value. How to determine if an ASF payload contains a key-frame is as specified in [ASF].

L (1 bit): This field MUST be set to one if the **Length/Offset** field specifies the size of the ASF data packet that follows this RTP payload format header. Otherwise, this field MUST be set to zero, and the **Length/Offset** field MUST specify an offset. The **L** field MUST be set to one in all RTP payload format headers that precede complete ASF data packets, and MUST be set to zero in all headers that precede fragmented ASF data packets.

R (1 bit): This field MUST be set to one if the **Relative Timestamp** field is present in the RTP payload format header. Otherwise, this field MUST be set to zero. In all RTP payload format headers that precede fragments of the same ASF data packet, the **R** field MUST be set to the same value.

D (1 bit): This field MUST be set to one if the **Duration** field is present in the RTP payload format header. Otherwise, this field MUST be set to zero. In all RTP payload format headers that precede fragments of the same ASF data packet, the **D** field MUST be set to the same value.

I (1 bit): This field MUST be set to one if the **LocationId** field is present in the RTP payload format header. Otherwise, this field MUST be set to zero.

RES (3 bits): This field MUST be set to zero and MUST be ignored by the receiver.

Length/Offset (3 bytes): If the **L** field is zero, the RTP payload contains a fragment of an ASF data packet, and the **Length/Offset** field MUST specify the byte offset of the fragment's first byte counted from the beginning of the complete ASF data packet. If the **L** field is one, the **Length/Offset** field MUST specify the size of the ASF data packet that directly follows the RTP payload format header in bytes.

If the **Length/Offset** field specifies the size of an ASF data packet, and that size is less than the remaining bytes in the RTP packet, another RTP payload format header MUST follow directly after the end of the ASF data packet.

Relative Timestamp (4 bytes): Optional. If this field is present, it MUST be set to the signed difference between the **Send Time** field of the ASF data packet that follows this RTP payload format header and the **Timestamp** field in the RTP header. If this field is not present, it SHOULD be assumed that the difference between the two fields is zero. If the difference between the two fields is nonzero, the **Relative Timestamp** field MUST be present. Otherwise, the **Relative Timestamp** field SHOULD NOT be present. The time scale used for the **Relative Timestamp** field MUST be the same as is used for the **Timestamp** field in the RTP header.

Where to find the **Send Time** field of an ASF data packet is as specified in [ASF] section 5.2.2.

Duration (4 bytes): Optional. If this field is present, it MUST specify the duration of the ASF data packet. The time scale used for the **Duration** field MUST be the same as that used for the **Timestamp** field in the RTP header. If this field is not present, the duration of the ASF data packet is unspecified, and a zero duration MUST NOT be assumed. (Duration information might be available in the ASF data itself or might be approximated in an implementation-specific manner.) In all RTP payload format headers that precede fragments of the same ASF data packet, the **Duration** field MUST be set to the same value.

Where to find the **Duration** field of an ASF data packet is as specified in [ASF] section 5.2.2.

LocationId (4 bytes): Optional. If this field is present, it MUST specify the index number of the ASF data packet in the original content from which the ASF data packet is extracted. The

first ASF packet in an ASF file MUST have **LocationId** 0x00000000, the second ASF packet in the file MUST have **LocationId** 0x00000001, and so on. Note that because a server may skip ASF packets, the value of the **LocationId** field may not be sequential from one RTP payload format header to the next. If the server does not have access to the ASF file (for example, in case of live content), the server MUST assume a virtual ASF file, incrementing **LocationId** (or decrementing it when rewinding the content) exactly as if a real ASF file existed. If the **LocationId** field is not present, the index number of the ASF data packet is unspecified, and the receiver SHOULD NOT make any assumptions about the value of the index number.

2.2.1.4 ASF Data Packet Payload

Each RTP payload format header is followed by a payload that contains an ASF data packet. The ASF data packet may be partial if it has been fragmented across multiple RTP payloads.

If the ASF data packet contains a **Padding Data** field, as specified in [ASF] section 5.2.4, that field SHOULD be removed before encapsulating the ASF data packet in an RTP packet. If the **Padding Data** field is removed, the **Padding Length** field in the ASF payload parsing information section (as specified in [ASF] section 5.2.2) MUST be updated to indicate a nonexistent **Padding Data** field.

2.2.2 RTP Payload Format for Forward Error Correction

This section defines an RTP payload format for FEC by using the Vandermonde matrix algorithm. RTP is as specified in [RFC3550].

2.2.2.1 General Usage

FEC is a technique that adds redundancy to a bit stream to help protect against corrupted or lost bits. The additional redundancy allows a receiver to recover the correct value of one or more incorrectly received bits from the bits that were received correctly.

The RTP payload format defined in this section is a specific application of the FEC technique to the delivery of RTP packets. The FEC data (redundant bits) generated by the FEC algorithm are transmitted as a separate stream of RTP packets. The original (source) RTP packets that are protected by the FEC algorithm are not modified by the use of this RTP payload format.

The RTP packets that contain FEC data are transmitted on the same RTP session that is used by the source RTP packets. To ensure that the RTP packets with FEC data can be distinguished from the source RTP packets, the RTP packets with FEC data MUST use a different value for the **Payload Type** field in the RTP header than what is used by the source RTP packets.

The RTP packets that contain FEC data consist of the regular RTP header, as specified in [RFC3550], followed by one RTP payload format header, as specified in section 2.2.2.4. The remainder of the RTP payload consists of FEC data. The FEC data payload is computed over the complete source RTP packets except for the RTP header specified in [RFC3550].

There is a need to be able to recover the values of some of the fields in the RTP header of the source RTP packets, so those fields are encoded separately and stored in the RTP payload format header of the FEC packets. For more information, see section 2.2.2.4.

The RTP payload format uses a 24-bit field to identify what source RTP packets are encoded into the FEC RTP packet. This means that at most 24 source RTP packets can be encoded into a single FEC RTP packet.

The algorithm used to compute the FEC data is the Vandermonde matrix algorithm.

2.2.2.2 Vandermonde Matrix Algorithm

The Vandermonde matrix algorithm allows *k* data packets (referred to as source packets) to be encoded into *n* encoded packets. The source packets are encoded in such a way that the reception of any subset of *k* encoded packets at the client end would suffice to recover all the source packets. If more than *n–k* encoded packets are lost, recovery of all the source packets is not possible.

The Vandermonde matrix algorithm always generates the first *k* encoded packets to be identical to the *k* source packets. This simplifies the decoding of the encoded packets and the recovery of the source packets in cases in which little or no packet loss occurs on the network.

It is also useful in cases in which more than *n–k* encoded packets are lost. For such cases, the recovery of all the source packets is not possible; however, because first *k* encoded packets are always the same as the *k* source packets, any of the first *k*-encoded packets received may be used by the receiver as the source packets.

The Vandermonde matrix algorithm uses the Reed-Solomon coding technique based on the Vandermonde matrix for encoding the data. The Reed-Solomon algorithm uses linear algebra principles for encoding and decoding the data.

2.2.2.2.1 Basic Principles Used in the Encoding Technique

Treat *k* source packets as variables labeled *xi...xk*, where *xi* equals the numerical value of the ith packet. The variables are arranged as a vector, *X*, with *k* rows.

$$\text{Where } X = \begin{pmatrix} x1 \\ \cdot \\ \cdot \\ xk \end{pmatrix}$$

Figure 1: RTSP encoding variables and formula (source matrix)

From the linear algebra principle, any *k* linearly independent equations involving *k* number of variables can be solved to obtain the values for those variables. Now, consider an *n* * *k* generator matrix *G*, where each row in *G* specifies the coefficients of an equation. Multiplying *G* with the vector *X* results in *k* linear equations.

If the values of the variables in vector *X* are known, multiplying *G* and *X* results in the vector *Y* with *n* elements.

$$GX = Y \text{ where } Y = \begin{pmatrix} y1 \\ \cdot \\ \cdot \\ yn \end{pmatrix}$$

Figure 2: RTSP encoding variables and formula (generator matrix)

Given the vector *Y* and the generator matrix *G*, the original vector *X* can be recalculated, provided that any *k* rows of matrix *G* are linearly independent, that is, any submatrix formed by taking *k* rows of matrix *G* is invertible. Any *k* rows of the matrix *G* can be chosen to generate *G'*. The multiplication of the inverse of *G'* with vector *Y* will result in the original vector *X*.

$$X * G^{-1}Y$$

Figure 3: RTSP encoding variables and formula (identity matrix for server, and inverse for client)

2.2.2.2.2 Generation of a Vandermonde Matrix

An *n* * *k* size Vandermonde matrix is of the following form.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{k-1} \end{bmatrix}$$

Figure 4: Vandermonde matrix using GF(n*k)

Xi are the elements of the Galois Field GF(pr). As is the case with all Galois Fields, *p* is a prime number, and *r* is an integer greater than or equal to 0. If all *Xi* are different, then the determinant of the matrix is formed by taking any *k* rows if the matrix is non-null and this submatrix is invertible.

The algorithm uses the following steps to generate the Vandermonde matrix.

1. The algorithm uses a field size of GF(28) to generate the matrix coefficients.
2. Assign Xi = I. This means that the values of the *n*th row becomes *1, n, n2, n3, ..., nk-1*.
3. The algorithm uses the Gauss-Jordan elimination method to convert the first *k* rows of the generator matrix to an identity matrix. The row reduction of the Vandermonde matrix results in the system error correction matrix. The systematic error correction is useful because it makes decoding and recovery easier, and it allows for at least some encoded packets to be decoded even if more than *n–k* encoded packets are lost and complete recovery is not possible.

Section 4.2 contains an example of how the algorithm is used.

2.2.2.3 RTP Header Usage for RTP FEC Data

The syntax of the RTP header is specified in [RFC3550]. The fields of the fixed RTP header have their usual meaning, as specified in [RFC3550] and by the RTP profile in use, with the following additional notes:

Marker (M): This bit MUST be set to 0.

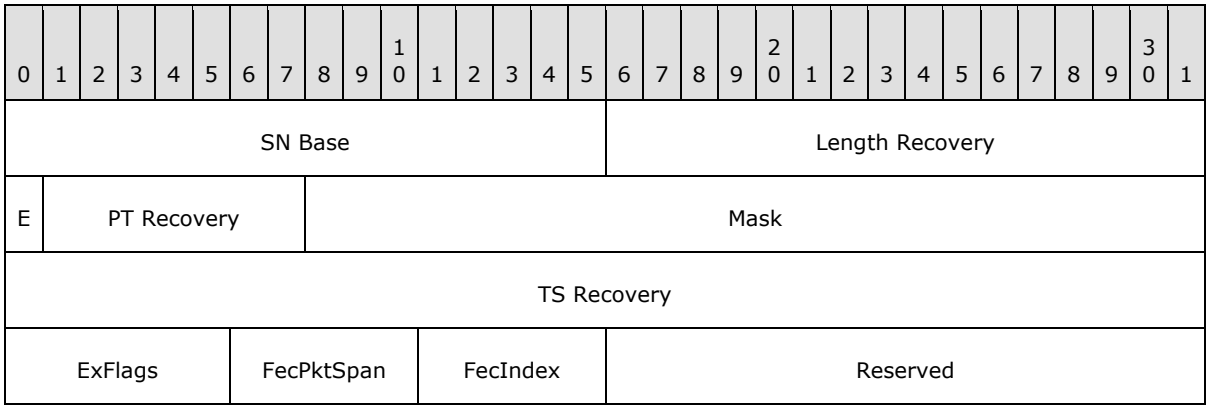
Payload Type (PT): There is no predefined RTP payload type number for this RTP payload format. Instead, this 7-bit field MUST be assigned to a number that is established through some mechanism outside of RTP. For example, it can be assigned by using the SDP, as specified in section 2.2.5.

Sequence Number: This 16-bit field MUST increment by one for each RTP packet that is transmitted in the RTP session.

Timestamp: This 32-bit field MUST be set to the value of the **Timestamp** field of the last source RTP packet in the span of source RTP packets that is encoded into this FEC RTP packet. The value of the **Timestamp** field is expressed by using the same time units used for the **Timestamp** field of the source RTP packets, unless otherwise specified (for example, through SDP).

2.2.2.4 RTP Packet Header FEC Extension

The fields in the RTP payload format header are transmitted in big-endian byte order, also called network byte order. The syntax of the RTP payload format header is as follows.



SN Base (2 bytes): This field MUST be set to the value of the **Sequence Number** field in the RTP header of the first source RTP packet that is encoded in this FEC RTP packet.

Length Recovery (2 bytes): This field MUST be set to the result of applying the FEC algorithm to a virtual 16-bit integer field in each source RTP packet. This virtual field, referred to here as the payload length field, MUST specify the payload length in bytes of each source RTP packet. The value of the **payload length** field MUST include the size of the RTP payload itself, as well as the sizes of the contributing source (CSRC, as specified in [RFC3550]) list, RTP extension, and RTP padding, if any. The **Length Recovery** field allows a receiver to recover the size of a reconstructed source RTP packet and makes it possible to use the FEC algorithms when the sizes of the source RTP packets vary.

E (1 bit): This field MUST be set to zero.

PT Recovery (7 bits): This field MUST be set to the result of applying the FEC algorithm to the value of the **Payload Type** field in the RTP header in each source RTP packet.

Mask (3 bytes): The purpose of this field is to indicate what source RTP packets are encoded into this FEC RTP packet. For each source RTP packet that is encoded into this FEC RTP packet, the bit with index I in the **Mask** field MUST be set to one where I is computed as the difference between the value of the **Sequence Number** field in the source RTP packet and the value of the **SN Base** field in this FEC RTP packet. All other bits in the **Mask** field MUST be set to zero. Index 0 MUST correspond to the least significant bit in the **Mask** field and index 23 to the most significant bit.

TS Recovery (4 bytes): This field MUST be set to the result of applying the FEC algorithm to the value of the **Timestamp** field in the RTP header in each source RTP packet.

ExFlags (6 bits): This field MUST be set to zero.

FecPktSpan (5 bits): This field MUST be set to the number of FEC RTP packets that are transmitted for the current span (group) of source RTP packets or set to zero if the span is unspecified. If the span is specified, the current FEC RTP packet MUST be included in the count of FEC RTP packets.

FecIndex (5 bits): This field MUST be set to the index of this FEC RTP packet among the FEC RTP packets that are transmitted for the current span (group) of source RTP packets. The first FEC RTP packet in the span has index 0, the second has index 1, and so on.

Reserved (2 bytes): This field MUST be set to 0x0004.

2.2.3 RTP Payload Format for Retransmitted RTP Packets and Packet-Pair Data

A client that discovers that it has lost one or more RTP packets might ask the server to retransmit the lost RTP packets. This section defines an RTP payload format that can be used for transmitting copies of RTP packets.

This RTP payload format can also be used for transmitting packet-pair data. Packet-pair data can be used by the receiver to estimate the bottleneck bandwidth in the network path between the transmitter and the receiver.

These two different usage modes of the RTP payload format are defined in the following two sections.

2.2.3.1 Transmitting Copies of RTP Packets

When this RTP payload format is used to transmit a copy of an RTP packet, the RTP payload format does not insert an RTP payload format header of its own into the RTP packet.

The fields in the RTP payload format headers used in the original RTP packet MUST NOT be modified. Also, the RTP header in the copied RTP packet MUST be identical to the RTP header of the original RTP packet.

RTP packets using this RTP payload format SHOULD be transmitted on an RTP session that is different from the one used for the original RTP packets. This allows a receiver to distinguish between the original RTP packets and retransmitted copies of the RTP packets.

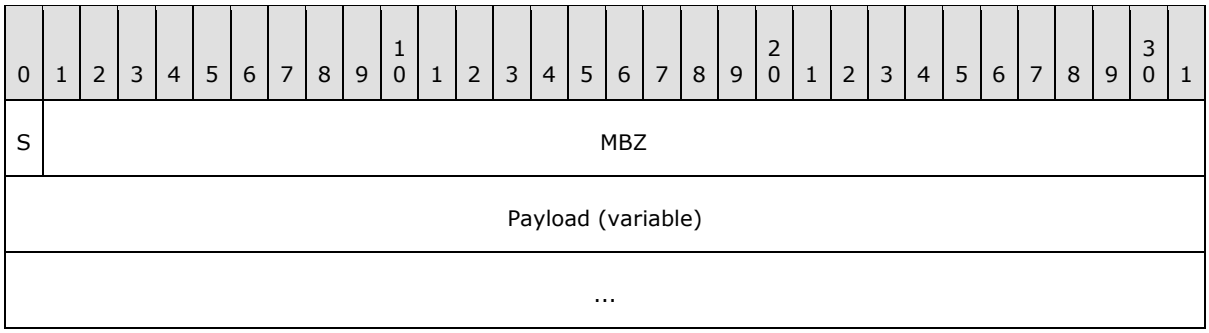
Also, because the RTP header is not changed, the value of the **Sequence Number** field in the RTP header of the retransmitted RTP packets does not necessarily increment monotonically. Transmitting the copied RTP packets on a separate RTP session avoids any confusion that might be caused by the **Sequence Number** field.

2.2.3.2 Transmitting Packet-Pair Data

When this RTP payload format is used for sending packet-pair data, one 4-byte RTP payload format header MUST be added directly after the normal RTP header, as specified in [RFC3550]. The RTP payload format header MUST be followed by highly entropic (random) data.

The RTP header MUST be filled in, following the rules specified in section 2.2.1.2, with the following exception: The value of the **Timestamp** field MUST always be set to 0x00000000.

The fields in the RTP payload format header are transmitted in big-endian byte order. The following diagram shows the RTP payload format header followed by the payload of data.



S (1 bit): This field MUST be set to 1 if this is the first RTP packet that contains packet-pair data. Otherwise, the field MUST be set to 0.

MBZ (31 bits): This field MUST be set to 0x00000000.

Payload (variable): The bytes in this array MUST be set to highly entropic (random) data, and the content of the **Payload** field in each RTP packet that contains packet-pair data MUST be different. The size of the **Payload** field depends on if the sender is transmitting two or three RTP packets with packet-pair data.

If the sender is transmitting three RTP packets with packet-pair data, and the receiver does not allow the sender to decide the **Payload** field size, the size of the **Payload** field MUST be 1,454 bytes for the first packet, 1,455 bytes for the second packet, and 1,456 bytes for the third packet.

If the sender is transmitting two RTP packets with packet-pair data, and the receiver does not allow the sender to decide the **Payload** field size, the size of the **Payload** field MUST be 1,455 bytes for the first packet and 1,456 bytes for the second packet.

If the receiver does allow the sender to decide the **Payload** field size, the size of the **Payload** field MUST NOT be less than 482 bytes. Additionally, the size of the **Payload** field MUST satisfy the equation given by the following table. (PayloadSize denotes the size in bytes of the **Payload** field.)

Packet	Sending two packet-pair packets	Sending three packet-pair packets
First packet	PayloadSize modulo 3 = 0	PayloadSize modulo 3 = 2
Second packet	PayloadSize modulo 3 = 1	PayloadSize modulo 3 = 0
Third packet	N/A	PayloadSize modulo 3 = 1

2.2.4 RTCP NACK Packet Syntax

The syntax for the RTCP NACK packets defined by RTSP Windows Media Extensions follows the syntax for RTCP packets, as specified in [RFC3550], and the syntax for generic NACK messages, as specified in [RFC4585] section 6.2.1, with the following exception.

Unlike what is specified in [RFC4585], which defines the **FMT** field of the RTCP feedback message to be a 5-bit field (as specified in [RFC4585] section 6.1), RTSP Windows Media Extensions define the **FMT** field to be 4 bits in size. The 4 least-significant bits, as specified in [RFC4585], of the **FMT** field are mapped to the 4-bit **FMT** field defined by RTSP Windows Media Extensions.

The most significant bit of the **FMT** field, as specified in [RFC4585], is redefined by RTSP Windows Media Extensions as the **E** field.

A client that sends an RTCP NACK packet SHOULD set the **E** field to 1. Servers MUST ignore the value of the **E** field.

According to the RTCP specification, as specified in [RFC3550], RTCP packets are compound packets consisting of multiple RTCP messages, and all RTCP packets MUST contain a source description (SDES) message with the **CNAME** field. RTSP Windows Media Extensions define the following additional requirement.

An RTCP packet that contains a generic NACK message, as specified in [RFC4585], MUST also contain an SDES message, as specified in [RFC3550], where the value of the **CNAME** field in the SDES message MUST adhere to the following syntax.

```
ssrc                    = 1*10DIGIT
sdes-value      = ssrc "@WMS:7ff42e07-3c7c-4eb5-9c17-6bdd11ad90de"
```

The preceding syntax is specified using Augmented Backus-Naur From (ABNF), as specified in [RFC4234].

The value of the *ssrc* parameter MUST be identical to the numerical value of the **ssrc** field that the server provided in the RTSP Transport (section 2.2.6.8) header in response to the SETUP request (as specified in [RFC2326] section 10.4) for the retransmission stream, expressed using decimal digits. For information on how to identify the retransmission stream, see section 2.2.5.2.5.

2.2.5 Session Description Protocol Extensions

This section defines extensions to the SDP. SDP is specified in [RFC4566], but there are constraints and extensions that apply when SDP is used in conjunction with RTSP, as specified in [RFC2326] Appendix C. RTSP Windows Media Extensions define additional extensions to SDP that apply when SDP is used in conjunction with RTSP.

This section defines the syntax of SDP fields by using ABNF syntax, as specified in [RFC4234].

2.2.5.1 Bandwidth Modifiers for the "b=" Field

The "b=" field is specified in [RFC4566] section 5.8.

2.2.5.1.1 "AS" Bandwidth Modifier

A "b=" field with the "AS" (Application-Specific) bandwidth modifier MUST be specified for each media description that corresponds to a stream in the ASF content. The bandwidth that is specified by the "b=" field MUST correspond to the peak bit rate of the ASF stream, if the ASF stream has a peak bit rate that is different from the average bit rate. Otherwise, the bandwidth that is specified by the "b=" field MUST correspond to the average bit rate of the ASF stream.

How a server determines the peak and average bit rates is implementation-specific. The ASF specification [ASF] provides various alternatives which might sometimes be used to determine the bit rates. For example, see [ASF] sections 3.12 and 4.1.

A "b=" field with the "AS" bandwidth modifier MUST also be specified once at the SDP session level. In this situation, the value of the attribute MUST be set to the peak bit rate required to stream all of

the ASF streams. If an ASF stream does not have an explicitly defined peak bit rate, its average bit rate MUST be used instead.

2.2.5.1.2 "RS" Bandwidth Modifier

A "b=" field with the "RS" bandwidth modifier, as specified in [RFC3556], MUST be specified at the session level or once for every media description. The bandwidth that is specified by the "b=" field MUST be 0.

2.2.5.1.3 "RR" Bandwidth Modifier

A "b=" field with the "RR" bandwidth modifier, as specified in [RFC3556], MUST be specified at the session level or once for every media description. The bandwidth that is specified by the "b=" field MUST be 0.

2.2.5.1.4 "X-AV" Bandwidth Modifier

The "X-AV" bandwidth modifier MUST be specified for each media description that corresponds to a stream in the ASF content, if that stream has an average bit rate that is different from the peak bit rate. In this case, the bandwidth that is specified by the "b=" field MUST correspond to the average bit rate of the stream in kilobits per second. If the average bit rate is identical to the peak bit rate, a "b=" field with the "X-AV" modifier SHOULD NOT be specified.

2.2.5.2 Attributes for the "a=" Field

RTSP Windows Media extensions define some new SDP attributes (that is, names that can be used in the SDP "a=" field) as well as extensions to some existing attributes.

2.2.5.2.1 Control URL Attribute ("a=control")

The control URL attribute ("a=control") MUST be specified for each media description except if the SDP contains only a single media description. The control URL attributes SHOULD be expressed as relative URLs, using the URL specified by using the control attribute at the SDP session level as the base URL.

When converting a relative URL from the media description level to an absolute URL, the URL specified by the SDP session-level control attribute MUST be used as the base URL. If the session-level control attribute is missing, the base URL MUST be determined by following the rules, as specified in [RFC2326] section C.1.1.

2.2.5.2.2 Max packetsize Attribute ("a=maxps")

The **maxps** attribute specifies the maximum ASF packet size in bytes. This attribute SHOULD be present at the SDP session level.

The ABNF syntax for this attribute is as follows:

```
maxps       = "a=maxps:" 1*10DIGIT
```

A packet size of 0 is invalid.

2.2.5.2.3 Program Parameters URL Attribute ("a=pgmpu")

The **pgmpu** (program parameters URL) attribute specifies a data URL, as specified in [RFC2397]. Such URLs are useful for encoding binary data or other kinds of data that use a syntax that may conflict with the SDP syntax.

The ABNF syntax for this attribute is as follows:

```
pgmpu      = "a=pgmpu:"
             dataurl              ; defined in [RFC2397] section 3
```

As specified in [RFC2397], the data URL specifies the kind of data it encodes using a MIME-format media type. RTSP Windows Media Extensions specify two different MIME types for use on data URLs.

2.2.5.2.3.1 application/vnd.ms.wms-hdr.asfv1

A data URL with the MIME type "application/vnd.ms.wms-hdr.asfv1" MUST be present at the SDP session level. The data URL MUST use the base64 encoding mode. The data URL MUST encode the ASF file header of the content being described by the SDP.

The ASF file header consists of the entire ASF Header object (as specified in [ASF] section 3.1), plus the 50-byte fixed initial portion of the ASF Data object (as specified in [ASF] section 5.1).

2.2.5.2.3.2 application/x-wms-contentdesc

A data URL with the MIME type "application/x-wms-contentdesc" SHOULD be present at the SDP session level. The data URL MUST NOT use the base64 encoding mode. The data URL MUST encode a content description list pertaining to the content being described by the SDP.

The syntax for the content description list is specified in [MS-WMSP] section 2.2.4. Because the content description list is an array of **Unicode** characters, the following conversion MUST be performed: The Unicode characters MUST be converted to UTF-8 characters, as specified in [RFC3629]. Any UTF-8 characters that cannot be used in a Uniform Resource Identifier (URI) MUST be encoded by using percent-encoding, as specified in [RFC3986] section 2.1.

2.2.5.2.4 Reliable Attribute ("a=reliable")

The Reliable attribute SHOULD be specified in a media description if it is preferable to transmit the RTP packets that carry the data for this ASF stream by using a reliable transport mechanism, such as TCP, instead of UDP.

Whether a stream should use reliable transport or not is specified in the Extended Stream Properties object, as specified in [ASF] section 4.1.

The ABNF syntax for this attribute is as follows.

```
Reliable          = "a=reliable"
```

2.2.5.2.5 Stream Number Attribute ("a=stream")

The **stream** attribute MUST be present for each media description. It MUST specify the ASF stream number that the media description refers to. The stream number specified by the **stream** attribute MUST be the same number that is specified for the stream in the ASF Stream Properties object in the ASF file header.

If the media description does not correspond to a stream in the ASF content (that is, because it is a retransmission stream or an FEC stream), the stream number specified by the **stream** attribute MUST be chosen according to the following table to correctly indicate if the media description is for a retransmission stream or an FEC stream. If there are multiple media descriptions of the same type, each MUST specify a different stream number.

Stream-num	Meaning
1-65534	Media description is for an ASF stream.
65536-131070	Media description is for a retransmission stream.
131072-196606	Media description is for an FEC stream.

The ABNF syntax for this attribute is as follows:

```
stream-num          = "a=stream:" 1*10DIGIT
```

A stream number that is not within the preceding ranges is invalid.

2.2.5.2.6 Type Attribute ("a=type")

The purpose of the type attribute is to specify a list of properties and capabilities that are applicable to the current content or **playlist** entry. The type attribute MUST be specified once at the SDP session level, except if specifying it would result in an empty "a=type:" field, in which case the attribute SHOULD be omitted.

The syntax of the type attribute is as follows.

```
wm-feat     = "broadcast" | "lastentry"
              | "notseekable" | "notstridable" | "playlist"
              | "skipbackward" | "skipforward"
type        = "a=type:" [wm-feat *6("," wm-feat)]
```

For example: a=type:notseekable,notstridable

2.2.5.2.6.1 broadcast

This property indicates that the content is being broadcast.

2.2.5.2.6.2 lastentry

This property indicates that the content is the last entry in a server-side playlist.

2.2.5.2.6.3 notseekable

This property indicates that the server does not support seeking within the content by using the RTSP Range header.

2.2.5.2.6.4 notstridable

This property indicates that the server does not support fast forward or rewind of the content by using the RTSP Scale header.

2.2.5.2.6.5 playlist

This property indicates to a client that the content is an entry (out of possibly multiple entries) in a server-side playlist.

2.2.5.2.6.6 skipbackward

This property indicates that the server supports skipping to the previous entry in the server-side playlist by using the **pl-offset** token on the X-Playlist (section 2.2.6.17) header.

2.2.5.2.6.7 skipforward

This property indicates that the server supports skipping to the next entry in the server-side playlist by using the **pl-offset** token on the X-Playlist (section 2.2.6.17) header.

2.2.5.3 RTP Payload Format for ASF Data Packets

The RTP payload format for ASF data packets, as specified in section 2.2.1, MUST be identified by the MIME type "audio/x-asf-pf" for audio streams, "video/x-asf-pf" for video streams, and "application/x-asf-pf" for streams that are neither audio nor video.

The RTP clock frequency MUST be 1,000 Hz.

The "a=fmtp" field MUST be specified, and the value of the format-specific parameters syntax element on the "a=fmtp" field (as specified in [RFC4566] section 6) MUST be the MIME type of the codec used to encode the data in this ASF stream. For audio and video streams, the MIME type of the codec MUST adhere to the syntax specified in [RFC2361].

For streams that are neither audio nor video, the MIME type of the codec for the purposes of the "a=fmtp" field MUST be generated according to the following ABNF syntax:

```
guid-value      = 8HEXDIG "-" 4HEXDIG "-" 4HEXDIG "-"
Codec-MIME      = "application/vnd.ms-asf;codec=" guid-value
```

The value of the preceding guid-value syntax element MUST correspond to the **globally unique identifier (GUID)**, as specified in [MS-DTYP], that identifies the codec used to encode the data in this ASF stream. The GUID can usually be obtained from the **Type-Specific Data** field in the Stream Properties object. If it is not possible to obtain the GUID that way, the guid-value syntax element MUST be set to the value of the **Stream Type** field of the Stream Properties object, as specified in [ASF] section 3.3.

Example:

```
m=audio 0 RTP/AVP 96
a=rtpmap:96 x-asf-pf/1000
a=fmtp:96 audio/vnd.wave;codec=123
```

2.2.5.4 RTP Payload Format for FEC Data

The RTP payload format for FEC, as specified in section 2.2.2, MUST be identified by the MIME type "audio/x-wms-fec" for audio streams, "video/x-wms-fec" for video streams, and "application/x-wms-fec" for streams that are neither audio nor video.

The RTP clock frequency MUST be 1,000 Hz.

The "a=reliable" field MUST NOT be specified.

The "a=fmtp" field MUST be specified, and the value of the format-specific parameters syntax element on the "a=fmtp" field (as specified in [RFC4566] section 6) MUST contain the URL that a client would specify in an RTSP SETUP request if it wants to receive the FEC packets. For more information, see [RFC2733] section 11.3.

The "a=fmtp" field specifies the control URL for the stream of FEC packets. If the URL is a relative URL, the client MUST follow the rules specified in section 2.2.5.2.1 for converting a relative URL to an absolute URL.

The "a=fmtp" field MUST also specify the FEC span and the number of FEC packets that will be transmitted per span. These are default values that the client might be able to override by using the Transport (section 2.2.6.8) header.

The syntax of the format-specific parameters syntax element on the "a=fmtp" field MUST adhere to the following ABNF syntax.

```
fecspan               = 1*2DIGIT
fecperspan            = 1*2DIGIT
format-spec-params    = URI-reference          ; section 4.1 of [RFC3986]
                        SP
                        fecspan
                        SP
                        fecperspan
```

The value of the *fecspan* parameter MUST be in the range 1 to 24, inclusive.

The value of the *fecperspan* parameter MUST be in the range 1 to 24, inclusive, and it MUST be less than or equal to the value of the *fecperspan* parameter.

When this RTP payload format is used, it MUST be specified as an additional RTP payload type in the media description of the stream that is being encoded into the FEC packets.

The following example shows the SDP syntax for a case in which RTP payload type 96 is used for ASF data packets containing audio data, and RTP payload type 98 is used for FEC packets generated from the payload type 96 source packets. The URL on the "a=control" field and on the "a=fmtp" field are both relative URLs. (The base URL is not shown in this example.)

```
m=audio 0 RTP/AVP 96 98
```

```
a=rtpmap:96 x-asf-pf/1000
a=fmtp:96 audio/vnd.wave;codec=123
a=control:audio
a=rtpmap:98 x-wms-fec/1000
a=fmtp:98 audio/fec 24 1
```

2.2.5.5 RTP Payload Format for Retransmitted RTP Packets and Packet-Pair Data

The RTP payload format for retransmitted RTP packets and packet-pair data, as specified in section 2.2.3, MUST be identified by the MIME type "application/x-wms-rtx".

The RTP clock frequency MUST be 1,000 Hz.

There are no format-specific parameters for this RTP payload format; therefore, the "a=fmtp" field SHOULD NOT be specified.

The "a=reliable" field MUST NOT be specified.

When this RTP payload format is used, it MUST be specified as a separate media description; that is, it MUST NOT share a media description with other ASF streams. That is because retransmitted RTP packets SHOULD be transmitted on an RTP session separate from the one used by the original RTP packets. For more information, see section 2.2.3.<3>

For example:

```
m=application 0 RTP/AVP 97
a=rtpmap:97 x-wms-rtx/1000
a=control:rtx
```

2.2.6 RTSP Header Fields

RTSP Windows Media Extensions defines several new headers that do not exist in RTSP, as specified in [RFC2326]. Some headers specified in [RFC2326] are further restrained by RTSP Windows Media Extensions in how they can be used. The new headers and the modified existing ones are defined in this section.

Unless otherwise specified, the headers specified in RTSP Windows Media Extensions, and any tokens (also called tags or directives) used on those headers, are defined for use in both requests and responses.

If a client or server receives an RTSP header that is not defined in this section, or if the header is not defined in the current context (for example, receiving a request-only header in a response), the header MUST be interpreted as specified in [RFC2326]. If a particular header is not specified in [RFC2326], it MUST be interpreted as specified in [RFC2616].

If a client or server receives an RTSP header that is defined in this section, and the header contains an unknown token, or if the token is not defined in the current context (for example, receiving a request-only token in a response), the token MUST be ignored.

This section defines the syntax of the RTSP headers by using the ABNF syntax, as specified in [RFC4234]. Any ABNF syntax rules that are not specified in [RFC4234] use the ABNF extensions that are specified in [RFC2326].

2.2.6.1 Bandwidth

The Bandwidth header MUST be used as specified in [RFC2326] section 12.6. Also, the bit rate expressed on the Bandwidth header MUST be in bits per second.

This header is defined for use only in requests sent to a server.

2.2.6.2 Cache-Control

The purpose of the Cache-Control header is to specify to clients, and any intermediate caches (that is, proxy servers), how they may cache the content. The syntax of the Cache-Control header specified in this section applies only when the header is included in an Announce (section 2.2.7.1) request, in the response to a Describe (section 2.2.7.2) request, and in the response to a GetContentInfo (section 2.2.7.4) request. In all other cases, the syntax specified in [RFC2326] section 12.8 applies.

The syntax of the Cache-Control header is as follows.

```
CCdir              = "max-age" | "must-revalidate" | "no-cache" |
                     "no-store" | "no-user-cache" | "private" |
                     "proxy-revalidate" | "public" /
                     "x-wms-event-subscription" | "x-wms-proxy-split" |
                     "x-wms-content-size" | "x-wms-stream-type"

Cache-Control      = "Cache-Control:" CCdir *11("," [SP] CCdir)
```

For example:

```
Cache-Control: no-cache, x-wms-content-size=638066, x-wms-event-subscription="remote-log"
```

2.2.6.2.1 max-age

This directive specifies how many seconds a cache is allowed to use the content without revalidating it with the server. For max-age, including the ABNF syntax, this information is specified in [RFC2616] section 14.9.

2.2.6.2.2 must-revalidate

This directive specifies that the cache MUST revalidate that the content is still refreshed before streaming or playing the content. The directive does not apply if caching has been disabled by using the no-cache (section 2.2.6.2.3) directive or the no-user-cache (section 2.2.6.2.5) directive. The must-revalidate directive MUST be ignored by caches that are acting as proxy servers.

2.2.6.2.3 no-cache

This directive specifies that the cache MUST NOT cache the content. The no-cache directive applies to both caches that are clients and caches that are proxy servers.

2.2.6.2.4 no-store

This directive specifies that the cache MUST NOT store the content on persistent storage, such as a hard drive.

2.2.6.2.5 no-user-cache

This directive specifies that a cache that is a client (not a proxy server) MUST NOT cache the content.

2.2.6.2.6 private

This directive specifies that the content MUST NOT be shared with other users on the device on which the client software is running, and it MUST NOT be shared by other proxy servers. The private directive applies to both caches that are clients and caches that are proxy servers.

2.2.6.2.7 proxy-revalidate

This directive specifies that if the cache is a proxy server, it MUST revalidate that the content is still refreshed before it is streamed. The directive does not apply if caching has been disabled by using the no-cache (section 2.2.6.2.3) directive or the private (section 2.2.6.2.6) directive. The proxy-revalidate directive MUST be ignored by caches that are not acting as proxy servers.

2.2.6.2.8 public

This directive specifies that the content is allowed to be cached and shared with other users on the device on which the client software is running as well as by proxy servers. The public directive applies to both caches that are clients and caches that are proxy servers.

2.2.6.2.9 x-wms-content-size

This directive specifies the approximate size of the content, in bytes, that is required if the content is cached in its entirety.

2.2.6.2.10 x-wms-event-subscription

This directive specifies a comma-separated list of remote event names that the server would like to receive. The list is enclosed in quotation marks. The SendEvent (section 2.2.7.11) request is used to send the remote events to the server.<4>

The syntax of the directive is as follows.

```
log-event    = ( "remote-open" / "remote-close" / "remote-log" )
Eventsub     = "x-wms-event-subscription="
               %x22 log-event *2( "," log-event ) %x22
```

2.2.6.2.11 x-wms-proxy-split

This directive indicates that the content is allowed to be split; that is, forwarded to multiple clients in real time. This is typically used for live content, for which caching is inappropriate or not allowed.<5>

2.2.6.2.12 x-wms-stream-type

This directive specifies a comma-separated list of properties that apply to the content. The list is enclosed in quotation marks.

The broadcast property specifies that the content is broadcast, or live (and therefore may be suitable for splitting to multiple downstream clients). The playlist property specifies that the content consists of possibly multiple entries from a server-side playlist.

The list of properties can be used by a caching proxy server to determine if the content is a suitable candidate for caching or splitting.<6>

The syntax of the directive is as follows.

```
stream-prop = ( "broadcast" / "playlist" )
StreamTypes = "x-wms-stream-type="
              %x22 stream-prop *2( "," stream-prop ) %x22
```

2.2.6.3 Content-Type

The Content-Type header specifies the media type of data that is included in the message payload (for example, the response to the DESCRIBE method or the message body of the ANNOUNCE and SET_PARAMETER methods). When used in a GET_PARAMETER request, the Content-Type header is sometimes used to specify the media type of the data that is expected in the response.

The syntax of the Content-Type header is as follows.

```
Ctype            = "application/sdp" |
 "application/x-rtsp-packetpair" |
 "application/x-rtsp-udp-packetpair" |
 "application/x-wms-contentdesc" |
 "application/x-wms-extension-cmd" |
                  "application/x-wms-getcontentinfo" |
 "application/x-wms-gettemplates" |
 "application/x-wms-Logconnectstats" |
 "application/x-wms-Logplaystats" |
 "application/x-wms-sendevent" |
 "application/x-wms-streamswitch"

Content-Type      = "Content-Type: " Ctype [";charset=UTF-8"]
```

For example:

```
Content-Type: application/x-wms-Logplaystats;charset=UTF-8
```

2.2.6.3.1 application/sdp

This media type is used in the response to the DESCRIBE request method and in the ANNOUNCE request method. It indicates that the message body in the response or request, respectively, contains a complete SDP description. For information on SDP syntax when used with RTSP Windows Media Extensions, see section 2.2.5.

2.2.6.3.2 application/x-rtsp-packetpair

This media type is used in a GET_PARAMETER request method and in the response to that request. When used in the GET_PARAMETER request, it identifies the request as a TcpPacketPair (section 2.2.7.12) request.

When used in the response to the GET_PARAMETER request method, this media type indicates that the message body contains three $P packets, as specified in [MS-WMSP] section 2.2.3.7.

2.2.6.3.3 application/x-rtsp-udp-packetpair

This media type is used in a SET_PARAMETER request method and serves to identify the SET_PARAMETER request as a UdpPacketPair (section 2.2.7.14) request. The message body of the UdpPacketPair request MUST follow the syntax as specified in section 2.2.7.14.

2.2.6.3.4 application/x-wms-extension-cmd

This media type specifies that the message body of the SET_PARAMETER request method adhere to the syntax as specified in section 2.2.7.3. This message body is used to provide details on why the server is sending an EndOfStream (section 2.2.7.3) request.

2.2.6.3.5 application/x-wms-getcontentinfo

This media type is used in a GET_PARAMETER request and serves to identify the GET_PARAMETER request as a GetContentInfo (section 2.2.7.4) request. The message body of the GetContentInfo request is not significant because the sole purpose of the request is to retrieve the Cache-Control (section 2.2.6.2) header in the response.

2.2.6.3.6 application/x-wms-Logconnectstats

This media type is used in a SET_PARAMETER request method and serves to identify the SET_PARAMETER request as a LogConnect (section 2.2.7.6) request. The message body of the LogConnect request MUST follow the syntax as specified in section 2.2.7.6.

2.2.6.3.7 application/x-wms-Logplaystats

This media type is used in a SET_PARAMETER request method and serves to identify the SET_PARAMETER request as a LogPlay (section 2.2.7.7) request. The message body of the LogPlay request MUST follow the syntax as specified in section 2.2.7.7.

2.2.6.3.8 application/x-wms-sendevent

This media type specifies that the message body of the SET_PARAMETER request contain a remote event message in the remote event format, as specified in [MS-WMSP] section 2.2.5. SET_PARAMETER requests that specify this media type are SendEvent (section 2.2.7.11) requests.

2.2.6.3.9 application/x-wms-streamswitch

This media type is used in a SET_PARAMETER request method and serves to identify the SET_PARAMETER request as a SelectStream (section 2.2.7.10) request. The message body of the SelectStream request MUST follow the syntax as specified in section 2.2.7.10.

2.2.6.4 Cookie

The syntax of the Cookie header MUST be as specified in [RFC2109].

This header is defined for use in requests sent to a server. Clients SHOULD share a single repository for RTSP cookies and HTTP cookies, and treat http:// and rtsp:// URLs as a single protocol.

This means that if a cookie is set for the URL http://example.com/ by using the HTTP protocol, and the client sends an RTSP Describe request for the URL rtsp://example.com/, the cookie SHOULD be included in the Describe request even though it was originally obtained through HTTP.

2.2.6.5 Range

The syntax of the Range header MUST follow the general rules as specified in [RFC2326] section 12.29. However, a Range header MUST NOT contain more than one ranges-specifier syntax element. Also, the *time* parameter and the utc-range ranges-specifier syntax element MUST NOT be used. RTSP Windows Media Extensions define two additional ranges-specifier tokens.

The ABNF syntax for the Range header is as follows.

```
ranges-specifier      = npt-range               ; [RFC2326] section 3.6
                      | smpte-range             ; [RFC2326] section 3.5
                      | byte-range              ; section 2.2.6.5.1
                      | packet-range            ; section 2.2.6.5.2
Range                 = "Range: " ranges-specifier
```

2.2.6.5.1 x-asf-byte

The x-asf-byte token is used to express a start position and, optionally, a stop position in units of bytes counted from the start of the content. Byte offsets are expressed as integer decimal numbers. The beginning of the ASF file has byte offset 0. The first ASF data packet is located at the byte offset that corresponds to the size of the ASF file header (as specified in section 2.2.5.2.3.1).

The ABNF syntax for the x-asf-byte token is as follows.

```
byte-offset       = 1*20DIGIT
byte-range        = "x-asf-byte=" byte-offset "-" [ byte-offset ]
```

The value of byte-offset MUST be an integer in the range 0 to 18,446,744,073,709,551,614, inclusive.

2.2.6.5.2 x-asf-packet

The x-asf-packet token is used to express a start position and, optionally, a stop position as an ASF data packet number. ASF data packet numbers are expressed as integer decimal numbers. The first ASF data packet in the ASF file has number 0, and each ASF data packet in the file increments by one.

The ABNF syntax for the x-asf-packet token is as follows.

```
packet-num        = 1*20DIGIT
packet-range      = "x-asf-packet=" packet-num "-" [ packet-num ]
```

The value of packet-num MUST be an integer in the range 0 to 18,446,744,073,709,551,614, inclusive.

2.2.6.6 Set-Cookie

The syntax of the Set-Cookie header MUST be as specified in [RFC2109].

This header is defined for use in responses sent to a client. Clients SHOULD share a single repository for RTSP cookies and HTTP cookies, and treat http:// and rtsp:// URLs as a single protocol.

This means that if a cookie is set for the URL http://example.com/, and a cookie with the same name is set for the URL rtsp://example.com/, the second cookie overrides the first cookie because the two URLs are considered equivalent.

2.2.6.7 Supported

The Supported header is used for specifying features of the protocol that are supported and that are allowed to be used in the current session. Different features may apply to different entries in a server-side playlist.

Some features are not to be used unless indicated by the Supported header.

If a feature is listed on the Supported header, the feature is supported, and the feature SHOULD be used, if appropriate. If a feature has been defined for use on the Supported header, but it is not listed on the Supported header, that feature MUST NOT be used.

A missing Supported header (from either a request or response) MUST NOT be interpreted as changing the list of features that are currently supported.

The syntax of the Supported header is as follows.

```
WMCfeat    = "com.microsoft.wm.eosmsg"
             | "com.microsoft.wm.fastcache"
             | "com.microsoft.wm.locid"
             | "com.microsoft.wm.packetpairssrc"
             | "com.microsoft.wm.predstrm"
             | "com.microsoft.wm.srvppair"
             | "com.microsoft.wm.sswitch"
             | "com.microsoft.wm.startupprofile"
Supported = "Supported: " WMCfeat *7["," [SP] WMCfeat]
```

For example:

```
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
   com.microsoft.wm.predstrm, com.microsoft.wm.startupprofile
```

The tokens that can be used on the Supported header are defined in the following sections.

2.2.6.7.1 com.microsoft.wm.eosmsg

The com.microsoft.wm.eosmsg token specifies support for the EndOfStream (section 2.2.7.3) request. If a client specifies this token, it means that the client supports the EndOfStream request

and the client expects the server to send such requests to it. If a server specifies the com.microsoft.wm.eosmsg token, it means that the server intends to send EndOfStream requests to the client.

If a server never sends the Supported header, or if a server does not specify the com.microsoft.wm.eosmsg token on the Supported header, clients SHOULD assume that the server will not send an EndOfStream request.

If a client never sends the Supported header, or if a client does not specify the com.microsoft.wm.eosmsg token on the Supported header, servers SHOULD assume that the client does not support the EndOfStream request. In this case, the server MAY send the EndOfStream request anyway, but MUST be prepared to handle an error response from the client.<7>

2.2.6.7.2 com.microsoft.wm.fastcache

The com.microsoft.wm.fastcache token specifies that the server permits the use of the Speed header, as specified in [RFC2326] section 12.35.

This token is defined for use only in responses sent to a client.

If a server never sends the Supported header, clients MUST NOT include the Speed header in RTSP requests sent to the server.

2.2.6.7.3 com.microsoft.wm.locid

The com.microsoft.wm.locid token specifies that the client wants the server to include the **LocationId** field in the RTP payload format headers of RTP packets. This applies only to RTP packets that use the RTP payload format for ASF data packets, as specified in section 2.2.1.

This token is defined only for use in requests sent to a server.

If a client never sends the Supported header, or if a client does not specify the com.microsoft.wm.locid token on the Supported header, servers MUST NOT include the **LocationId** field in the RTP payload format headers of RTP packets when the RTP payload format for ASF data packets is used.

2.2.6.7.4 com.microsoft.wm.packetpairssrc

The com.microsoft.wm.packetpairssrc token specifies that when the server sends packet-pair data over the RTP by using the RTP payload format, as specified in section 2.2.3.2, the **SSRC** field in the RTP header is set to the same **SSRC** value specified by the server in the RTSP Transport header in the response to the SETUP request for the retransmission stream. For information on the retransmission stream, see section 2.2.5.5.

This token is defined only for use in responses sent to a client.

If a server does not specify the com.microsoft.wm.packetpairssrc token on the Supported header, clients SHOULD ignore the **SSRC** field in the RTP header of RTP packets that contain packet-pair data and that use the RTP payload format as specified in section 2.2.3.2.

2.2.6.7.5 com.microsoft.wm.predstrm

The com.microsoft.wm.predstrm token specifies support for predictive stream selection. This is a technique in which the server selects a set of streams from the next entry in a server-side playlist on the client's behalf, and starts streaming those streams. When predictive stream selection is not

used, the server will not start streaming the next entry in the server-side playlist until the client has sent a Play (section 2.2.7.9) request.

Although a client might support predictive stream selection, it might not always want it to be used. For example, if the client prefers having full control over what streams are selected for a new playlist entry instead of having the server select the streams on its behalf. Therefore, if a Supported header does not include the com.microsoft.wm.predstrm token, then the client does not want predictive stream selection to be used at the moment.

If a client never sends the Supported header, or if a client does not specify the com.microsoft.wm.predstrm token on the Supported header, servers MUST NOT use predictive stream selection.

2.2.6.7.6 com.microsoft.wm.srvppair

The com.microsoft.wm.srvppair token specifies support for Packet-Pair. This is a technique in which the server sends two or three packets with random data to the client over UDP or TCP. The client measures the time it takes for the second packet to arrive and can use this to estimate the bottleneck bandwidth on the network path between the server and the client.

If a server never sends the Supported header, or if a server does not specify the com.microsoft.wm.srvppair token on the Supported header, clients MUST NOT send a TcpPacketPair (section 2.2.7.12) request or a UdpPacketPair (section 2.2.7.14) request to the server.

If a client never sends the Supported header, or if a client does not specify the com.microsoft.wm.srvppair token on the Supported header, servers SHOULD assume that the client does not intend to send a TcpPacketPair request or a UdpPacketPair request.

2.2.6.7.7 com.microsoft.wm.sswitch

The com.microsoft.wm.sswitch token specifies support for the SelectStream (section 2.2.7.10) message body.

If a server never sends the Supported header, or if a server does not specify the com.microsoft.wm.sswitch token on the Supported header, clients SHOULD assume that the server does not support receiving a SelectStream request that includes a nonempty message body.

If a client never sends the Supported header, or if a client does not specify the com.microsoft.wm.sswitch token on the Supported header, servers SHOULD assume that the client does not support sending a SelectStream request with a nonempty message body.

2.2.6.7.8 com.microsoft.wm.startupprofile

This token specifies support for the X-StartupProfile (section 2.2.6.25).

If a server never sends the Supported header, or if a server does not specify the com.microsoft.wm.startupprofile token on the Supported header, clients MUST assume that the server will not send the X-StartupProfile header.

If a client never sends the Supported header, or if a client does not specify the com.microsoft.wm.startupprofile token on the Supported header, servers MUST NOT send the X-StartupProfile header to the client.<8>

2.2.6.8 Transport

The syntax of the Transport header MUST adhere to the ABNF syntax, as specified in [RFC2326] section 12.39, with the following modification. The transport-spec syntax element has been extended with an optional fec-parameters element. The modified syntax for transport-spec is as follows.

```
fec-parameters  = ";FecSpan=" 1*2HEXDIG
                  ";FecPerSpan=" 1*2HEXDIG
                  ";FecBurstMargin=" HEXDIG
transport-spec  = transport-protocol "/" profile ; [RFC2326] section 12.39
                  [ "/" lower-transport ]        ; [RFC2326] section 12.39
                  2*parameter                    ; [RFC2326] section 12.39
                  [ fec-parameters ]
```

The syntax elements unicast and client_port, which are included in the parameter syntax element, as specified in [RFC2326] section 12.39, MUST be present in each transport-spec syntax element.

The value of the *FecSpan* parameter MUST be in the range 0x01 to 0x18, inclusive.

The value of the *FecPerSpan* parameter MUST be in the range 0x01 to 0x18, inclusive, and it MUST also be less than or equal to the value of the *FecSpan* parameter.

The value of the *FecBurstMargin* parameter MUST be in the range 0x1 to 0xC, inclusive. Also, the value of *FecSpan* minus 1 multiplied by the value of *FecBurstMargin* MUST NOT exceed 0x18.

The fec-parameters syntax element SHOULD be present in a SETUP request for a stream that uses the RTP payload format for FEC (for more information, see section 2.2.2).

When used in a request, the numerical value after the *FecSpan* parameter specifies the number of RTP source packets that the client wants to be included in a span. There is at least one RTP FEC packet for each span, so a small span implies higher-loss tolerance because each RTP FEC packet protects few RTP source packets. But a small span also implies higher overhead because RTP FEC packets are transmitted more frequently than with a large span.

When used in a request, the numerical value after the *FecPerSpan* parameter specifies the number of RTP FEC packets that the client wants to be included in each span. Each span will include at least one RTP FEC packet, and any additional RTP FEC packets provide additional protection against lost RTP source packets (at the overhead of having to transmit the RTP FEC packet).

When used in a request, the numerical value after the *FecBurstMargin* parameter specifies the distance between RTP source packets in a single span. A value of 1 means that adjacent RTP source packets belong to the same span. A value of 2 means that every second RTP source packet belongs to the same span (that is, two spans are interleaved). Interleaving spans provides additional protection against burst errors, which is a kind of packet-loss event that causes multiple adjacent RTP packets to be lost.

The fec-parameters syntax element SHOULD be present in the response to a SETUP request for a stream that uses the RTP payload format for FEC (for more information, see section 2.2.2).

When used in a response, the *FecSpan*, *FecPerSpan*, and *FecBurstMargin* parameters specify the actual values for these parameters that the server has chosen.

2.2.6.9 User-Agent

The User-Agent header specifies the major and minor version numbers of the Microsoft software product that is sending the RTSP request.

This header is defined only for use in requests sent to a server.

The syntax of the User-Agent header is as follows.

```
client-token     = ( "WMPlayer" | "WMServer" | "WMCacheProxy" )
major            = 1*2DIGIT
minor            = 1*2DIGIT ["." 1*4DIGIT "." 1*4DIGIT]
guid-value       = 8HEXDIG "-" 4HEXDIG "-" 4HEXDIG "-"
                   4HEXDIG "-" 12HEXDIG
client-guid      = "guid/" guid-value
user-agent-data      = client-token "/" major "." minor
                       [ SP client-guid ]
                       *( SP product ) ; defined in section 3.8 of [RFC2616]
User-Agent       = "User-Agent: " user-agent-data
```

The guid-value syntax element specifies an identifier that uniquely identifies the client software installation that originated the request. The identifier MUST be identical for all requests belonging to the same streaming session. The identifier MUST be a GUID, as specified in [MS-DTYP]. As shown in the preceding ABNF syntax, the GUID is expressed in registry format and is not enclosed in quotation marks.

Clients MUST assign the values of the client-token, major, and minor ABNF syntax elements as shown in the following table.

Product name	Client-token	Major	Minor
Windows Media Services 9.0	WMServer	9	0
	WMPlayer	9	0
	WMCacheProxy	9	0
Windows Media Services 9.1	WMServer	9	1
	WMPlayer	9	1
	WMCacheProxy	9	1
Windows Media Format 9 Series SDK	WMPlayer	9	0
Windows Media Format 9.5 SDK	WMPlayer	10	0
Windows Vista	WMPlayer	11	0

2.2.6.10 X-Accelerate-Streaming

The X-Accelerate-Streaming header specifies an amount of multimedia data (in millisecond units) and a transmission rate (in bits per second).

When used in a request, the client is requesting the server to transmit the specified amount of data at the specified transmission rate.

When used in a response, the header states the server's intent to transmit the specified amount of data at the specified transmission rate.

This header is defined only for use in requests sent to a server and responses sent to a client.

The ABNF syntax for the X-Accelerate-Streaming header is as follows.

```
X-Accelerate-Streaming  = "X-Accelerate-Streaming: AccelDuration="
                          1*10DIGIT
                          ";AccelBandwidth="
                          1*10DIGIT
```

The value of each of the two numerical parameters MUST be an integer in the range 0 to 2,147,483,647, inclusive.

2.2.6.11 X-Accept-Authentication

The X-Accept-Authentication header specifies the authentication schemes that the client supports.

This header is defined only for use in requests sent to a server.

For more information on the X-Accept-Authentication header, including syntax, see [MS-WMSP] section 2.2.1.9.

2.2.6.12 X-Accept-Proxy-Authentication

The X-Accept-Proxy-Authentication header specifies the authentication schemes that the client supports when challenged by a proxy server.

This header is defined only for use in requests sent to a server.

For more information on the X-Accept-Proxy-Authentication header, including syntax, see [MS-WMSP] section 2.2.1.10.

2.2.6.13 X-Broadcast-Id

The purpose of the X-Broadcast-Id header is to provide a numerical identifier for the source of the current entry in the server-side playlist, if the current entry is broadcast.

The value on the X-Broadcast-Id header MUST uniquely identify the source of broadcast content within the scope of the current server-side playlist. For example, if the same live content source is used in multiple entries in a single playlist, the value on the X-Broadcast-Id header MUST be the same in each of those entries. There is no requirement that the values of the X-Broadcast-Id header should be unique across different server-side playlists. If a playlist entry is not using a broadcast content source, the X-Broadcast-Id header MUST be either omitted or specified with a 0 value.

This header is defined only for use in requests and responses sent to a client.

The ABNF syntax for the X-Broadcast-Id header is as follows.

```
X-Broadcast-Id            = "X-Broadcast-Id: " 1*10DIGIT
```

The value of the numerical parameter MUST be an integer in the range 0 to 4,294,967,295, inclusive.

2.2.6.14 X-Burst-Streaming

The X-Burst-Streaming header specifies an amount of multimedia data (in millisecond units) and a transmission rate (in bits per second).

When this header is used in a request, the client is requesting that the server transmit the specified amount of data at the specified transmission rate.

The client that is sending the request is normally an intermediate device that is relaying a request on behalf of another client. The X-Accelerate-Streaming (section 2.2.6.10) header, if specified, is the one provided by the original client. The X-Burst-Streaming header specifies the amount of multimedia data and the transmission rate requested by the intermediate device.

When used in a response, the header states the server's intent to transmit the specified amount of data at the specified transmission rate.

This header is defined only for use in requests sent to a server and responses sent to a client.

The ABNF syntax for the X-Burst-Streaming header is as follows.

```
X-Burst-Streaming     = "X-Burst-Streaming: BurstDuration="
                        1*10DIGIT
                        ";BurstBandwidth="
                        1*10DIGIT
```

The value of each of the two numerical parameters MUST be an integer in the range 0 to 2,147,483,647, inclusive.

2.2.6.15 X-Notice

The X-Notice header SHOULD be included in EndOfStream (section 2.2.7.3) requests sent by the server. This header is defined only for requests sent to a client.

The ABNF syntax of the X-Notice header is as follows.

```
X-Notice      = "X-Notice: 2101" SP %x22 "End-of-Stream Reached" %x22
```

2.2.6.16 X-Player-Lag-Time

The X-Player-Lag-Time header indicates to the server the amount of time (in milliseconds) by which the client might be lagging behind the server as a result of predictive stream selection. During

predictive stream selection, after having streamed all of the content in one server-side playlist entry, the server will start to stream the content in the next playlist entry without waiting for the Play (section 2.2.7.9) request from the client. The time that lapses until the client eventually sends the Play request is the amount of time that the client is lagging behind the server.

As a result of receiving this header in a Play request, the server is expected to slow down the pushing of new ASF file headers to the client by several milliseconds to ensure that the client does not fall too far behind.

For more information on predictive stream selection, see section 2.2.6.7.5.

This header is defined only for requests sent to a server.

The syntax of the X-Player-Lag-Time header is as follows.

```
X-Player-Lag-Time = "X-Player-Lag-Time: " 1*10DIGIT
```

The numerical value MUST be an integer in the range 0 to 4,294,967,295, inclusive.

2.2.6.17 X-Playlist

The X-Playlist header indicates to the server whether to move forward or backward to an entry in a playlist relative to the entry ID specified in the X-Playlist-Seek-Id (section 2.2.6.20) header. If the numerical value specified on the X-Playlist header is 1, the server is requested to move forward to the next entry. A value of -1 means that the server is requested to move backward to the previous entry.

This header is defined only for requests sent to a server.

The syntax of the X-Playlist header is as follows.

```
X-Playlist = "X-Playlist: pl-offset=" ( "1" | "-1" )
```

2.2.6.18 X-Playlist-Change-Notice

The X-Playlist-Change-Notice header is defined onlyfor use in the response to a Play request sent to a client.

The header notifies the client that no RTP packets will be transmitted after the Play response. Instead, the response is immediately followed by an EndOfStream request and an Announce request.

The syntax of the X-Playlist-Change-Notice header is as follows.

```
X-Playlist-Change-Notice = "X-Playlist-Change-Notice: true"
```

2.2.6.19 X-Playlist-Gen-Id

The X-Playlist-Gen-Id header specifies the identifier of the playlist entry that the current request or response applies to.

Once the client has obtained the identifier for the current playlist entry (typically through the response to the Describe (section 2.2.7.2) request), the client will include it in most subsequent RTSP requests for the current session.

The syntax of the X-Playlist-Gen-Id header is as follows.

```
X-Playlist-Gen-Id = "X-Playlist-Gen-Id: " 1*10DIGIT
```

The numerical value MUST be an integer in the range 1 to 4,294,967,295, inclusive.

2.2.6.20 X-Playlist-Seek-Id

The X-Playlist-Seek-Id header requests the server to seek the playlist entry with the ID specified as the numerical value of this header.

If a client includes this header in a request, the numerical value MUST be either the identifier of the current playlist entry or the identifier of the previous playlist entry. (Identifiers for playlist entries are obtained from the X-Playlist-Gen-Id (section 2.2.6.19) header.)

This header is defined only for requests sent to a server.

The syntax of the X-Playlist-Seek-Id header is as follows.

```
X-Playlist-Seek-Id = "X-Playlist-Seek-Id: " 1*10DIGIT
```

The numerical value MUST be an integer in the range 1 to 4,294,967,295, inclusive.

2.2.6.21 X-Proxy-Client-Agent

The X-Proxy-Client-Agent header is sent by intermediate devices (such as proxy servers) and specifies the information that the original client who initiated the RTSP request specified in the User-Agent header.

Because each intermediate device replaces the information on the User-Agent header with its own information, the X-Proxy-Client-Agent header enables the information on the original User-Agent header to be preserved and forwarded across possibly multiple intermediate devices to the server.

This header is defined only for requests sent to a server by an intermediate device, such as a proxy server that is acting as a client.

The syntax of the X-Proxy-Client-Agent header is as follows.

```
X-Proxy-Client-Agent = "X-Proxy-Client-Agent: "
                        user-agent-data      ; defined in section 2.2.6.9.
```

2.2.6.22 X-Proxy-Client-Verb

The X-Proxy-Client-Verb header is sent by intermediate devices (such as proxy servers) and specifies the name of the HTTP or RTSP request method used by the client that triggered the RTSP request sent by the intermediate device.

The header is useful when digest authentication (as specified in [RFC2617]) is used, and one or more of the intermediate devices is performing a protocol translation between HTTP and RTSP, or vice versa. When digest authentication is used, the name of the HTTP or RTSP request method might be a part of the authentication challenge. Intermediate devices may change the request method. In particular, this is true if the intermediate device translates requests between different streaming protocols. The X-Proxy-Client-Verb header enables the name of the request method to be preserved and forwarded across possibly multiple intermediate devices to the server.

This header isdefined only for requests sent to a server by an intermediate device, such as a proxy server that is acting as a client.

The syntax of the X-Proxy-Client-Verb header is as follows.

```
X-Proxy-Client-Verb = "X-Proxy-Client-Verb: "
                      ( Method     ; defined in section 6.1 of [RFC2326]
                      =| Method )  ; defined in section 5.1.1 of [RFC2616]
```

2.2.6.23 X-Receding-PlaylistChange

The X-Receding-PlaylistChange header is defined only for use in an Announce (section 2.2.7.1) request sent to a client. The header indicates that the playlist entry described by the SDP in the Announce request is the previous entry in the server-side playlist.

The server MUST NOT send this header unless it was rewinding the content (that is, streaming the content backward) and sent the Announce request as a result of reaching the beginning of the playlist entry.

The syntax of the X-Receding-PlaylistChange header is as follows:.

```
X-Receding-PlaylistChange = "X-Receding-PlaylistChange: 1"
```

2.2.6.24 X-RTP-Info

The X-RTP-Info header is sent by servers in the Announce request when predictive stream selection is used. This header is also sent by clients in the Play request when predictive stream selection is used. For more information on predictive stream selection, see section 2.2.6.7.5.

When sent by the server in the Announce request, the X-RTP-Info header specifies what streams the server has selected for the new playlist entry and specifies transport parameters for each stream. When sent by the client in the Play request, the header lists the predicted streams that the client wants to continue to receive. Any streams that the client no longer wants to receive are omitted from the X-RTP-Info header.

The syntax of the X-RTP-Info header is modeled after the syntax of the RTP-Info header, as specified in [RFC2326] section 12.33, but it contains additional syntax elements that are normally found in the Transport (section 2.2.6.8) header.

The syntax of the X-RTP-Info header is as follows.

```
parameter    = ";blocksize=" 1*4DIGIT
             | ( ";client_port="
```

```
                    port [ "-" port ] )     ; section 12.39 of [RFC2326]
                | fec-parameters             ; section 2.2.6.8
                | ( "interleaved="
                    channel "-" channel )    ; section 12.39 of [RFC2326]
                | ";mode=PLAY"
                | ";rtptime=" 1*10DIGIT      ; section 12.33 of [RFC2326]
                | ( ";server_port="
                    port [ "-" port ] )     ; section 12.39 of [RFC2326]
                | ";seq=" 1*4DIGIT           ; section 12.33 of [RFC2326]
                | ";ssrc=" 8HEXDIG           ; section 12.39 of [RFC2326]
                | ( ";thinlevel=" ( "0" | "1" | "2" ) )
                | ( ";transport=RTP/AVP/" ( "UDP" | "TCP" ) )
                | ";unicast"

X-RTP-Info    = "X-RTP-Info: "
                  1# ( stream-url          ; section 12.33 of [RFC2326]
                       1*parameter )
```

When the X-RTP-Info header is sent in a Play request, any parameter syntax elements MUST NOT be present.

The *unicast* parameter MUST be specified in the X-RTP-Info header.

The value of the *blocksize* parameter, if specified, is the maximum size of the RTP packets that the server will send for the new playlist entry described by the SDP in the Announce request.

The numerical value of the *thinlevel* parameter MUST be set as follows.

Value	Meaning
0	All ASF media objects for the stream will be transmitted.
1	Only ASF media objects that are marked as containing key-frame data are to be transmitted.
2	No ASF media objects for the stream will be transmitted.

If the *thinlevel* parameter is missing from the X-RTP-Info header in an Announce request, a value of 0 MUST be assumed.

2.2.6.25 X-StartupProfile

The X-StartupProfile header specifies a list of streaming bit rates. For each bit rate, this header specifies the maximum amount of data that the audio and video decoders will need, and the time stamp of the ASF payload at which this maximum occurs. It is recommended that clients buffer at least the amount of data that this header indicates to prevent buffer underflow.

This header is defined only for use in responses sent to a client.<9> For more information about the X-StartupProfile header, including syntax, see [MS-WMSP] section 2.2.1.12.

2.2.7 Request Types

RTSP Windows Media Extensions define logical requests that are sent from the client to the server, or from the server to the client.

The requests from the client and the corresponding responses from the server are exchanged by using RTSP request methods. Each request type is mapped by using one of the following RTSP methods: ANNOUNCE, DESCRIBE, GET_PARAMETER, PAUSE, PLAY, SET_PARAMETER, SETUP, and TEARDOWN. For example, the EndOfStream request is mapped to the SET_PARAMETER method.

All RTSP methods specify a URI on the request line (as specified in [RFC2326] section 6). Unless otherwise specified, when an RTSP method is used to implement one of the logical requests defined in this section, the URI specified in the RTSP request line MUST be the aggregate control URL (that is, the URL that references the entire presentation rather than an individual stream).

This section defines the syntax of the RTSP headers by using the ABNF syntax, as specified in [RFC4234]. Any ABNF syntax rules that are not specified in [RFC4234] use the ABNF extensions specified in [RFC2326].

2.2.7.1 Announce

The purpose of the Announce request is to send information to the client on a new entry in a server-side playlist that the server will start to stream. The Announce request describes the playlist entry by using SDP syntax (for more information, see section 2.2.5), providing (among other things) the ASF file header as well as a URL for each stream, and information on the RTP payload format for each stream.

The Announce request is implemented by using the ANNOUNCE request method (sent by the server to the client) and MUST adhere to the syntax for ANNOUNCE (as specified in [RFC2326] section 10.3).

The Announce request MUST include the Session header (as specified in [RFC2326] section 12.37) and MUST include the X-Playlist-Gen-Id (section 2.2.6.19) header. If applicable, the X-Broadcast-Id (section 2.2.6.13) header MUST also be included (applies only to broadcast content).

If the new playlist entry that this Announce request describes is the previous entry in the playlist (as opposed to the next entry), the X-Receding-PlaylistChange (section 2.2.6.23) MUST be included in the request. If the new playlist entry is the next entry (the normal case during streaming in the forward direction), then the X-Receding-PlaylistChange MUST NOT be included.

The Announce request SHOULD specify the Cache-Control (section 2.2.6.2) header, because different cache control directives can apply to each entry in a server-side playlist.

The message body in the Announce request MUST contain the SDP, as specified in section 2.2.5. The SDP MUST be identified by the media type application/sdp on the Content-Type (section 2.2.6.3) header.

If the server is using predictive stream selection (that is, it will start streaming the content described by the SDP without waiting for a Play request from the client), then the Announce request MUST include the X-RTP-Info header (section 2.2.6.24). For information on predictive stream selection, see section 2.2.6.7.5.

The following example shows an Announce request (message body omitted).

```
ANNOUNCE rtsp://myserver.com/ServerSidePlaylist.wsx RTSP/1.0
Content-Type: application/sdp
Vary: Accept
Session: 13856065358275910855
X-Playlist-Gen-Id: 5353
X-Broadcast-Id: 73
```

```
X-RTP-Info: url=rtsp://myserver.com/ServerSidePlaylist.wsx
/audio;transport=RTP/AVP/UDP;unicast;server_port=5004;
client_port=1790;ssrc=90cbcaac;mode=PLAY;
blocksize=5994;thinlevel=0;seq=26968;rtptime=0,
url=rtsp://myserver.com/ServerSidePlaylist.wsx/rtx;
transport=RTP/AVP/UDP;
unicast;server_port=5004-5005;client_port=1788-1789;
ssrc=e740fe80;mode=PLAY;thinlevel=0
Content-Length: 61337
Date: Tue, 10 Sep 2002 23:09:36 GMT
CSeq: 2
User-Agent: WMServer/9.0.0.3191
```

2.2.7.2 Describe

The purpose of the Describe request is to request information on one particular piece of multimedia content, which is identified by a URL. The client sends this request before it asks the server to start streaming the content. The server's response describes the content using SDP syntax (for more information, see section 2.2.5), providing (among other things) the ASF file header as well as a URL for each stream and information on the RTP payload format for each stream. If the URL in the Describe request identifies a server-side playlist, the SDP in the Describe response describes only the first entry in the playlist.

The Describe request is implemented by using the DESCRIBE request method, and MUST adhere to the syntax for DESCRIBE, as specified in [RFC2326] section 10.2.

The Describe request also MUST include the User-Agent (section 2.2.6.9) header, and SHOULD include the Supported (section 2.2.6.7) header and the X-Accept-Authentication (section 2.2.6.11) header. The Cookie (section 2.2.6.4) header MUST be included if there are any applicable cookies to send to the server.

The response to the Describe request SHOULD specify the Cache-Control (section 2.2.6.2) header.

The message body in the response to the Describe request MUST contain the SDP in accordance with the rules specified in section 2.2.5. The SDP MUST be identified by the media type application/sdp on the Content-Type (section 2.2.6.3) header.

The following example shows a Describe request.

```
DESCRIBE rtsp://myserver.com/mycontent.wmv RTSP/1.0
User-Agent: WMPlayer/9.0.0.2833 guid/B64345F5-8C45-4818-8A1A-4775F0923FAC
Accept: application/sdp
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-US, *;q=0.1
CSeq: 1
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.eosmsg, com.microsoft.wm.predstrm
```

2.2.7.3 EndOfStream

The purpose of the EndOfStream request is to inform the client that the server has transmitted the last RTP packet for all of the selected streams in the content.

The EndOfStream request is implemented by using the SET_PARAMETER request method (sent by the server to the client) and MUST adhere to the syntax for SET_PARAMETER, as specified in [RFC2326] section 10.9.

If the content is a part of a server-side playlist, the SET_PARAMETER request MUST include the X-Playlist-Gen-Id (section 2.2.6.19) header. If applicable, the X-Broadcast-Id (section 2.2.6.13) header MUST also be included (this applies only to broadcast content).

If one or more audio or video streams were selected by the client for the content, the SET_PARAMETER request MUST include the RTP-Info header, as specified in [RFC2326] section 12.33. The *seq* parameter MUST be specified for each stream-url on the RTP-Info header. The *rtptime* parameter SHOULD NOT be specified.

The purpose of the RTP-Info header is to allow the client to determine the RTP sequence number of the last RTP packet transmitted for each stream. Because the *seq* parameter specifies the RTP sequence number of the first RTP packet transmitted after the SET_PARAMETER request, it follows that the RTP sequence number of the last RTP packet transmitted prior to the SET_PARAMETER request is equal to the value of the *seq* parameter minus 1, modulo 65536.

The X-RTP-Info header MUST NOT include a stream-url section for streams used for retransmitted RTP packets and packet-pair data. For information on how to determine the types of different streams, see section 2.2.5.2.5.

The SET_PARAMETER request MUST include the X-Notice (section 2.2.6.15) header.

The Content-Type header MUST be present in the request and MUST specify the media type application/x-wms-extension-cmd.

The message body of the SET_PARAMETER request MUST adhere to the following syntax (all characters MUST be US-ASCII).

```
message-body       = "Session: " session-id      ; Section 3.4 of [RFC2326]
                     CRLF
                     "EOF: true" CRLF
                     [ ( "AdministrativeDisconnection: true"
                       | "End-Of-Playlist-Entry: true"
                       | "RecedingEos: true" )
                    CRLF ]
```

If the content is part of a server-side playlist, and the server intends to send an Announce request after this EndOfStream request, then the syntax element "End-Of-Playlist-Entry: true" MUST be present.

If the server is rewinding the content (that is, streaming the content for playback in the reverse direction) and reached the start of the content (or playlist entry), the syntax element "RecedingEos: true" MUST be present.

If the server is sending the EndOfStream request because a system administrator decided that the server should stop streaming to the client, then the "AdministrativeDisconnection: true" syntax element SHOULD be present.

Otherwise, the preceding three syntax elements MUST NOT be present.

The following example shows an EndOfStream request.

```
SET_PARAMETER rtsp://myserver.com/ServerSidePlaylist.wsx RTSP/1.0
Content-Type: application/x-wms-extension-cmd
X-Notice: 2101 "End-of-Stream Reached"
RTP-Info: url=rtsp://myserver.com/ServerSidePlaylist.wsx/audio;
Seq=26968, url=rtsp://myserver.com/ServerSidePlaylist.wsx/video;
Seq=46497
X-Playlist-Gen-Id: 5351
Content-Length: 71
Date: Tue, 10 Sep 2002 23:09:36 GMT
CSeq: 5
User-Agent: WMServer/9.0.0.3191

Session: 13856065358275910855
EOF: true
End-Of-Playlist-Entry: true
```

2.2.7.4 GetContentInfo

The purpose of the GetContentInfo request is to retrieve cache-control information from the server without incurring the overhead of a Describe request. This request is normally sent only by clients that are acting as intermediate devices (for example, a caching proxy server).

The GetContentInfo request is implemented by using the GET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for GET_PARAMETER, as specified in [RFC2326] section 10.8.

The Content-Type header MUST be present in the request, and MUST specify the media type application/x-wms-getcontentinfo (section 2.2.6.3.5). The size of the message body MUST be 1 byte. The value of this byte MUST be 0x00.

A server that receives a GET_PARAMETER request MUST treat it as a GetContentInfo request if the Content-Type header specifies the application/x-wms-getcontentinfo media type.

The response to a GetContentInfo request MUST specify the Cache-Control (section 2.2.6.2) header, and MUST contain a zero-length message body.

The following example shows a GetContentInfo request (message body omitted).

```
GET_PARAMETER rtsp://myserver.com/mycontent.wmv RTSP/1.0
Content-Length: 1
User-Agent: WMCacheProxy/9.0.0.3191
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-US, *;q=0.1
Content-Type: application/x-wms-getcontentinfo
CSeq: 1
```

2.2.7.5 KeepAlive

The purpose of the KeepAlive request is to request that the RTSP session specified by the Session header be kept alive. Clients need to send this request if they have not sent any other requests during the time interval specified by the server in the Session header, as specified in [RFC2326]

section 12.37. Clients that are receiving RTP packets over TCP do not need to send this request while the server is sending RTP packets.

The KeepAlive request is implemented by using the GET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for GET_PARAMETER, as specified in [RFC2326] section 10.8.

The GET_PARAMETER request method MUST include the Session header (as specified in [RFC2326] section 12.37).

The Content-Type (section 2.2.6.3) MUST NOT be included in the GET_PARAMETER request method.

The request MUST NOT have a message body.

A server that receives a GET_PARAMETER request MUST treat it as a KeepAlive request if the Content-Type header is not specified in the request.

The server's response to the KeepAlive request SHOULD NOT have a message body.

The following example shows a KeepAlive request.

```
GET PARAMETER rtsp://myserver.com/mycontent.wmv RTSP/1.0
User-Agent: WMPlayer/9.0.0.2833 guid/B64345F5-8C45-4818-8A1A-4775F0923FAC
Accept-Charset: UTF-8, *;q=0.1
Accept-Language: en-US, *;q=0.1
Session: 13856065358275910855
CSeq: 10
```

2.2.7.6 LogConnect

The purpose of the LogConnect request is to submit statistics on the client to the server. This request is normally sent to the server just as streaming starts for the first time, so the logging message does not include any information on the content being streamed. Instead, the logging message contains information on the client software and client operating system.

The LogConnect request is implemented by using the SET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for SET_PARAMETER, as specified in [RFC2326] section 10.9.

The Content-Type header MUST be present in the request and MUST specify the media type application/x-wms-Logconnectstats (section 2.2.6.3.6).

The remote event included in the message body of the LogConnect request MUST be an XML-format connect-time log, as specified in [MS-WMLOG].

A server that receives a SET_PARAMETER request MUST treat it as a LogConnect request if the Content-Type header specifies the application/x-wms-Logconnectstats media type.

The response to a LogConnect request MUST NOT have a message body.

2.2.7.7 LogPlay

The purpose of the LogPlay request is to submit statistics on the streamed content to the server. The request specifies parameters such as streaming quality and packet transmission statistics.

The LogPlay request is implemented by using the SET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for SET_PARAMETER, as specified in [RFC2326] section 10.9.

The Content-Type header MUST be present in the request and MUST specify the media type application/x-wms-Logplaystats (section 2.2.6.3.7).

The remote event included in the message body of the LogPlay request MUST be an XML-format streaming log or legacy log, as specified in [MS-WMLOG].

A server that receives a SET_PARAMETER request MUST treat it as a LogPlay request if the Content-Type header specifies the application/x-wms-Logplaystats media type.

The response to a LogPlay request MUST NOT have a message body.

2.2.7.8 Pause

The purpose of the Pause request is to request that the server stop streaming RTP packets for all of the currently selected streams.

The Pause request is implemented by using the PAUSE request method and MUST adhere to the syntax for PAUSE, as specified in [RFC2326] section 10.6.

The Pause request MUST include the Session header (as specified in [RFC2326] section 12.37).

The Range (section 2.2.6.5) header MUST NOT be present in the Pause request.

The following example shows a Pause request.

```
PAUSE rtsp://myserver.com/mycontent.wmv RTSP/1.0
User-Agent: WMPlayer/9.0.0.2833 guid/B64345F5-8C45-4818-8A1A-4775F0923FAC
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-us, *;q=0.1
Session: 3358283865419300849
CSeq: 7
```

2.2.7.9 Play

The main purpose of the Play request is to ask the server to start streaming RTP packets for the currently selected streams. If the server has switched to a new entry in a server-side playlist and is using predictive stream selection to select streams on the client's behalf, the Play request is also used as a way for the client to confirm the stream selection made by the server and to confirm that it has started to play the RTP packets for the new playlist entry.

The Play request is implemented by using the PLAY request method and MUST adhere to the syntax for PLAY, as specified in [RFC2326] section 10.5.

The Play request MUST include the Session header (as specified in [RFC2326] section 12.37).

The Play request MUST specify either a Range (section 2.2.6.5) header or an X-Playlist (section 2.2.6.17) header.

If the stream is a part of a server-side playlist, the Play request MUST include the X-Playlist-Gen-Id (section 2.2.6.19) header.

If the Play request is sent in response to an Announce request from the server, and the server is using predictive stream selection (that is, it has already started streaming the streams listed in the X-RTP-Info (section 2.2.6.24) header in the Announce request), the following rule applies: If the streams listed by the server on the X-RTP-Info header include streams that the client wants to receive, the client MUST include the X-RTP-Info header in the Play request; otherwise, the client MUST NOT include the X-RTP-Info header. The X-RTP-Info header in a Play request MUST NOT include any parameter syntax elements.

If predictive stream selection is used, and the client has information on the bottleneck bandwidth in the network path between the server and the client, then the Play request SHOULD include the Bandwidth (section 2.2.6.1) header.

For information on predictive stream selection, see section 2.2.6.7.5.

The response to the Play request SHOULD specify the Cache-Control (section 2.2.6.2) header.

The following example shows a Play request.

```
PLAY rtsp://myserver.com/ServerSidePlaylist.wsx RTSP/1.0
X-Playlist-Seek-Id: 5353
User-Agent: WMPlayer/9.0.0.2868 guid/832BF8C6-D8E4-40D4-A058-C31F3D4A3B65
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-us, *;q=0.1
Session: 13856065358275910855
CSeq: 8
Range: npt=0.000-
Bandwidth: 2147483647
X-Accelerate-Streaming: AccelDuration=8000;AccelBandwidth=1024128
X-RTP-Info: url=rtsp://myserver.com/ServerSidePlaylist.wsx/audio,
 url=rtsp://myserver.com/ServerSidePlaylist.wsx/rtx
```

2.2.7.10 SelectStream

The purpose of the SelectStream request is to ask the server to modify the streaming state of one or two streams in the content. It is possible to ask the server to start or stop streaming a particular stream. It is also possible to ask the server to replace one stream with another and to specify if a stream should be thinned. A thinned stream is a stream for which the server transmits only ASF media objects that are marked as containing key-frame data.

If the server is to start streaming a new stream without replacing some other stream with the new stream, then the client MUST implement the SelectStream request by using the SETUP request method, as specified in section 2.2.7.10.1.

If the server is to stop streaming a stream without replacing that stream with some new stream, then the client MUST implement the SelectStream request by using the TEARDOWN request method, as specified in section 2.2.7.10.2.

If the server is already streaming a particular stream, and the client wants the server to start or stop thinning that stream (that is, start or stop transmitting only ASF media objects that are marked as containing key-frame data), then the client MUST implement the SelectStream request by using the SET_PARAMETER request method, as specified in section 2.2.7.10.3.

If the server is to replace one stream with another stream, and both streams can use the same RTP session, then the client MUST implement the SelectStream request by using the SET_PARAMETER request method, as specified in section 2.2.7.10.3.

It is not possible to use a SelectStream request to replace a pair of streams if they use different RTP sessions, for example, if one stream uses TCP and the other uses UDP. In this case, the client will have to send two SelectStream requests instead, one mapped to the TEARDOWN method and the other one mapped to the SETUP method.

Regardless of the RTSP request method used, the URL specified on the RTSP request line MUST be the stream URL, not the one of the complete RTSP presentation.

If the stream is a part of a server-side playlist, then the SelectStream request MUST include the X-Playlist-Gen-Id (section 2.2.6.19) header.

If the request contains a message body, then the Content-Type header MUST specify the media type application/x-wms-streamswitch (section 2.2.6.3.9).

The message body, when present, MUST adhere to the following ABNF syntax.

```
OldStream          = 1*10DIGIT
NewStream          = 1*10DIGIT
ThinLevel          = "0" | "1" | "2"
OldStreamURI       = rtsp_URL                  ; section 3.2 of [RFC2326]
NewStreamURI       = rtsp_URL                  ; section 3.2 of [RFC2326]
message-body       = "SSEntry:" SP OldStream SP NewStream SP ThinLevel
                     [ SP OldStreamURI ]
                     [ SP NewStreamURI ]
                     CRLF
```

The value of the *OldStream* and *NewStream* parameters MUST be an integer in the range 0 to 4,294,967,295, inclusive. The value MUST be set to the stream number of the stream being referenced, as determined by the "a=stream" attribute (for more information, see section 2.2.5.2.5) in the corresponding SDP media description. If there is no applicable stream number (because a stream is only added or only removed), then the value MUST be set to 4,294,967,295.

The *OldStreamURI* parameter MUST NOT be present if the value of *OldStream* is 4,294,967,295. Otherwise, *OldStreamURI* MUST be present.

The *NewStreamURI* parameter MUST NOT be present if the value of *NewStream* is 4,294,967,295. Otherwise, *NewStreamURI* MUST be present.

The *ThinLevel* parameter specifies the thinning level of the stream identified by the *NewStreamURI* parameter, as described in the following table.

Value	Meaning
0	All ASF media objects for the stream given by *NewStreamURI* are to be transmitted.
1	Only ASF media objects for the stream given by *NewStreamURI* that are marked as containing key-frame data are to be transmitted.
2	No ASF media objects for the stream given by *NewStreamURI* are to be transmitted.

If the value of the *NewStream* parameter is 4,294,967,295, then the value of the *ThinLevel* parameter SHOULD be 0 and MUST be ignored by the server.

The server's response to a SelectStream request MUST NOT include a message body.

2.2.7.10.1 SelectStream Using SETUP

The syntax for the SETUP request method is as specified in [RFC2326] section 10.4.

The SelectStream request MUST include the Transport (section 2.2.6.8) header.

For SelectStream requests for non-retransmission streams, the following rule applies: For transport-spec syntax elements on the Transport header that specify UDP as the transport protocol, each stream MUST specify the same port value on the *client_port* parameter. If the *client_port* parameter specifies a pair of port values, the two values in the pair MUST be different; but for each transport-spec and for every Transport header, the pair MUST be identical.

For retransmission streams, for transport-spec syntax elements on the Transport header that specify UDP as the transport protocol, each stream SHOULD specify a port value on the *client_port* parameter that is different from the port value used by any other stream. (This makes it possible to distinguish retransmitted RTP packets from non-retransmitted RTP packets. For more information, see section 2.2.3.1.)

For information on how to determine if a stream is a retransmission stream, see section 2.2.5.2.5.

If the server is supposed to transmit only ASF media objects that contain key-frame data for the stream identified by the URL in the SETUP request line, then the SETUP request method MUST include the message body, as specified in section 2.2.7.10. Otherwise, the message body MUST NOT be included.

When the message body is included, the value of the *OldStream* parameter MUST be 4,294,967,295 and the value of the *ThinLevel* parameter MUST be 1. The value of the *NewStreamURI* parameter MUST be identical to the URL specified on the SETUP request line.

The response to the SelectStream request MUST include the Transport (section 2.2.6.8) header.

The following example shows a SETUP request (without message body).

```
SETUP rtsp://myserver.com/mycontent.wmv/audio RTSP/1.0
User-Agent: WMPlayer/9.0.0.2833 guid/B64345F5-8C45-4818-8A1A-4775F0923FAC
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-US, *;q=0.1
CSeq: 3
X-Playlist-Gen-Id: 2
Transport: RTP/AVP/TCP;unicast;interleaved=0-1;ssrc=6095d7d7;mode=PLAY
```

2.2.7.10.2 SelectStream Using TEARDOWN

The syntax for the TEARDOWN request method is as specified in [RFC2326] section 10.7.

The URL specified on the TEARDOWN request line MUST be the same URL used in the corresponding SETUP request (for more information, see section 2.2.7.10.1).

If the URL of the stream being deselected by the TEARDOWN request is different from the URL specified on the TEARDOWN request line, then the TEARDOWN request method MUST include the message body, as specified in section 2.2.7.10. Otherwise, the message body MUST NOT be included.

When the message body is included, the value of the *NewStream* parameter MUST be 4,294,967,295 and the value of the *ThinLevel* parameter SHOULD be 0. The value of the *OldStreamURI* parameter MUST be the stream URL of the stream that is being deselected.

For example, if a client first sends a SETUP request for stream 1, and then sends a SelectStream request to replace stream 1 with stream 2 (this is implemented by using the SET_PARAMETER method), and then wants to deselect stream 2, then the URL on the TEARDOWN request line must still be that of stream 1. The URL for stream 2 will be specified in the message body of the TEARDOWN request.

2.2.7.10.3 SelectStream Using SET_PARAMETER

The syntax for the SET_PARAMETER request method is as specified in [RFC2326] section 10.9.

The SET_PARAMETER request method MUST include the message body, as specified in section 2.2.7.10.

If the server is already streaming a particular stream, and the client wants the server to start or stop thinning that stream, that is, start or stop transmitting only ASF media objects that are marked as containing key frame data, then the *OldStream* and *NewStream* parameters of the message body MUST be set to the same value (the stream number of the stream whose thinning status is being modified).

Otherwise, the *OldStream* parameter MUST be set to the stream number of the stream being deselected and the *NewStream* parameter MUST be set to the stream number of the stream that is being selected.

The URL specified on the SET_PARAMETER request line MUST be the same URL that is used to select the stream identified by the *OldStream* parameter in the message body. Note that the stream identified by the *OldStream* parameter may have been selected by either a SETUP request or a previous SET_PARAMETER request.

For example, if a client first sends a SETUP request for stream 1, and then uses SET_PARAMETER to replace stream 1 with stream 2, the URL on the request line must be that of stream 1. If the client subsequently uses SET_PARAMETER to replace stream 2 with stream 3, the URL on the request line must still be that of stream 1, because the URL used to select stream 2 was that of stream 1.

2.2.7.11 SendEvent

The purpose of the SendEvent request is to submit a remote event to the server. The most common remote event is remote-log, which specifies rendering statistics independently of streaming statistics. It is possible for clients to send remote-log events to a server after playing content entirely from a cache without having a streaming connection to the server.

The SendEvent request is implemented by using the SET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for SET_PARAMETER, as specified in [RFC2326] section 10.9.

The Content-Type header MUST be present in the request and MUST specify the media type application/x-wms-sendevent (section 2.2.6.3.8).

The remote event included in the message body of the SendEvent request MUST be one of the types previously specified by the server using the x-wms-event-subscription directive (for more information, see section 2.2.6.2.10) in the Cache-Control header.

The message body of the SET_PARAMETER request MUST adhere to the syntax for remote events, as specified in [MS-WMSP] section 2.2.5.

A server that receives a SET_PARAMETER request MUST treat it as a SendEvent request if the Content-Type header specifies the application/x-wms-sendevent media type.

The response to a SendEvent request MUST NOT have a message body.

2.2.7.12 TcpPacketPair

The purpose of the TcpPacketPair request is to request packet-pair data delivered over the TCP connection. The server's response to this request will contain three packets of random data. The server attempts to ensure that each packet is delivered in a separate TCP segment. By measuring the time it takes for the second TCP segment to arrive, the client can estimate the bottleneck bandwidth in the network path between the server and the client.

The TcpPacketPair request is implemented by using the GET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for GET_PARAMETER, as specified in [RFC2326] section 10.8.

A client MUST NOT send a TcpPacketPair request unless the server has indicated that it supports this using the Supported (section 2.2.6.7) header.

The Content-Type header MUST be present in the request and MUST specify the media type application/x-rtsp-packetpair (section 2.2.6.3.2). The request MUST have an empty message body. That the size of the message body is 0 bytes MUST be indicated with the Content-Length header, as specified in [RFC2326] section 12.14.

A server that receives a GET_PARAMETER request MUST treat it as a TcpPacketPair request if the Content-Type header specifies the application/x-rtsp-packetpair media type.

The Content-Type header MUST be present in the response and MUST specify the media type application/x-rtsp-packetpair (section 2.2.6.3.2). The message body of the response MUST consist of three $P packets, as specified in [MS-WMSP] section 2.2.3.7. The sizes of the $P packets MUST be as specified in [MS-WMSP]; however, when computing the size of the **Reason** field in the $P packets, the size of the RTSP response MUST be used instead of the size of the HTTP response.

The following example shows a TcpPacketPair request.

```
GET_PARAMETER rtsp://myserver.com/mycontent.wmv RTSP/1.0
Content-Length: 0
User-Agent: WMPlayer/9.0.0.2833 guid/B64345F5-8C45-4818-8A1A-4775F0923FAC
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-US, *;q=0.1
Content-Type: application/x-rtsp-packetpair
CSeq: 2
```

2.2.7.13 Teardown

The purpose of the Teardown request is to deselect all streams that were previously selected by using SelectStream requests. It also invalidates the RTSP session state.

The Teardown request is implemented by using the TEARDOWN request method (sent by the client to the server) and MUST adhere to the syntax for TEARDOWN, as specified in [RFC2326] section 10.7.

The Teardown request MUST include the Session header (as specified in [RFC2326] section 12.37).

The response to the Teardown request MUST NOT include a message body.

2.2.7.14 UdpPacketPair

The purpose of the UdpPacketPair request is to request packet-pair data delivered as RTP packets over UDP, using the RTP payload format for packet-pair data, as specified in section 2.2.3. The server's response to this request contains an empty message body because the packet-pair data is transmitted over UDP. By measuring the time it takes for the second RTP packet to arrive, the client can estimate the bottleneck bandwidth in the network path between the server and the client.

The client needs to have sent a SelectStream request to establish an RTP session for a retransmission stream before it sends the UdpPacketPair request because the RTP packets with packet-pair data will be transmitted over that RTP session. For information on how to determine if a stream is a retransmission stream, see section 2.2.5.2.5.

The UdpPacketPair request is implemented by using the SET_PARAMETER request method (sent by the client to the server) and MUST adhere to the syntax for SET_PARAMETER, as specified in [RFC2326] section 10.9.

A client MUST NOT send a UdpPacketPair request unless the server has indicated that it supports this using the Supported (section 2.2.6.7) header.

The Content-Type header MUST be present in the request and MUST specify the media type application/x-rtsp-udp-packetpair (section 2.2.6.3.3).

The message body of the SET_PARAMETER request MUST adhere to the following syntax (all characters MUST be US-ASCII).

```
message-body        = "type:" SP "high-entropy-packetpair"
                      [ SP "variable-size" ]
```

The optional syntax element *variable-size* SHOULD be specified. If *variable-size* is specified, the server SHOULD decide the size of the RTP packets according to the table specified in section 2.2.3.2. Otherwise, the size of the **Payload** field MUST be 1,456 bytes for the last RTP packet, 1,455 bytes for the second-to-last packet, and so on. For information on rules, see section 2.2.3.2.<10> A server that receives a SET_PARAMETER request MUST treat it as a UdpPacketPair request if the Content-Type header specifies the application/x-rtsp-udp-packetpair media type.

The response to the UdpPacketPair request SHOULD NOT have a message body.

The following example shows a UdpPacketPair request.

```
SET PARAMETER rtsp://myserver.com/mycontent.wmv RTSP/1.0
Content-Length: 43
User-Agent: WMPlayer/9.0.0.2833 guid/B64345F5-8C45-4818-8A1A-4775F0923FAC
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-US, *;q=0.1
Content-Type: application/x-rtsp-udp-packetpair
CSeq: 3

type: high-entropy-packetpair variable-size
```

3 Protocol Details

The following sections specify details of RTSP Windows Media Extensions, including abstract data models and message processing rules.

3.1 Client Details

The state machine for RTSP clients is as specified in [RFC2326] section A.1 and depicted in the following figure. RTSP Windows Media Extensions define an additional state transition: An EndOfStream request can cause the client to transition from PLAYING to READY state. The presence of a caching proxy server introduces an additional state to the RTSP client state machine, as specified in section 4.6. Transitions in and out of the RECORD state are not described in this specification.<11>

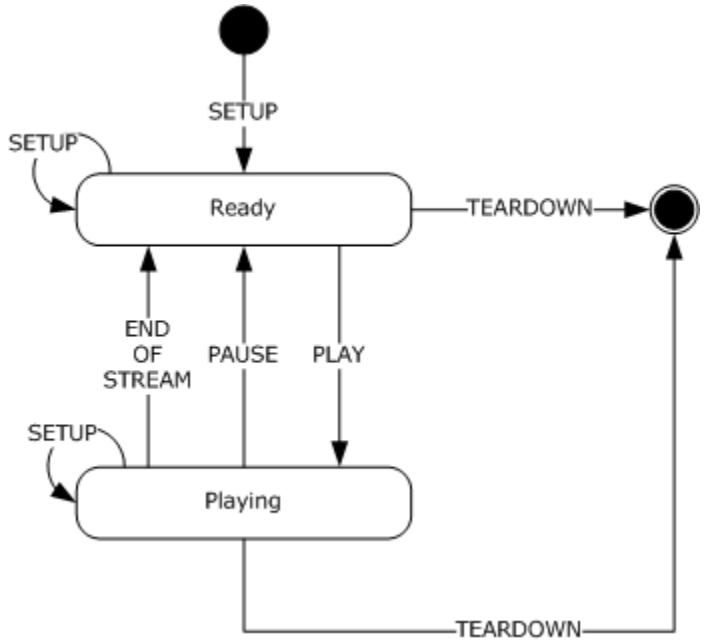


Figure 5: RTSP state diagram (client perspective)

Unless otherwise specified, the protocol reports the occurrence of an error to the higher layer, stops all timers, and stops processing further messages. Possible errors include failure to connect to the server, unexpected closure of the connection to the server, or the response to a request indicating an error.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Keepalive-timeout: This variable stores the frequency at which the client will send KeepAlive requests. The default value is 60 seconds.

Playlist-gen-id: The value of this variable is an identifier assigned by the server to identify the current playlist entry. The default value is 0.

Server-features: This variable stores the capabilities that the server specified on the most recently received Supported header. The default value of this variable is that the server does not support any of the capabilities as specified in section 2.2.6.7.

State: This variable stores the client's state. Possible values are INIT, READY, and PLAYING.

3.1.2 Timers

Firewall: This timer is used when waiting for the RTP packets that contain packet-pair data that the server transmits by using UDP. The minimum-allowed value for the time-out period is 1 second, and the maximum value is 30 seconds.

Keepalive: This timer is used for sending KeepAlive requests. The time-out period is controlled by the Keepalive-timeout variable, as specified in section 3.1.1. The minimum allowed value for the time-out period is 10 seconds. The maximum value of the time-out period is 4,294,967,295 milliseconds.

3.1.3 Initialization

Initialization of the protocol occurs as the result of a higher layer asking for information on multimedia content located on a server. That event is as specified in section 3.1.4.2.

The variables defined by the abstract data model MUST initially assume their default values. Variables that do not have a default defined MUST be initialized as follows.

The State variable MUST be set to INIT.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Request to Retrieve Caching Information

This event can occur when the application is a caching proxy server. The event can occur if the higher layer wants to check if content is available, if it can be cached, and so on, but does not necessarily want to stream it.

The higher layer MUST provide the URL that will be used in the request.

If this will be the first request that is sent by the client, the client MUST perform the initialization of the protocol, as specified in section 3.1.3.

The client MUST then establish a TCP connection to the server by using the IP address and port number obtained by parsing the URL.

The client MUST send a GetContentInfo request to the server, adhering to the syntax specified in section 2.2.7.4.

In addition, the common processing steps, as specified in section 3.1.5.1, MUST be followed when sending the GetContentInfo request.

After sending the request, the client MUST wait for the response to be received. How to process the response is specified in section 3.1.5.3.

3.1.4.2 Request to Retrieve Content Information

This event causes the client to send a Describe request to the server. The following are the most common scenarios in which an application would ask the client for information on multimedia content:

- A media player application that intends to play multimedia content that will be streamed from a server. The media player knows the URL to the content, and it might already know at what time position and at what rate it intends to play the content. However, before it can start playing the content, it needs to, for example, retrieve information on what audio and video streams are included in the content and what decoders will be needed to decompress the content.
- A cache that already has a copy of the content but wants to retrieve information on the content from the server to determine if the cached copy is still fresh.
- A server or intermediate device, such as a noncaching proxy, that is asking for information on behalf of another client.

The higher layer MUST provide the URL that will be specified in all requests sent by the client.

If this will be the first request that is sent by the client, the client MUST perform the initialization of the protocol, as specified in section 3.1.3.

The client MUST then establish a TCP connection to the server, using the IP address and port number obtained by parsing the URL. Next, the client MUST send the Describe request to the server, as specified in section 3.1.4.2.1.

3.1.4.2.1 Sending the Describe Request

The Describe request MUST adhere to the Describe syntax, as specified in section 2.2.7.2.

In addition, the common processing steps, as specified in section 3.1.5.1, MUST be followed when sending the Describe request.

After sending the request, the client MUST wait for the response to be received. How to process the response is specified in section 3.1.5.4.

3.1.4.3 Request to Start Streaming Content

This higher-layer triggered event can occur when the client is not currently streaming from the server. The event causes the client to send one or more SelectStream requests to the server followed by a Play request. The following are the most common scenarios in which an application would ask the client to request the server to start streaming content:

- A media player application that has examined the ASF file header that was received from the server (for more information, see section 3.1.5.4) and determined that it can decompress and play the multimedia content.
- A cache that has determined that the currently cached copy of the content, if any, is either stale or incomplete.
- A server or intermediate device, such as a noncaching proxy, that is asking for content to be streamed on behalf of another client.

Next, the client MUST send a SelectStream request to the server, as specified in section 3.1.5.9.1.

3.1.4.3.1 Sending a SelectStream Request

Because the ASF file header will typically specify multiple streams, the higher layer MUST select exactly what streams that are listed in the ASF file header should be streamed from the server. The client MUST send one SelectStream request for each stream that the higher layer wants to select and that is not yet selected on the server.

All SelectStream requests MUST adhere to the syntax as specified in section 2.2.7.10.

In addition, the common processing steps specified in section 3.1.5.1 MUST be followed when sending a SelectStream request.

The higher layer MUST specify, for each stream, if the stream is supposed to be streamed over UDP or TCP.

For streams that are to be streamed over UDP, the client MUST specify the same UDP port number, or pair of UDP port numbers, in the *client_port* parameter on the Transport header. For more information, see section 2.2.7.10.1.

For each stream that is to be streamed over UDP, if the SDP media description for that stream indicates that the server can transmit RTP packets that contain FEC data, the client SHOULD send a SelectStream request to select RTP packets containing FEC data. (RTP packets containing FEC data have a separate stream URL that has to be explicitly selected by using SelectStream for the server to transmit the FEC RTP packets.) For information on how to determine if FEC RTP packets can be selected and for how to determine the URL to use in the SelectStream request, see section 2.2.5.4.

If the value of the State variable is READY or PLAYING, all of the SelectStream requests SHOULD be pipelined; that is, if there is more than one SelectStream request to send, the client SHOULD send all of them at once without waiting for the response to one request before sending the next one.

If the value of the State variable is INIT, the client MUST only send one SelectStream request. (Subsequent SelectStream requests will be sent after the response to the first one is received.)

The client MUST wait for the response to a SelectStream request. How to process the response is specified in section 3.1.5.9. If the value of the State variable is PLAYING, the client MUST continue to process incoming RTP packets while it is waiting for the response to a SelectStream request.

3.1.4.4 Request to Change Currently Selected Streams

This event occurs when the higher layer wants to change the streams that are currently being streamed. For example, the higher layer may have decided to switch from an English language audio stream to a Spanish language audio stream, or it may have decided to switch to a stream with higher-quality video.

For every stream that is being replaced by another stream, the client MUST send a SelectStream request. For each stream that is to be added without being replaced by another stream, and for each stream that is to be removed without being replaced by another stream, the client MUST also send a SelectStream request.

Each SelectStream request MUST be sent by following the rules specified in section 3.1.5.9.1.

3.1.4.5 Streams to Play from the New Playlist Entry Are Selected

This event occurs after the client has received the Announce request and has delivered the ASF file header for the new playlist entry to the higher layer. (For more information, see section 3.1.5.15.) When this event occurs, the higher layer is ready to start processing the ASF packets for the new

playlist entry. This is the higher layer's opportunity to select the streams that it wants to receive from the new playlist entry. Because the bit rate needed to stream each playlist entry depends on how the content was encoded and on what streams are selected, the higher layer may also want to specify new values for the parameters that control how much faster than real time (if at all) the content is streamed.

The client MUST compare the list of streams that the higher layer has specified against the streams listed by the server on the X-RTP-Info header in the Announce request. If that header was present in the Announce request, any streams listed on the header are already selected by the server, and any streams not listed are not selected. If the Announce request did not include an X-RTP-Info header, it means that no streams in the new playlist entry have been selected by the server.

If the server has selected a stream for the new playlist entry that the client does not want to receive, the client MUST send a SelectStream request to deselect that stream.

All SelectStream requests MUST adhere to the syntax as specified in section 2.2.7.10.

In addition, the common processing steps, as specified in section 3.1.5.1, MUST be followed when sending a SelectStream request.

The client MUST send a SelectStream request for each stream that it wants to select for the new playlist entry that the server has not already selected. The client SHOULD NOT send a SelectStream request to select streams that the server has already selected except if it wants to change the transport parameters.

For example, if the streams in the first playlist entry are all delivered over UDP, when the server sends an Announce request for the next playlist entry, any streams that the server lists on the X-RTP-Info header will also be delivered over UDP (to exactly the same UDP port specified by the client in an earlier SelectStream request). If the client wants to receive the listed streams over UDP, it does not need to send a SelectStream request. But for any stream that it wants delivered over TCP instead of UDP, it will have to send a SelectStream request.

All of the SelectStream requests MUST be pipelined; that is, if there is more than one SelectStream request to send, the client MUST send all of them at once without waiting for the response to one request before sending the next one.

The client MUST wait for the response to each of the SelectStream requests that it has sent. How to process the response is specified in section 3.1.5.16. The client MUST continue to process incoming RTP packets while it is waiting for the response to the SelectStream requests.

If the client did not send any SelectStream requests, it MUST now send a Play request, as specified in section 3.1.5.16.1.

3.1.4.6 Request to Retransmit Lost RTP Packets

This event occurs if the higher layer has detected that one or more RTP packets containing ASF data packets has been lost, and the higher layer wants to ask the server to resend one or more of the lost RTP packets.

For this higher-layer triggered event to be possible, the value of the State variable in the abstract data model MUST be PLAYING, the client MUST have requested that at least one of the streams be delivered over UDP by using the SelectStream request, and the client MUST have sent a SelectStream request for the retransmission stream (for more information, see section 3.1.5.4).

The **Sequence Number** field in the RTP packets that belong to the same RTP session increment sequentially, and this allows the higher layer to detect if an RTP packet has been lost.

To request that the server resend one or more lost packets, the client MUST fill in the fields of an RTCP NACK message, as specified in section 2.2.4.

The RTCP packet that contains the NACK message MUST also contain an SDES message. The value of the *ssrc* parameter, in the syntax for the **CNAME** field in the SDES message, MUST be set to the numerical value of the ssrc field that the server specified in the Transport (section 2.2.6.8) header in the response to the SelectStream request for the retransmission stream.

The RTCP packet that contains the NACK message MUST be sent to the UDP port for RTCP packets that the server specified in the *server_port* parameter of the Transport header that the server included in the response to the SelectStream request for the retransmission stream.

3.1.4.7 Request to Stop Streaming

This event occurs if the higher layer wants to stop streaming. The end user may have requested that streaming should stop. Or the end user may have requested to seek to some position in the content while the client is currently streaming multimedia content from a different position.

If the value of the State variable is PLAYING, the client MUST send a Pause request, adhering to the syntax as specified in section 2.2.7.8. Otherwise, the client MUST NOT send a Pause request.

The common processing steps, as specified in section 3.1.5.1, MUST be followed when sending the Pause request.

If the value of the State variable is PLAYING, the client MUST send a LogPlay request, adhering to the syntax as specified in section 2.2.7.7. Otherwise, the client MUST NOT send a LogPlay request.

The common processing steps, as specified in section 3.1.5.1, MUST be followed when sending the LogPlay request.

If the client will submit remote-log remote events by using the SendEvent request (as specified in section 3.1.4.9), the logging information included in the LogPlay request MUST be a streaming log, as specified in [MS-WMLOG]. Otherwise, the logging information MUST be a legacy style log, as specified in [MS-WMLOG].

If requests were sent, the client MUST wait for the response to the Pause request. How to process the response is specified in section 3.1.5.17.

3.1.4.8 Request to Change Playback Position

This event occurs when the higher layer wants to start streaming from some specific position in the content.

If the value of the State variable is READY, this event MUST be treated the same way as a request to start streaming, which is as specified in section 3.1.4.3.

If the value of the State variable is PLAYING, the client MUST first request the server to stop streaming, as specified in section 3.1.4.7. Once this has completed, the client MUST request to start streaming at the new playback position, as specified in section 3.1.4.3.

3.1.4.9 Playback of Content Has Finished

This event occurs when the application software is a media player that has finished rendering (that is, playing back) the content in the current playlist entry.

If the client specified the Speed header (as specified in [RFC2326] section 12.35) in the Play request for the current playlist entry, and the server specified the remote-log remote event in the "x-wms-event-subscription" directive (for more information, see section 2.2.6.2.10) on the Cache-Control header, the client MUST send a SendEvent (section 2.2.7.11) request to the server.

After sending the request, the client MUST be prepared for the response to be received. How to process the response is specified in section 3.1.5.19.

If the value of the State variable is PLAYING, the client MUST also be prepared to receive RTP packets and an EndOfStream request. Information on how to process RTP packets is as specified in section 3.1.5.12, and information on how to process an EndOfStream request is as specified in section 3.1.5.13.

3.1.4.10 Request to Finish Streaming Session

This event occurs if the higher layer wants to finish the streaming session. Possible causes are the end user requesting that different content start streaming or the end user attempting to exit the client software application.

If the value of the State variable in the abstract data model is READY or PLAYING, the client MUST send a Teardown (section 2.2.7.13) request.

The common processing steps, as specified in section 3.1.5.1, MUST be followed when sending the Teardown request.

If the Teardown request is sent, the client MUST wait for the response to be received. If the value of the **State** variable is not READY and is not PLAYING, then the client MUST close the TCP connection to the server. Information on how to process the response is specified in section 3.1.5.20.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Sending a Request (All Request Types)

This section specifies common steps that MUST be performed whenever the client sends a request of any of the types, as specified in section 2.2.7, to the server.

If the KeepAlive timer is running, it MUST be restarted (reset).

The request sent by the client MUST NOT specify any of the headers and tokens as specified in section 2.2.7 that are defined only for use in responses.

All headers that are specified in [RFC2326] that are defined as mandatory for requests MUST be included in the request.

The client MUST specify the User-Agent (section 2.2.6.9) header in the request. The client-token syntax element on the User-Agent header MUST be set to "WMCacheProxy" if the client is acting as a caching proxy server. If the client is a server that will relay the content to one or more other clients without caching it, the client-token syntax element on the User-Agent header MUST be set to "WMServer". Otherwise, it MUST be set to "WMPlayer".

If the client-token syntax element on the User-Agent header is set to "WMServer", the User-Agent header MAY also include the client-guid syntax element. Otherwise, the client-guid syntax element MUST be included.<12>

The value of the guid-value syntax element, when specified, MUST be the same GUID for all requests belonging to the same RTSP streaming session. The client MAY specify a different GUID on the guid-value syntax element for different streaming sessions.<13>

The client SHOULD specify the Supported (section 2.2.6.7) header in the request if the request is using the DESCRIBE, SETUP, or PLAY method. Otherwise, the client MAY specify the Supported header in the request. If the Supported header is specified, the header MUST correctly reflect the features that are supported by the client.

The client MUST support the following features: com.microsoft.wm.eosmsg and com.microsoft.wm.sswitch. The client SHOULD support the following features: com.microsoft.wm.predstrm, com.microsoft.wm.srvppair, and com.microsoft.wm.startupprofile.<14>

If the client intends to forward the content to another client by using a different streaming protocol, such as the Windows Media HTTP Streaming Protocol, the client SHOULD specify the com.microsoft.wm.locid token on the Supported header. Otherwise, that token MUST NOT be specified.

If the request is any of the types Describe, SelectStream, Teardown, or UdpPacketPair, and if the Playlist-gen-id variable in the abstract data model has a nonzero value, the client MUST specify the X-Playlist-Gen-Id (section 2.2.6.19) header in the request. The numerical value specified on the X-Playlist-Gen-Id header MUST be equal to the value of the Playlist-gen-id variable.

The client SHOULD specify the X-Accept-Authentication (section 2.2.6.11) and X-Accept-Proxy-Authentication (section 2.2.6.12) headers in the request.

If the client is acting as a proxy server and relaying a request from another client, the request MUST include the Via header (as specified in [RFC2326] section 12.43) in the request.

If the client is acting as a proxy server and relaying a request from another client, the request SHOULD include the X-Proxy-Client-Verb (section 2.2.6.22) header in the request.

If the client is acting as a proxy server and relaying a request from another client, and if that request contains either a User-Agent header or a X-Proxy-Client-Agent header, the client MUST include the X-Proxy-Client-Agent (section 2.2.6.21) header in the request.

When the client is sending the X-Proxy-Client-Agent header in a request, it MUST be identical to the X-Proxy-Client-Agent header that was received in the original request (that is, the request that the client is relaying). If the original request does not have a X-Proxy-Client-Agent header, when the client sends the X-Proxy-Client-Agent header, the user-agent-data syntax element on that header MUST be identical to the user-agent-data syntax element of the User-Agent header in the original request.<15>

If there are any cookies to send for the URL specified on the RTSP request line, the Cookie (section 2.2.6.4) header MUST be included in the request.

If the request contains a non-empty message body, the client MUST specify the Content-Type (section 2.2.6.3) header.

3.1.5.2 Receiving a Response (All Request Types)

This section specifies common steps that MUST be performed whenever the client receives the response to a request that it has sent. These steps MUST be performed prior to any processing that is specific to a particular request type.

The client SHOULD assume that the responses are received in exactly the same order in which the requests are sent. The CSeq header (as specified in [RFC2326] section 12.17) MAY be used to order incoming responses in the unlikely event that they would arrive in the wrong order.

The client MUST check the status code in the response to determine if the request succeeded. The sections dealing with responses for specific request types can have rules for how to handle certain failed requests, and any such rules MUST be followed. However, if the status code indicates that the request failed, the response is for a Describe, Play, or SelectStream request, and no other rules describe how to handle the failure, then this MUST be treated as an error and reported as such to the higher layer. In other cases, responses that indicate a failure SHOULD be processed the same way as if they indicated a success.

The client MUST process the Supported header, if present. Each feature token on the header MUST be added to the Server-features variable in the abstract data model. If the header is present, any feature token not listed on the header MUST be removed from the Server-features variable.

The client MUST process the X-Playlist-Gen-Id (section 2.2.6.19) header, if present. If it is present, the Playlist-gen-id variable in the abstract data model MUST be set to the numerical value specified on that header.

The client MUST process the *timeout* parameter on the Session header, if present. If the Session header is present, the Keepalive-timeout variable in the abstract data model MUST be set to the value of the delta-seconds syntax element. If the *timeout* parameter is missing, the KeepAlive-timeout variable MUST be set to the default value for the delta-seconds syntax element. The Session header is as specified in [RFC2326] section 12.37.

The client SHOULD adhere to the directives specified by the Cache-Control (section 2.2.6.2) header and MUST NOT cache the content unless explicitly allowed by the appropriate directive (that is, must-revalidate, public, or proxy-revalidate).

If the Set-Cookie header is present in the response, the cookies on that header MUST be processed according to the rules specified in section 2.2.6.6.

3.1.5.3 Receiving a GetContentInfo Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the RTSP status code indicates that the request succeeded, the server SHOULD report the information in the Cache-Control header to the higher layer.

3.1.5.4 Receiving a Describe Response

The client MUST first follow the steps as specified in section 3.1.5.2.

An RTSP status code in the response in the range 300 to 305 indicates that the server is asking the client to connect to another server. The client MUST connect to the server specified in the response by following the rules as specified in [RFC2326] section 11.2. This is a brief summary of those rules: If the status code is 305, the URL on the Location header (as specified in [RFC2326] section 12.25) is for a proxy, and the URL used in the Describe request MUST remain unchanged. For status codes 300 to 304, the URL on the Location header MUST replace the URL used in the Describe request. The server MUST close the current TCP connection and establish a new TCP connection to the server or proxy server, as appropriate, depending on the status code. The client MUST then continue by following the steps, as specified in section 3.1.4.2.1.

If the RTSP status code in the response is 401, then the server requires authentication; if the status code is 407, then the proxy server requires authentication. The rules for access authentication, as

specified in [RFC2616] section 11, MUST be followed. When the client is ready to resubmit the Describe request with the authentication challenge, it MUST continue by following the steps defined in section 3.1.4.2.1.<16>

If the RTSP status code indicates that the request succeeded, the client MUST perform the following steps:

The client MUST extract the ASF file header from the SDP (for more information, see section 2.2.5.2.3.1) and SHOULD make it available to the higher layer. If the SDP contains a content description list (for more information, see section 2.2.5.2.3.2), it SHOULD also be made available to the higher layer.

If the higher layer allows RTP packets to be streamed over UDP, and the SDP specifies a retransmission stream (for more information, see section 2.2.5.5), the client SHOULD send a SelectStream request to select that retransmission stream. The SelectStream request MUST specify UDP as the transport protocol.

Otherwise, if the higher layer wants to measure the bottleneck bandwidth on the network path between the server and the client, and the Server-features variable in the abstract data model indicates that the server supports com.microsoft.wm.srvppair (section 2.2.6.7.6), the client MUST send a TcpPacketPair request to the server.

If the client has sent the SelectStream request, it MUST wait for the response to the SelectStream request. How to process the response is specified in section 3.1.5.6.

If the client has sent the TcpPacketPair request, it MUST wait for the response to the TcpPacketPair request. How to process the response is specified in section 3.1.5.5.

Otherwise, the client MUST wait until a higher-layer triggered event occurs. (Typically, the next higher-layer triggered event will be a request to start streaming content, as specified in section 3.1.4.3.)

3.1.5.5 Receiving a TcpPacketPair Response

The client MUST first follow the steps as specified in section 3.1.5.2.

Section 2.2.7.12 specifies that the message body of the response consists of three $P packets. As specified in [MS-WMSP] section 2.2.3.7, the client MUST process the $P packets as soon as each $P packet is received, as opposed to waiting for the entire message body to be received.

As soon as the first $P packet has been completely received, the client SHOULD start measuring the time until the second $P packet is received. When the second $P packet has been completely received, the client can use the time elapsed between receiving the first $P packet and the second $P packet to compute the bit rate at which the second $P packet was transferred. When the entire message body has been received, the client SHOULD make this information available to a higher layer.

The client MUST now wait until a higher-layer triggered event occurs. (Typically, the next higher-layer triggered event will be a request to start streaming content, as specified in section 3.1.4.3.)

3.1.5.6 Receiving a SelectStream Response for the Retransmission Stream

The client MUST first follow the steps as specified in section 3.1.5.2.

If the value of the State variable is INIT, it MUST be set to READY.

If the Server-features variable indicates that the server supports com.microsoft.wm.srvppair (section 2.2.6.7.6), the client SHOULD send a UdpPacketPair request to the server.

If the client has sent the UdpPacketPair request, the client MUST start the Firewall timer. The time-out of the Firewall timer SHOULD be 10 seconds plus (if it can be determined) half the round-trip time between the server and the client.

If the client has sent the UdpPacketPair request, the client MUST wait until one of the following happens: It receives the response to the UdpPacketPair request, it receives an RTP packet containing packet-pair data, or the Firewall timer expires. Information on how to process the response to the UdpPacketPair request is as specified in section 3.1.5.7. How to process RTP packets containing packet-pair data is as specified in section 3.1.5.8.

Otherwise, the client MUST wait until a higher-layer triggered event occurs. (Typically, the next higher-layer triggered event will be a request to start streaming content, which is specified in section 3.1.4.3.)

3.1.5.7 Receiving a UdpPacketPair Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the Firewall timer is still running, the client MUST wait for an RTP packet containing packet-pair data to be received. How to process RTP packets containing packet-pair data is specified in section 3.1.5.8.

Otherwise, the client MUST wait until a higher-layer triggered event occurs. (Typically, the next higher-layer triggered event will be a request to start streaming content, as specified in section 3.1.4.3.)

3.1.5.8 Receiving an RTP Packet Containing Packet-Pair Data

The client MUST verify that the RTP packet is compliant with the syntax as specified in section 2.2.3.2. If the value of the Server-features variable in the abstract data model indicates that the server supports the com.microsoft.wm.srvppair (section 2.2.6.7.6), the client SHOULD validate that the SSRC field in the RTP packet is identical to the value of the "ssrc" parameter on the Transport header in the server's response to the SelectStream request for the retransmission stream. Otherwise, the client MUST ignore the value of the SSRC field in the RTP packet.

If this is the first RTP packet received, the client SHOULD start measuring the time until the second RTP packet is received.

After receiving the first RTP packet, the client MUST wait for the second RTP packet to be received, and then process the rules as previously specified in this section.

If this is the second RTP packet received, the client can use the time elapsed between receiving the first RTP packet and the second RTP packet to compute the bit rate at which the second RTP packet was transferred. The client SHOULD make this information available to a higher layer.

The client can determine if an RTP packet that contains packet-pair data is the last or the second-to-last such RTP packet by examining the size of the **Payload** field. For more information, see section 2.2.3.2.

If this is the second-to-last RTP packet containing packet-pair data, the client MUST wait for the last RTP packet containing packet-pair data to be received, and then process the rules as specified previously in this section.

After the last RTP packet containing packet-pair data has been received, the client MUST stop the Firewall timer if it is running. The client MUST then wait until a higher-layer triggered event occurs. (Typically, the next higher-layer triggered event will be a request to start streaming content, as specified in section 3.1.4.3.)

3.1.5.9 Receiving a SelectStream Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the value of the State variable is INIT, the State variable MUST be set to READY.

If any responses to previously transmitted requests are still pending, the client MUST be prepared to receive a response. If the value of the State variable is PLAYING, the client MUST also be prepared to receive an RTP packet.

Otherwise, if there are no pending responses to previously transmitted requests, and if the higher layer has selected streams for which the client has not yet sent a SelectStream request, the client MUST now send one or more additional SelectStream requests by processing the rules as specified in section 3.1.4.3.1.

Otherwise, if the value of the State variable is PLAYING, the client MUST wait for an RTP packet to be received or for a higher-layer triggered event to occur.

How to process RTP packets is specified in section 3.1.5.12.

Otherwise, if the value of the State variable is READY, the client MUST send a Play request, following the rules as specified in section 3.1.5.9.1.

3.1.5.9.1 Sending a Play Request in READY State

The Play request MUST adhere to the syntax as specified in section 2.2.7.9.

In addition, the common processing steps as specified in section 3.1.5.1 MUST be followed when sending the Play request.

The higher layer SHOULD provide either the time position or the ASF packet number from which the server should be asked to start streaming. If a time position is provided, the client MUST send this information by using the npt-range syntax element on the Range (section 2.2.6.5) header. If an ASF packet number is provided, the client MUST send this information by using either the x-asf-byte syntax element (for more information, see section 2.2.6.5.1) or the x-asf-packet syntax element (for more information, see section 2.2.6.5.2) on the Range header. Otherwise, the Range header MUST be omitted.

The higher layer MUST specify the playlist entry ID of the current playlist entry on the X-Playlist-Seek-Id (section 2.2.6.20) header. If the X-Playlist-Seek-Id header is present, the playlist entry ID on the X-Playlist-Seek-Id header identifies the entry that the client wants to skip from. Otherwise, if the Range header is present, the playlist entry ID on the X-Playlist-Seek-Id header identifies the entry that the Range header applies to.

Usually the playlist entry identified by the X-Playlist-Seek-Id header will be the same playlist entry given by the client's Playlist-gen-id variable. But it can happen that the client has recently received an Announce request and updated its Playlist-gen-id variable, and that the higher layer has not yet processed the fact that the playlist entry has changed. In that case, the X-Playlist-Seek-Id header will identify the previous playlist entry from the server's point of view even though it is still the current playlist entry from the client's point of view.

The higher layer MUST specify at what rate the multimedia content should be played back. For example, if the higher layer wants to play the multimedia content in reverse, it MUST specify this and the rate of playback. The client MUST send this information by using the Scale header, as specified in [RFC2326] section 12.34.

The higher layer SHOULD specify both an amount of data that should be streamed faster than real time and the bit rate at which the server should stream this data. The client SHOULD send this information to the server by using the X-Accelerate-Streaming (section 2.2.6.10) header. The higher layer SHOULD also specify the bit rate that can be used for streaming between the server and the client. The client SHOULD send this information to the server by using the Bandwidth (section 2.2.6.1) header.

If the client is acting as a proxy and will relay the content to another client, the higher layer SHOULD specify both an alternate amount of data that should be streamed faster than real time and an alternate bit rate at which the server should stream this data. These alternate amounts are for the client's own behalf, as opposed to the values specified on the X-Accelerate-Streaming header, which are on the behalf of the other (external) client. The client SHOULD send the alternate amount and alternate bit rate to the server by using the X-Burst-Streaming (section 2.2.6.14) header.

If the client supports the X-StartupProfile (section 2.2.6.25) header, and the value of the Server-features variable indicates that the server supports the com.microsoft.wm.startupprofile feature, the client SHOULD specify the com.microsoft.wm.startupprofile token on the Supported header when it sends the Play request. A client that does not support the X-StartupProfile header MUST NOT include this token in the Supported header.

The higher layer SHOULD specify that the entire content should be streamed faster than real time at some transmission rate chosen by the higher layer. If the Server-features variable indicates that the server supports the com.microsoft.wm.fastcache (section 2.2.6.7.2) feature, the client SHOULD send this information to the server by using the Speed header, as specified in [RFC2326] section 12.35. The Speed header MUST NOT be included in the request unless the server has explicitly specified that it supports the com.microsoft.wm.fastcache feature.

After sending the Play request, if the client has not previously sent a LogConnect request in this RTSP session, the client MUST send a LogConnect request.

The client MUST now wait for the response to the Play request to be received. How to process the response is specified in section 3.1.5.10.

3.1.5.10 Receiving a Play Response

The client MUST first follow the steps as specified in section 3.1.5.2.

The value of the State variable MUST be set to PLAYING.

If at least one of the selected streams is delivered over TCP, the KeepAlive timer SHOULD be stopped.

The client MUST be prepared to receive RTP packets. How to process RTP packets is specified in section 3.1.5.12. The client MUST also be prepared to receive an EndOfStream request. How to process this request is specified in section 3.1.5.13.

If the client sent a LogConnect request immediately after sending the Play request, the client MUST now be prepared to receive the LogConnect response. How to process that response is specified in section 3.1.5.11.

3.1.5.11 Receiving a LogConnect Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the value of the State variable is PLAYING, the client MUST be prepared to receive RTP packets. How to process RTP packets is specified in section 3.1.5.12. The client MUST also be prepared to receive an EndOfStream request. How to process this request is specified in section 3.1.5.13.

Otherwise, the client MUST wait for a higher-layer triggered event.

3.1.5.12 Receiving RTP Packets

The client MUST verify that the RTP packet is compliant with the RTP payload format syntax. The RTP payload format for ASF data packets is as specified in section 2.2.1. The RTP payload format for FEC packets is as specified in section 2.2.2. The RTP payload format for retransmitted RTP packets is as specified in section 2.2.3.1.

Information on how to determine what RTP payload format is used for a packet is as specified in [RFC3550]. In summary, it involves determining the RTP session by examining the UDP destination port number (if UDP is used) and the **SSRC** field in the RTP header. It then involves checking the value of the **Payload Type** field in the RTP header and cross-referencing this with the RTP payload format for that value, as specified in SDP [RFC2327].

When receiving RTP packets that contain FEC data, the client SHOULD use the FEC data to reconstruct any RTP packets that were lost.

When receiving RTP packets that contain ASF data packets, the ASF data packets MUST be extracted from the RTP packets. If an ASF data packet has been split up across multiple RTP packets, the client MUST recombine the ASF data packet once all the relevant RTP packets are received. For information on how to determine if an RTP packet contains a complete ASF data packet or a fragmented ASF data packet, see section 2.2.1.3.

The client SHOULD make the ASF data packet and the value of the **Sequence Number** field in the RTP header available to the higher layer. (The value of the **Sequence Number** field can help the higher layer determine if an RTP packet has been lost.)

If the client has sent any requests for which it has not yet received a response, it MUST be prepared to receive the response. The client MUST also be prepared to receive more RTP packets and to follow the rules specified in this section for each received RTP packet.

The client MUST also be prepared for a higher-layer triggered event to occur and to receive an EndOfStream request. How to process an EndOfStream request is specified in section 3.1.5.13.

3.1.5.13 Receiving an EndOfStream Request

The client MUST validate that the request adheres to the syntax as specified in section 2.2.7.3.

The client MUST send a response to the EndOfStream request.

If the RTP-Info header is present in the request, for each stream specified on that header, the client MUST compare the value of the *seq* parameter against the value of the **Sequence Number** field in the RTP header of the most recent RTP packet. If the *seq* parameter indicates that the server has transmitted one or more RTP packets that have not yet been received by the client, the client SHOULD wait for those remaining RTP packets to be received and process the rules, as specified in section 3.1.5.12, for each RTP packet that is received. Processing of the rules, as specified in section 3.1.5.13, MUST continue once the missing RTP packets are received.

If the message body of the request does not contain the "End-Of-Playlist-Entry: true" syntax element, the State variable MUST be set to READY.

If the Keepalive timer is stopped, it MUST be started.

If the most recently received Play response contains the X-Playlist-Change-Notice (section 2.2.6.18) header, the client MUST NOT send a LogPlay request. Otherwise, the client MUST send a LogPlay (section 2.2.7.7) request.

If the client will submit remote-log remote events by using the SendEvent request, as specified in section 3.1.4.9, the logging information included in the LogPlay request MUST be a streaming log, as specified in [MS-WMLOG]. Otherwise, the logging information MUST be a legacy style log, as specified in [MS-WMLOG].

If the client sent a LogPlay request, the client MUST be prepared to receive the response to that request. How to process that response is specified in section 3.1.5.14.

If the State variable is PLAYING, the client MUST also be prepared to receive an Announce request. How to process this request is specified in section 3.1.5.15.

Otherwise, the client MUST be prepared for a higher-layer triggered event to occur.

3.1.5.14 Receiving a LogPlay Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the value of the State variable is PLAYING, the client MUST wait for an Announce request to be received. How to process this request is specified in section 3.1.5.15.

Otherwise, the client MUST wait for a higher-layer triggered event.

3.1.5.15 Receiving an Announce Request

The client MUST validate that the request adheres to the syntax as specified in section 2.2.7.1.

The client MUST process Supported header, if present. Each feature token on the header MUST be added to the Server-features variable in the abstract data model. If the header is present, any feature token not listed on the header MUST be removed from the Server-features variable.

The client MUST process the X-Playlist-Gen-Id (section 2.2.6.19) header, if present. If it is present, the Playlist-gen-id variable in the abstract data model MUST be set to the numerical value specified on that header.

The client MUST extract the ASF file header from the SDP (for more information, see section 2.2.5.2.3.1), and SHOULD make it available to the higher layer. If the SDP contains a content description list (for more information, see section 2.2.5.2.3.2), it SHOULD also be made available to the higher layer.

The client MUST send a response to the Announce request.

If the X-RTP-Info header is present in the request, it indicates the streams that the server has selected for the playlist entry described by the SDP. The client needs to remember this because it will need this information once it is ready to select the streams for this playlist entry that it wants to receive. For more information, see section 3.1.4.5.

If the value of the State variable is PLAYING, the client MUST be prepared to receive RTP packets from any of the streams listed on the X-RTP-Info header. How to process RTP packets is specified in section 3.1.5.12.

The client MUST also be prepared to receive a higher-layer triggered event. (Typically, the next higher-layer triggered event will be a request to select the streams from the new playlist entry. This event is as specified in section 3.1.4.5.)

3.1.5.16 Receiving a SelectStream Response After Announce

The client MUST first follow the steps as specified in section 3.1.5.2.

If any responses to previously transmitted requests are still pending, the client MUST be prepared to receive a response. For any SelectStream request for which a response is pending, the client MUST follow the rules, as specified in this section. The client MUST also be prepared to receive an RTP packet.

How to process RTP packets is specified in section 3.1.5.12.

If there are no pending responses to previously transmitted requests, the client MUST now send a Play request following the rules as specified in section 3.1.5.16.1.

3.1.5.16.1 Sending a Play Request in PLAYING State

The Play request MUST adhere to the syntax as specified in section 2.2.7.9.

In addition, the common processing steps as specified in section 3.1.5.1 MUST be followed when sending the Play request.

The Range header SHOULD NOT be included in the request.

The higher layer MUST include the X-Playlist-Seek-Id (section 2.2.6.20) header in the request, and the numerical value on the header MUST be set to the value of the Playlist-gen-id variable in the abstract data model.

If the server has selected any streams for the current playlist entry (as indicated by the X-RTP-Info (section 2.2.6.24) header in the Announce request received from the server for this playlist entry), and the client has not sent a SelectStream request for one or more of those streams, the client MUST now include the X-RTP-Info header in the Play request. The X-RTP-Info header MUST list the streams that the server selected for which the client has not sent a SelectStream request (to select or deselect the stream).

The client SHOULD include the X-Player-Lag-Time (section 2.2.6.16) header if the delay between receiving the Announce request for this playlist entry and sending this Play request is greater than normal. (If the start of playback is initially delayed by a few seconds due to buffering, each Play request would normally be delayed by the same amount. This is considered the normal delay, and the client should send only the X-Player-Lag-Time header if the Play request is delayed more than normal.)

The higher layer MUST specify at what rate the multimedia content should be played back. For example, if the higher layer wants to play the multimedia content in reverse, it MUST specify this and the rate of playback. The client MUST send this information by using the Scale header, as specified in [RFC2326] section 12.34.

The client MUST now wait for the response to the Play request to be received. How to process the response is specified in section 3.1.5.10.

3.1.5.17 Receiving a Pause Response

The client MUST first follow the steps as specified in section 3.1.5.2.

The value of the State variable MUST be set to READY.

If the LogPlay request was sent, and the response has not yet been received, the client MUST wait for the response to be received. How to process this response is as specified in section 3.1.5.14.

Otherwise, the client MUST wait for a higher-layer triggered event.

3.1.5.18 Receiving a KeepAlive Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the value of the State variable is PLAYING, the client MUST be prepared to receive RTP packets. How to process RTP packets is specified in section 3.1.5.12. The client MUST also be prepared to receive an EndOfStream request. How to process this request is specified in section 3.1.5.13.

Otherwise, the client MUST wait for a higher-layer triggered event.

3.1.5.19 Receiving a SendEvent Response

The client MUST first follow the steps as specified in section 3.1.5.2.

If the value of the State variable is PLAYING, the client MUST be prepared to receive RTP packets. How to process RTP packets is specified in section 3.1.5.12. The client MUST also be prepared to receive an EndOfStream request. How to process this request is specified in section 3.1.5.13.

Otherwise, the client MUST wait for a higher-layer triggered event.

3.1.5.20 Receiving a Teardown Response

The client MUST first follow the steps as specified in section 3.1.5.2.

The client MUST close the TCP connection to the server.

3.1.6 Timer Events

3.1.6.1 Firewall Timer Expires

The client MUST report to the higher layer that it is not possible to receive RTP packets streamed over UDP. The likely cause is that a firewall is blocking UDP packets.

After this, the client MUST wait for a higher-layer triggered event to occur. (The higher layer may give up, close the TCP connection, and display an error to the user; or the higher layer may simply decide that all streams will be streamed by using TCP instead of UDP.)

3.1.6.2 Keepalive Timer Expires

When the KeepAlive timer expires, the following actions MUST take place:

1. The client MUST send a KeepAlive (section 2.2.7.5) request.

2. After sending the request, the client MUST wait for the response to be received.

Information on how to process the response is specified in section 3.1.5.18.

3.1.7 Other Local Events

3.1.7.1 TCP Connection Is Disconnected

If the TCP connection to the server is disconnected, and the client did not initiate the disconnection, then the client MUST report this as an error to the higher layer.

3.2 Server Details

The state machine for RTSP servers is as specified in [RFC2326] section A.2 and as depicted in the following illustration. RTSP Windows Media Extensions define an additional state transition: A higher-layer event can cause the server to transition from the PLAYING to the READY state. The presence of a caching proxy server introduces an additional state to the RTSP server state machine; this state is specified in section 4.6. Transitions in and out of the RECORD state are not described in this specification.<17>

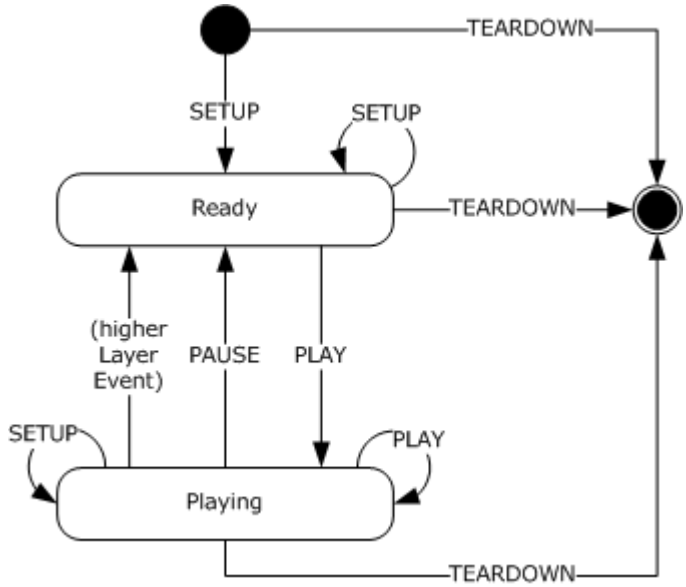


Figure 6: RTSP state diagram (server perspective)

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Client-features: This variable stores the capabilities that the client specified on the most recently received Supported header. The default value of this variable is that the client does not support any of the capabilities as specified in section 2.2.6.7.

Playlist-gen-id: The value of this variable is an identifier assigned by the server to identify the current playlist entry. The default value is 0.

State: This variable stores the server's state. Possible values are INIT, READY, and PLAYING.

3.2.2 Timers

Idle-Timeout: This timer is used for cleaning up unused session state. If no requests are received from the client, the Idle-Timeout timer will expire, and the server is then free to delete the session state. The minimum allowed value for the timeout period is 10 seconds.

3.2.3 Initialization

Initialization of the protocol occurs when a Describe or SelectStream request is received and the request does not specify the Session header (as specified in [RFC2326] section 12.37).

The variables defined by the abstract data model MUST initially assume their default values. Variables that do not have a default defined MUST be initialized as follows:

The State variable MUST be set to INIT.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Notification that the Last RTP Packet Has Been Sent

When the higher layer notifies the server that the last RTP packet has been sent, the server MUST send an EndOfStream (section 2.2.7.3) request to the client.

The higher layer MUST specify if a new ASF header is forthcoming (that is, there are additional entries in the server-side playlist) or if this was the last entry in the playlist, so that the message body in the request can be filled in correctly. For more information, see section 2.2.7.3.

The value of the State variable in the abstract data model MUST be changed to READY.

The Idle-Timeout timer MUST be started.

After sending the EndOfStream request, the server MUST wait for the response to be received. How to process the response is specified in section 3.2.5.13. While waiting for the response, the server MUST also be prepared to receive RTCP packets. How to handle RTCP packets is specified in section 3.2.5.10.

3.2.4.2 Notification that a New ASF File Header Is Available

As a prerequisite for this event, the higher layer MUST already have notified the server that it has sent the last RTP packet for the previous playlist entry, as specified in section 3.2.4.1.

If the most recently received Play request specified the X-Player-Lag-Time (section 2.2.6.16) header, the server SHOULD delay sending the Announce request by the amount of time specified by the X-Player-Lag-Time header.

The server MUST now send an Announce (section 2.2.7.1) request to the client. The SDP in the message body of the Announce request MUST include the ASF file header of the new playlist entry. For more information, see section 2.2.5.

The server MUST change the value of the Playlist-gen-id variable in the abstract data model such that each playlist entry gets a different identifier. This variable MUST be used as the value for the X-Playlist-Gen-Id header in the Announce request. For more information, see section 2.2.7.1.

The server SHOULD specify the Supported (section 2.2.6.7) header in the request. If the Supported header is specified, the header MUST correctly reflect the features that are supported by the server. For information on what feature tokens to list on the Supported header, see section 3.2.5.4.

If the value of the Client-features variable specifies that the client supports the com.microsoft.wm.predstrm (section 2.2.6.7.5) feature, the server has also specified that it supports this feature, and the server has received a Play request for the previous playlist entry, the server MUST select suitable streams from the ASF file header of the new playlist entry. The server SHOULD use the most recently specified value on the Bandwidth (section 2.2.6.1) header to choose what streams to select from the ASF file header.

If the server selected any streams, the Announce request MUST include the X-RTP-Info header, and that header MUST specify what streams the server has selected. See section 2.2.6.24.

If the server selected any streams, the State variable MUST be set to PLAYING.

Otherwise, the State variable MUST be set to READY, and if the Idle-Timeout timer is not running, it MUST be started.

If the State variable is set to PLAYING, the server MUST start streaming RTP packets to the client. RTP packets for the new playlist entry MUST be sent to the same UDP port or TCP channel that was previously specified by the client. For information on rules to follow when sending RTP packets, see section 3.2.5.8.

If the State variable is set to PLAYING, the Idle-Timeout timer MUST be started if all RTP packets are sent over UDP. If at least some RTP packets are sent over TCP, the Idle-Timeout timer MUST be stopped if it is running.

After sending the Announce request, the server MUST wait for the response to be received. How to process the response is specified in section 3.2.5.14. While waiting for the response, the server MUST also be prepared to receive RTCP packets. How to handle RTCP packets is specified in section 3.2.5.10.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Receiving a Request (All Request Types)

This section specifies common steps that MUST be performed whenever the server receives a request from a client. These steps MUST be performed prior to any processing that is specific to a particular request type.

The server MUST validate that the request is of one of the types specified in section 2.2.7 and that the request is using the appropriate RTSP request method. If the validation fails, the server MUST respond with some RTSP error status code such as 400 or 501, as appropriate.

If the request includes the Session header, the server MUST load the state associated with the RTSP session ID specified on that header. If the matching state cannot be found, the server MUST treat this as an error and respond with status code 454 (as specified in [RFC2326] section 12.37).

The server MUST process the Supported (section 2.2.6.7) header, if present. Each feature token on the header MUST be added to the Client-features variable in the abstract data model. If the header is present, any feature token not listed on the header MUST be removed from the Client-features variable.

If the Idle-Timeout timer is running, it MUST be stopped.

3.2.5.2 Receiving a GetContentInfo Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not valid, this is an error, and the server MUST respond with some suitable RTSP error status code such as 404.

The GetContentInfo request does not require any server state. Hence, if the Session header is missing, this MUST NOT be treated as an error.

The GetContentInfo response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.4.

3.2.5.3 Receiving a Describe Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not valid, this is an error, and the server MUST respond with some suitable RTSP error status code such as 404.

The server MUST check with the higher layer to determine if the client is to be redirected to a different server or to a proxy server. (The presence, or absence, of a Via header in the request can be used to determine if the request was delivered directly by the client or through a proxy server. Information on the Via header is as specified in [RFC2326] section 12.43.)

If the higher layer indicates that the client is to be redirected to another server, the server MUST respond with status code 302. If the client is to be redirected to a proxy server, the server SHOULD respond with status code 305. In both cases, the URL of the server, or proxy server, MUST be specified on the Location header in the response. Information on the Location header is as specified in [RFC2326] section 12.25. The RTSP REDIRECT request method MUST NOT be used to redirect the client.

After sending a response with status codes 302 or 305, the server MUST delete the session state, if any, and close the TCP connection to the client.

If the higher layer requires the client to authenticate itself, the server MUST process the Authorization header (as specified in [RFC2326] section 12.5) if it is present in the request. If the server is acting as a proxy server, it MUST process the Proxy-Authorization header (as specified in [RFC2616] section 14.34) instead of the Authorization header.

If it is necessary to send an authentication challenge to the client (for example, because the Authorization header was missing or specified incorrect credentials), the server SHOULD use one of the authentication schemes that the client listed on the X-Accept-Authentication (section 2.2.6.11) header in the request, if that header is present. If the server is acting as a proxy server, it MUST use the X-Accept-Proxy-Authentication (section 2.2.6.12) header instead of the X-Accept-Authentication header.

If the server sends an authentication challenge to the client, it MUST be specified by using the WWW-Authenticate header (as specified in [RFC2326] section 12.44), and the status code of the response MUST be 401. If the server is acting as a proxy server, it MUST specify the Proxy-Authenticate header (as specified in [RFC2326] section 12.26) instead of the WWW-Authenticate header, and the status code of the response MUST be 407.

After sending a response with status code 401 or 407, the server MUST NOT close the TCP connection to the client because the client is expected to resubmit the Describe request with the appropriate credentials.

If the server is not sending a response with an error status code, and if the request does not specify a Session header, the server MUST create new state by performing the initialization procedure as specified in section 3.2.3.

The Describe response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.2.

After sending the response, if the status code indicates success (for example, 200), the server MUST wait for a TcpPacketPair request or a SelectStream request to be received. How to process a TcpPacketPair request is specified in section 3.2.5.5. How to process a SelectStream request is specified in section 3.2.5.6.

If the status code of the response was 401 or 407, the server MUST wait for another Describe request and process it as specified in this section..

3.2.5.4 Sending a Response (All Request Types)

This section specifies common steps that MUST be performed whenever the server sends a response to a request from the client.

The response sent by the server MUST NOT specify any of the headers and tokens, as specified in section 2.2.7, that are defined only for use in requests or responses sent to a server.

All headers that are specified in [RFC2326], that are defined as mandatory for responses, MUST be included in the response.

The server SHOULD specify the Supported (section 2.2.6.7) header in the response. If the Supported header is specified, the header MUST correctly reflect the features that are supported by the server. The server MUST support the com.microsoft.wm.eosmsg (section 2.2.6.7.1) and com.microsoft.wm.sswitch (section 2.2.6.7.7) features. The server SHOULD support the com.microsoft.wm.fastcache (section 2.2.6.7.2), com.microsoft.wm.packetpairssrc (section 2.2.6.7.4), com.microsoft.wm.predstrm (section 2.2.6.7.5), com.microsoft.wm.srvppair (section 2.2.6.7.6), and com.microsoft.wm.startupprofile (section 2.2.6.7.8) features.<18>

When responding to a GetContentInfo, SelectStream, or Play request, the server SHOULD specify the Cache-Control (section 2.2.6.2) header in the response.

When the server includes the Session header in the response, it SHOULD also include the *timeout* parameter (as specified in [RFC2326] section 12.37). The value of the *timeout* parameter MUST be set to a value less than or equal to the timeout interval of the Idle-Timeout timer. It is recommended that the value of the token be at least a few seconds less than the timeout interval to allow for processing delays and network delays.

If the Playlist-gen-id variable in the abstract data model has a nonzero value, the client MUST specify an X-Playlist-Gen-Id (section 2.2.6.19) header in the response. The numerical value specified on the header MUST be equal to the value of the Playlist-gen-id variable.

If the response contains a non-empty message body, the server MUST specify the Content-Type (section 2.2.6.3) header.

The Idle-Timeout timer MUST be started unless the value of the State variable in the abstract data model is PLAYING and RTP packets are transmitted over TCP.

3.2.5.5 Receiving a TcpPacketPair Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The TcpPacketPair request does not require any server state. Hence, if the Session header is missing, this MUST NOT be treated as an error.

The TcpPacketPair response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.12.

As specified in section 2.2.7.12, the response consists of three $P packets. The server SHOULD deliver the $P packets to the TCP layer such that each $P packet is transmitted in a separate TCP segment.

After sending the response, the server MUST wait for another request to be received. Normally, a SelectStream request will be received at this point. How to process this request is specified in section 3.2.5.6.

3.2.5.6 Receiving a SelectStream Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The SelectStream request MUST follow the rules as specified in section 2.2.7.10.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. It MUST be one of the URLs specified by the **control** attribute in SDP (for more information, ee section 2.2.5.2.1), or the URL for one of the FEC streams (for more information, see section 2.2.5.4). If it is not, this is an error, and the server MUST respond with some suitable RTSP error status code such as 400.

If the value of the State variable in the abstract data model is INIT, then it MUST be set to READY. At this point, the server MUST also choose a session ID value for use on the Session header in the response.

Any new streams that are transmitted as a result of the request MUST be transmitted beginning with an ASF key-frame payload. Any streams that are replaced by different streams MUST continue to be transmitted until the first ASF key-frame payload of the new stream is transmitted. Information on how to determine if an ASF payload contains a key-frame is as specified in [ASF].

The SelectStream response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.10.

After sending the response, the server MUST wait for another request to be received.

If the value of the State variable is READY, then another SelectStream request, a UdpPacketPair request, a Play request, a KeepAlive request, a SendEvent request, or a Teardown request are all possible.

If the value of the State variable is PLAYING, then another SelectStream request, a Pause request, a KeepAlive request, a SendEvent request, or a Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a UdpPacketPair request is specified in section 3.2.5.7.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.7 Receiving a UdpPacketPair Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The UdpPacketPair request MUST follow the rules as specified in section 2.2.7.14.

The server SHOULD send three RTP packets containing packet-pair data to the client, or the server MAY send two RTP packets containing packet-pair data to the client. The RTP packets MUST use the RTP payload format for packet-pair data as specified in section 2.2.3.

If the UdpPacketPair request specified the *variable-size* parameter, the first RTP packet sent by the server SHOULD have a 1,283-byte **Payload** field. The size of the **Payload** field of each RTP packet MUST be chosen in accordance with the rules as specified in section 2.2.3.2.<19>

The RTP packets MUST be sent to the UDP port that the client specified as the RTP UDP port in the "client_port" parameter of the Transport (section 2.2.6.8) header that the client included in the SelectStream request for the retransmission stream.

The UdpPacketPair response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.14.

After sending the response, the server MUST wait for another request to be received. Normally, a SelectStream request will be received at this point. How to process this request is specified in section 3.2.5.6.

3.2.5.8 Receiving a Play Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The Play request MUST follow the rules as specified in this section.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not valid, this is an error, and the server MUST respond with some suitable RTSP error status code such as 404.

The value of the State variable in the abstract data model MUST be set to PLAYING.

The Play response MUST follow the rules as specified in this section and in section 3.2.5.4.

The server SHOULD specify the X-Accelerate-Streaming (section 2.2.6.10) header in the response if the client sent that header in the Play request and if the server supports changing the transmission rate based on that header's presence in the request. Otherwise, the server SHOULD NOT specify the X-Accelerate-Streaming header in the response.

The server SHOULD specify theX-Burst-Streaming (section 2.2.6.14) header in the response if the client sent that header in the Play request and if the server supports changing the transmission rate based on that header's presence in the request. Otherwise, the server SHOULD NOT specify the X-Burst-Streaming header in the response.

If the value of the Client-features variable in the abstract data model indicates that the client supports the com.microsoft.wm.startupprofile (section 2.2.6.7.8) feature, the server SHOULD specify the X-StartupProfile (section 2.2.6.25) header in the response.

The server MUST NOT specify the X-StartupProfile header in the response unless the client has specified support for the com.microsoft.wm.startupprofile feature.

If the server will immediately send an EndOfStream request after this response (for example, this would normally happen if the client included the X-Playlist header in the Play request), the Play response MUST include the X-Playlist-Change-Notice (section 2.2.6.18) header.

After sending the response, the server MUST start sending RTP packets to the client (or continue sending them, if the State variable was already PLAYING when the Play request was received).

The RTP packets MUST use the RTP payload for ASF data packets, as specified in section 2.2.1.

The ASF payloads in the ASF packets MUST be filtered such that only ASF payloads that belong to streams that have been selected are included in the ASF packets.

If the Client-features variable indicates that the client supports the com.microsoft.wm.locid (section 2.2.6.7.3) feature, the **LocationId** field of the RTP payload format header MUST be present in all RTP payload format headers that are contained in an RTP packet in which the **M** field in the RTP header is set to 1. Otherwise, the **LocationId** field of the RTP payload format header SHOULD NOT be present.

The Idle-Timeout timer MUST be started if all RTP packets are sent over UDP. If at least some RTP packets are sent over TCP, the Idle-Timeout timer MUST be stopped if it is running.

While sending RTP packets, the server MUST be prepared for another request to be received.

A LogConnect, SelectStream, Pause, KeepAlive, SendEvent, or Teardown request are all possible. The server MUST also be prepared to receive RTCP packets.

How to process a LogConnect request is specified in section 3.2.5.9.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.9 Receiving a LogConnect Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The LogConnect request MUST follow the rules as specified in section 2.2.7.6.

The server SHOULD communicate the logging information submitted by the client to the higher layer.

The LogConnect response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.6.

After sending the response, the server MUST wait for another request to be received.

If the value of the State variable is READY, then a SelectStream, Play, KeepAlive, SendEvent, or Teardown request are all possible.

If the value of the State variable is PLAYING, then a SelectStream, Pause, KeepAlive, SendEvent, or Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.10 Receiving an RTCP Packet

The server MUST verify that the RTCP packet contains a Generic NACK message adhering to the syntax as specified in section 2.2.4.

The server MUST ignore any RTCP packet that does not contain a generic NACK message or an SDES message.

The server MUST verify that the value of the *ssrc* parameter on the **CNAME** field in the SDES message is identical to the numerical value of the ssrc field that the server provided in the Transport (section 2.2.6.8) header in response to the SelectStream request for the retransmission stream.

The server can use the value of the *ssrc* parameter to find the appropriate RTSP session state, but this depends on how the server chose the value for the *ssrc* parameter from the Transport header in the SelectStream response. This is implementation-specific depending on the server and the system involved.

If the value of the **CNAME** field or the value of the *ssrc* parameter does not match what is expected by the server, the server MUST ignore the RTCP packet.

For each generic NACK message that is included in the RTCP packet, the server SHOULD retransmit a copy of the RTP packets that the client specified in the generic NACK message.

Retransmitted RTP packets MUST be sent to the UDP port that the client specified as the RTP UDP port in the *client_port* parameter of the Transport header in the SelectStream request for the retransmission stream.

After this, the server MUST wait for a request to be received.

If the value of the State variable is READY, then a SelectStream, Play, KeepAlive, SendEvent, or Teardown request are all possible.

If the value of the State variable is PLAYING, then a SelectStream, Pause, KeepAlive, SendEvent, or Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.11 Receiving a Pause Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The Pause request MUST follow the rules as specified in section 2.2.7.7.

The State variable MUST be set to READY.

The Pause response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.7.

After sending the response, the server MUST wait for another request to be received.

A LogPlay, SelectStream, Play, KeepAlive, SendEvent, or Teardown request are all possible.

How to process a LogPlay request is specified in section 3.2.5.12.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.12 Receiving a LogPlay Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The LogPlay request MUST follow the rules as specified in section 2.2.7.7.

The server SHOULD communicate the logging information submitted by the client to the higher layer.

The LogPlay response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.7.

After sending the response, the server MUST wait for another request to be received.

If the value of the State variable is READY, then a SelectStream, Play, KeepAlive, SendEvent, or Teardown request are all possible.

If the value of the State variable is PLAYING, then a SelectStream, Pause, KeepAlive, SendEvent, or Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.13 Receiving an EndOfStream Response

The server MUST validate that the response adheres to the syntax as specified in section 2.2.7.3.

The status code in the response SHOULD be ignored. Therefore, normal processing continues even if the client indicates that the EndOfStream (section 2.2.7.3) request failed.

The server MUST then wait for another request or response to be received.

If the server has sent an Announce (section 2.2.7.1) request, it should have received a response to that request by this time. If the server has not yet received a response to the Announce (section 2.2.7.1) request, the server MUST wait to receive this response, as previously stated. The client MUST send a response to the Announce (section 2.2.7.1) request, as specified in section 3.1.5.15. How to process this response is specified in section 3.2.5.14.

If the value of the State variable is READY, then a SelectStream (section 2.2.7.10), Play (section 2.2.7.9), LogPlay (section 2.2.7.7), KeepAlive (section 2.2.7.5), SendEvent (section 2.2.7.11), or Teardown (section 2.2.7.13) request are all possible.

If the value of the State variable is PLAYING, then a LogPlay, SelectStream, Pause (section 2.2.7.8), KeepAlive, , or request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a LogPlay request is specified in section 3.2.5.12.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.14 Receiving an Announce Response

The server MUST validate that the response adheres to the syntax as specified in section 2.2.7.1.

The server MUST then wait for another request to be received.

If the value of the State variable is READY, then a SelectStream (section 2.2.7.10), Play (section 2.2.7.9), LogPlay (section 2.2.7.7), KeepAlive (section 2.2.7.5), SendEvent (section 2.2.7.11), or Teardown (section 2.2.7.13) request are all possible.

If the value of the State variable is PLAYING, then a LogPlay, SelectStream, Pause (section 2.2.7.8), KeepAlive, SendEvent, or Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a LogPlay request is specified in section 3.2.5.12.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.15 Receiving a KeepAlive Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The KeepAlive request MUST follow the rules as specified in section 2.2.7.5.

The KeepAlive response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.5.

After sending the response, the server MUST wait for another request to be received.

If the value of the State variable is READY, then a SelectStream (section 2.2.7.10), Play (section 2.2.7.9), KeepAlive (section 2.2.7.5), SendEvent (section 2.2.7.11), or Teardown (section 2.2.7.13) request are all possible.

If the value of the State variable is PLAYING, then a SelectStream, Pause (section 2.2.7.8), KeepAlive, SendEvent, or Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.16 Receiving a SendEvent Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The SendEvent request MUST follow the rules as specified in section 2.2.7.11.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not, this is an error, and the server MUST respond with an appropriate RTSP error status code, such as 404.

The SendEvent request does not require any server state. Therefore, if the Session header is missing, this MUST NOT be treated as an error.

The server SHOULD communicate the logging information submitted by the client to the higher layer.

The SendEvent response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.11.

After sending the response, the server MUST wait for another request to be received.

If the value of the State variable is READY, then a SelectStream (section 2.2.7.10), Play (section 2.2.7.9), LogPlay (section 2.2.7.7), KeepAlive (section 2.2.7.5), SendEvent (section 2.2.7.11), or Teardown (section 2.2.7.13) request are all possible.

If the value of the State variable is PLAYING, then a LogPlay, SelectStream, Pause (section 2.2.7.8), KeepAlive, SendEvent, or Teardown request are all possible. The server MUST continue transmitting RTP packets while in the PLAYING state and MUST be prepared to receive RTCP packets.

How to process a LogPlay request is specified in section 3.2.5.12.

How to process a SelectStream request is specified in section 3.2.5.6.

How to process a Play request is specified in section 3.2.5.8.

How to process a Pause request is specified in section 3.2.5.11.

How to process a KeepAlive request is specified in section 3.2.5.15.

How to process a SendEvent request is specified in section 3.2.5.16.

How to process a Teardown request is specified in section 3.2.5.17.

How to process RTCP packets is specified in section 3.2.5.10.

3.2.5.17 Receiving a Teardown Request

The server MUST first follow the steps as specified in section 3.2.5.1.

The Teardown request MUST follow the rules as specified in section 2.2.7.13.

The Teardown response MUST follow the rules as specified in sections 3.2.5.4 and 2.2.7.13.

After sending the response, the server MUST close the TCP connection to the server and delete the session state.

3.2.6 Timer Events

3.2.6.1 Idle-Timeout Timer Expires

When the Idle-Timeout timer expires, the server MUST close the TCP connection to the client if the connection is still open. After that, the server MUST delete the session state.

3.2.7 Other Local Events

3.2.7.1 Client Closes TCP Connection

If the value of the State variable in the abstract data model is READY or PLAYING and the client disconnected its TCP connection, then the State variable MUST be set to READY and the Idle-Timeout timer MUST be started. If the value of the State variable is INIT, then the session state MUST be deleted.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of RTSP Windows Media Extensions.

4.1 RTP Packet Syntax

The size of the RTP payload format header, as specified in section 2.2.1, varies from 4 to 16 bytes, depending on how the **R**, **D**, and **I** fields are set. When none of the fields are set to 1, the size of the RTP payload format header is 4 bytes. When only one of the fields is set to 1, the size of the RTP payload format header is 8 bytes. When two fields are set to 1, the size of the RTP payload format header is 12 bytes. When all three fields are set to 1, the size of the RTP payload format header is 16 bytes.

The following illustration represents an ASF data packet as it would be sent on the network. In this typical scenario, the total packet size is 1,500 bytes, which includes the following headers and payloads:

- IP header = 20 bytes
- UDP header = 8 bytes
- RTP header = 12 bytes
- RTP payload format header = 4 bytes
- ASF data packet header = 12 bytes
- ASF payload headers and compressed media data = 1,444 bytes

Therefore, the header overhead is approximately 3.73 percent. For smaller RTP packet sizes, such as 1,000 bytes, the overhead is increased to approximately 5.6 percent.

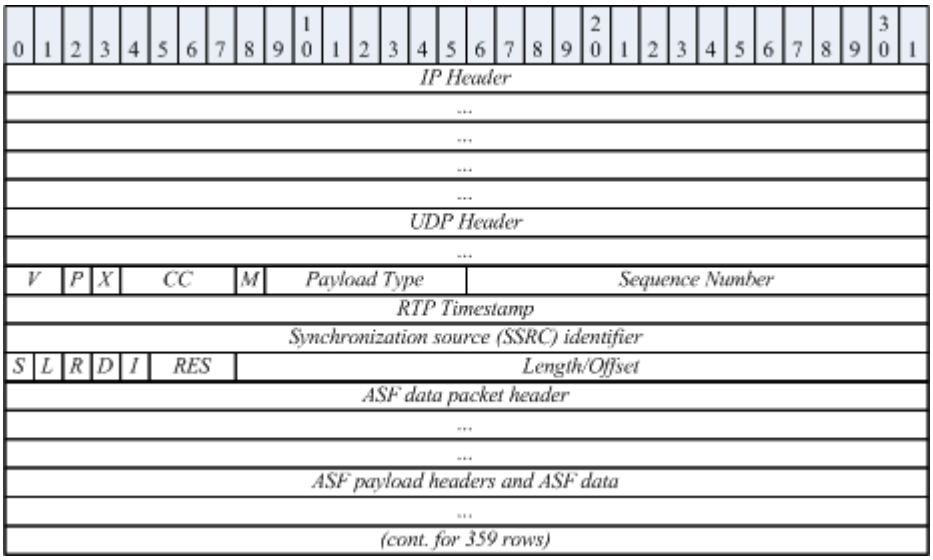


Figure 7: RTP packet diagram

4.2 Vandermonde Matrix Algorithm

The Vandermonde matrix is created by following the steps as specified in section 2.2.2.2.2.

All computations needed to perform encoding and decoding of the data are based on the finite field GF(28). The following shows the tables for log() and exp() over a GF(28).

exp(x)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	4	8	10	20	40	80	1D	3A	74	E8	CD	87	13	26
1	4C	98	2D	5A	B4	75	EA	C9	8F	3	6	C	18	30	60	C0
2	9D	27	4E	9C	25	4A	94	35	6A	D4	B5	77	EE	C1	9F	23
3	46	8C	5	A	14	28	50	A0	5D	BA	69	D2	B9	6F	DE	A1
4	5F	BE	61	C2	99	2F	5E	BC	65	CA	99	F	1E	3C	78	F0
5	FD	E7	D3	BB	6B	D6	B1	7F	FE	E1	DF	A3	5B	B6	71	E2
6	D9	AF	43	86	11	22	44	88	D	1A	34	68	D0	BD	67	CE
7	81	1F	3E	7C	F8	ED	C7	93	3B	76	EC	C5	97	33	66	CC
8	85	17	2E	5C	B8	6D	DA	A9	4F	9E	21	42	84	15	2A	54
9	A8	4D	9A	29	52	A4	55	AA	49	92	39	72	E4	D5	B7	73
A	E6	D1	BF	63	C6	91	3F	7E	FC	E5	D7	B3	7B	F6	F1	FF
B	E3	DB	AB	4B	96	31	62	C4	95	37	6E	DC	A5	57	AE	41
C	82	19	32	64	C8	8D	7	E	1C	38	70	E0	DD	A7	53	A6
D	51	A2	59	B2	79	F2	F9	EF	C3	9B	2B	56	AC	45	8A	9
E	12	24	48	90	3D	7A	F4	F5	F7	F3	FB	EB	CB	8B	B	16
F	2C	58	B0	7D	FA	E9	CF	83	1B	36	6C	D8	AD	47	8E	1

log(x)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	512	0	1	25	2	50	26	198	3	223	51	238	27	104	199	75
1	4	100	227	14	52	141	239	129	28	193	105	248	200	8	76	113
2	5	138	101	47	225	36	15	33	53	147	142	218	240	18	130	69
3	29	181	194	125	106	39	249	185	201	154	9	120	77	228	114	166
4	6	191	139	98	102	221	48	253	226	152	37	179	16	145	34	136
5	54	208	148	206	143	150	219	189	241	210	19	92	131	56	70	64
6	30	66	182	163	195	72	126	110	107	58	40	84	250	133	186	61
7	202	94	155	159	10	21	121	43	78	212	229	172	115	243	167	87
8	7	112	192	247	140	128	99	13	103	74	222	237	49	197	254	24
9	227	165	153	119	38	184	180	124	17	68	146	217	35	32	137	46
A	55	63	209	91	149	188	207	205	144	135	151	178	220	252	190	97
B	242	86	211	171	20	42	93	158	132	60	57	83	71	109	65	162
C	31	45	67	216	183	123	164	118	196	23	73	236	127	12	111	246
D	108	161	59	82	41	157	85	170	251	96	134	177	187	204	65	90
E	203	89	95	176	156	169	160	81	11	245	22	235	122	117	44	215
F	79	174	213	233	230	231	173	232	116	214	244	234	168	80	88	175

Figure 8: Galois fields table in hexadecimal and decimal (log and exp)

The following is a 10 * 6 size Vandermonde matrix created from the Galois Field tables.

1	1	1	1	1	1
1	2	4	8	10	20
1	3	5	f	11	33
1	4	10	40	1d	74
1	5	11	55	1c	6c
1	6	14	78	d	2e
1	7	15	6b	c	24
1	8	40	3a	cd	26
1	9	41	73	cc	e2
1	a	44	92	dd	1

Figure 9: Vandermonde matrix table in hexadecimal, using GF28

It is then reduced to the following identity matrix by using standard linear algebra.

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
ba	69	d3	d2	68	bb
fe	60	89	60	f7	81
56	3a	7b	93	ac	29
d8	2	cd	25	95	a6

Figure 10: Vandermonde identity matrix table in hexadecimal, using GF28

In this example, a group of 6 RTP packets labeled SourcePacket0 through SourcePacket5 are used to create 4 FEC packets labeled FECPacket0 through FECPacket3.

The reduced form of the previous identity matrix is used as the generator matrix and is subsequently used to generate 10 encoded packets. The first 6 encoded packets will be identical to SourcePacket0 through SourcePacket5, and the last 4 encoded packets will be the FEC packets.

SourcePacket0		Identity0		SourcePacket0
SourcePacket1		Identity1		SourcePacket1
SourcePacket2		Identity2		SourcePacket2
SourcePacket3		Identity3		SourcePacket3
SourcePacket4	=	Identity4	*	SourcePacket4
SourcePacket5		Identity5		SourcePacket5
FECPacket0		Identity6		
FECPacket1		Identity7		
FECPacket2		Identity8		
FECPacket3		Identity9		

Figure 11: Vandermonde-generated data equation using GF28 (definition)

The server multiplies the generator matrix, which has 10 rows and 6 columns, with a source matrix with 6 rows and 8 columns. Each row in the source matrix corresponds to one of the 6 RTP packets that will be encoded, and each column is 1 byte from the packet on the corresponding row. To reduce the size of the matrices, this example uses 8-byte packets. Each byte is expressed as a hexadecimal number in the following illustration.

The result of the matrix multiplication is another matrix with 10 rows and 6 columns, each row corresponding to an encoded RTP packet. The first 6 rows are identical to the RTP packets in the source matrix, and the last 4 rows are the FEC RTP packets.

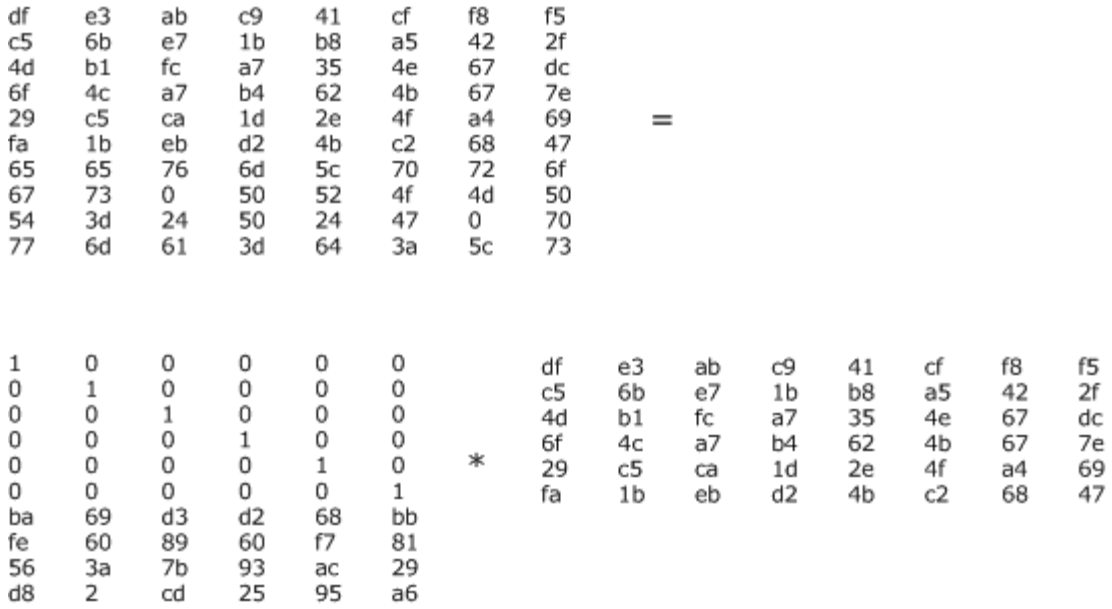


Figure 12: Vandermonde-generated data equation using GF28 (implementation)

A client that has lost some RTP packets arranges the RTP packets that it received as the result matrix, and multiplies it with the inverse of the identity matrix to obtain the source matrix.

4.3 SDP Examples

4.3.1 Retransmission Stream

Windows Media Services is capable of retransmitting lost RTP packets. Windows Media Services indicates that it supports retransmission of RTP packets by including a retransmission stream in the Session Description Protocol (SDP) description. The SDP description specifies a payload format of "x-wms-rtx" in the **rtpmap** attribute, as shown in the following example.

```
m=application 0 RTP/AVP 96
a=rtpmap:96 x-wms-rtx/1000
a=control:rtx
a=stream:65536
```

4.4 RTSP Examples

4.4.1 SETUP Request

To recover lost RTP packets, the client selects the retransmission stream by sending the SETUP request (section 2.2.7.10.1) to the server. In the SETUP request (section 2.2.7.10.1), the client specifies the RTP port to receive retransmitted RTP packets. In the response to this request, the server specifies the RTCP port to send the generic NACK messages. For example, the following SETUP request (section 2.2.7.10.1) specifies an RTP port of 4958.

```
SETUP rtsp://server/foo/rtx RTSP/1.0
Transport: RTP/AVP/UDP;unicast;client_port=4958-4959;ssrc=b58db0de
      ;mode=PLAY
```

4.4.2 Packet-Pair Bandwidth Estimation Using UDP

The following sequence occurs between a client and server when conducting the packet-pair experiment over UDP:

1. The client sends a Describe (section 2.2.7.2) request.
2. The server sends a "200 OK:" response that includes the SDP record.
3. The client sends a SelectStream (section 2.2.7.10) request by using the Setup method to establish an RTP session for a retransmission stream.
4. The server responds with "200 OK".
5. The client sends a UdpPacketPair (section 2.2.7.14) request with a content type of application/x-rtsp-udp-packetpair (section 2.2.6.3.3).
6. The server sends a "200 OK" response with the message body empty.
7. The server transmits two or three RTP packets with packet-pair data over UDP (for more information, see sections 2.2.3.2 and 2.2.7.14).

The following illustration shows the sequence described above.

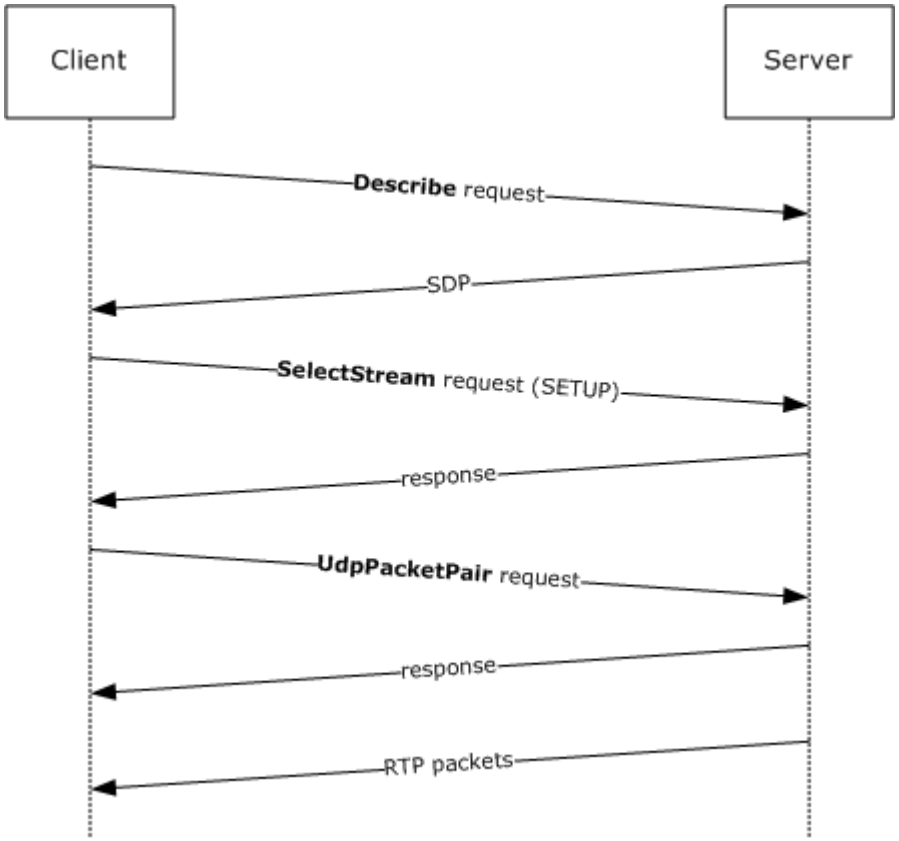


Figure 13: Packet-pair experiment sequence over UDP

The following example shows a client's packet-pair experiment request. Note that some headers extraneous to this example have been omitted for brevity.

```
DESCRIBE rtsp://wms4708/test RTSP/1.0
User-Agent: WMPlayer/10.0.0.4332
Accept: application/sdp
CSeq: 5
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.eosmsg, com.microsoft.wm.predstrm,
com.microsoft.wm.startupprofile

RTSP/1.0 200 OK
Content-Type: application/sdp
CSeq: 5
Server: WMServer/9.5.5732.6324
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.eosmsg, com.microsoft.wm.fastcache,
com.microsoft.wm.packetpairssrc, com.microsoft.wm.startupprofile

v=0
...
m=application 0 RTP/AVP 96
a=rtpmap:96 x-wms-rtx/1000
a=control:rtx
```

```
a=stream:65536

SETUP rtsp://wms4708/test/rtx RTSP/1.0
User-Agent: WMPlayer/10.0.0.4332
CSeq: 6
Transport: RTP/AVP/UDP;unicast;client_port=3236-
3237;ssrc=16a4ffff;mode=PLAY

RTSP/1.0 200 OK
Transport: RTP/AVP/UDP;unicast;server_port=5004-5005;client_port=3236-
3237;ssrc=2f73682b;mode=PLAY
CSeq: 6
Session: 8134007897615700187;timeout=60
Server: WMServer/9.5.5732.6324
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.eosmsg, com.microsoft.wm.fastcache,
com.microsoft.wm.packetpairssrc, com.microsoft.wm.startupprofile

SET_PARAMETER rtsp://wms4708/test RTSP/1.0
User-Agent: WMPlayer/10.0.0.4332
Accept-Language: en-US, *;q=0.1
Session: 8134007897615700187
Content-Type: application/x-rtsp-udp-packetpair;charset=UTF-8
CSeq: 7

type: high-entropy-packetpair

RTSP/1.0 200 OK
Content-Type: application/x-rtsp-udp-packetpair;charset=UTF-8
CSeq: 7
Session: 8134007897615700187;timeout=60
Server: WMServer/9.5.5732.6324

type: high-entropy-packetpair
```

4.4.3 Packet-Pair Bandwidth Estimation Using TCP

The following sequence occurs between a client and server when conducting the packet-pair experiment over TCP:

1. The client sends a Describe (section 2.2.7.2) request.
2. The server sends a "200 OK" response that includes the SDP record.
3. The client sends a TcpPacketPair (section 2.2.7.12) request with a content type of application/x-rtsp-packetpair (section 2.2.6.3.2).
4. The server sends a "200 OK" response with three $P packets in the message body (for more information, see section 2.2.7.12).

The following illustration shows the sequence described above.

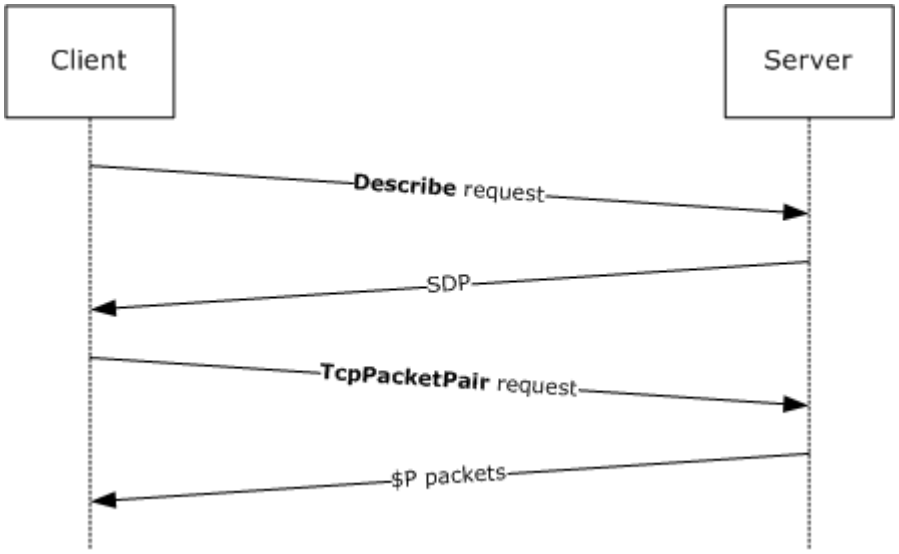


Figure 14: Packet-pair experiment sequence over TCP

4.4.4 Predictive Stream Selection and SelectStream

The server may switch from streaming one piece of content to another. This may occur, for example, when a server-side playlist is used. When the server transitions from one entry to the next, it may attempt to predict what streams the client would select based on the client's previous stream selections.

The server announces the entry change with an EndOfStream (section 2.2.7.3) request sent to the client indicating end-of-stream (and playlist entry) followed by an Announce (section 2.2.7.1) request with SDP in the message body. It then sends the streams that the server predicts the client would select. Note that the client can send a SelectStream (section 2.2.7.10) request by using the SET_PARAMETER method at any time to reduce the transmission rate while streaming. This action does not interrupt the streaming.

If the client accepts the streams, it must include the respective URLs for those streams in the X-RTP-Info (section 2.2.6.24) header of the corresponding Play (section 2.2.7.9) request. If the client rejects a stream, it must send a SelectStream (section 2.2.7.10) request by using the TEARDOWN method before sending any subsequent Play (section 2.2.7.9) requests. In this case, the client should not include the URL of the rejected stream in the X-RTP-Info (section 2.2.6.24) header of the Play (section 2.2.7.9) request.

4.4.4.1 SelectStream Using SET_PARAMETER

A client can request the server to switch streams. The stream switch requests are performed by submitting a SelectStream (section 2.2.7.10) request to the server. The SelectStream (section 2.2.7.10) request must specify the application/x-wms-streamswitch (section 2.2.6.3.9) media type:

```
Content-Type: application/x-wms-streamswitch
```

This is an example of the message body in the request:

```
SSEntry: 7 6 0 rtsp://myserver.com/myMBRcontent.wmv/stream=7
rtsp://myserver.com/myMBRcontent.wmv/stream=6
```

In this case, the client is requesting that the server stop sending stream 7, and instead start sending stream 6 in its place. The client is requesting that the server perform no thinning (that is, thinning level of 0).

The following example illustrates the conversation between the server and client during a stream switch request from stream 7 to 6 with no thinning.

Client to server:

```
SET PARAMETER rtsp://myserver.com/myMBRcontent.wmv/stream=7
RTSP/1.0
Content-Length: 112
User-Agent: WMPlayer/9.0.0.2899 guid/3300AD50-2C39-46C0-AE0A-FF4DD9402916
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest, Basic
Accept-Language: en-us, *;q=0.1
Session: 14828520034371861638
Content-Type: application/x-wms-streamswitch
CSeq: 8
X-Playlist-Gen-Id: 1547
If-Match: "{0279ED14-1413-E1EE-14DF-E327B7519C52}"

SSEntry: 7 6 0 rtsp://myserver.com/myMBRcontent.wmv/stream=7
rtsp://myserver.com/myMBRcontent.wmv/stream=6
```

Server to client:

```
RTSP/1.0 200 OK
Date: Wed, 09 Oct 2002 19:26:03 GMT
CSeq: 8
Session: 14828520034371861638;timeout=60
Server: WMServer/9.0.0.3239
.
.
... continue receiving content...
```

4.4.4.2 SelectStream Using TEARDOWN

When the SelectStream (section 2.2.7.10) request uses the TEARDOWN and SETUP methods, the method can also contain an SSEntry message body (indicated by the application/x-wms-streamswitch (section 2.2.6.3.9) media type on the Content-Type (section 2.2.6.3) header). The application must send the SSEntry message body after predictive stream selection if the stream URLs for an old entry are different from the stream URLs for a new entry.

For example, an application sets up the following stream:

```
SETUP rtsp://host/abc/audio
```

```
Server
```

It then sends an Announce (section 2.2.7.1) request where the X-RTP-Info (section 2.2.6.24) header indicates that the stream "rtsp://host/abc/audio=5" has been selected.

Later, if the application wants to send a SelectStream (section 2.2.7.10) request to de-select the /audio=5 stream, it sends the following:

```
TEARDOWN rtsp://host/abc/audio
Content-Type: application/x-wms-streamswitch

5 5 2 rtsp://host/abc/audio=5 rtsp://host/abc/audio=5
```

Note The value 5 is the ASF stream ID for the audio=5 stream.

4.4.4.3 SelectStream After Predictive Stream Selection

Assume for the first entry that the client selects the following streams: /audio, /video. For the second entry, the server predicts streams /audio=2, /video=5. If the client wants to deselect the stream /audio=2, or switch from /audio=2 to /audio=3, the client must choose one of the originally selected streams (/audio or /video), and then send a TEARDOWN (section 2.2.7.13) or SET_PARAMETER (section 2.2.7.10.3) request by using the original stream URL with the SSEntry message body referencing the new stream. Both the original and the new stream must have the same transport (UDP or TCP) and use the same RTP payload format.

Looking at a simple case, if all streams (/audio, /video, /audio=2, /video=5) use the same transport and RTP payload format, the client is free to select any stream (/audio or /video).

For example, the client can use the following request to deselect the /audio=2 stream:

```
TEARDOWN rtsp://host/abc/video
Content-Type: application/x-wms-streamswitch

2 2 2 rtsp://host/abc/audio=2 rtsp://host/abc/audio=2
```

4.4.4.4 Client Requests FEC Stream from Server

From the SDP description sequence, the client can select audio or video streams, and for each of these it can select associated FEC streams.

The following example illustrates a request/response exchange in which the client requests FEC using a packet span of 24 and 4 FEC packets per span (FecSpan=4). The third parameter, the **FecBurstMargin** field, is used to buffer a set of packet spans and associated FEC packets in the form of a Vandermonde matrix (section 2.2.2.2) (a calculation commonly applied to error correction problems).

The example also shows another FEC parameter, known as the FEC burst margin (FecBurstMargin=6), used to recover from burst losses.

Client to server:

```
SETUP rtsp://myserver.com/mycontent.wmv/stream=5/fec98 RTSP/1.0
Transport:
RTP/AVP/UDP;unicast;client port=2408;ssrc=6dded651;mode=PLAY;FecSpan=4;
FecPerSpan=1;FecBurstMargin=6, RTP/AVP/TCP;unicast;interleaved=0-1;
ssrc=6dded651;mode=PLAY
If-Match: "{83A04BD0-FD30-1984-4994-0A22CA116ED3}"
Date: Fri, 23 Mar 2001 04:28:14 GMT
CSeq: 8
Session: 1077055954
User-Agent: WMPlayer/9.0.0.197 guid/CB131790-CC16-4CCE-A234-6D29BEE21FCE
Accept-Language: en-us, *;q=0.1
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: NTLM, Digest
```

Server to client:

```
RTSP/1.0 200 OK
Transport:
RTP/AVP/UDP;unicast;source=157.56.216.159;server_port=2410;client_port=2408;
ssrc=3874dd27;mode=PLAY;FecSpan=4;FecPerSpan=1;FecBurstMargin=6
Date: Fri, 23 Mar 2001 04:28:14 GMT
CSeq: 8
Timestamp: 1 0.031
Session:1077055954;timeout=60
Server: WMServer/ 9.0.0.197
Cache-Control: must-revalidate, proxy-revalidate
```

4.4.5 Server-Side Playlist Entry Switching

Content is identified as sourcing from a server-side playlist in the SDP included in the response to the Describe (section 2.2.7.2) request.

How a server-side playlist is represented on the server is implementation-specific. However, the following example shows how a server-side playlist may look using an XML-based syntax.

```
<?wsx version="1.0"?>
<smil>
<media role="Advertisement" noSkip="TRUE" src="Ad_1.wmv
<media src="preview.wmv"/>
<media src="movie.wmv"/>
<media role="Advertisement" noSkip="TRUE" src="Ad_2.wmv"/>
</smil>
```

In the following example, the server streams an advertisement that the client is unable to skip past. Immediately after the ad is a movie preview followed by a movie. At the end, another advertisement plays, and the client is again unable to skip past it.

For the client to successfully stream this server-side playlist, it must use the following sequence.

1. When the server transitions from one server-side playlist entry to the next, it sends an EndOfStream (section 2.2.7.3) request to the client. The message body of the request contains the following:

```
EOF: True
End-Of-Playlist-Entry: True
```

2. The client sends a LogPlay (section 2.2.7.7) request to submit the logging statistics to the server.
3. The server sends an Announce (section 2.2.7.1) request to the client providing the SDP record for the next playlist entry.
4. The client sends a SelectStream (section 2.2.7.10) request by using the TEARDOWN method to the server to de-select any old streams.
5. The client sends a SelectStream (section 2.2.7.10) request by using the SETUP method to the server to select new streams.
6. The client repeats steps 4 and 5 until all old streams are de-selected, and the new streams have been selected.
7. The client submits a Play (section 2.2.7.9) request to start receiving the RTP packets.

The following illustration shows the previously described sequence.

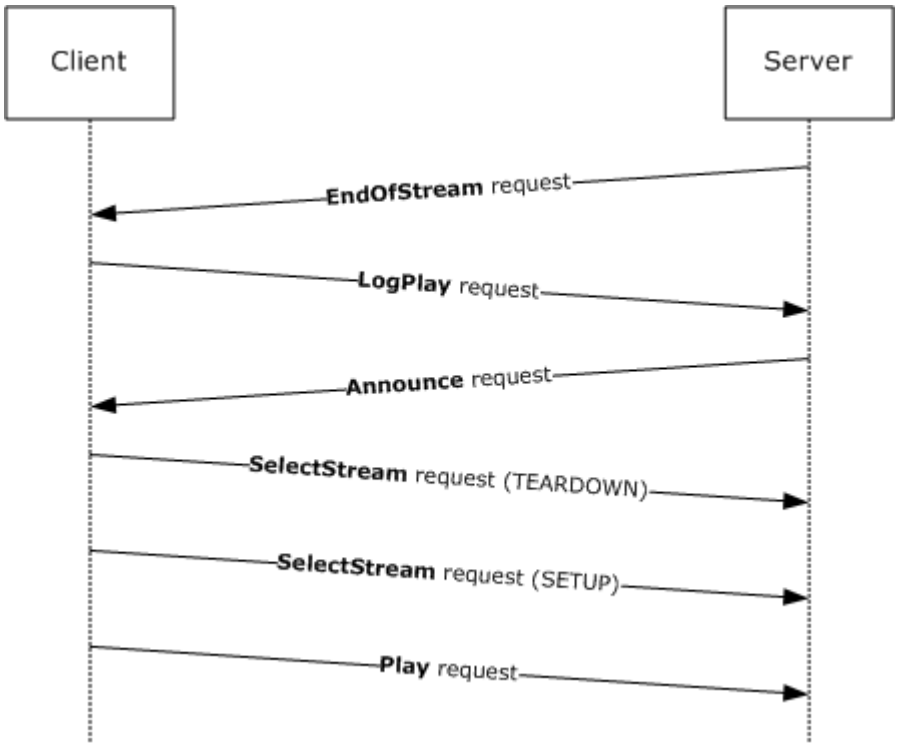


Figure 15: Streaming sequence between client and server for a server-side playlist

4.4.6 Stream Playback with Authentication

The following sequence occurs between a client and server if the server requires the client to be authenticated:

1. The client sends a Describe (section 2.2.7.2) request.
2. The server responds with a "401 Unauthorized" response. The server and client exchange access authentication messages, as specified in [RFC1945] section 11.

   Note The request/response exchanges required for authentication are defined by the selected authentication scheme.
3. If authentication has succeeded, the server sends a "200 OK:" response that includes the SDP record.
4. The client sends a SelectStream (section 2.2.7.10) request by using the Setup method for each selected stream.
5. The server responds with "200 OK".
6. The client submits a Play (section 2.2.7.9) request to start receiving the RTP packets.
7. The server responds with "200 OK" and begins sending the RTP packets.
8. The client submits connect-time statistics using a LogConnect request.

The following illustration shows the previously described sequence.

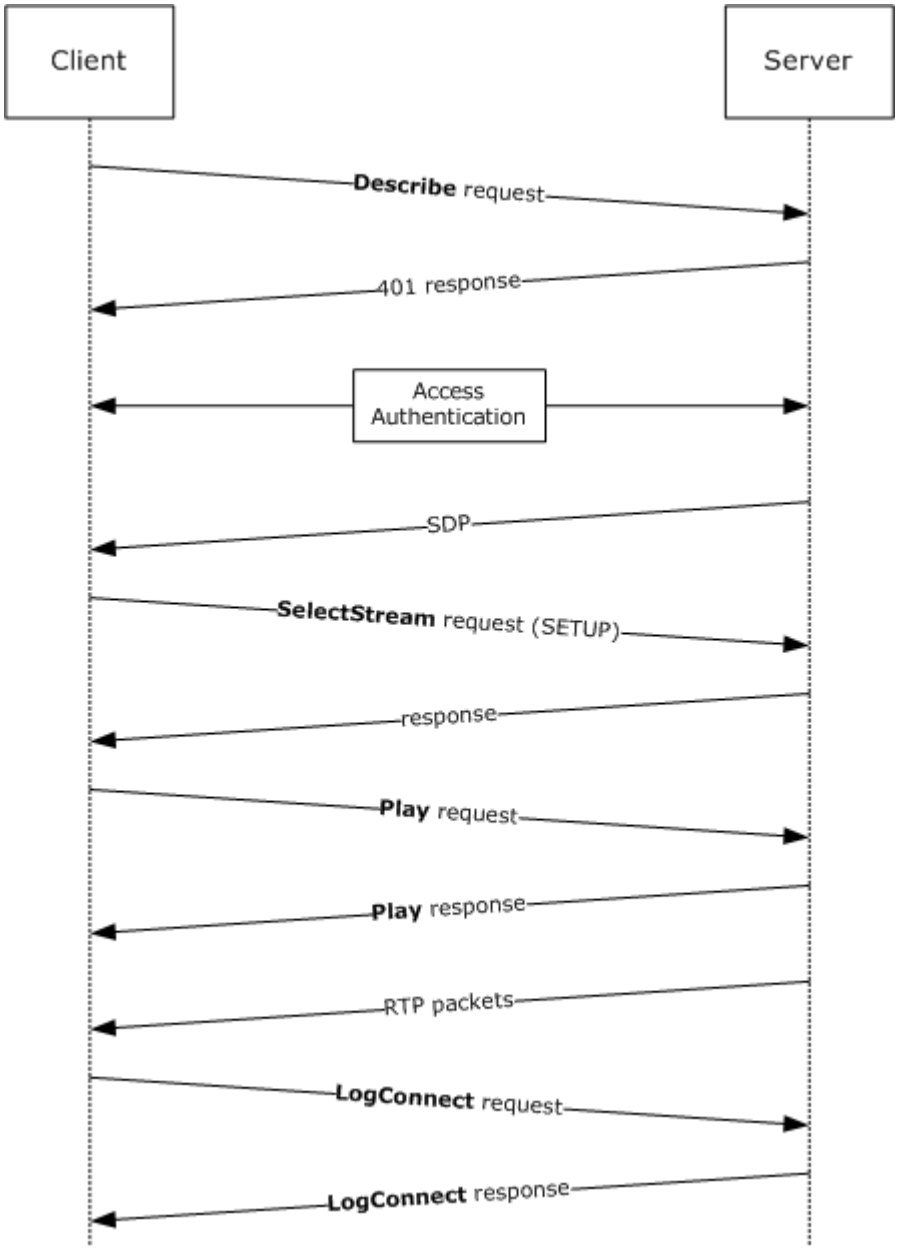


Figure 16: RTSP client-server stream playback sequence with authentication

4.4.7 Streaming, Pausing, Fast-Forwarding, and Stopping Playback

The following sequence occurs between a client and server while performing other transport operations in between streaming and then stopping a file:

- The client sends a Describe (section 2.2.7.2) request.
- The server responds with a "200 OK:" response that includes the SDP record.

- The client sends a SelectStream (section 2.2.7.10) request using the Setup method for each selected stream.
- The server responds with "200 OK".
- The client submits a Play (section 2.2.7.9) request to start receiving the RTP packets.
- The server responds with "200 OK" and begins sending the RTP packets.
- The client submits a Play (section 2.2.7.9) request using the Scale header (as defined in [RFC2326] section 12.34) to request playback of the content at a rate other than normal playback speed.
- The server responds with "200 OK" and continues sending RTP packets.
- The client submits a Pause (section 2.2.7.8) request to suspend receiving the RTP packets.
- The server responds with "200 OK".
- The client submits play statistics using a LogPlay (section 2.2.7.7) request.
- The server responds with "200 OK".
- The client submits a Teardown (section 2.2.7.13) request to stop receiving RTP packets and deselects the streams.
- The server responds with "200 OK".

The following illustration shows the previously described sequence.

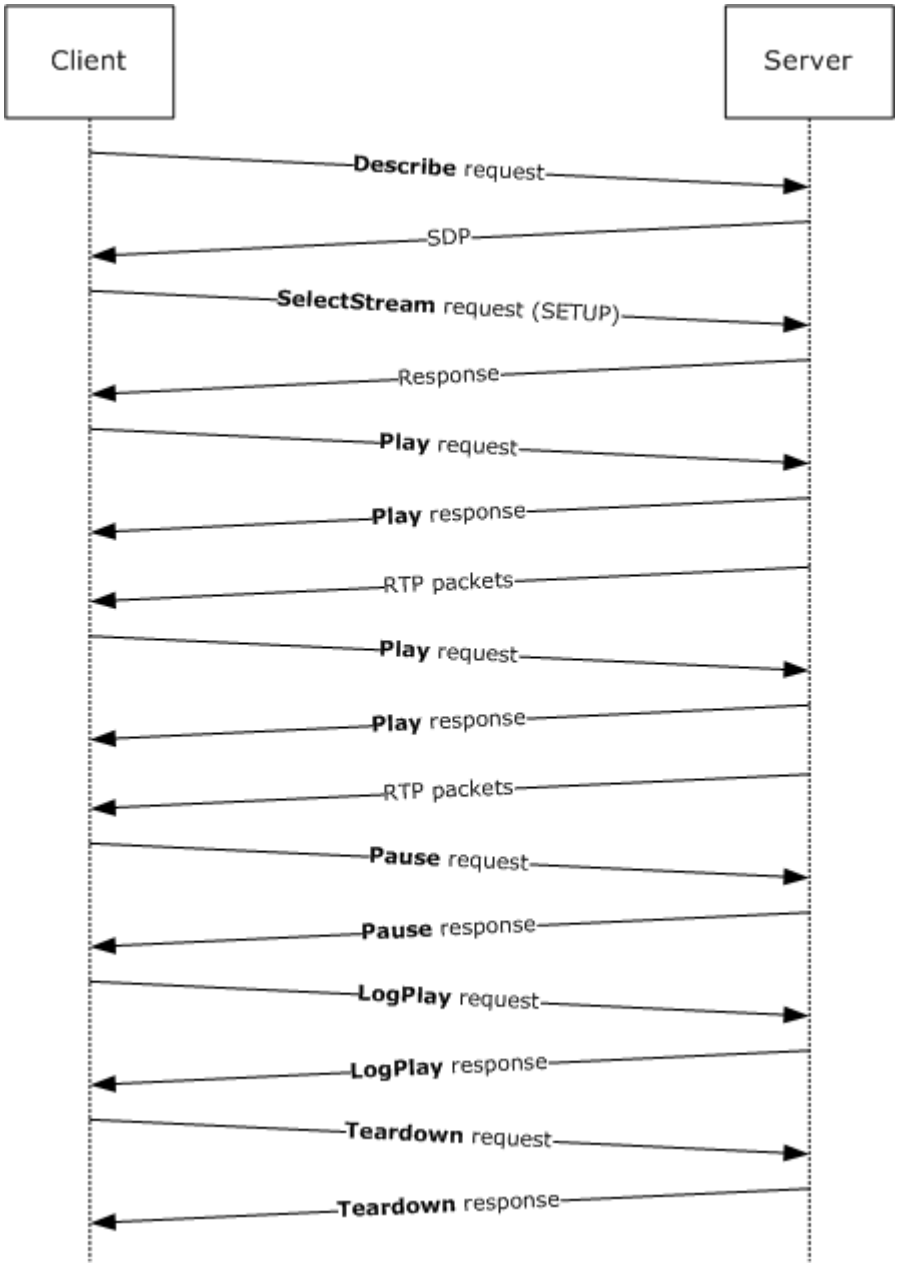


Figure 17: RTSP client-server stream playback sequence with transport operations

4.5 Logging and RTSP

4.5.1 Submitting Connect-Time Statistics

Submitting the connect-time statistics log is done by using the SET_PARAMETER (section 2.2.7.10.3) request method as shown in the following example.

Client to server (note the absence of content in the <Summary> </Summary> tags):

```
SET_PARAMETER rtsp://myserver.com/mycontent.wmv RTSP/1.0
Content-Length: 218
User-Agent: WMPlayer/9.0.0.2833 guid/3300AD50-2C39-46C0-AE0A-C4D98694D7B4
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-us, *;q=0.1
Session: 2828013918854793989
Content-Type: application/x-wms-Logconnectstats;charset=UTF-8
CSeq: 6

<XML><Summary></Summary><c-dns>-</c-dns><c-ip>0.0.0.0</c-ip>
<c-os>Windows XP</c-os><c-osversion>5.1.0.2600</c-osversion>
<date>2002-07-30</date><time>15:42:30</time><c-cpu>Pentium</c-cpu>
<transport>TCP</transport></XML>
```

Server to client:

```
RTSP/1.0 200 OK
Date: Tue, 30 Jul 2002 15:47:10 GMT
CSeq: 6
Session: 2828013918854793989;timeout=60
Server: WMServer/9.0.0.3101
```

4.5.2 Submitting a Play Log

Client to server (note the <Summary></Summary> tags in this example):

```
SET PARAMETER rtsp://myserver.com/mycontent.wmv RTSP/1.0
Content-Length: 2067
User-Agent: WMPlayer/9.0.0.2683 guid/3300AD50-2C39-46C0-AE0A-C4D98694D7B4
Accept-Charset: UTF-8, *;q=0.1
X-Accept-Authentication: Negotiate, NTLM, Digest
Accept-Language: en-us, *;q=0.1
Session: 2828013918854793989
Content-Type: application/x-wms-Logplaystats;charset=UTF-8
CSeq: 8
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.eosmsg, com.microsoft.wm.predstrm

<XML><Summary>0.0.0.0 2002-04-22 23:31:24 -
rtsp://myserver.com/mycontent.wmv 0 18 1 200 {3300AD50-2C39-46c0-
AE0A-C4D98694D7B4} 9.0.0.2683 en-US
WMFSDK/9.0.0.2683_WMPlayer/9.0.0.2683 - wmplayer.exe 9.0.0.2683
Windows_.Net_Server 5.1.0.3604 Pentium 18 501880 144932 rtsp UPD
Windows Media Audio V2 Microsoft MPEG-4 Video Codec V3 - - 136436 -
156 0 0 0 0 0 0 1 0 100 - - - - rtsp://myserver.com/mycontent.wmv
mycontent.wmv -</Summary><c-ip>0.0.0.0</c-ip><date>2002-04-22</date>
22</date><time:23:31:24</time><c-dns>-</c-dns>
.
.
.
</XML
```

Server to client:

```
RTSP/1.0 200 OK
Date: Mon, 22 Apr 2002 23:31:33 GMT
CSeq: 8
Session: 2828013918854793989;timeout=60
Server: WMServer/9.0.0.3067
```

4.6 RTSP Proxy Server Interaction

A server that is configured to operate as a **proxy** server provides the service of routing client requests to one or more origin servers that publish the streaming media content. In this case, the proxy server behaves as a client to the origin server. A proxy server might support caching of content. When the client requests content from a caching proxy server, it can either transmit the content from its local cache or obtain the content from the origin server and then transmit it to the requesting client.

A Cache-Control (section 2.2.6.2) header contains directives about the content that indicates to the proxy server how it must handle the content. For example, the x-wms-stream-type directive is used to determine whether the requested content is broadcast or on-demand. The header may include one or more directives and can be passed from the origin server to the client through the intermediate proxy server. When the proxy server is caching content, the Cache-Control header MUST be saved. When streaming this content to the requesting client, the proxy server MUST add the previously saved Cache-Control header to the relevant response messages. For information on the Cache-Control header and the directives, see section 2.2.6.2.

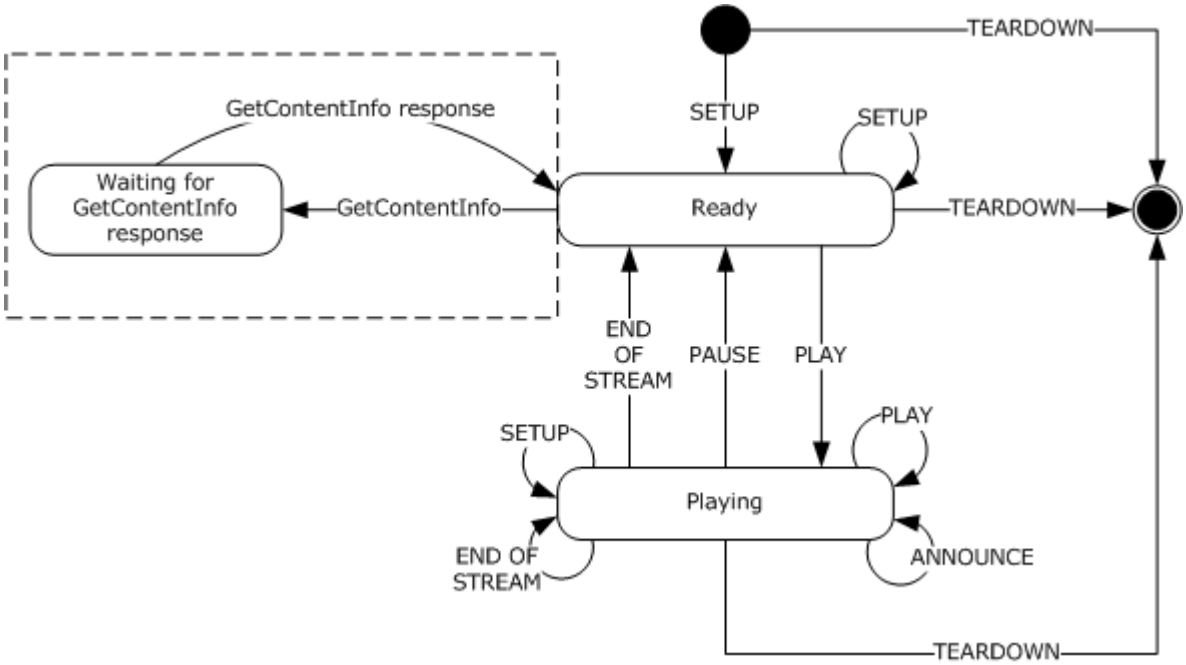


Figure 18: RTSP state diagram with caching proxy server (client perspective)

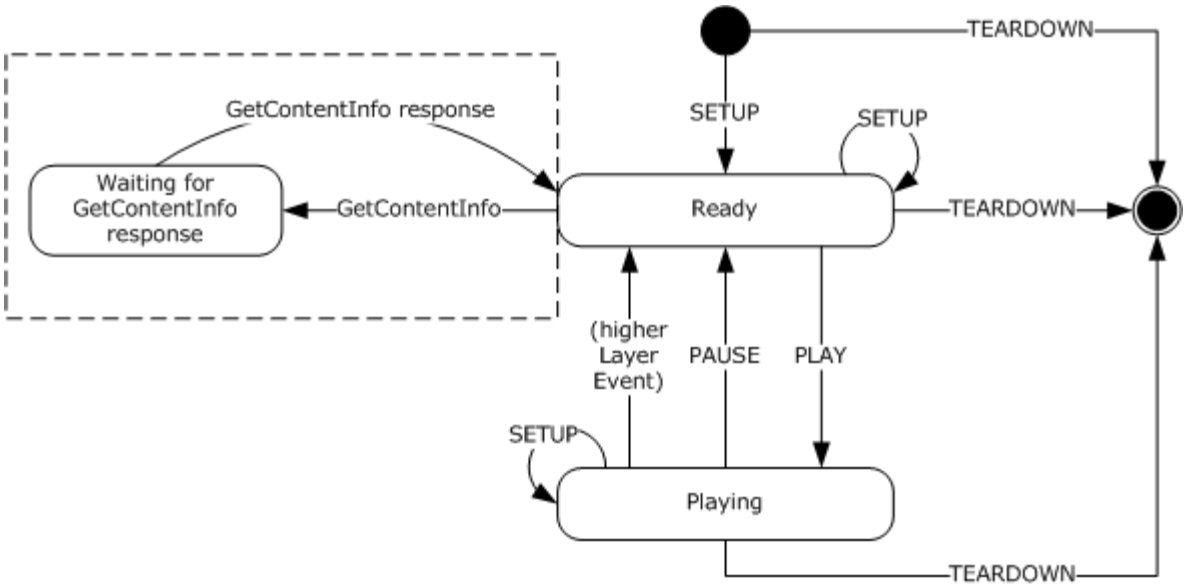


Figure 19: RTSP state diagram with caching proxy server (server perspective)

A proxy server, whether acting as a server or as a client, is largely identical in state to an origin server (as specified in illustrations in sections 3.1 and 3.2.) That is, when streaming to a client, the proxy server is acting as a regular (origin) server. When acting as a logical client, the proxy server is forwarding requests to the origin server. As indicated in Figure 20, much of the decision matrix previously described occurs as a result of the Setup request that causes the initial transition into the "Ready" state. One additional state—the Waiting for GetContentInfo response state—becomes available with the introduction of a caching proxy server. This state, which is indicated by the dotted boxes in Figure 18 and Figure 19, is applicable only if the caching proxy server is acting as a client to the origin server and only if the content on the cache has expired. The caching proxy server remains in the "Waiting for GetContentInfo response" state until it receives the GetContentInfo response from the origin server. The response to the GetContentInfo request determines whether the session is streamed from the cache or from the origin server. In either case, both the origin server and the proxy server transition back to the "Ready" state.

When a client requests on-demand content from a caching proxy server, the proxy server should first check if the content exists locally and if the content is valid. If both conditions are true, then the proxy server may transmit the content from its local cache to the client. If the content has expired, the proxy server should establish a connection to the origin server to determine if the cached copy of the content is still valid. If the proxy server is able to determine that the cached content is still valid, then the proxy server is allowed to transmit the content to the client. If the cached copy of the content is invalid and caching of the content is allowed, then the proxy server might replace its cached copy by downloading the content from the origin server into the cache. The proxy server would then be able to transmit the content to the requesting client.

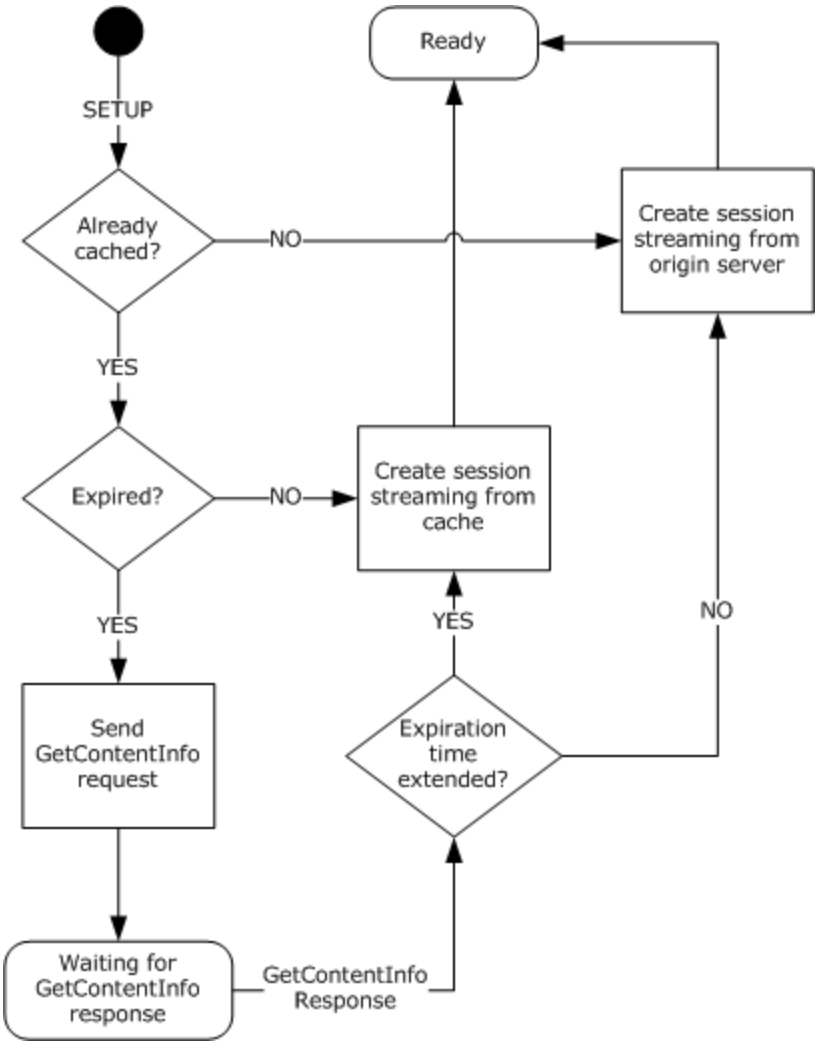


Figure 20: Caching proxy server states

4.6.1 Sequencing for Playlist Content Delivery

The following sequence occurs when a client is requesting playlist content. A proxy server handles requests and content delivery to and from the client, connecting with the playlist origin server if needed.

1. The client sends a Describe (section 2.2.7.2) request.
2. The proxy server sends a "200 OK" response that includes the SDP record and the Cache-Control (section 2.2.6.2) header.
3. The client sends a SelectStream (section 2.2.7.10) request using the Setup method for each selected stream.
4. The proxy server checks whether the requested streams are in its local cache. If the content is not in the cache, the proxy server opens a connection with the origin server. These steps are described in section 4.6.3.

5. The proxy server checks whether the content in the local cache is valid. If the content is not valid, the proxy server revalidates the content. If the content is still not valid, then the proxy server opens a connection with the origin server. These steps are described in section 4.6.3.

6. The proxy server sends a "200 OK" response to the client that includes the Cache-Control (section 2.2.6.2) header.

7. The client submits a Play request to start receiving the RTP packets.

8. The proxy server responds with "200 OK" and begins sending the RTP packets for the requested streams from its local cache to the client.

9. The proxy server communicates any subscribed-to events to the origin server.

10. The client submits play statistics using a LogPlay (section 2.2.7.7) request.

11. The server responds with "200 OK".

The following illustration shows the sequencing that occurs when the client requests a file from a media server that is configured as a proxy server.

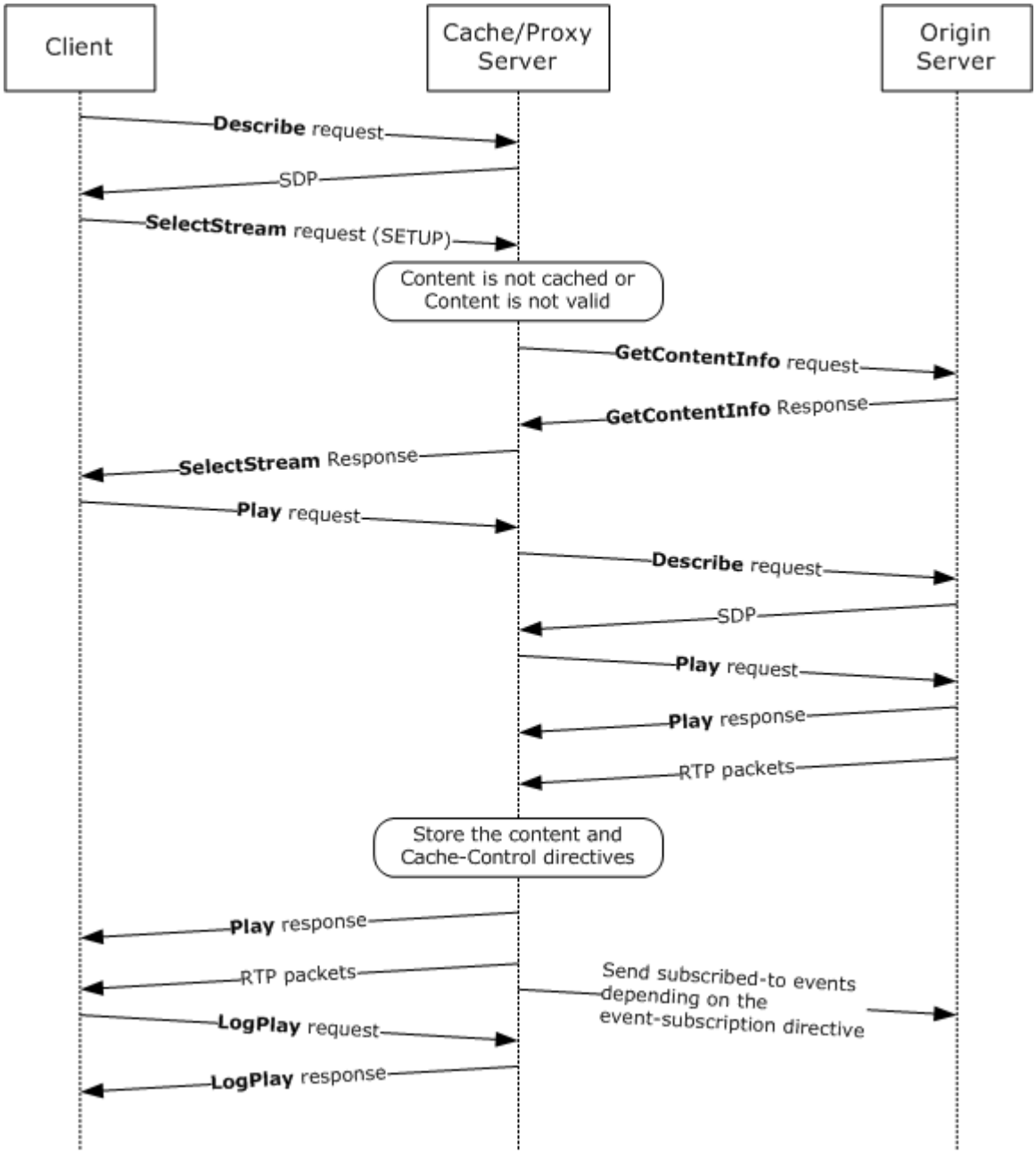


Figure 21: Playlist content delivery

4.6.2 Sequencing for Broadcast Content Delivery

The following sequence occurs when a client is requesting broadcast content. A proxy server handles requests and content delivery between the client and the origin server that generates the broadcast.

1. The client sends a Describe (section 2.2.7.2) request.

2. The proxy server sends a "200 OK" response that includes the SDP record and the Cache-Control header.

3. The client sends a SelectStream (section 2.2.7.10) request using the Setup method for each selected stream.

4. The proxy server checks whether the stream can be split, as determined by the x-wms-proxy-split directive on the Cache-Control (section 2.2.6.2) header. If the content cannot be split, then the proxy server opens a connection with the origin server. These steps are described in section 4.6.3.

5. The proxy server checks whether it is currently receiving the content. If the content is not being received, then the proxy opens a connection with the origin server. These steps are described in section 4.6.3.

6. The proxy server checks whether the content is valid. If the content is not valid, the proxy server revalidates the content.

   If the content is still not valid after revalidation, then the proxy server opens a connection with the origin server. These steps are described in section 4.6.3.

7. The proxy server splits the content and sends the RTP packets that it receives from the origin server to the client.

8. The proxy server communicates any subscribed-to events to the origin server.

9. The client submits play statistics using a LogPlay (section 2.2.7.7) request.

10. The server responds with "200 OK".

The following illustration shows the sequencing that occurs when the client requests broadcast content from a media server that is configured as a proxy server.

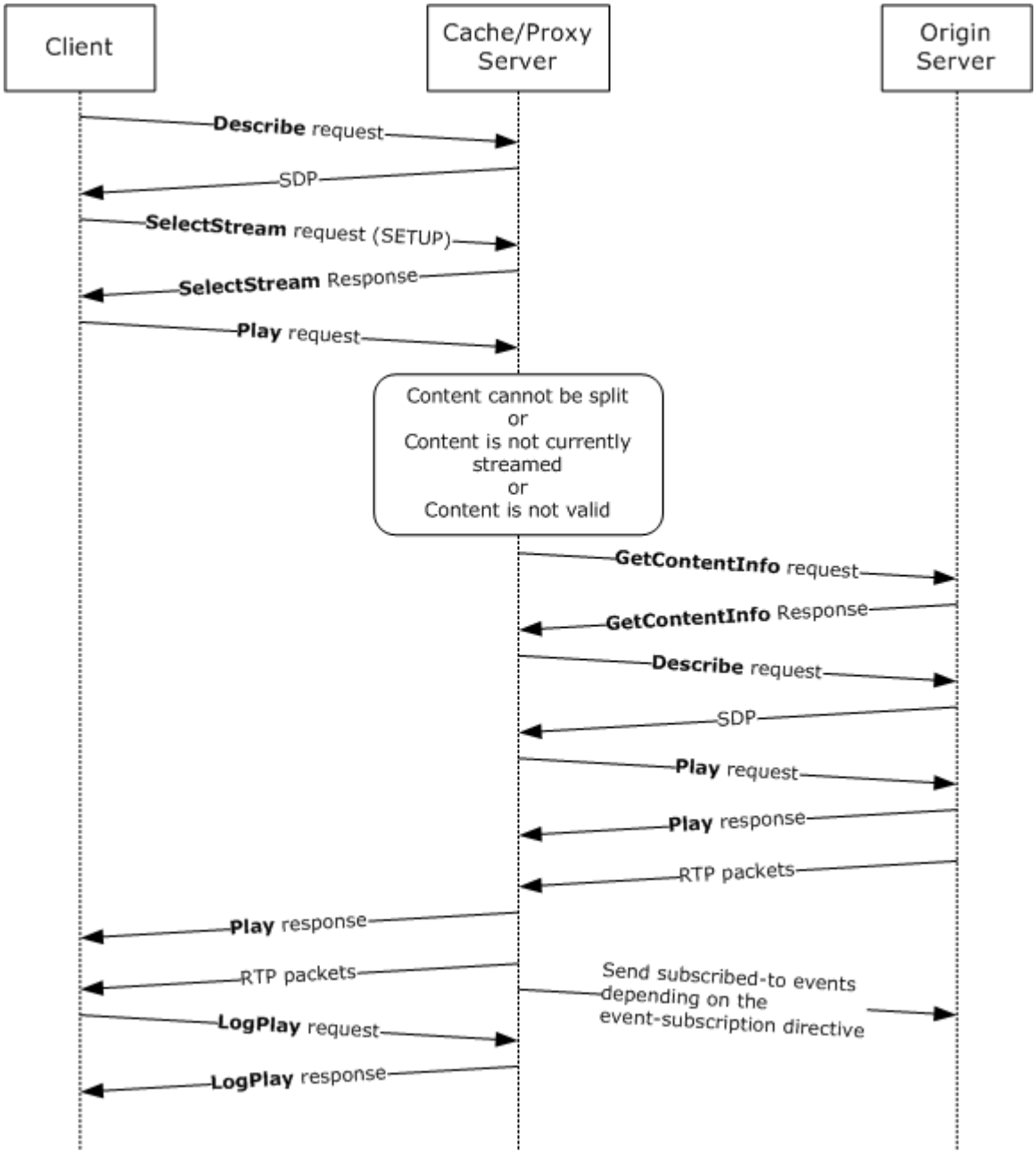


Figure 22: Broadcast content delivery

4.6.3 Proxy Server and Origin Server Communication

The following sequence occurs when a proxy server is requesting content from an origin server.

1. The proxy server requests content information from the origin server by sending a GetContentInfo (section 2.2.7.4) request.

2. The origin server responds with the Cache-Control (section 2.2.6.2) header for the content.

3. Optional. Additional handshaking can occur depending on the implementation of the origin server.

4. The proxy server determines whether to cache the content that the origin server is about to stream. If the content is to be cached, the proxy server stores the Cache-Control header information and the content in the local cache.

5. If the content is not fully cached in the proxy server's local cache, or if the content will not be cached, the proxy server sends a Describe (section 2.2.7.2) request followed by a Play (section 2.2.7.9) request to the origin server.

6. The proxy server sends the RTP packets for the requested content to the client. Depending on whether the content is already fully cached, the content source is either the proxy server's local cache or the origin server.

7. The proxy server communicates any subscribed-to events to the origin server.

5 Security

The following sections specify security considerations for implementers of RTSP Windows Media Extensions.

5.1 Security Considerations for Implementers

RTSP Windows Media Extensions are at risk of an attack in which the attacker spoofs RTCP packets containing generic NACK messages, causing the server to flood the client unnecessarily with retransmitted RTP packets. To mitigate against the attack, the server should choose the value of the *ssrc* parameter in the Transport (section 2.2.6.8) header in such a way that it is difficult for an attacker to predict its value. The server can also impose a limit on how many RTP packets per second that it will retransmit to a client.

5.2 Index of Security Parameters

The only security parameter, HTTP access authentication, is found in section 2.1.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista
- Windows Server 2003
- Windows XP SP2
- Windows XP SP1

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1: Windows Media Format 9 Series Software Development Kit (SDK), Windows Media Format 9.5 SDK, Windows Vista, and Windows Media Services on Windows Server 2003 support only RTSP using TCP.

<2> Section 2.1: Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista support NTLM, Digest (as specified in [RFC2617]), and Negotiate (as specified in [RFC4559]) authentication. Basic authentication is supported only when challenged by a proxy server. Windows Media Services on Windows Server 2003 supports NTLM, Digest, and Negotiate authentication. Authentication protocols on the server are disabled by default and may be selectively enabled by the server administrator.

<3> Section 2.2.5.5: For compatibility with Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Media Services on Windows Server 2003, and for compatibility with all of the RTSP implementations on Windows Vista, the URL specified on the **control** attribute for this stream MUST be "rtx".

<4> Section 2.2.6.2.10: The Windows Media Format 9 Series SDK, the Windows Media Format 9.5 SDK, and Windows Vista support only the remote-log event.

<5> Section 2.2.6.2.11: This directive is only supported by Windows Media Services on Windows Server 2003.

<6> Section 2.2.6.2.12: This directive is supported only by Windows Media Services on Windows Server 2003.

<7> Section 2.2.6.7.1: Windows Media Services on Windows Server 2003 and Windows Server 2003 SP1 ignores this token and always sends the EndOfStream request.

<8> Section 2.2.6.7.8: The X-StartupProfile header is supported by Windows Media Format 9.5 SDK, Windows Vista, and by the server implementation in Windows Media Services on Windows Server 2003 SP1.

<9> Section 2.2.6.25: This header is supported by the Windows Media Format 9.5 SDK, Windows Vista, and Windows Media Services on Windows Server 2003 SP1.

<10> Section 2.2.7.14: The *variable-size* parameter is added only by Windows Vista.

<11> Section 3.1: The Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and Windows Media Services on Windows Server 2003 do not support transitions in or out of the RECORD state.

<12> Section 3.1.5.1: The client-guid syntax element is not sent by Windows Media Format 9.5 SDK. A patch is available from Microsoft to correct this problem.

<13> Section 3.1.5.1: Whether the same GUID is used for all sessions or changed between different sessions is determined by how the user has configured Windows Media Player and/or the Windows Media Format SDK.

<14> Section 3.1.5.1: The com.microsoft.wm.startupprofile token is only specified by Windows Media Format 9.5 SDK and Windows Vista.

<15> Section 3.1.5.1: Only Windows Media Services on Windows Server 2003 supports sending the X-Proxy-Client-Agent and X-Proxy-Client-Verb headers.

<16> Section 3.1.5.4: The Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Media Services on Windows Server 2003, and Windows Vista support NTLM, Digest (as specified in [RFC2617]), and Negotiate (as specified in [RFC4559]) authentication. Basic authentication (as specified in [RFC2617]) is only supported when challenged by a proxy server.

<17> Section 3.2: Windows Media Services on Windows Server 2003 does not support transitions in or out of the RECORD state.

<18> Section 3.2.5.4: The com.microsoft.wm.startupprofile token is specified only by Windows Media Services on Windows Server 2003 SP1.

<19> Section 3.2.5.7: The *variable-size* parameter is not understood by any currently available versions of Windows Media Services.

7 Index

S

T

U

V

W

X