RTMP Commands Messages
draft-rtmpcommandmessages-01.txt

Abstract

This document describes the different types of messages and commands that are exchanged between the server and the client to communicate with each other.

Table of Contents

1. Introduction

The different types of messages that are exchanged between the server and the client include audio messages for sending the audio data, video messages for sending video data, data messages for sending any user data, shared object messages, and command messages. Shared objects messages provide a general purpose way to manage distributed data among multiple clients and a server.
Command messages carry the AMF encoded commands between the client and the server. A client or a server can request Remote Procedure Calls (RPC) over streams that are communicated using the command messages to the peer.

2. Definitions

Message stream:
A logical channel of communication in which messages flow.

Message stream ID:
Each message has an ID associated with it to identify the message stream to which it belongs.

Remote Procedure Calls (RPC)
A request that allows a client or a server to call a subroutine or procedure at the peer end.

Metadata
A description about the data. The metadata of a movie includes the movie title, duration, date of creation, and so on.

Application instance
The instance of the application at the server with which the clients connect by sending the connect request.

Action Message Format (AMF)
A compact binary format that is used to serialize ActionScript object graphs. Formats Specifications:
AMF0(http://opensource.adobe.com/wiki/download/attachments/1114283/amf0_spec_121207.pdf?version=1) and
AMF3(http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf?version=1).

3. Types of messages

The server and the client send messages over the network to communicate with each other. The messages can be of any type which includes audio messages, video messages, command messages, shared object messages, data messages, and user control messages.

3.1. Command message

Command messages carry the AMF-encoded commands between the client and the server.These messages have been assigned message type value of 20 for AMF0 encoding and message type value of 17 for AMF3 encoding. These messages are sent to perform some operations like connect, createStream, publish, play, pause on the peer. Command messages like onstatus, result etc. are used to inform the sender about the status of the requested commands. A command message consists of command name, transaction ID, and command object that contains related parameters. A client or a server can request Remote Procedure Calls (RPC) over streams that are communicated using the command messages to the peer.

3.2. Data message

The client or the server sends this message to send Metadata or any user data to the peer. Metadata includes details about the data(audio, video etc.) like creation time, duration, theme and so on. These messages have been assigned message type value of 18 for AMF0 and message type value of 15 for AMF3.

3.3. Shared object message

A shared object is a Flash object (a collection of name value pairs) that are in synchronization across multiple clients, instances, and so on. The message types kMsgContainer=19 for AMF0 and kMsgContainerEx=16 for AMF3 are reserved for shared object events. Each message can contain multiple events.

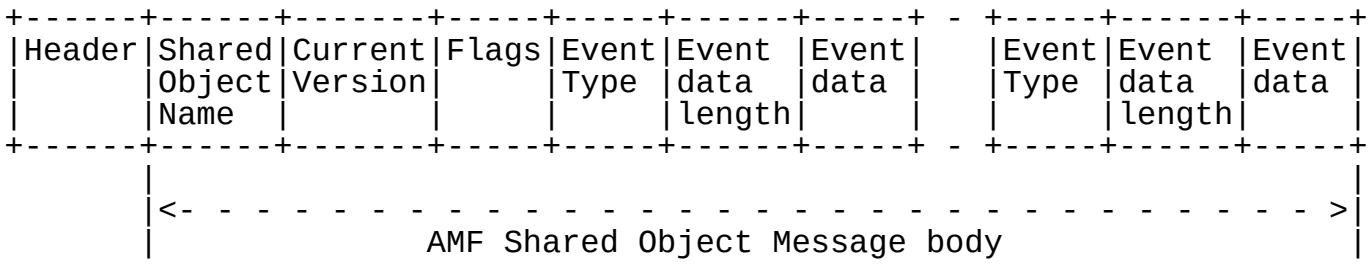

Figure 1 The shared object message format

The following event types are supported:

Event	Description
Use(=1)	The client sends this event to inform the server about the creation of a named shared object.
Release(=2)	The client sends this event to the server when the shared object is deleted on the client side.
Request Change (=3)	The client sends this event to request that the change the value associated with a named parameter of the shared object.
Change (=4)	The server sends this event to notify all clients, except the client originating the request, of a change in the value of a named parameter.
Success (=5)	The server sends this event to the requesting client in response to RequestChange event if the request is accepted.
SendMessage (=6)	The client sends this event to the server to broadcast a message. On receiving this event, the server broadcasts a message to all the clients, including the sender.
Status (=7)	The server sends this event to notify clients about error conditions.
Clear (=8)	The server sends this event to the client to clear a shared object. The server also sends this event in response to Use event that the client sends on connect.
Remove (=9)	The server sends this event to have the client delete a slot.
Request Remove (=10)	The client sends this event to have the client delete a slot.

Use Success (=11)	The server sends this event to the client on a successful connection.

3.4. Audio message

The client or the server sends this message to send audio data to the peer. The message type value of 8 is reserved for audio messages.

3.5. Video message

The client or the server sends this message to send video data to the peer. The message type value of 9 is reserved for video messages. These messages are large and can delay the sending of other type of messages. To avoid such a situation, the video message is assigned the lowest priority.

3.6. Aggregate message

An aggregate message is a single message that contains a list of sub-messages. The message type value of 22 is reserved for aggregate messages.

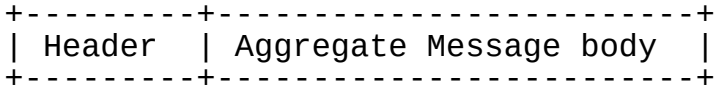

Figure 2 The aggregate message format

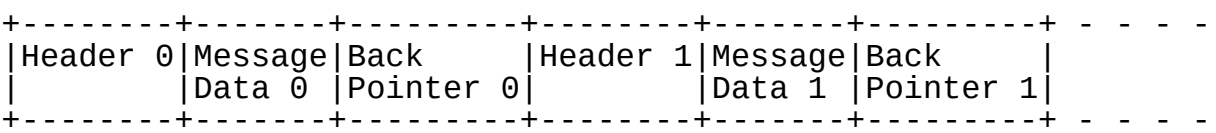

Figure 3 The aggregate message body format

The back pointer contains the size of the previous message including its header. It is included to match the format of FLV file and is used for backward seek.

Using aggregate messages has several performance benefits:

- The chunk stream can send at most a single complete message within a chunk. Therefore, increasing the chunk size and using the aggregate message reduces the number of chunks sent.

- The sub-messages can be stored contiguously in memory. It is more efficient when making system calls to send the data on the network.

3.7. User Control message

The client or the server sends this message to notify the peer about the user control events. For information about the message format, refer to the User Control Messages section in the RTMP Message Foramts draft.

The following user control event types are supported:

Event	Description
Stream Begin (=0)	The server sends this event to notify the client that a stream has become functional and can be used for communication. By default, this event is sent on ID 0 after the application connect command is successfully received from the client. The event data is 4-byte and represents the stream ID of the stream that became functional.
Stream EOF (=1)	The server sends this event to notify the client that the playback of data is over as requested on this stream. No more data is sent without issuing additional commands. The client discards the messages received for the stream. The 4 bytes of event data represent the ID of the stream on which playback has ended.
StreamDry (=2)	The server sends this event to notify the client that there is no more data on the stream. If the server does not detect any message for a time period, it can notify the subscribed clients that the stream is dry. The 4 bytes of event data represent the stream ID of the dry stream.
SetBuffer Length (=3)	The client sends this event to inform the server of the buffer size (in milliseconds) that is used to buffer any data coming over a stream. This event is sent before the server starts processing the stream. The first 4 bytes of the event data represent the stream ID and the next 4 bytes represent the buffer length, in milliseconds.
StreamIs Recorded (=4)	The server sends this event to notify the client that the stream is a recorded stream. The 4 bytes event data represent the stream ID of the recorded stream.
PingRequest (=6)	The server sends this event to test whether the client is reachable. Event data is a 4-byte timestamp, representing the local server time when the server dispatched the command. The client responds with kMsgPingResponse on receiving kMsgPingRequest.

PingResponse (=7)	The client sends this event to the server in response to the ping request. The event data is a 4-byte timestamp, which was received with the kMsgPingRequest request.

4. Types of commands

The client and the server exchange commands which are AMF encoded. The sender sends a command message that consists of command name, transaction ID, and command object that contains related parameters. For example, the connect command contains ‘app’ parameter, which tells the server application name the client is connected to. The receiver processes the command and sends back the response with the same transaction ID. The response string is either _result, _error, or a method name, for example, verifyClient or contactExternalServer.

A command string of _result or _error signals a response. The transaction ID indicates the outstanding command to which the response refers. It is identical to the tag in IMAP and many other protocols. The method name in the command string indicates that the sender is trying to run a method on the receiver end.

The following class objects are used to send various commands:

- NetConnection – An object that is a higher-level representation of connection between the server and the client.
- NetStream – An object that represents the channel over which audio streams, video streams and other related data are sent. We also send commands like play , pause etc. which control the flow of the data.

4.1. NetConnection commands

The NetConnection manages a two-way connection between a client application and the server. In addition, it provides support for asynchronous remote method calls.

The following commands can be sent on the NetConnection :

- connect

- call
- close
- createStream

4.1.1. connect

The client sends the connect command to the server to request connection to a server application instance.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command. Set to “connect”.
Transaction ID	Number	Always set to 1.
Command Object	Object	Command information object which has the name-value pairs.
Optional User Arguements	Object	Any optional information

Following is the description of the name-value pairs used in Command Object of the connect command.

Property	Type	Description	Example Value
app	String	The Server application name the client is connected to.	testapp
flashver	String	Flash Player version. It is the same string as returned by the ApplicationScript getversion () function.	FMSc/1.0
swfUrl	String	URL of the source SWF file making the connection.	file://C:/FlvPlayer.swf
tcUrl	String	URL of the Server. It has the following format. protocol://servername:port/appName/appInstance	rtmp://localhost:1935/testapp/instance1
fpad	Boolean	True if proxy is being used.	true or false
audioCodecs	Number	Indicates what audio codecs the client supports.	SUPPORT_SND_MP3
videoCodecs	Number	Indicates what video codecs are supported.	SUPPORT_VID_SORENSON
pageUrl	String	URL of the web page from where the SWF file was loaded.	http://somehost/sample.html
object Encoding	Number	AMF encoding method.	kAMF3

Values for the audio codecs property:

Source Code Constant	Usage	Value
SUPPORT_SND_NONE	Raw sound, no compression	0x0001
SUPPORT_SND_ADPCM	ADPCM compression	0x0002
SUPPORT_SND_MP3	mp3 compression	0x0004
SUPPORT_SND_INTEL	Not used	0x0008
SUPPORT_SND_UNUSED	Not used	0x0010
SUPPORT_SND_NELLY8	NellyMoser at 8-kHz compression	0x0020
SUPPORT_SND_NELLY	NellyMoser compression (5, 11, 22, and 44 kHz)	0x0040
SUPPORT_SND_G711A	G711A sound compression (Flash Media Server only)	0x0080
SUPPORT_SND_G711U	G711U sound compression (Flash Media Server only)	0x0100
SUPPORT_SND_NELLY16	NellyMouser at 16-kHz compression	0x0200
SUPPORT_SND_AAC	Advanced audio coding (AAC) codec	0x0400
SUPPORT_SND_SPEEX	Speex Audio	0x0800
SUPPORT_SND_ALL	All RTMP-supported audio codecs	0x0FFF

Values for the videoCodecs Property:

Source Code Constant	Usage	Value
SUPPORT_VID_UNUSED	Obsolete value	0x0001
SUPPORT_VID_JPEG	Obsolete value	0x0002
SUPPORT_VID_SORENSON	Sorenson Flash video	0x0004
SUPPORT_VID_HOMEBREW	V1 screen sharing	0x0008
SUPPORT_VID_VP6 (On2)	On2 video (Flash 8+)	0x0010
SUPPORT_VID_VP6ALPHA (On2 with alpha channel)	On2 video with alpha channel	0x0020
SUPPORT_VID_HOMEBREWV (screensharing v2)	Screen sharing version 2 (Flash 8+)	0x0040
SUPPORT_VID_H264	H264 video	0x0080
SUPPORT_VID_ALL	All RTMP-supported video codecs	0x00FF

Values for the video function property:

Source Code Constant	Usage	Value
SUPPORT_VID_CLIENT_SEEK	Indicates that the client can seek frame-accurate on the client	1

Values for the object encoding property:

Source Code Constant	Usage	Value
kAMF0	AMF0 object encoding supported by Flash 6 and later	0
kAMF3	AMF3 encoding from Flash 9 (AS3)	3

The command structure from server to client is as follows:

Field Name	Type	Description
Command Name	String	_result or _error; indicates whether the response is result or error.
Transaction ID	Number	Transaction ID is 1 for call connect responses
Properties	Object	Name-value pairs that describe the properties(fmsver etc.) of the connection.
Information	Object	Name-value pairs that describe the response from|the server. ‘code’, ‘level’, ‘description’ are names of few among such information.

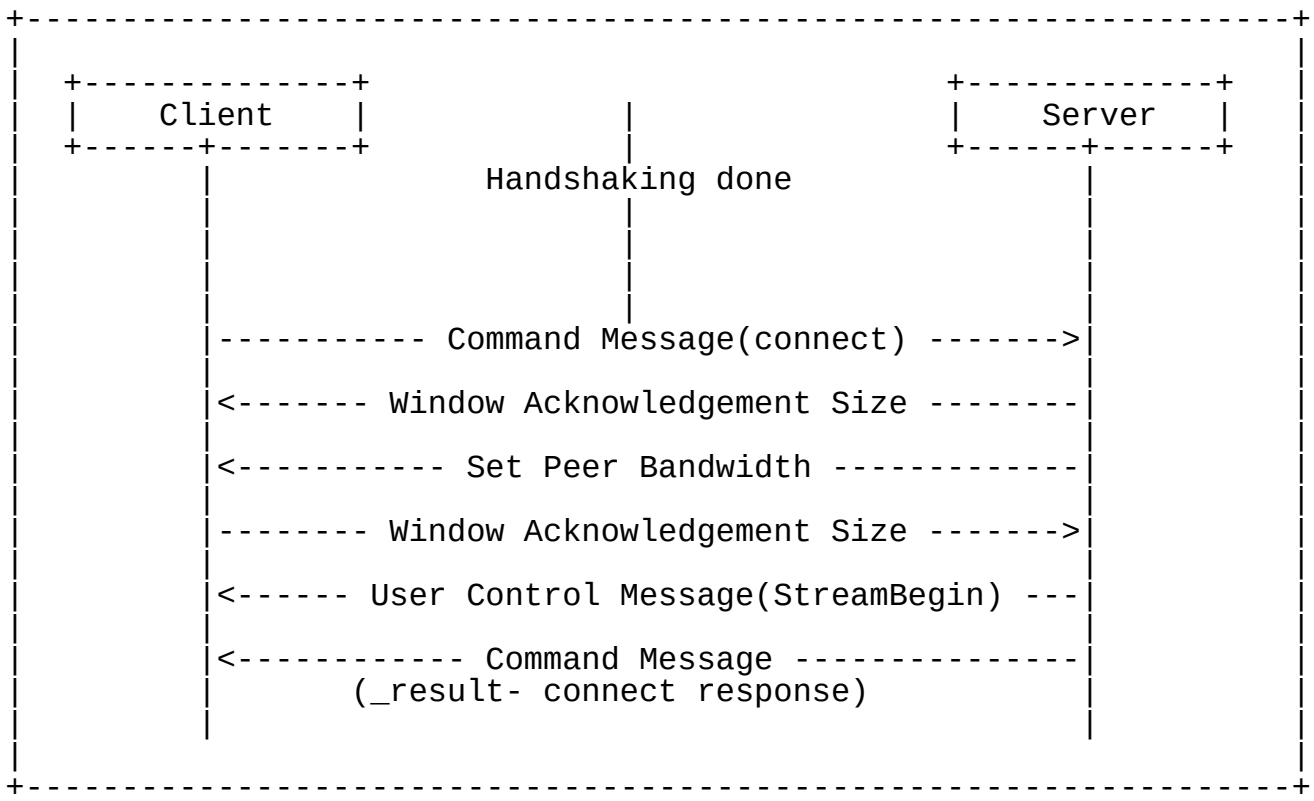

Figure 4 Message flow in the connect command

The message flow during the execution of the command is:

- Client sends the connect command to the server to request to connect with the server application instance.
- After receiving the connect command, the server sends the protocol message 'Window Acknowledgement Size' to the client. The server also connects to the application mentioned in the connect command.
- The server sends the protocol message 'Set Peer Bandwidth' to the client.
- The client sends the protocol message 'Window Acknowledgement Size' to the server after processing the protocol message 'Set Peer Bandwidth'.
- The server sends an another protocol message of type User Control Message(StreamBegin) to the client.

- The server sends the result command message informing the client of the connection status (success/fail). The command specifies the transaction ID (always equal to 1 for the connect command). The message also specifies the properties, such as Flash Media Server version (string), capabilities (number) In addition it specificies other connection response related information like level(string), code(string), description (string), objectencoding (number)etc.

4.1.2. Call

The call method of the NetConnection object runs remote procedure calls (RPC) at the receiving end. The called RPC name is passed as a parameter to the call command.

The command structure from the sender to the receiver is as follows:

Field Name	Type	Description
Procedure Name	String	Name of the remote procedure that is called.
Transaction ID	Number	If a response is expected we give a transaction Id. Else we pass a value of 0
Command Object	Object	If there exists any command info this is set, else this is set to null type.
Optional Arguements	Object	Any optional arguments to be provided

The command structure of the response is as follows:

Field Name	Type	Description
Command Name	String	Name of the command.
Transaction ID	Number	ID of the command, to which the response belongs to
Command Object	Object	If there exists any command info this is set, else this is set to null type.
Response	Object	Response from the method that was called.

4.1.3. createStream

The client sends this command to the server to create a logical channel for message communication The publishing of audio, video, and metadata is carried out over stream channel created using the createStream command.

NetConnection is the default communication channel, which has a stream ID 0. Protocol and a few command messages, including createStream, use the default communication channel.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command. Set to “createStream”.
Transaction ID	Number	Transaction ID of the command.
Command Object	Object	If there exists any command info this is set, else this is set to null type.

The command structure from server to client is as follows:

Field Name	Type	Description
Command Name	String	_result or _error; indicates whether the response is result or error.
Transaction ID	Number	ID of the command that response belongs to.
Command Object	Object	If there exists any command info this is set, else this is set to null type.
Stream ID	Number	The return value is either a stream ID or an error information object.

4.2. NetStream commands

The NetStream defines the channel through which the streaming audio, video, and data messages can flow over the NetConnection that connects the client to the server. A NetConnection object can support multiple NetStreams for multiple data streams.

The following commands can be sent on the NetStream :

- play
- play2
- deleteStream
- closeStream
- receiveAudio
- receiveVideo
- publish
- seek
- pause

4.2.1. play

The client sends this command to the server to play a stream. A playlist can also be created using this command multiple times.

If you want to create a dynamic playlist that switches among different live or recorded streams, call play more than once and pass false for reset each time. Conversely, if you want to play the specified stream immediately, clearing any other streams that are queued for play, pass true for reset.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command. Set to “play”.
Transaction ID	Number	Transaction ID set to 0.
Command Object	Null	Command information does not exist. Set to null type.
Stream Name	String	Name of the stream to play. To play video (FLV) files, specify the name of the stream without a file extension (for example, "sample"). To play back MP3 or ID3 tags, you must precede the stream name with mp3: (for example, "mp3:sample". To play H.264/AAC files, you must precede the stream name with mp4: and specify the file extension. For example, to play the file sample.m4v,specify "mp4:sample.m4v"
Start	Number	An optional parameter that specifies the start time in seconds. The default value is -2, which means the subscriber first tries to play the live stream specified in the Stream Name field. If a live stream of that name is not found,it plays the recorded stream specified in the Stream Name field. If you pass -1 in the Start field, only the live stream specified in the Stream Name field is played. If you pass 0 or a positive number in the Start field, a recorded stream specified in the Stream Name field is played beginning from the time specified in the Start field. If no recorded stream is found, the next item in the playlist is played.
Duration	Number	An optional parameter that specifies the duration of playback in seconds. The default value is -1. The -1 value means

		a live stream is played until it is no longer available or a recorded stream is played until it ends. If u pass 0, it plays the single frame since the time specified in the Start field from the beginning of a recorded stream. It is assumed that the value specified in the Start field is equal to or greater than 0. If you pass a positive number, it plays a live stream for the time period specified in the Duration field. After that it becomes available or plays a recorded stream for the time specified in the Duration field. (If a stream ends before the time specified in the Duration field, playback ends when the stream ends.) If you pass a negative number other than -1 in the Duration field, it interprets the value as if it were -1.
Reset	Boolean	An optional Boolean value or number that specifies whether to flush any previous playlist.

The command structure from the server to the client is as follows:

Field Name	Type	Description
Command Name	String	Name of the command. If the play command is successful, the command name is set to onStatus.
Description	String	If the play command is successful, the client receives OnStatus message from server which is NetStream.Play.Start. If the specified stream is not found , NetStream.Play.StreamNotFound is received.

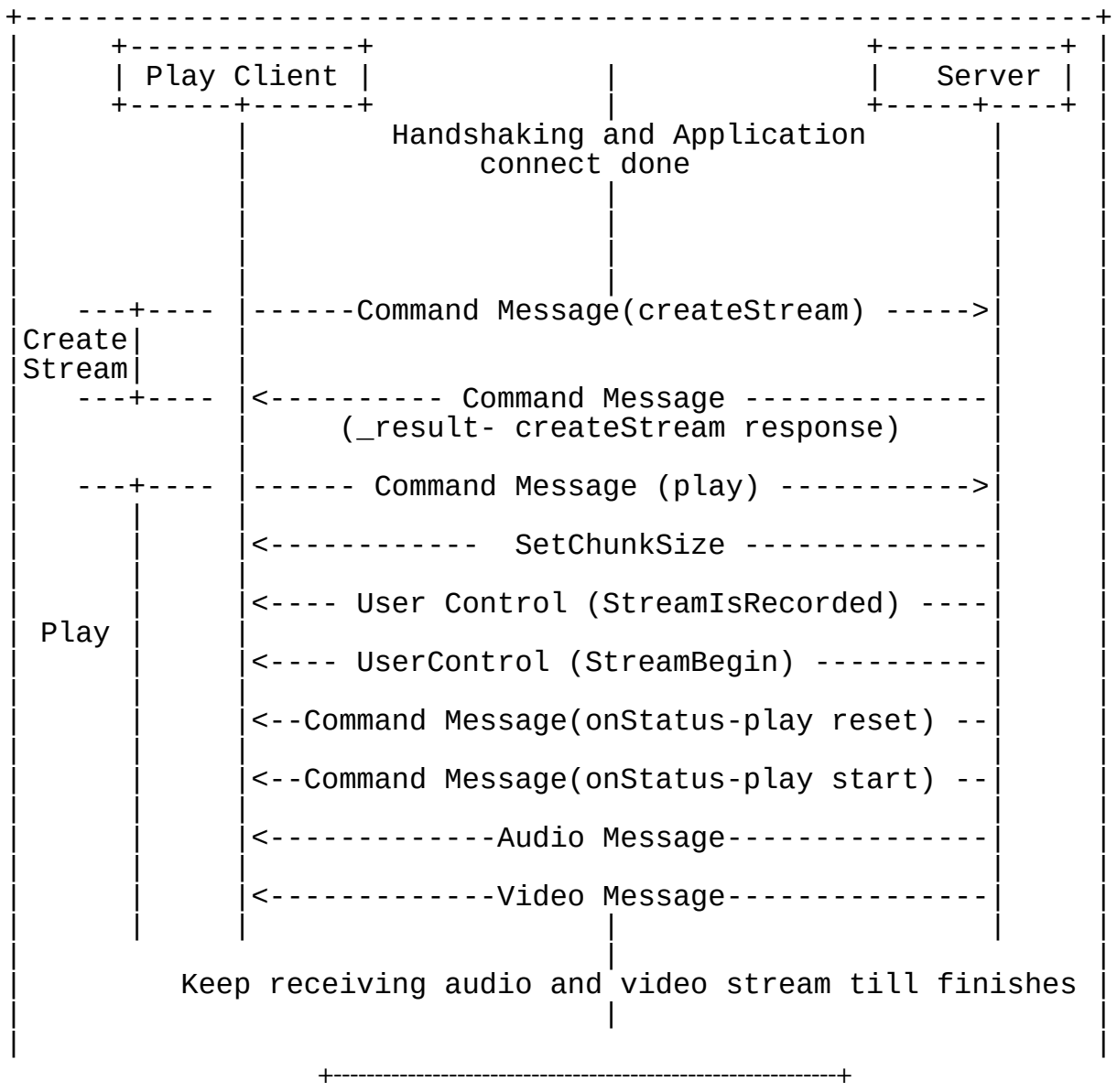

Figure 5 Message flow in the play command

The message flow during the execution of the command is:

- The client sends the play command after receiving result of the createStream command as success from the server.
- On receiving the play command, the server sends a protocol message to set the chunk size.

- The server sends another protocol message (user control) specifying the event ‘StreamIsRecorded’ and the stream ID in that message. The message carries the event type in the first 2 bytes and the stream ID in the last 4 bytes.

- The server sends another protocol message (user control) specifying the event ‘StreamBegin’, to indicate beginning of the streaming to the client.

- The server sends OnStatus command messages NetStream.Play.Start & NetStream.Play.Reset if the play command sent by the client is successful. NetStream.Play.Reset is sent by the server only if the play command sent by the client has set the reset flag. If the stream to be played is not found, the Server sends the onStatus message NetStream.Play.StreamNotFound.

After this, the server sends audio and video data, which the client plays.

4.2.2. play2

Unlike the play command, play2 can switch to a different bit rate stream without changing the timeline of the content played. The server maintains multiple files for all supported bitrates that the client can request in play2.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “play2”.
Transaction ID	Number	Transaction ID set to 0.
Start Time	Object	A AMF encoded object that stores a number value. The value in this field specifies the beginning position of the stream, in seconds. If 0 is passed in the Start Time field, the stream is played from the current timeline.
oldStreamName	Object	A AMF encoded object that stores a string value. Its value is a string containing the stream name parameter and the old stream name.
Stream Name	Object	A AMF encoded object that stores a string value. It stores the name of the stream that is played.
Duration	Object	A AMF encoded object that stores a number value. The value stored in it specifies the total duration of playing the stream.
Transition	Object	A AMF encoded object that stores a string value. Its value defines the playlist transition mode (switch or. swap mode)switch:Performs multi-bitrate streaming by switching 1-bit rate version of a stream to another. swap: Replaces the value in oldStreamName with the value in streamName, and stores the remaining playlist queue as is. However, in this case, the server does not make any assumptions about the content of the streams and treats them like different content. Hence, it either switches at the stream boundary or never.

The message flow for the command is shown in the following illustration.

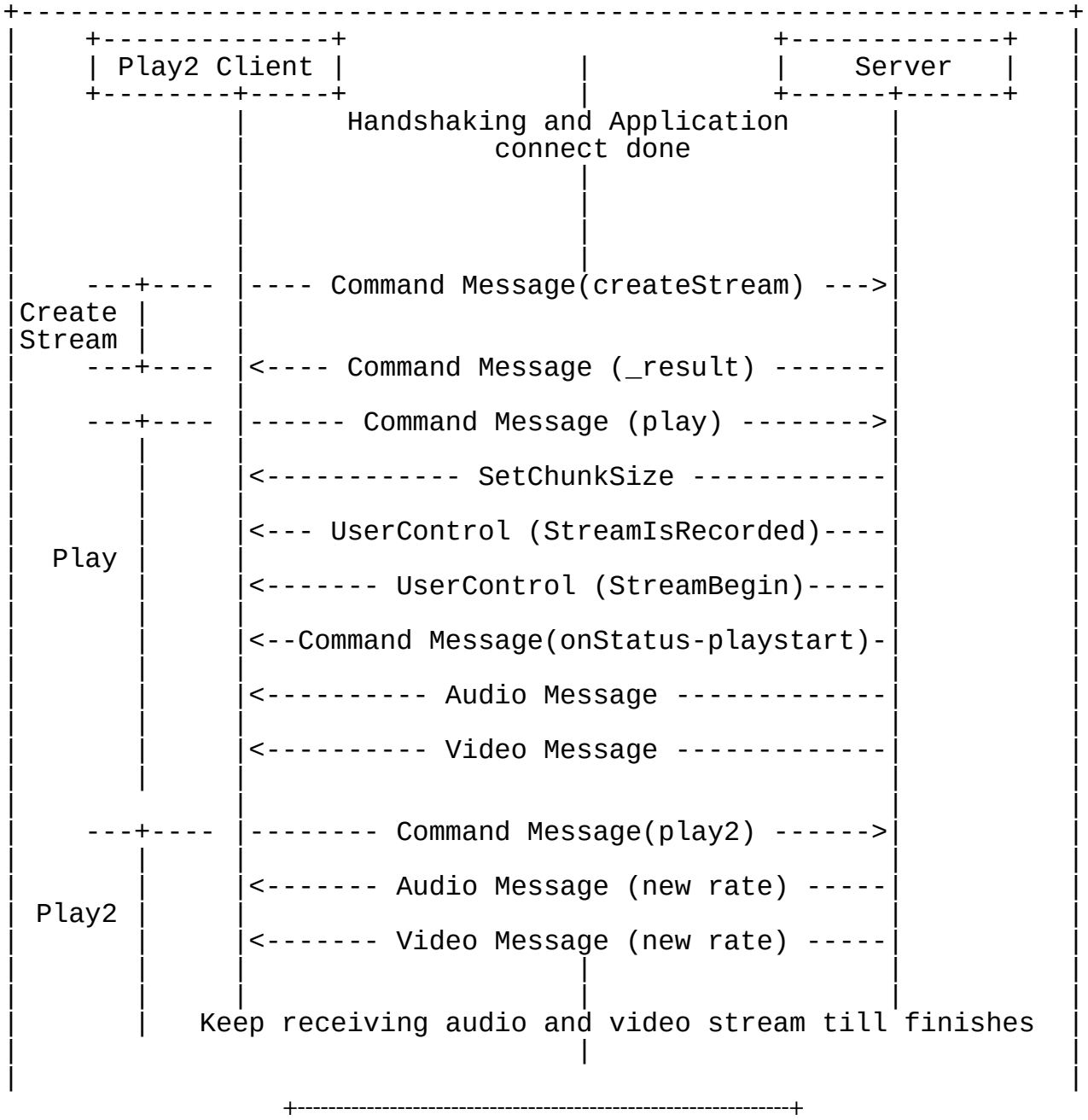

Figure 1 Message flow in the play2 command

4.2.3. deleteStream

NetStream sends the deleteStream command when the NetStream object is getting destroyed.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “deleteStream”.
Transaction ID	Number	Transaction ID set to 0.
Command Object	Null	Command information object does not exist. Set to null type.
Stream ID	Number	The ID of the stream that is destroyed on the server.

The server does not send any response.

4.2.4. receiveAudio

NetStream sends the receiveAudio message to inform the server whether to send or not to send the audio to the client.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “receiveAudio”.
Transaction ID	Number	Transaction ID set to 0.
Command Object	Null	Command information object does not exist. Set to null type.
Bool Flag	Boolean	true or false to indicate whether to receive audio or not.

The server does not send any response.

4.2.5. receiveVideo

NetStream sends the receiveVideo message to inform the server whether to send the video to the client or not.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “receiveVideo”.
Transaction ID	Number	Transaction ID set to 0.
Command Object	Null	Command information object does not exist. Set to null type.
Bool Flag	Boolean	true or false to indicate whether to receive video or not.

The server does not send any response.

4.2.6. Publish

The client sends the publish command to publish a named stream to the server. Using this name, any client can play this stream and receive the published audio, video, and data messages.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “publish”.
Transaction ID	Number	Transaction ID set to 0.
Command Object	Null	Command information object does not exist. Set to null type.
Publishing Name	String	Name with which the stream is published.
Publishing Type	String	Type of publishing. Set to “live”, “record”, or “append”. record: The stream is published and the data is recorded to a new file.The file is stored on the server in a subdirectory within the directory that contains the server application. If the file already exists, it is overwritten. append: The stream is published and the data is appended to a file. If no file is found, it is created. live: Live data is published without recording it in a file.

The server responds with the OnStatus command to mark the beginning of publish.

4.2.7. seek

The client sends the seek command to seek the offset (in milliseconds) within a media file or playlist.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “seek”.
Transaction ID	Number	Transaction ID set to 0.
Command Object	Null	There is no command information object for this command. Set to null type.
milliSeconds	Number	Number of milliseconds to seek into the playlist.

The server sends a status message NetStream.Seek.Notify when seek is successful. In failure, it returns an _error message.

4.2.8. pause

The client sends the pause command to tell the server to pause or start playing.

The command structure from the client to the server is as follows:

Field Name	Type	Description
Command Name	String	Name of the command, set to “pause”.
Transaction ID	Number	There is no transaction ID for this command. Set to 0.
Command Object	Null	Command information object does not exist. Set to null type.
Pause/Unpause Flag	Boolean	true or false, to indicate pausing or resuming play
milliSeconds	Number	Number of milliseconds at which the the stream is paused or play resumed. This is the current stream time at the Client when stream was paused. When the playback is resumed, the server will only send messages with timestamps greater than this value.

The server sends a status message NetStream.Pause.Notify when the stream is paused. NetStream.Unpause.Notify is sent when a stream in un-paused. In failure, it returns an _error message.

5. Message exchange example

Here are a few examples to explain message exchange using RTMP.

5.1. Publish recorded video

The example illustrates how a publisher can publish a stream and then stream the video to the server. Other clients can subscribe to this published stream and play the video.

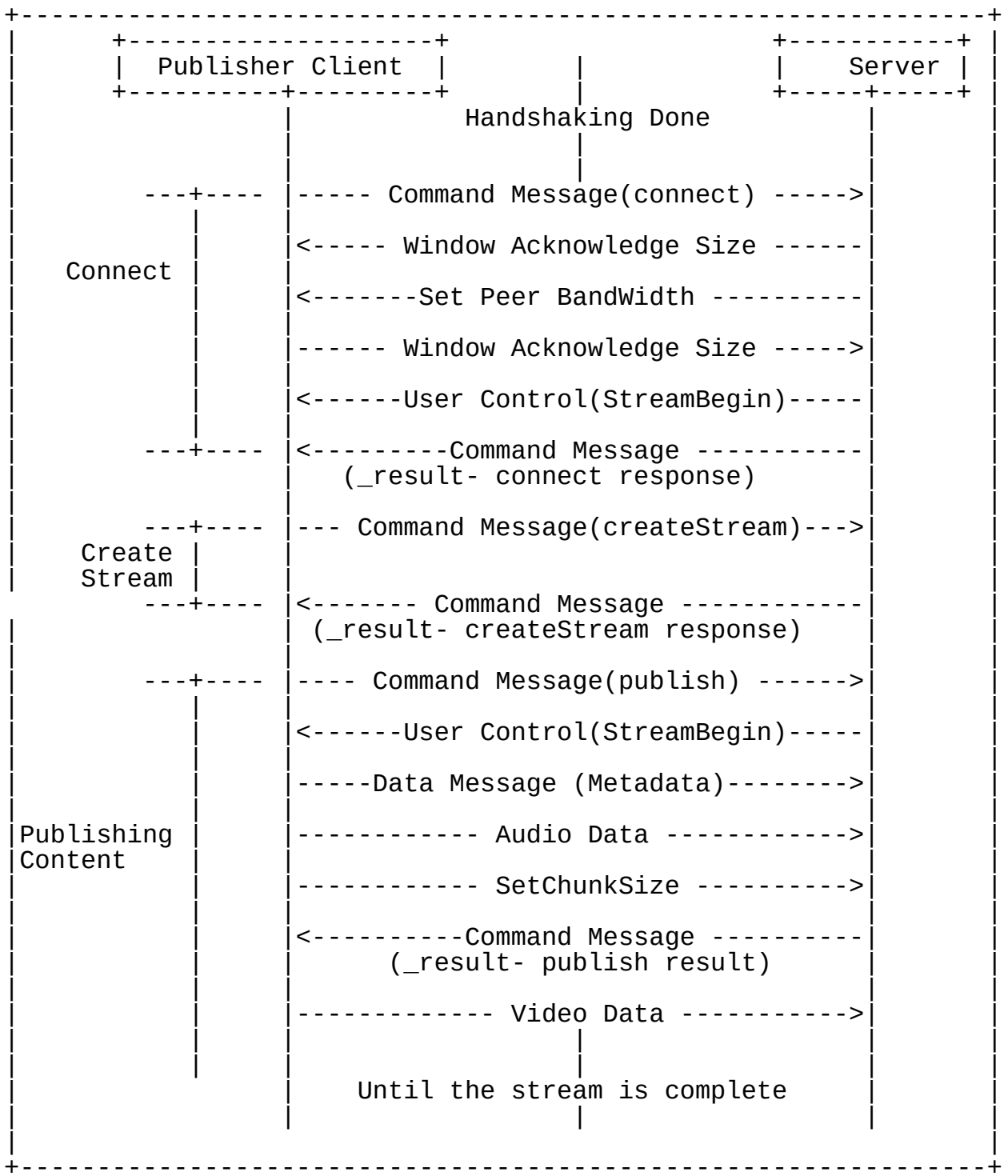

Figure 1 Message flow in publishing a video stream

5.2. Broadcasting a shared object message

The example illustrates the messages that are exchanged during the creation and changing of shared object. It also illustrates the process of shared object message broadcasting.

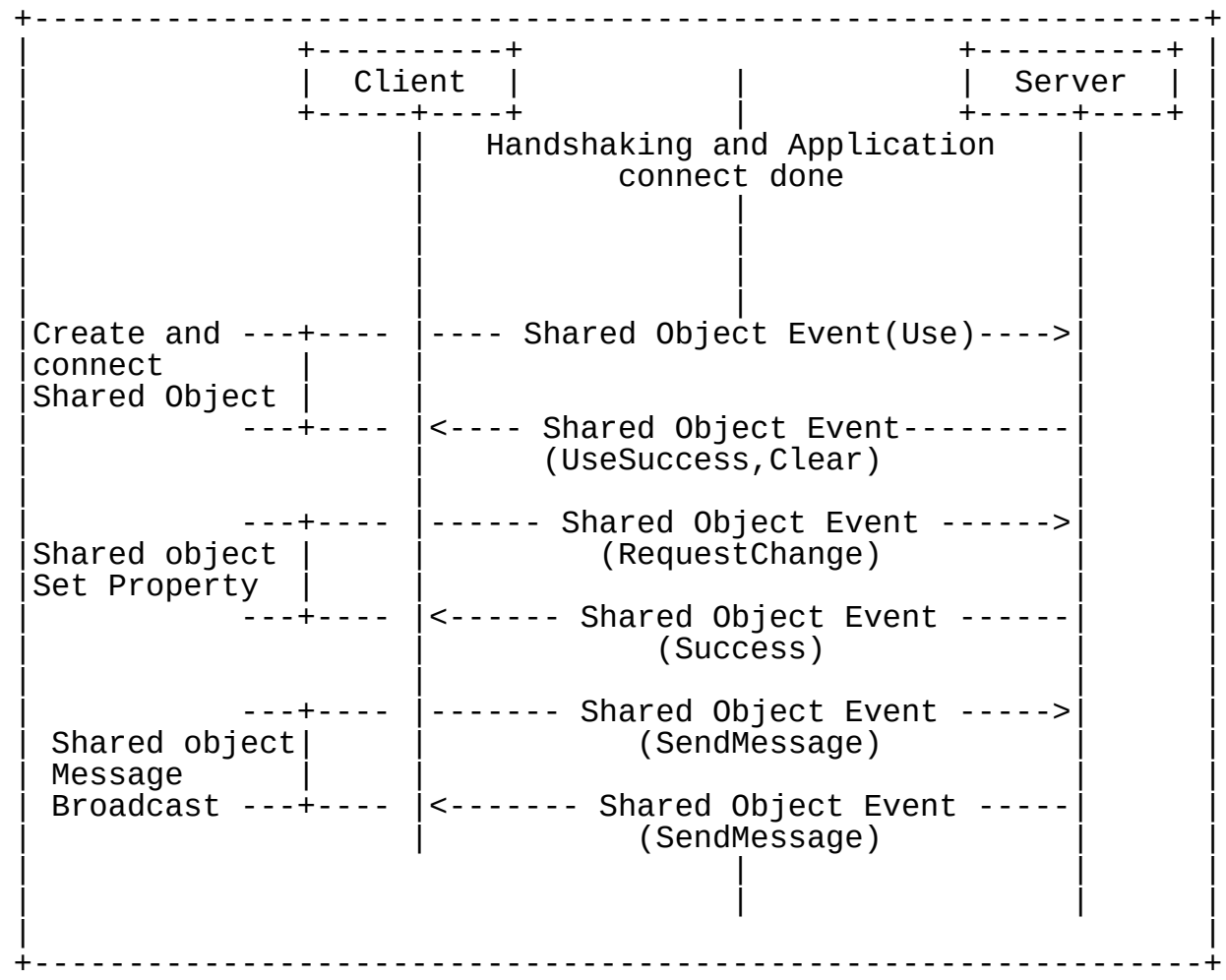

Figure 1 Shared object message broadcast

5.3. Publish MetaData from recorded stream

This example describes the message exchange for publishing metadata.

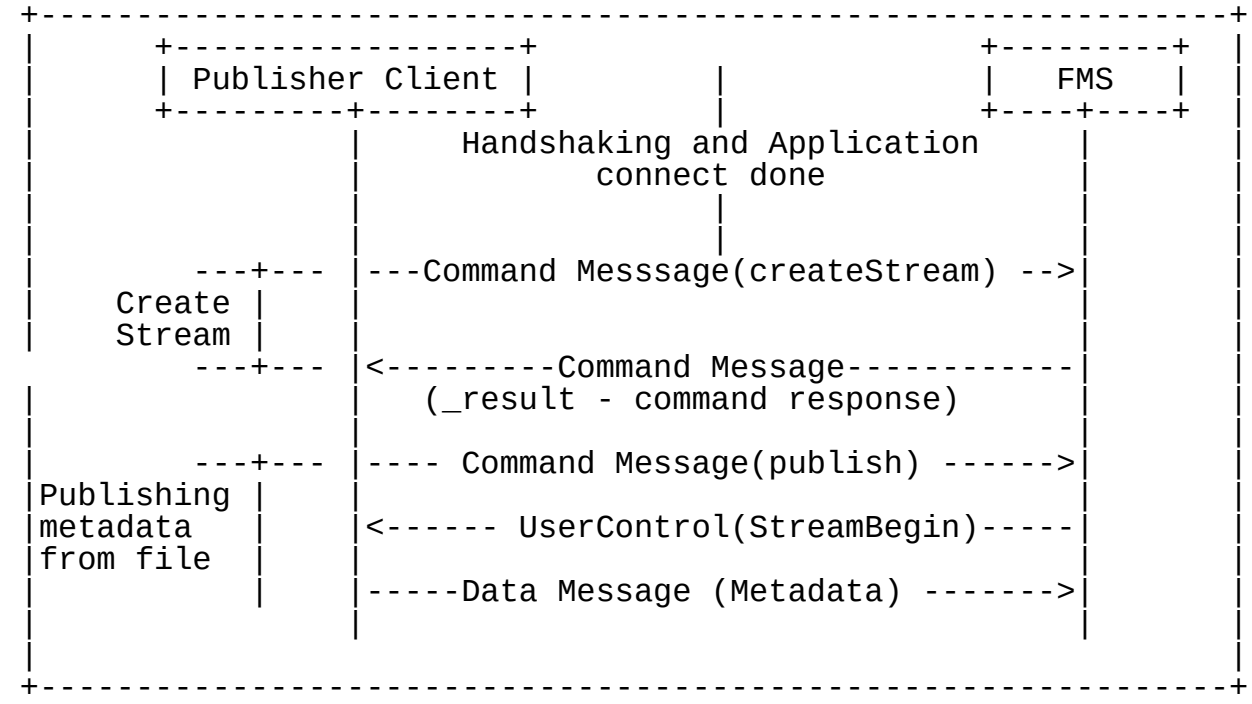

Figure 2 Publishing metadata

6. References

6.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

6.2. Informative References

[3] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

[Fab1999] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

7. Acknowledgments

Address:

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704