Real Time Messaging Protocol (RTMP) Message Formats
draft-rtmp-01.txt

Copyright Notice

Abstract

This memo describes the Real Time Messaging Protocol, an application-level protocol designed for multiplexing and packetizing multimedia transport streams (such as audio, video, and interactive content) over a suitable transport protocol, such as RTMP chunking stream protocol.

Table of Contents

1. Introduction

The document specifies the Real Time Messaging Protocol, which specifies the format of the messages that are transferred between entities on a network using a lower level transport layer.

While RTMP was designed to work with the Real Time Messaging Chunk Stream Protocol, it can send the messages using any other transport protocol. RTMP Chunk Stream and RTMP together are suitable for a wide variety of audio-video applications, from one-to-one and one-to-many live broadcasting to video-on-demand services to interactive conferencing applications.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP14], [RFC2119].

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 in which it is flowing.

Payload:
The data contained in a packet, for example audio samples or compressed video data. The payload format and interpretation are beyond the scope of this document.

Packet:
A data packet consists of the fixed header and the payload data. Some underlying protocols may require an encapsulation of the packet to be defined.

3. Byte Order, Alignment, and Time Format

All integer fields are carried in network byte order, byte zero is the first byte shown, and bit zero is the most significant bit in a word or field. This byte order is commonly known as big-endian. The transmission order is described in detail in [STD5]. Unless otherwise noted, numeric constants in this document are in decimal (base 10).

Except as otherwise specified, all data in RTMP is byte-aligned, for example, a 16-bit field may be at an odd byte offset. Where padding is indicated, padding bytes SHOULD have the value zero.

Timestamps in RTMP are given as an integer number of milliseconds, relative to an unspecified epoch. Typically, each message stream will start with a message having timestamp 0, but this is not required, as long as the two endpoints agree on the epoch. Note that this means that any synchronization across multiple streams (especially from separate hosts) requires some additional mechanism outside of RTMP.

Timestamps MUST be monotonically increasing, and SHOULD be linear in time, to allow applications to handle synchronization, bandwidth measurement, jitter detection, and flow control.

Because timestamps are generally only 32 bits long, they will roll over after fewer than 50 days. Because streams are allowed to run continuously, potentially for years on end, an RTMP application MUST use modular arithmetic for subtractions and comparisons, and SHOULD be capable of handling this wraparound heuristically. Any reasonable method is acceptable, as long as both endpoints agree. An application could assume, for example, that all adjacent timestamps are within 2^31 milliseconds of each other, so 10000 comes after 4000000000, while 3000000000 comes before 4000000000.

Timestamp deltas are also specified as an unsigned integer number of milliseconds, relative to the previous timestamp. Timestamp deltas may be either 24 or 32 bits long.

4. RTMP Message Format

The server and the client send RTMP messages over the network to communicate with each other. The messages could include audio, video, data, or any other messages.

The RTMP message has two parts, a header and its payload.

4.1. Message Header

The message header contains the following:

Message Type:
One byte field to represent the message type. A range of type IDs (1-7) are reserved for protocol control messages.

Length:
Three-byte field that represents the size of the payload in bytes. It is set in big-endian format.

Timestamp:
Four-byte field that contains a timestamp of the message. The 4 bytes are packed in the big-endian order.

Message Stream Id:
Three-byte field that identifies the stream of the message. These bytes are set in big-endian format.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Message Type|                Payload length                   |
|   (1 byte)  |                  (3 bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Timestamp                            |
|                          (4 bytes)                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Stream ID                     |
|                (3 bytes)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1 Message header

4.2. Message Payload

The other part which is the payload is the actual data that is contained in the message. For example, it could be some audio samples or compressed video data. The payload format and interpretation are beyond the scope of this document.

5. Protocol Control Messages

RTMP reserves message type IDs 1-7 for protocol control messages. These messages contain information needed by the RTM Chunk Stream protocol or RTMP itself. Protocol messages with IDs 1 & 2 are reserved for usage with RTM Chunk Stream protocol. Protocol messages with IDs 3-6 are reserved for usage of RTMP. Protocol message with ID 7 is used between edge server and origin server.

Protocol control messages MUST have message stream ID 0 (called as control stream) and chunk stream ID 2, and are sent with highest priority.

Each protocol control message type has a fixed-size payload.

5.1. Set Chunk Size (1)

Protocol control message 1, Set Chunk Size, is used to notify the peer a new maximum chunk size to use.

The value of the chunk size is carried as 4-byte message payload. A default value exists for chunk size, but if the sender wants to change this value it notifies the peer about it through this protocol message. For example, a client wants to send 131 bytes of data and the chunk size is at its default value of 128. So every message from the client gets split into two chunks. The client can choose to change the chunk size to 131 so that every message get split into two chunks. The client MUST send this protocol message to the server to notify that the chunk size is set to 131 bytes.

The maximum chunk size can be 65536 bytes. Chunk size is maintained independently for server to client communication and client to server communication.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          chunk size (4 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Figure 2 Pay load for the protocol message ‘Set Chunk Size’

chunk size: 32 bits
  This field holds the new chunk size, which will be used for all future chunks sent by this chunk stream.

5.2. Abort Message (2)

Protocol control message 2, Abort Message, is used to notify the peer if it is waiting for chunks to complete a message, then to discard the partially received message over a chunk stream and abort processing of that message. The peer receives the chunk stream ID of the message to be discarded as payload of this protocol message. This message is sent when the sender has sent part of a message, but wants to tell the receiver that the rest of the message will not be sent.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    chunk stream id (4 bytes)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Figure 3 Pay load for the protocol message ‘Abort Message’.

chunk stream ID: 32 bits
  This field holds the chunk stream ID, whose message is to be discarded.

5.3. Acknowledgement (3)

The client or the server sends the acknowledgment to the peer after receiving bytes equal to the window size. The window size is the maximum number of bytes that the sender sends without receiving acknowledgment from the receiver. The server sends the window size to the client after application connects. This message specifies the sequence number, which is the number of the bytes received so far.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    sequence number (4 bytes)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Figure 4 Pay load for the protocol message ‘Acknowledgement’.

sequence number: 32 bits
  This field holds the number of bytes received so far.

5.4. User Control Message (4)

The client or the server sends this message to notify the peer about the user control events. This message carries Event type and Event data.

```
+------------------------------+-------------------------
|     Event Type ( 2- bytes )  | Event Data
+------------------------------+-------------------------
```
Figure 5 Pay load for the ‘User Control Message’.

The first 2 bytes of the message data are used to identify the Event type. Event type is followed by Event data. Size of Event data field is variable.

5.5. Window Acknowledgement Size (5)

The client or the server sends this message to inform the peer which window size to use when sending acknowledgment. For example, a server expects acknowledgment from the client every time the server sends bytes equivalent to the window size. The server updates the client about its window size after successful processing of a connect request from the client.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Acknowledgement Window size (4 bytes)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6 Pay load for ‘Window Acknowledgement Size’.

5.6. Set Peer Bandwidth (6)

The client or the server sends this message to update the output bandwidth of the peer. The output bandwidth value is the same as the window size for the peer. The peer sends ‘Window Acknowledgement Size’ back if its present window size is different from the one received in the message.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Acknowledgement Window size                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Limit type   |
+-+-+-+-+-+-+-+-+
```

Figure 7 Pay load for ‘Set Peer Bandwidth’

The sender can mark this message hard (0), soft (1), or dynamic (2) using the Limit type field. In a hard (0) request, the peer must send the data in the provided bandwidth. In a soft (1) request, the bandwidth is at the discretion of the peer and the sender can limit the bandwidth. In a dynamic (2) request, the bandwidth can be hard or soft.

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.

Address:

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