Table of Contents

Introduction 1.1

1. 简介 1.2

2. 贡献者 1.3

3. 定义 1.4

4. 字节顺序，校准和时间格式 1.5

5. RTMP块流 1.6

    5.1 消息格式 1.6.1

    5.2 握手 1.6.2

    5.3 分块 1.6.3

    5.4 协议控制消息 1.6.4

6. RTMP消息格式 1.7

7. RTMP指令消息 1.8

    7.1 消息类型 1.8.1

    7.2 指令类型 1.8.2

RTMP协议规范翻译工作

翻译过程中，参考了RTMP规范翻译1.0的翻译， 对其进行了补充。

1. RTMP协议规范

本文档是Adobe公司RTMP协议规范的中文翻译版本， 原版下载地址:
http://www.adobe.com/cn/devnet/rtmp.html

引言

本备忘录用来描述Adobe公司的实时消息协议（RTMP）， RTMP是一种应用层协议， 被设计用来对基于底层传输协议（如TCP）的多媒体传输流（如音频、视频和交互数据）进行复用和分包。

简介

RTMP在可靠流式传输（TCP）的基础上提供了双向消息多路复用功能，用于在通讯双方之间运载音频、视频和数据消息，及其相关时间信息的并行流数据。 协议实现方通常为不同的消息类型指定不同的优先级， 这样在网络带宽受限时改变底层传输顺序。

本文档描述RTMP协议的语法和使用规范。

术语

本文档中的关键词“必须”，“一定不”，“要求”，“可以”，“不可以”，“应该”，“不应该”，“推荐”，“不推荐”，“可能”和“可选”的解释，参考文档RFC2119

2. 贡献者

Rajesh Mallipeddi, Adobe公司前员工， 是本协议规范的原作者， 也是贡献最多的贡献者。

Mohit Srivastava, Adobe公司员工， 继续开发和修订本协议。

陈立朝， 中文翻译者

3. 定义

负载: 包中所承载的数据。 例如音频或视频数据。 负载格式和解释不再本文档描述范围内

包: 一个数据包由固定头部和所承载的数据组成。一些底层协议可能需要定义数据包的封装格式

端口: 在一个给定计算机中用于区分不同目标的抽象定义。 在TCP/IP协议中用一个小的正整数来表示端口。 OSI传输层的传输选择器等同于端口的概念。

传输地址: 用于唯一标识一个传输终端的网络地址和端口的组合， 例如IP地址和TCP端口的组合。 数据包从源地址传输到目的地址。

消息流: 允许消息传播的逻辑通讯通道。

消息流**ID**: 每个消息都会有一个关联的ID，用于标识其所在的消息流。

块: 消息的一个片段。 消息在网络上传输之前会被分成更小的片段，并交错存取。 分块确保在多个流中安全的按照时间戳顺序端到端传输。

块流: 允许块向某一确定方向传播的逻辑通讯通道。 块流可以是客户端到服务端， 也可以是服务端到客户端。

块流**ID**: 每个块都会有一个关联ID，用于标识其所在的块流。

复用: 将分开的音频/视频数据整合为统一的音视频流， 以使多个音视频流可以同步传输的过程。

复用分离: 复用的逆向过程。将交错的音视频数据分离为原始的音频和视频数据的过程。

远程过程调用（**RPC)**: 允许客户端或服务端调用另一端程序过程的请求。

元数据: 数据的描述信息。 一个影片的元数据包括名称、时长、创建时间等信息。

应用实例: 服务器上允许客户端发起连接请求的应用程序实例。

动作消息格式（**AMF**）: 用于序列化ActionScript对象图的一种紧凑二进制格式。 AMF有两个版本： AMF0和AMF3。

字节序: 顾名思义字节的顺序，既多字节类型的数据在内存中的存放顺序。TCP/IP各层协议将字节序定义为大字节序，因此TCP/IP协议中使用的字节序通常称之为网络字节序。

大字节序: (Big-Endian)高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

小字节序: (Little-Endian)低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

4. 字节顺序, 校准和时间格式

所有整数都是以网络字节序来承载的， 零字节是第一个字节，同时零位是一个字或字段中最显著的位。 这种字节序就是所谓的“big-endian"。 这种传输顺序的详细描述在IP协议[RFC0791]。 除非另行说明， 本文档中的所有数字都是十进制数。

在没有特殊说明的情况下， RTMP中的数据都是按字节对齐的； 例如： 一个16位的字段可能在奇数字节偏移。 在标有延拓的地方， 延拓字节应该赋予零值。

RTMP中的时间戳是用一个整数来表示的， 代表相对于一个未规定的起始时间的毫秒数。 通常，每个流的时间戳都从0开始， 但这不是必须的， 只要通讯的双方用统一的起始时间就可以了。 要注意的是，跨流的时间同步（特别是不同主机之间）需要额外的机制来实现。

由于时间戳的长度只有32位， 所以只能在50天内循环（49天17小时2分钟47.296秒）。 而流是可以不断运行的，可能多年才会结束。所以RTMP应用在处理时间戳是应该使用连续的数字算法， 并且应该支持回环处理。 例如：一个应用可以假设所有相邻的时间戳间隔不超过2^31-1毫秒， 那么 10000 在 4000000000， 3000000000在4000000000之前。

时间戳增量也是以毫秒为单位的无符号整数。时间戳增量可以是24位长度也可以是32位长度。

5. RTMP块流

本章节定义RTMP协议的块流，它为高级流媒体协议提供复用和分包的功能。

RTMP块流为配合RTMP协议而设计，它可以服务于任何发送消息流的协议。每个消息包含时间戳和负载类型信息。RTMP块流和RTMP组合适合多重音视频应用，从一对一或一回多直播到视频会议都能很好的满足。

当使用可靠传输协议（如TCP）时，RTMP块流为所有消息提供了可靠的跨流端对端按时间戳顺序发送的机制。RTMP块流不提供优先级控制，但是可以由上层协议提供这样的优先级。例如：一个视频直播服务器，当某个客户端网络比较慢时，可能会选择抛弃视频消息来保证声音消息能够及时接收，这是根据每个消息的发送时间或确认时间来确定的。

RTMP块流除自身内置的协议控制消息外，还为上层协议提供了用户控制消息的机制。

5.1 消息格式

消息格式依赖于上层协议， 消息可以被分成多个块以支持复用。 消息格式应该包含创建分块所必须的字段， 如下：

- 时间戳: 消息的时间戳， 占4个字节。
- 长度: 消息有效负载的长度， 如果消息头不能被省略， 则消息头的长度也应该包含在长度中。 占3个字节
- 类型**ID**: 消息类型ID。 一些类型ID是为协议控制消息保留的，这些消息所表示的信息同时供RTMP块流协议和上层协议使用。所有其他类型ID都用于上层协议， RTMP块流对这些ID做不透明处理。 实际上， RTMP块流不需要用这些值来区分类型； 所有消息都可以是相同的类型， 应用也可以用本字段来区分同步轨道而不是区分类型。 本字段占1个字节
- 消息流**ID**: 消息流ID可以是任意值。 被复合到同一个块流的消息流，依据消息的消息流ID进行分离。 另外， 就相关的块流而言， 这个值是不透明的。 这个字段在块头信息中占4个字节，并且使用小字节序。

5.2 握手

一个RTMP连接以握手开始。这里的握手和其他协议的握手不同， 这里的握手由3个固定大小的块组成， 而不是可变大小的块加上头。

5.2.1 握手流程

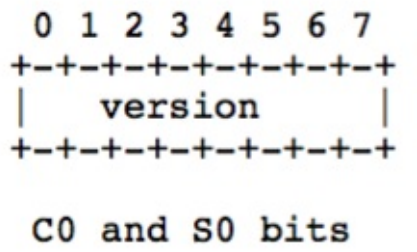
握手由客户端发送 `C0` 和 `C1` 块开始。

客户端必须等接收到 `S1` 之后才可以发送 `C2` 。 客户端必须等接收到 `S2` 之后才可以发送其他数据。

服务器必须等接收到 `C0` 之后才可以发送 `S0` 和 `S1` ， 也可能是接收到 `C1` 之后发送。 服务器必须等接收到 `C1` 之后才可以发送 `S2` 。 服务器必须等接收到 `C2` 之后才可以发送其他数据。

5.2.2 `C0` 和 `S0` 格式

`C0` 和 `S0` 是单独的一个字节， 可以当做一个8bit的整数字段来对待。

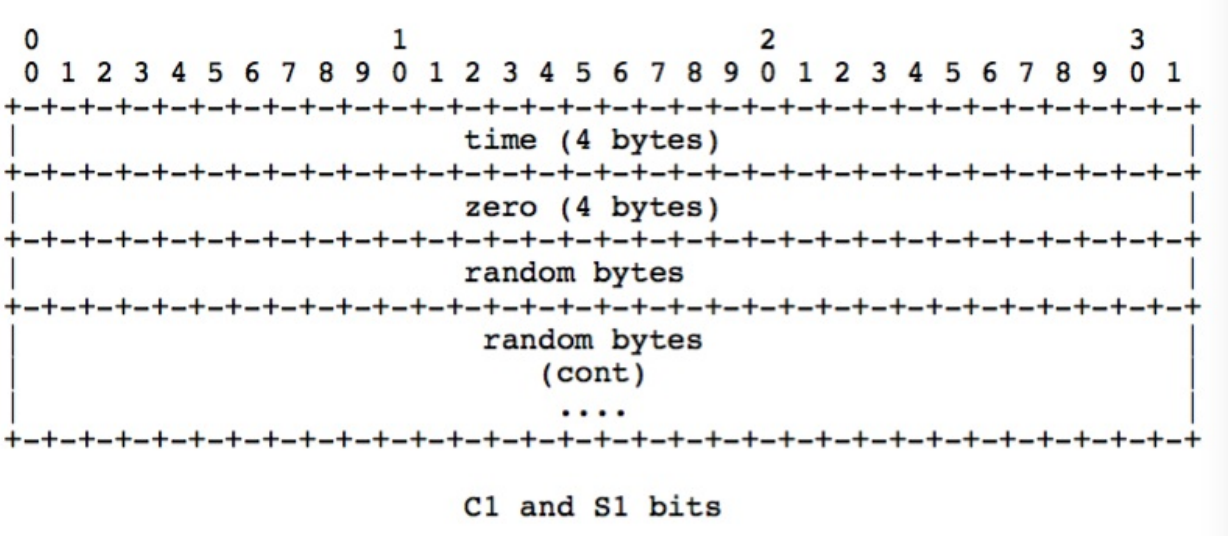


以下是 `C0` 和 `S0` 包的字段解释：

- 版本号（**8**位）： 在 `C0` 包中， 该字段表示客户端请求的RTMP版本。 在 `S0` 中， 该字段表示服务器选择的RTMP版本。 本规范所定义的版本是 `3` 。 可选值中， 0-2是早期版本所用的， 已被丢弃； 4-31保留在未来使用； 32-255不允许使用（为了区分其他以某一可见字符开始的文本协议）。 如果服务器不能识别客户端请求的版本， 应该返回 `3` ， 客户端可能选择降级到版本 `3` ， 也可能放弃握手。

5.2.3 `C1` 和 `S1` 格式

`C1` 和 `S1` 包的长度固定为1536字节， 包含以下字段：

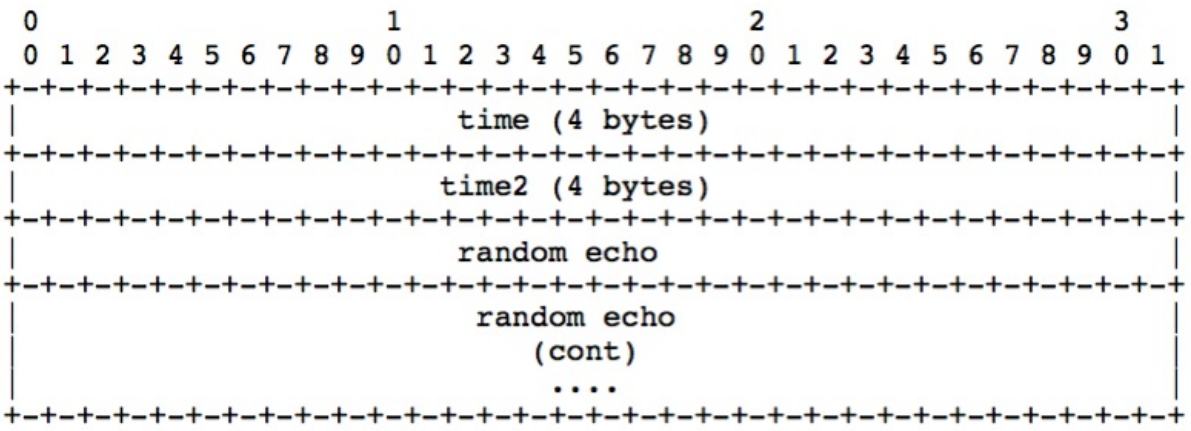


C1 and S1 bits

- 时间戳**(4**字节**)**：该字段承载一个时间戳，该时间戳应该作为发送端点所有后续块的时间戳起始时间。可以是0，也可以是其他的任意值。为了同步多个块流，端点可能会发送其他块流时间戳的当前值。
- 零值**(4**字节**)**：该字段所有值都必须为0。
- 随机数据**(1528**字节**)**：该字段可以是任意值。因为每个端点都需要区分自己和其他端点初始化的握手响应，所以此数据应该有充分的随机性，但是没必要使用加密安全的随机值或动态值。

5.2.4 C2 和 S2 格式

`C2` 和 `S2` 包的长度固定为1536字节，基本上分别是 `S1` 和 `C1` 的回显，包含以下字段：

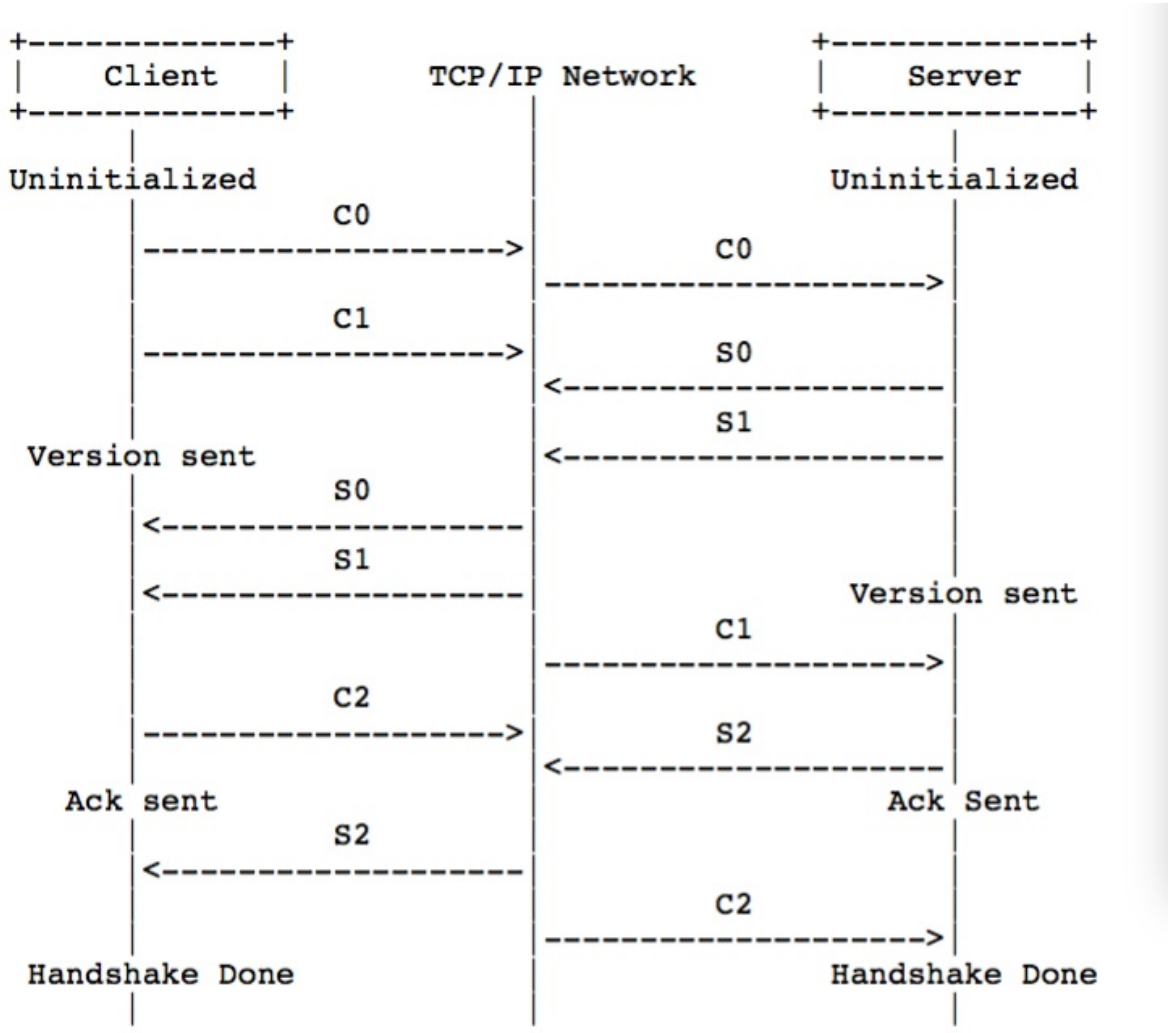


C2 and S2 bits

- 时间戳**(4**字节**)**：该字段必须包含对等端发来的时间戳（对 `C2` 来说是 `S1`，对 `S2` 来说是 `C1`。
- 时间**2**（**4**字节）：该字段必须包含先前发送的并被对端读取的包的时间戳。（`C1` 或 `S1`）。

- 随机数据回显(1528字节)：该字段必须包含对端发送过来的随机数据字段值（对 `C2` 来说是 `S1` ，对 `S2` 来说是 `C1` ）。任何一端都可以用时间戳和时间戳2两个字段值和当前时间戳来快速的估算带宽和延迟，但这样可能并不实用。

5.2.5 握手流程示意图



Pictorial Representation of Handshake

下面是对图中提到的状态的解释：

- **Uninitialized**: 未初始化状态。在该阶段发送协议版本，客户端和服务端都未初始化。客户端在 `C0` 包中发送RTMP协议版本，如果服务器支持此版本，服务器将在响应中发送 `S0` 和 `S1` ；如果不支持，服务器采用适当的行为作为响应，在RTMP规范中是终止连接。
- **Version Send**: 版本已发送状态。在未初始化状态之后客户端和服务端都进入版本已发送状态。客户端等待接收 `S1` 包，服务端等待接收 `C1` 包。收到所等待的包后，客户端发送 `C2` 包，服务端发送 `S2` 包。之后状态进入发送确认状态。
- **Ack Send**: 客户端和服务端等待接收 `S2` 和 `C2` 包，收到后进入握手完成状态。

- **Handshake Done**: 握手完成， 客户端和服务端开始交换消息。

5.3 分块

握手完成后，连接复用一个或多个块流，每个块流承载来自同一个消息流的同一类消息。创建的每个块都有一个唯一的块流ID，这些块通过网络进行传输。 在传输过程中，必须一个块发送完毕之后再发送下一个块。在接收端，将所有块根据块中的块流ID组装成消息。

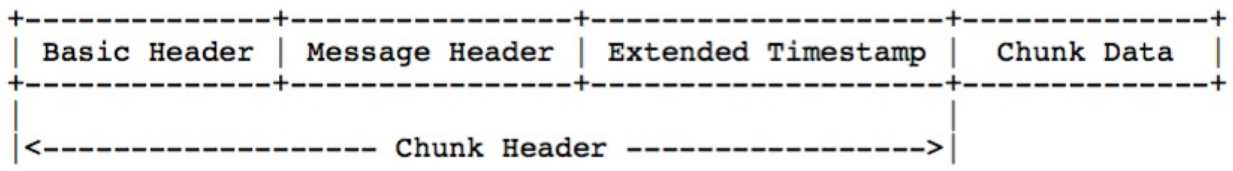
分块将上层协议的大消息分割成小的消息，保证大的低优先级消息（比如视频）不阻塞小的高优先级消息（比如音频或控制消息）。

分块还能降低消息发送的开销，它在块头中包含了压缩的原本需要在消息中所包含的信息。

块大小是可配置的，这个可以通过一个设置块大小控制消息进行设定修改，详细信息在5.4.1章描述。越大的块CPU使用率越低，但是在低带宽的情况下，大的写入会阻塞其他内容的写入；而小一些的块不适合高比特率的流。块大小在每个方向上保持独立。

5.3.1 块格式

块由块头和数据组成，块头包含3部分：基本头、消息头和扩展时间戳。



Chunk Format

- 基本头**(1-3**字节**)**: 该部分编码块流ID和块类型，块类型决定了消息头的编码格式。该部分的长度取决于块流ID，块流ID是一个变长字段。
- 消息头**(0,3,7**或**11**字节**)**: 该部分编码所发送消息的描述信息（无论是整个消息还是一部分）。该部分的长度取决于基本头中指定的块类型。
- 扩展时间戳**(0**或**4**字节**)**: 该部分只有在某些特殊情况下才会使用，是否使用取决与块消息头中的时间戳或时间戳增量，详细信息请参阅5.3.1.3。
- 块数据**(**变长**)**: 块承载的有效数据，长度最大为配置的块大小。

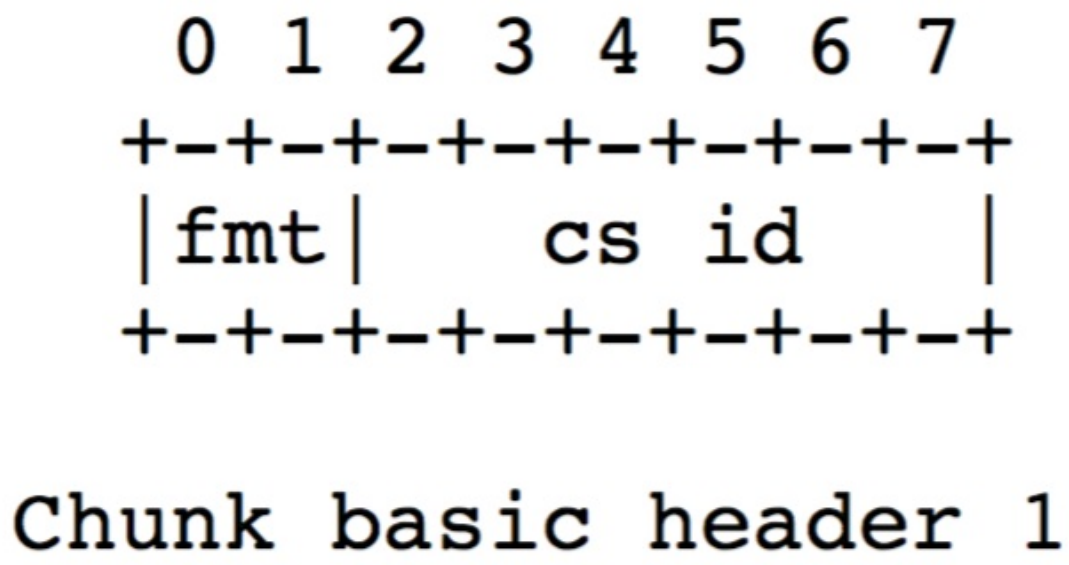
5.3.1.1 块基本头

块基本头编码块流ID和块类型（在下图中用fmt字段表示），块类型决定了消息头的编码格式，块基本头长度可能是1，2或3字节，这取决于块流ID的长度。

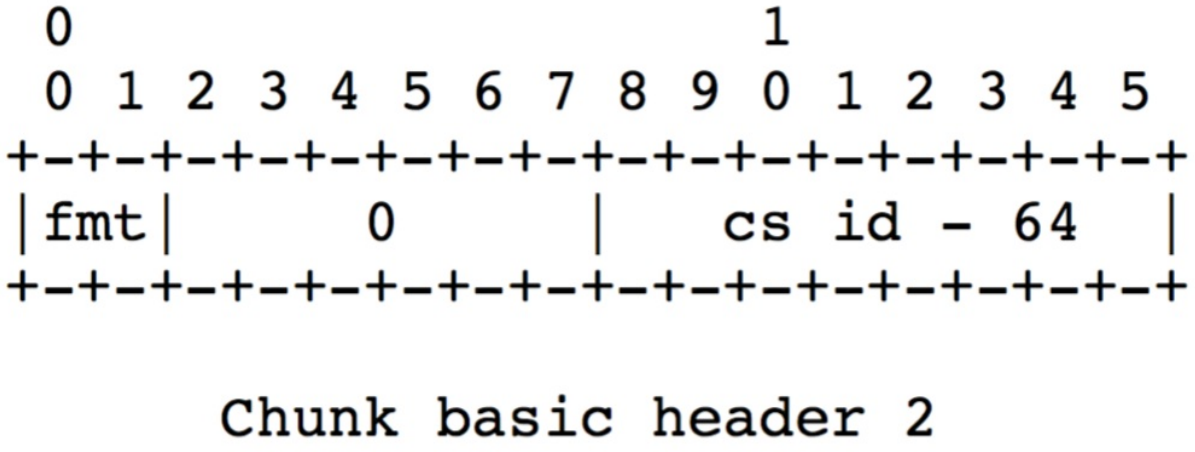
协议实现方应该用能够承载块流ID的最短表示法来表示块流ID。

RTMP最多支持65597个流，ID在3-65599范围内。ID0，1，2为保留值。0表示块基本头为2个字节，并且块流ID范围在64-319之间（第二个字节+64）；1表示块基本头为3个字节，并且ID范围在64-65599之间（第三个字节*256 + 第二个字节 + 64）；取值在3-63之间的ID表示完整的流ID。值2是为低版本协议保留的，用于协议控制消息和命令，第0-5位（不重要的）表示块流ID。

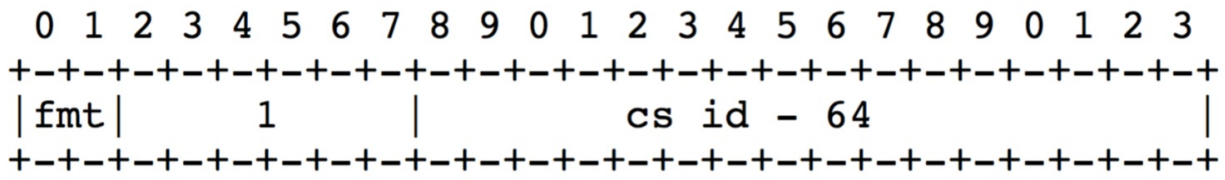
2-63范围内的块流ID可以用1个字节来编码。



64-319范围内的块流ID可以用2个字节来编码，块流ID为计算所得，公式为：第二个字节值 + 64.



64-65599范围内的块流ID可以用3个字节来编码，块流ID为计算所得，公式为：第三个字节值 *255 + 第二个字节值 + 64.



Chunk basic header 3

- **fmt**: 该字段表明了块消息头可以使用的4种格式的哪一种，每种块类型的消息头详情，在下一节描述。
- **cs id(6位)**: 该字段承载了完整的块流ID，取值在2-63之间。0，1两个值是保留值，用来表示块基本头是2字节还是3字节长度。
- **cs id - 64(8位或16位)**: 该字段承载块流ID，取值在64-63399之间。 例如：ID365应该在cs id字段位置取值为1，并在该16位字段位置取值301。

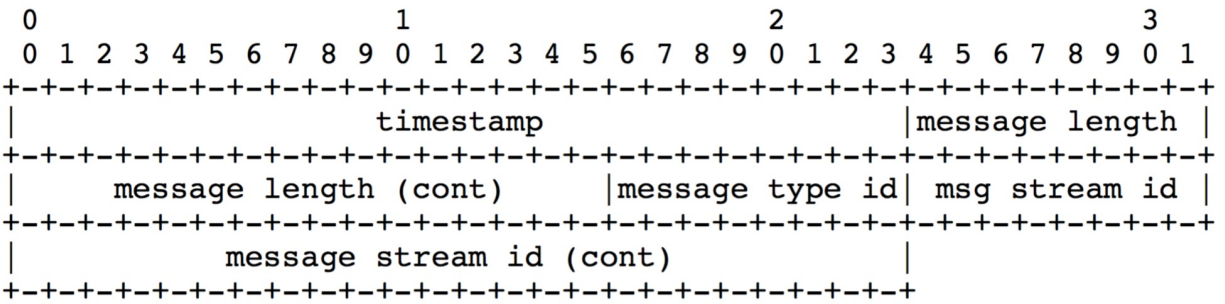
64-319范围内的块流ID可以用2字节长度来编码，也可以用3字节长度来编码。

5.3.1.2 块消息头

块消息头共有4种不同的格式，根据块基本头中的"fmt"字段值来选择。协议实现方应该用最紧凑的格式来表示块消息头。

5.3.1.2.1 块类型0

0类型的块消息头占11个字节长度，该类型必须用在一个块流的开头， 和每当块流时间戳后退的时候（例如向后搜索的操作）。



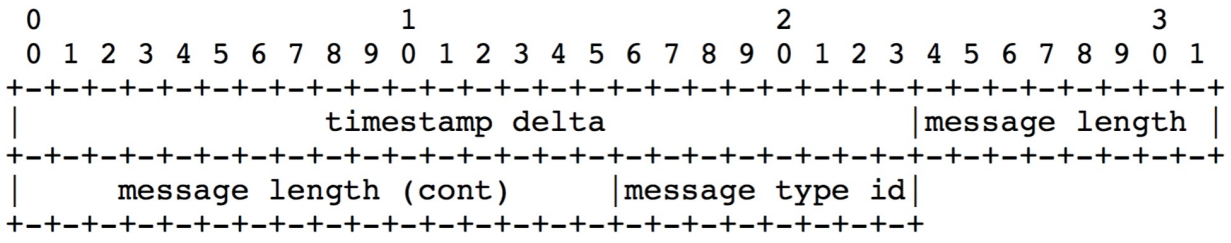
Chunk Message Header - Type 0

- **timestamp(3字节)**: 对于0类型的消息块， 消息的绝对时间戳在这里发送。 如果时间戳大于或等于16777215(0xFFFFFF)，改字段值必须为16777215，并且必须设置扩展时间戳来共同编码32位的时间戳。否则该字段就是完整的时间戳。

其他字段描述请参考**5.3.1.2.5**节

5.3.1.2.2 块类型1

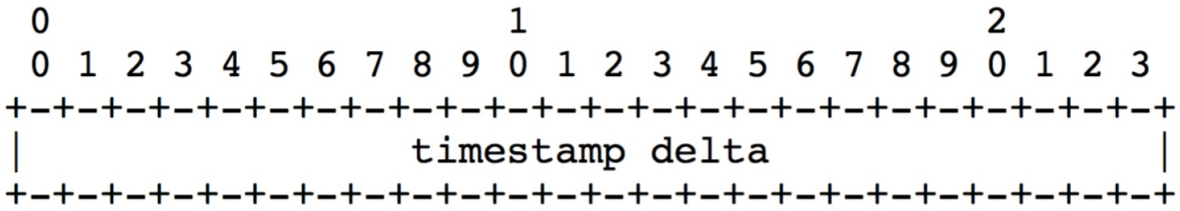
1类型的块消息头占用7个字节长度，不包含消息流ID，该块沿用上一个消息的消息流ID。对于传输大小可变消息的流（如多数视频格式），在发送第一个消息之后的每个消息，第一个块都应该使用该类型格式。



Chunk Message Header - Type 1

5.3.1.2.3 块类型2

2类型的块消息头占用3个字节长度，不包含消息流ID和消息长度，沿用上一个块的消息流ID和消息长度。对于传输固定大小消息的流（如音频和数据格式），在发送第一个消息之后的每一个消息，第一个块都应该使用该类型格式。



Chunk Message Header - Type 2

5.3.1.2.4 块类型3

3类型的块没有消息头，消息流ID、消息长度和时间戳增量都不指定，该类型的块都使用上一个块相同的块流ID。当一个消息被分割成块时，除了第一个块，其他块都应该使用该类型。示例请参考5.3.2.2节。由相同大小、消息流ID和时间间隔的消息组成的流，在类型2的块之后所有块都应该使用该类型格式。示例请参考5.3.2.1节。如果第一个消息和第二消息之间的时间增量与第一个消息的时间戳相同，则0类型的块之后可以马上发送3类型的块，而不必使用2类型的块来注册时间增量。如果类型3的块跟在类型0的块后面，那么3类型块的时间戳增量与0类型块的时间戳相同。

5.3.1.2.5 公共头字段

描述块消息头的其他字段：

timestamp delta(3字节): 时间戳增量。类型1和类型2的块包含此字段，表示前一个块的timestamp字段和当前块timestamp间的差值。 如果时间戳增量大于或等于16777215(0xFFFFFF)，该字段必须为16777215，并且必须设置扩展时间戳，来共同表示32位的时间戳增量，否则该字段值就是实际的时间戳增量。 **message length(3字节)**: 消息长度，类型0和类型1的块包含此字段，表示消息的长度。要注意的是，通常该长度与块负载长度并不相同。块负载长度除了最后一个块，都与块最大长度相同。 **message type id(3字节)**: 消息类型id，类型0和类型1的块包含此字段，表示消息的类型。 **message stream id(4字节)**: 消息流ID，类型0的块包含此字段，表示消息流ID。消息流ID以小字节序存储。通常，相同块流中的消息属于用一个消息流。虽然，不同的消息流复用相同的块流会导致消息头无法有效压缩，但是当一个消息流已关闭，才打开另外一个消息流，就可以通过发送一个新的0类型块来实现复用。

5.3.1.3 扩展时间戳

扩展时间戳用来辅助编码超过16777215(0xFFFFFF)的时间戳或时间戳增量，也就说0，1或2类型的块，无法用24位数字来表示时间戳或时间戳增量时，既0类型块的时间戳字段或1，2类型的时间戳增量字段值为16777215(0xFFFFFF)时。当最近的属于相同块流ID的0类型块、1类型块或2类型块有此字段时有此字段时，3类型块也应该有此字段。

5.3.2 示例

5.3.2.1 示例1

这是一个简单的音频流消息，这是示例示范了信息冗余。

	Message Stream ID	Message TYpe ID	Time	Length
Msg # 1	12345	8	1000	32
Msg # 2	12345	8	1020	32
Msg # 3	12345	8	1040	32
Msg # 4	12345	8	1060	32

Sample audio messages to be made into chunks

下图展示该消息流分割成的块。从3类型块开始了数据传输优化，之后的块只附加了一个字节。

	Chunk Stream ID	Chunk Type	Header Data	No.of Bytes After Header	Total No.of Bytes in the Chunk
Chunk#1	3	0	delta: 1000 length: 32, type: 8, stream ID: 12345 (11 bytes)	32	44
Chunk#2	3	2	20 (3 bytes)	32	36
Chunk#3	3	3	none (0 bytes)	32	33
Chunk#4	3	3	none (0 bytes)	32	33

Format of each of the chunks of audio messages

5.3.2.2 示例2

该示例展示了一个超过128字节长度的消息，消息被分割成了数个块。

	Message Stream ID	Message TYpe ID	Time	Length
Msg # 1	12346	9 (video)	1000	307

Sample Message to be broken to chunks

下图是被分割成的块

	Chunk Stream ID	Chunk Type	Header Data	No. of Bytes after Header	Total No. of bytes in the chunk
Chunk#1	4	0	delta: 1000 length: 307 type: 9, stream ID: 12346 (11 bytes)	128	140
Chunk#2	4	3	none (0 bytes)	128	129
Chunk#3	4	3	none (0 bytes)	51	52

Format of each of the chunks

第一个块的头信息指明了消息总大小为307字节。

注意这两个示例，3类型块可以在两种情况下使用。第一种情况是指明消息还未结束；另一种情况是指明新消息开始时，可以根据当前状态信息推导出新消息的头信息。

5.4 协议控制消息

RTMP块流用消息类型ID 1，2，3，5和6来作为协议控制消息，这些消息包含RTMP块流协议所需要的信息。

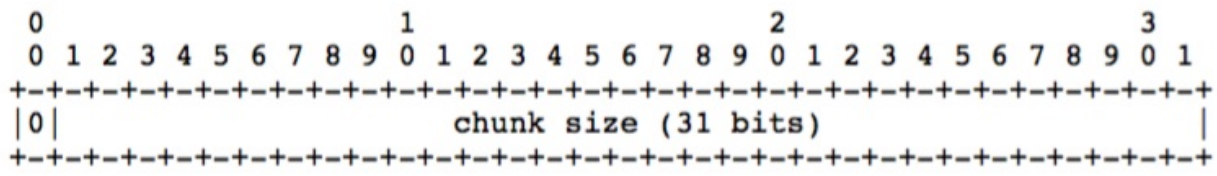
这些协议控制消息必须用 `0` 作为消息流ID(控制流ID)，并在ID为2的块流中发送。协议控制消息收到后立即生效，它们的时间戳信息是被忽略的。

5.4.1 设置块大小(`1`)

协议控制消息 `1` ，设置块大小，用于通知另一端新的最大块大小。

最大块大小默认为128字节，客户端或服务端可以修改此值，并用该消息通知另一端。例如，假设一个客户端想要发送131字节的音频数据，而最大块大小为128。在这种情况下，客户端可以向服务端发送该消息，通知它最大块大小被设置为了131字节。这样客户端只用一个块就可以发送这些音频数据。

最大块大小不能小于1字节，通常应该不低于128字节。每个方向上的最大块大小是独立的。

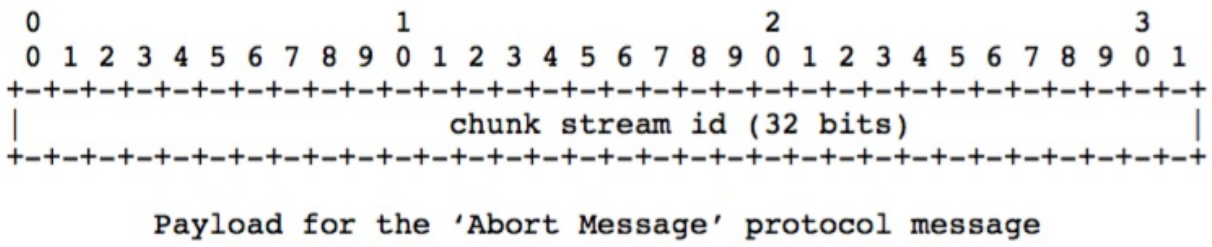


Payload for the 'Set Chunk Size' protocol message

0: 该位必须为0 **chunk size(31位)**: 该字段以字节形式保存新的最大块大小，该值将用于后续的所有块的发送，知道收到新的通知。该值可取值范围为1-2147483647(0x7FFFFFFF)，但是所有大于1677215(0xFFFFFF)的值都是等同的，因为任何块不可能比消息大，而消息长度不能大于16777215字节。

5.4.2 终止消息(`2`)

协议控制消息 `2` ，终止消息，通知正在等待消息后续块的另一端，可以丢弃通过指定块流接收到的部分数据，块流ID为该消息有效负载。应用可能在关闭的时候发送该消息，用来表明后面的消息没有必要继续处理了。

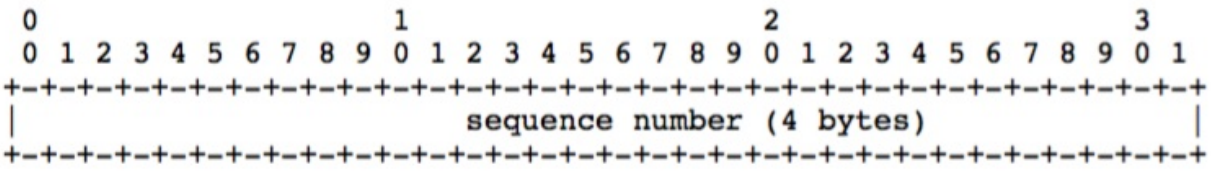


Payload for the 'Abort Message' protocol message

chunk stream id(32字节): 该字段承载可以丢弃当前消息的块流ID。

5.4.3 确认消息(3)

客户端或服务器在收到数据后必须向对端发送与视窗大小相等的确认消息。视窗大小是发送端发送的最大字节数，无论有没有收到接收端发送的确认消息。该消息代表序列号，是到当前时间为止接收到的字节总数。

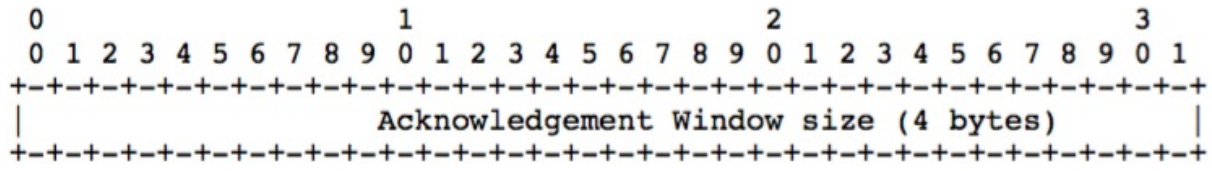


Payload for the 'Acknowledgement' protocol message

sequence number(32字节): 到当前时间位置接收到的字节总数

5.4.4 视窗大小确认(5)

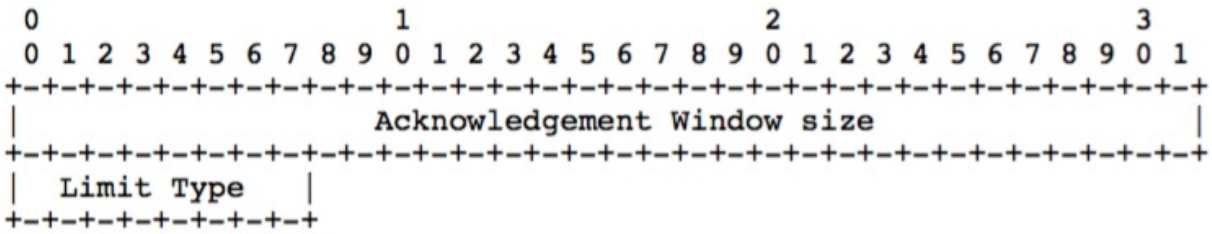
客户端或服务端发送该消息来通知对端发送确认消息所使用的视窗大小，并等待接收端发送确认消息。接收端在接收到视窗大小后必须发送确认消息(5.4.3)。



Payload for the 'Window Acknowledgement Size' protocol message

5.4.5 设置对等带宽(6)

客户端或服务端发送该消息来限制对端的输出带宽。接收端收到消息后，通过将已发送但尚未被确认的数据总数限制为该消息指定的视窗大小，来实现限制输出带宽的目的。如果视窗大小与上一个视窗大小不同，则该消息的接收端应该向该消息的发送端发送视窗大小确认消息。



Payload for the 'Set Peer Bandwidth' protocol message

Limit Type（限制类型）有以下可选值：

- **0 - Hard**: 消息接收端应该将输出带宽限制为指定视窗大小
- **1 - Soft**: 消息接收端应该将输出贷款限制为指定视窗大小和当前视窗大小中较小的值
- **2 - Dynamic**: 如果上一个消息的限制类型为Hard，则该消息同样为Hard，否则抛弃该消息。

6 RTMP消息格式

本章描述RTMP消息遵循底层协议（比如RTMP块流）在网络中断之间传输时的消息格式。

虽然RTMP被设计成使用RTMP块流传输，但是它也可以使用其他传输协议来发送消息。RTMP块流协议和RTMP协议配合时，非常适合音视频应用，包括单播、一对多实时直播、视频点播和视频会议等。

6.1 RTMP消息格式

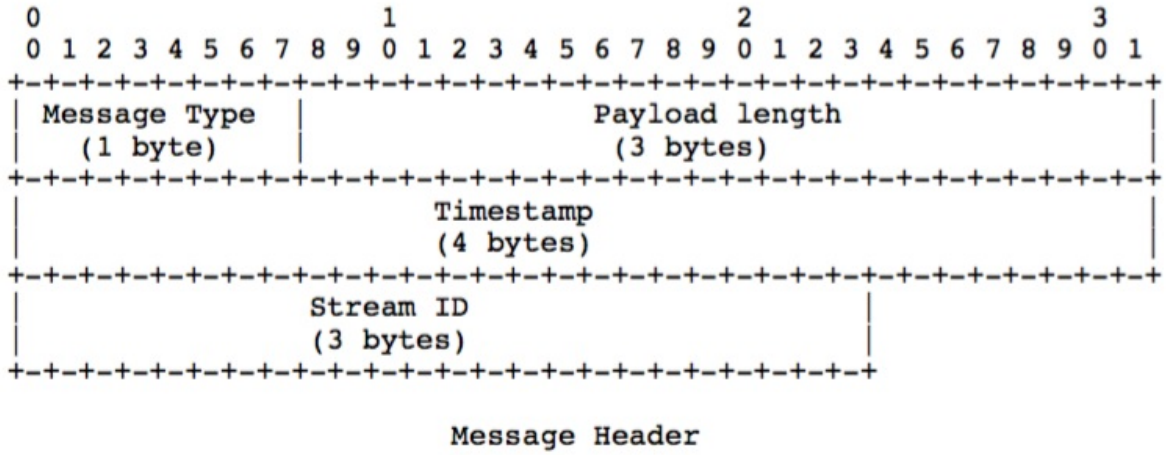
服务端和客户端通过在网络上发送RTMP消息实现之间的交互，消息包括但不限于音频、视频、数据。

RTMP消息包含两部分，1个消息头和有效负载

6.1.1 消息头

消息头包含以下信息：

- **Message Type**: 消息类型，占用1个字节。1-6的消息类型ID是为协议控制消息保留的。
- **Length**: 有效负载的字节数，占用3个字节。该字段是用大字节序表示的。
- **Timestamp**: 时间戳，占用4个字节，用大字节序表示。
- **Message Stream Id**: 消息流ID，标识消息所使用的流，用大字节序表示。



Message Header

6.1.2 消息有效负载

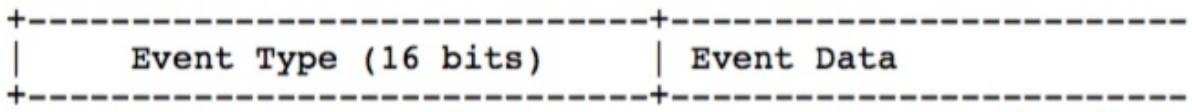
消息的另一部分就是有效负载，也是消息包含的实际数据，可以是音频样本或者压缩的视频数据。负载的格式不再本文档的讨论范围之内。

6.2 用户控制消息(4)

RTMP协议将消息类型 4 作为用户控制消息ID，这些消息包含RTMP流所需的必要信息。消息类型1，2，3，5和6由RTMP块流协议使用。

用户控制消息应该使用ID为0的消息流（控制流），并且通过RTMP块流传输时使用ID为2的块流。用户控制消息收到后立即生效，它们的时间戳信息会被忽略。

客户端或服务端通过发送该消息告知对方用户控制事件。该消息携带事件类型和事件数据两部分。



Payload for the 'User Control' protocol message

开头的2个字节用于指定事件类型，紧跟着是事件数据。事件数据字段长度可变，但是如果用RTMP块流传输，则消息总长度不能超过最大块大小，以使消息可以使用一个单独的块进行传输。

事件类型和对应的事件数据格式，在7.1.7章节详细介绍。

7 RTMP指令消息

本章描述客户端和服务端进行交互的消息类型和指令类型。

各种类型的消息在客户端和服务端之间进行交换，包括用于发送音频数据的音频消息，用于发送视频数据的视频消息，用于发送任意用户数据的数据消息，共享对象消息和指令消息等。共享对象消息为管理分布与不同客户端和相同服务器的共享数据提供了规范途径。指令消息携带客户端与服务端之间的AMF编码指令，客户端或服务端也可以通过指令消息来实现远程过程调用（RPC)。

7.1 消息类型

客户端和服务端通过在网络上发送消息来实现交互，消息可以是任意类型，包括但不限于音频消息、视频消息、指令消息、共享对象消息、数据消息和用户控制消息。

7.1.1 指令消息(20,17)

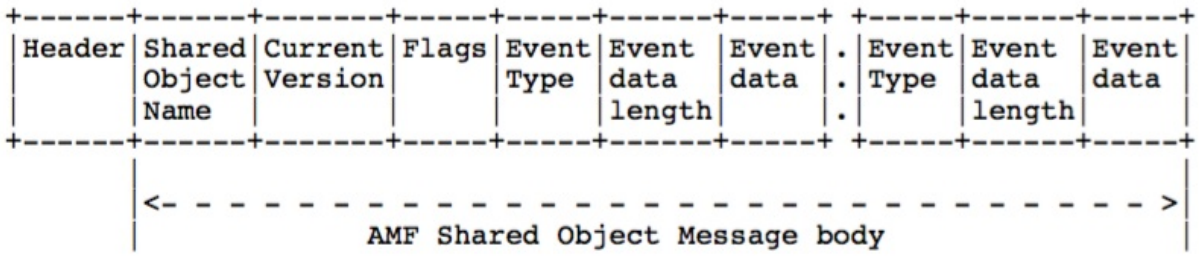
指令消息在客户端和服务端之间传递AMF编码的指令，消息类型20代表AMF0编码，消息类型17代表AMF3编码。发送这些消息来完成连接、创建流、发布、播放、暂停等操作。像状态、结果这样的指令消息，用于通知发送方请求的指令状态。一条指令消息由指令名、事务ID和包含相关参数的指令对象。客户端或服务端还可以通过指令消息来实现远程过程调用(RPC)。

7.1.2 数据消息(18,15)

客户端或服务端通过该消息来发送元数据或其他用户数据。元数据包括数据(音频、视频)的创建时间、时长、主题等详细信息。消息类型18代表AMF0编码，消息类型15代表AMF3编码。

7.1.3 共享对象消息(19,16)

共享对象是一个在多个客户端、示例之间进行同步的Flash对象(键值对集合)。消息类型19代表AMF0编码，消息类型16代表AMF3编码。每个消息都可以包含多个事件。



The shared object message format

支持以下事件类型：

名称	取值	描述
创建	1	客户端向服务端发送，请求创建指定名称的共享对象
释放	2	客户端通知服务端，共享对象已在本地删除
请求更新	3	客户端请求修改共享对象的属性值
更新	4	服务端向除请求发送方外的其他客户端发送，通知其有属性的值发生了变化。
成功	5	“请求更新”事件被接受后，服务端向发送请求的客户端回复此事件
发送消息	6	客户端向服务端发送此事件，来广播一个消息。服务端收到此事件后向所有客户端广播一条消息，包括请求方客户端
状态	7	服务端发送此事件来通知客户端错误信息
清除	8	服务端向客户端发送此事件，通知客户端清除一个共享对象。服务端在回复客户端的“创建”事件时也会发送此事件
移除	9	服务端发送此事件，使客户端删除一个插槽
请求移除	10	客户端删除一个插槽时发送此事件
创建成功	11	当连接成功时服务端向客户端发送此事件

7.1.4 音频消息(8)

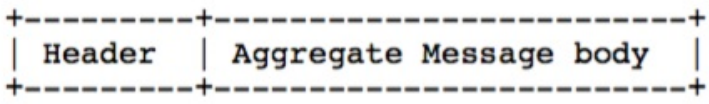
客户端或服务端通过发送此消息来发送音频数据给对方，消息类型8是为音频消息预留的。

7.1.5 视频消息(9)

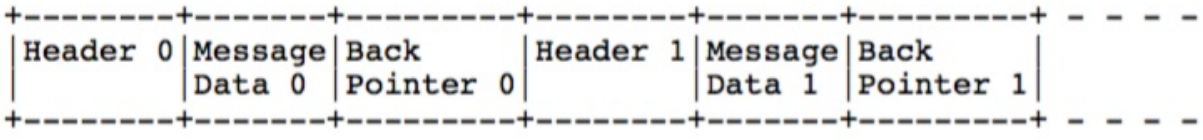
客户端或服务端通过发送此消息来发送视频数据给对方，消息类型9是为视频消息预留的。

7.1.6 组合消息(22)

组合消息，是一个消息包含多个子RTMP消息，子消息符合6.1章节所描述的消息格式。消息类型22用于组合消息。



The Aggregate Message format



The Aggregate Message body format

组合消息的消息流ID会覆盖其中子消息的消息流ID。

组合消息的时间戳和其中第一个子消息的时间戳的差值，是用来将所有子消息的时间戳重整为流时间标尺的位移量。位移量会加到每一个子消息的时间戳上来换算出正常的流时间。第一个子消息的时间戳应该与组合消息的时间戳相同，所以位移量应该为0。

Back Pointer(反向指针)包含前一个消息的长度（包括消息头），这样就能符合flv文件格式，并用于进行后退操作。

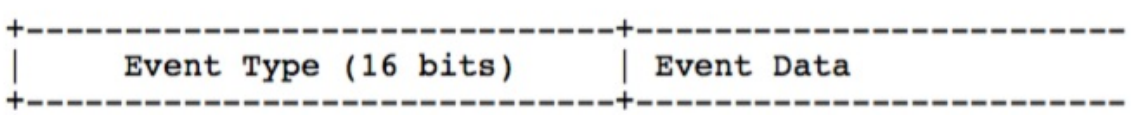
使用组合消息有以下好处：

- 块流协议中，一个块最多只能发送一个消息，这样就使用组合消息，加大块大小，从而降低发送的块数量。
- 子消息在内存中连续存放，这样系统调用网络发送数据的性能更高。

7.1.7 用户控制消息事件

客户端或服务器通过该消息发送用户控制事件，用户控制消息格式请参考6.2章节。

> 用户控制消息格式



Payload for the 'User Control' protocol message

用户控制消息支持以下事件：

名称	取值	描述
流开始	0	服务端发送该事件，用来通知客户端一个流已经可以用来通讯了。默认情况下，该事件是在收到客户端连接指令并成功处理后发送的第一个事件。事件的数据使用4个字节来表示可用的流的ID
流结束	1	服务端发送该事件，用来通知客户端其在流中请求的回放数据已经结束了。如果没有额外的指令，将不会再发送任何数据，而客户端会丢弃之后从该流接收到的消息。事件数据使用4个字节来表示回放完成的流的ID。
流枯竭	2	服务端发送该事件，用来通知客户端流中已经没有更多的数据了。如果服务端在一定时间后没有探测到更多数据，它就可以通知所有订阅该流的客户端，流已经枯竭。事件数据用4个字节来表示枯竭的流的ID。
设置缓冲区大小	3	客户端发送该事件，用来告知服务端用来缓存流中数据的缓冲区大小(单位毫秒)。该事件在服务端开始处理流数据之前发送。事件数据中，前4个字节用来表示流ID，之后的4个字节用来表示缓冲区大小（单位毫秒）。
流已录制	4	服务端发送该事件，用来通知客户端指定流是一个录制流。事件数据用4个字节表示录制流的ID
ping请求	6	服务端发送该事件，用来探测客户端是否处于可达状态。事件数据是一个4字节的时间戳，表示服务端分发该事件时的服务器本地时间。客户端收到后用 `ping响应` 回复服务端。
ping响应	7	客户端用该事件回复服务端的 `ping请求` ，事件数据为收到的 `ping请求` 中携带的4字节的时间戳。

7.2 指令类型