专题报告：RTMP协议

# 

目录

[专题报告：RTMP协议 1](#_Toc378337493)

[一：什么是rtmp 2](#_Toc378337494)

[二：RTMP消息格式 3](#_Toc378337495)

[三：RTMP握手过程 7](#_Toc378337496)

[三．协议控制消息 15](#_Toc378337497)

[四：消息交换的例子 17](#_Toc378337498)

写在前面红色字体是重点必读，蓝色字体是分点便于区分，绿色字体是次分点便于区分

# 

# 一：什么是rtmp

**RTMP协议**

　　Real Time Messaging Protocol（实时消息传送协议协议)实时消息传送协议是Adobe Systems公司为Flash播放器和服务器之间音频、视频和数据传输开发的私有协议。它有三种变种：

　　1)工作在TCP之上的明文协议，使用端口1935；

　　2)RTMPT封装在HTTP请求之中，可穿越防火墙；

　　3)RTMPS类似RTMPT，但使用的是HTTPS连接；

　　介绍:

* + RTMP协议是被Flash用于对象,视频,音频的传输.该协议建立在TCP协议或者轮询HTTP协议之上.
  + RTMP协议就像一个用来装数据包的容器,这些数据可以是AMF格式的数据,也可以是FLV中的视/音频数据.
  + 一个单一的连接可以通过不同的通道传输多路网络流.这些通道中的包都是按照固定大小的包传输的.

RTMP中定义了两种通信单元：消息（message）和消息块（chunk）.RTMP消息是协议中实现各种流媒体控制和应用的基本逻辑信息单元，消息从种类上可以分为协议控制消息、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、发送用户数据的数据消息、共享对象消息以及命令消息，属于相同逻辑通道的消息组成一个消息流，这个逻辑通道通过消息格式中的“消息流ID”字段来标识。

作为应用层协议，RTMP协议架构在TCP层之上，但RTMP消息并不是直接封装在TCP中，而是通过一个被称为消息块的封装单元进行传输。消息在网络上发送之前往往要分割成多个较少的部分，这些较小的部分就是消息块，属于不同消息流的消息块可以在网络上交叉发送。这样做可以保证各个消息流中的高优先级消息块能够严格按照时间顺序达到通信的对端。比如某个较长消息的实时性要求比较低，如果不进行消息块处理，等长消息都发送完毕后再发送实时性要求高的短消息，则会对流媒体的播放质量造成影响。

# 二：RTMP消息格式



RTMP消息包含两个部分，包头和有效负载。包头包含时间戳、消息长度、消息类型以及消息流ID。有效负载是包含在消息中的实际数据，例如：它可以是一些音频样本或者压缩的视频数据。

(1)、时间戳delta（timestamp delta）：消息的时间戳，占3个字节

(2)、消息长度（message length）：3个字节。消息的有效负载的长度，如果消息头不能被省略，他应该包含在长度中，这个字段在消息块包头中。注意这一般和消息块的有效负载长度是不一样的。消息块的有效负载长度是除了最后一个消息块外其他所有消息块都取消息块大小的最大值（通常默认为128字节）。

(3)、消息类型 id（message type id）：1个字节。协议控制消息的类型字段的范围是被保留的，这些传播信息的消息被RTMP消息块和高层协议处理，所有其他的类型ID可被高层协议使用，被RTMP消息块当做不透明的值

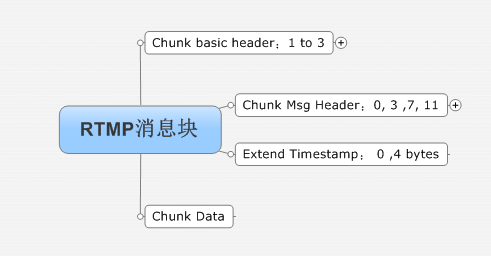
(4)消息流ID（message stream id）：4个字节。消息流ID可以是任意值，不同的消息复用成同一消息块流以及解复用都是基于消息流ID。

**消息分块**

(1)、在握手以后，连接一个或者多个复用消息块，每个消息块流负载来自消息流中同一类的消息，每一个消息块的产生配有一个唯一的ID，被称为消息块流ID，消息块流在网络上传输。传输的时候每一个消息块在下一个消息块之前必须完全被发送，当接收结束，消息块基于消息块流ID被组装成消息。消息分块允许高层协议的大的消息被分割成小的消息，消息分块也允许小的消息被传送时带有小的包头，消息块的包头包含了信息的压缩表示，否则信息必须在消息自身中包含。

(2)、消息块的字节顺序：  
      RTMP的字节序和大多数网络协议一样是大端序，也有一些字段是小端序的，不过都有特殊的说明。  
1).RTMP的head组成

         RTMP的head在协议中的表现形式是chunk head，前面已经说到一个Message+ head可以分成一个和多个chunk，为了区分这些chunk，肯定是需要一个chunk head的，具体的实现就把Message  head的信息和chunk head的信息合并在一起以chunk head的形式表现。

        一个完整的chunk的组成如下图所示  
  
                                                        

Chunk basic header：1 to 3

该字段包含chunk的stream ID和 type 。chunk的Type决定了消息头的编码方式。该字段的长度完全依赖于stream ID，该字段是一个可变长的字段。

chunk basic head的长度为1~3个字节，具体长度主要是依赖chunk stream ID的长度，所谓chunk stream ID是flash server用来管理连接的客户端的信令交互的标识，协议最大支持65597个streamID 从3~65599。ID 0，1，2为协议保留，0代表ID是64~319（第二个byte + 64）；1代表chunk stream ID为64~65599（（第三个byte）\* 256 + 第二个byte + 64）（小端表示）；2代表该消息为低层的协议（在RTMP协议中控制信令的chunk stream ID都是2）。3~63的chunk stream ID就是该byte的值。没有附加的字段来标识chunk stream ID。  
目前chunk basic head的长度一般为1个字节。这一个字节由两部分组成  
  
fmt占两个bit用来标识紧跟其后的chunk Msg Header的长度，cs id占六个bit。

两位的fmt取值为 0~3，分别代表的意义如下：  
case 0：chunk Msg Header长度为11；  
case 1：chunk Msg Header长度为7；  
case 2：chunk Msg Header长度为3；  
case 3：chunk Msg Header长度为0；所以 只有一个字节的chunk basic header取值为 chunk basic header = (fmt << 6) | (cs id).

其他的两种的字节情况分别为：





Chunk Msg Header：0, 3 ,7, 11

该字段包含了将要发送的消息的信息（或者是一部分，一个消息拆成多个chunk的情况下是一部分）该字段的长度由chunk basic header中的fmt决定。共有四种类型

类型0：11个字节的完整包头，如图所示



Timestamp：3bytes

对于type 0的chunk，绝对时间戳在这里表示，如果时间戳值大于等于0xffffff（16777215），该值必须是0xffffff,且时间戳扩展字段必须发送，其他情况没有要求。

message length：3bytes

Message的长度，注意这里的长度并不是跟随chunk head其后的chunk data（Payload）的长度，而是前文提到的一条信令或者一帧视频数据或音频数据的长度。前文提到过信令或者媒体数据都称之为Message，一条Message可以分为一条或者多条chunk。

message type  id：1byte

message stream id：4bytes

message stream id的字节序是小端序，这个字段是为了解复用而设计的

类型1：7 个字节 的chunk head

该类型不包含stream ID，该chunk的streamID和前一个chunk的stream ID是相同的，变长的消息，例如视频流格式，在第一个新的chunk以后使用这种类型，注意其中时间戳部分是相对时间，为与上一个绝对时间之间的差值 如图所示：

 类型2：3 字节的chunk head

该类型既不包含stream ID 也不包含消息长度，这种类型用于stream ID和前一个chunk相同，且有固定长度的信息，例如音频流格式，在第一个新的chunk以后使用该类型。如图所示：



  类型3： 0字节的chunk head:

这种类型的chunk从前一个chunk得到值信息，当一个单个消息拆成多个chunk时，这些chunk除了第一个以外，其他的都应该使用这种类型。



Extend Timestamp： 0 ,4 bytes

该字段发送的时候必须是正常的时间戳设置成0xffffff时，当正常时间戳不为0xffffff时，该字段不发送。当时间戳比0xffffff小该字段不发送，当时间戳比0xffffff大时该字段必须发送，且正常时间戳设置成0xffffff。

Chunk Data实际数据（Payload），可以是信令，也可以是媒体数据。

# 三：RTMP握手过程

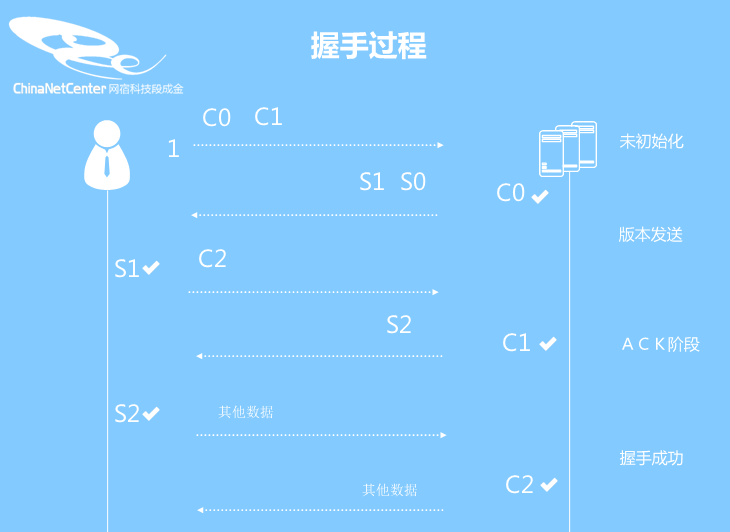
**握手**

一个RTMP通信以握手开始，握手不像其他的协议，他包含三个固定长度的消息块。

客户（初始化通信的终端）和服务器每放发送同样的三个消息块，说明一下，被客户段发送的消息块被指定为C0，C1，C2，被服务器端发送的消息被指定为S0，S1，S2。

握手的顺序

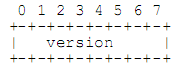
握手以客户端发送C0和C1消息块位开始，客户端必须等到S1到达在发送C2。客户端必须等到S2接收到才可以发送其他的数据；服务端必须等到C0到达才发送S0和S1，在C1之后也会等待。服务端必须等到C1到达才发送S2，服务端必须等到C2到达后才发送其他数据。



|  |  |
| --- | --- |
| 阶段 | 描述 |
| 未初始化 | 协议版本在这个阶段中被发送，客户端个服务端没有初始化，客户端在C0中发送协议版本，如果服务端支持版本，就在回应中发送S0和S1，如果不能，服务端通过适当的行为进行恢复，在RTMP中，这个行为是连接结束 |
| 版本发送 | 在未初始化阶段以后客户端个服务端在版本发送阶段，客户端等待S1包，服务端等待C1包，当接收到想要的包，客户端发送C2，服务端发送S2，此时阶段变成了ACK的发送。 |
| ACK发送 | 客户端和服务端分别等待S2和C2 |
| 握手完成 | 客户端和服务交换消息。 |

C0和S0格式

C0和S0包长8个字节

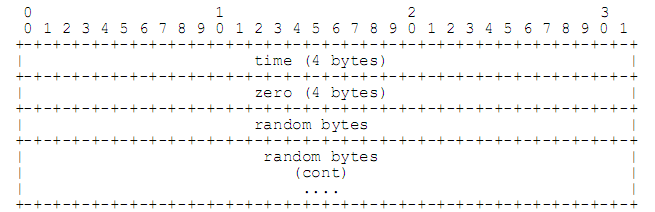


版本：8比特

在C0中，这个字段识别客户端需求的RTMP的版本，在S0中，这个字段识别服务器端选择的RTMP的版本，被定义的是版本3，0到2是被早前的版本使用的，4到31保留被用作未来的用途，32到255还没有被允许。不能区分客户的请求的版本的服务应该以3返回，客户端或许会选择3一下的版本，或者放弃握手。

C1和S1的格式：

C1和S1的数据包有1536个字节长，由以下几个字段组成。



时间：4个字节

这个字段包含时间戳，被当做以后消息块从终端发送的时间点，也许是0，或者一些任意的值，用来同步多重的消息块流，终端或许希望发送其他消息块流的时间戳的当前值。

0：4个字节

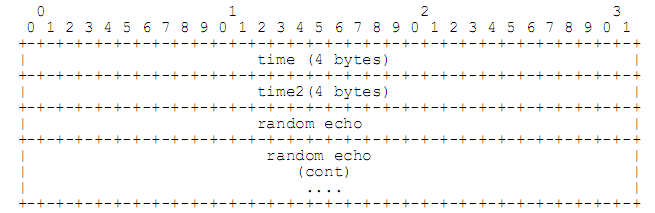
这个字段必须全0。

随机数据：1528个字节

这个字段可以包含任何任意的值，因为每个终端必须区分自己初始化的握手的返回数据和对方初始化的握手的返回数据，这个数据应该发送一些随机数。但是没有必要密码保护随机数和动态值。

C2和S2的格式

C2和S2数据有1536字节长度，近似S1和C1的回声，由一下几个字段组成。



时间：4个字节

这个字段必须包含由每方发送的S1（对应C2）或者C1（对应S2）的时间戳.

时间2：4个字节

这个字段必须包含先前的由每一方发送数据包（S1或者C1）被读到的时间戳。

随机返回：1528个字节

这个字段必须包含在每方发送的S1（对应C2）或者S2（对应C1）的随机数据字段。每一方可以利用时间和时间2字段和当前时间戳组成作为连接的带宽或者延迟的评估。但是似乎没有用。

**命令消息**

RTMP服务器和客户端通过命令消息传递双方的命令信息，这些命令信息均采用AMF编码方式。AMF编码方式是Adobe公司开发出的一种通信协议，它采用二进制形式，为基于Flash的播放器和远端服务器提供一种轻量级的、高效能够的通信方式。目前AMF已经从AMF0版本发展到了AMF3版本。通过命令消息，用户可以执行连接、创造流、发布、播放、暂停的操作，接受方也可以通过命令消息向操作方发送返回命令的状态。发送方还可以通过命令消息来向接收方请求远程成语调用

|  |  |
| --- | --- |
| 命令消息 | 消息描述 |
| Connect消息 | 客户端向服务器发送连接命令来请求与服务器的一个应用程序建立连接 |
| Call命令消息 | 发送方可以通过这个命令进行接收方的远端程序调用。调用命令的参数中包含了被调用程序的名称 |
| CreatStream命令消息 | 客户端向服务器发送这个命令来创造一条用于消息通信的逻辑通道。音频、视频和元数据的发布都通过这条流通道来承载 |
| Play命令消息 | 客户端发送这个命令给服务器请求播放一个流。如果你创建了一个动态播放列表，各个节目需要不断切换和播放，这是需要调用play命令多次，并在命令的reset属性传递false属性。相反，如果你只想播放一个列表中的特定节目，可以在试用这个命令时将reset属性设置为true |
| Play2命令消息 | 与play命令不一样，使用play2命令可以在无须改变播放内容时间线的情况下将流切换成另外一个不同的码率。服务器会针对同一播放内容的几个不同码率维持几个文件，客户端可以通过play2来请求不同码率 |
| DeleteStream命令消息 | 当netstream对象被破坏以后发送这个命令 |
| ReceiveVideo命令消息 | 这个命令消息用来通知服务器是否给客户端发送视频流 |
| Publish命令消息 | 客户端向服务器发送这个命令消息来表示要发布一个命名的流 |
| Seek命令消息 | 客户端可以通过seek命令告诉服务器从某个特定时间点开始发送某个媒体文件或者播放列表流数据。如果服务器操作成功，则向客户端返回成功的通知，否则返回错误消息 |
| Pause命令消息 | 客户端可以通过这个命令告诉服务器暂停或者继续播放 |

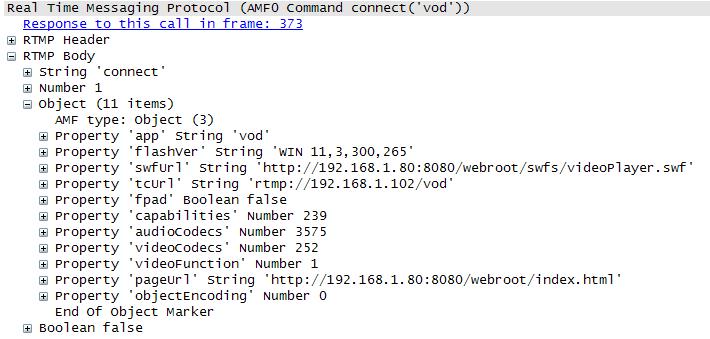
 下面是在连接命令的命令对象中使用的名值对的描述：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 类型 | 描述 | 示例值 |
| App | 字符串 | 客户端要连接到的服务应用名 | Testapp |
| Flashver | 字符串 | Flash播放器版本。和应用文档中getversion（）函数返回的字符串相同。 | FMSc/1.0 |
| SwfUrl | 字符串 | 发起连接的swf文件的url | file://C:/ FlvPlayer.swf |
| TcUrl | 字符串 | 服务url。有下列的格式。protocol://servername:port/appName/appInstance | rtmp://localhost::1935/testapp/instance1 |
| fpad | 布尔值 | 是否使用代理 | true or false |
| audioCodecs | 数字 | 指示客户端支持的音频编解码器 | SUPPORT\_SND\_MP3 |
| videoCodecs | 数字 | 指示支持的视频编解码器 | SUPPORT\_VID\_SORENSON |
| pageUrl | 字符串 | SWF文件被加载的页面的Url | http:// somehost/sample.html |
| objectEncoding | 数字 | AMF编码方法 | kAMF3 |

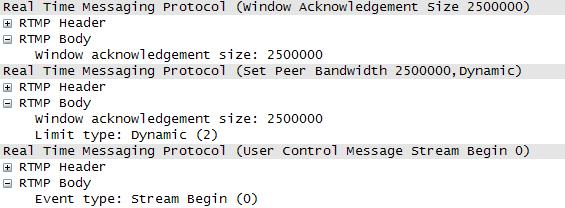
Connect命令的流程控制描述如下：

1. 客户端发送连接命令给服务器，请求与服务器上的一个应用程序建立连接

Connect

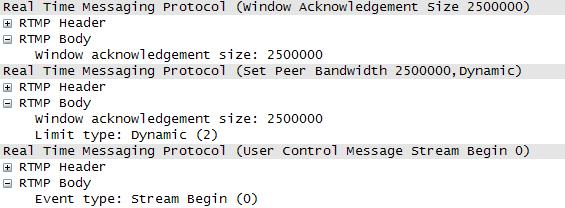


windows acknowledgement size- set peer bandwidth- user control message(begin 0)- result(connect response)

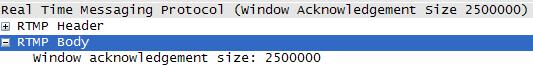


1. 服务器收到连接命令后，服务器发送窗口确认大小的消息给客户端。服务器也与连接命令中提到的应用程序建立连接。
2. 服务器发送设置带宽的消息给客户端。

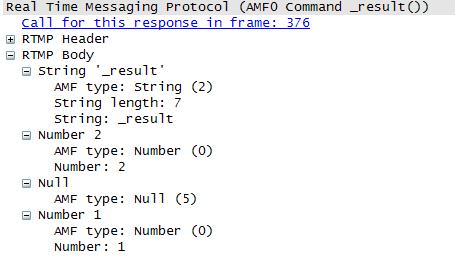
windows acknowledgement size- set peer bandwidth- user control message(begin 0)- result(connect response)



1. 客户端收到设置带宽的消息后，发送窗口确认大小的消息给服务器。

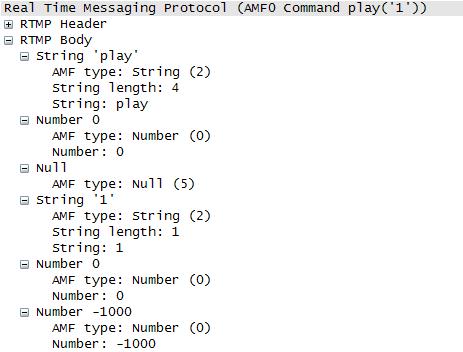


1. 服务器在发送另一个用户控制消息给客户端，比如streambegin
2. 服务器载发送一个控制命令通知客户端连接状态。这个消息也包括了一些特殊属性，比如Flash流媒体服务器的版本、能力等。



Play命令的流程描述如下：

客户端在收到creatstream命令后向服务器发送play命令。

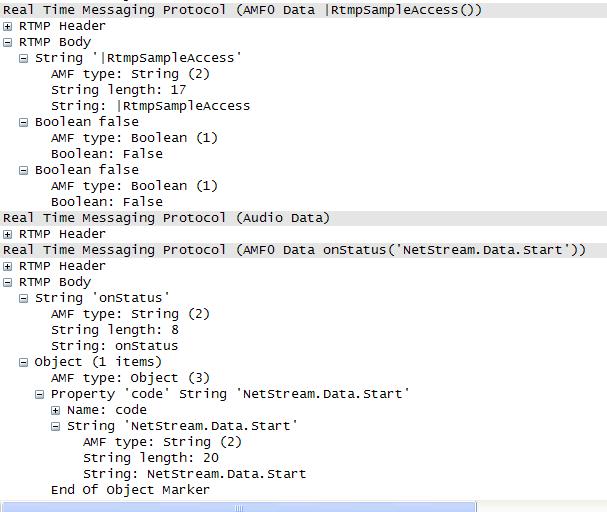


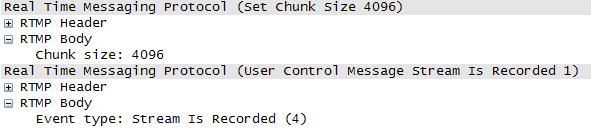
服务器收到play命令后发送协议控制消息来设置消息块的大小。

服务器发送另一个用户控制消息，说明“StreamRecorder”以及相关消息流ID

服务器发送另一个用户控制消息，通过“streambegin“向客户端表示流已经开始。

如果客户端前面的play命令发送成功。服务器在发送onstatus的命令消息。如果play命令请求的相关流媒体没有找到，服务器会在onstatus的命令消息中显示相关信息。





在以上信息交互完成后，服务器就开始向客户端发送视频和音频数据。

# 三．协议控制消息

服务端和客户端交换的消息的不同类型包括用于发送音频数据的音频消息，发送视频的数据的视频消息，发送用户数据的数据消息，共享对象消息，和命令消息。共享对象消息一般提供在多个的客户和一个服务段之间管理分布式数据的方法。

RTMP为协议控制消息保留了1-7的类型ID，这些消息包含RTMP消息块流协议(有两个)或者RTMP本身需要的信息。

ID为1和2时：协议消息被保留用作RTMP消息块流协议的使用；

ID为3-6：协议消息被保留用作RTMP的使用；

ID为7时：协议消息被用在边缘服务和原服务之间

协议控制消息必须有消息流ID0（称为控制流）和消息块流ID2，发送时带有最高优先级。



Message ype是RTMP包里面的数据的类型，占用1个字节。例如音频包的类型为8，视频包的类型为9。下面列出的是常用的数据类型：

|  |  |  |
| --- | --- | --- |
| 0×01 | Chunk Size | changes the chunk size for packets |
| 0×02 | Unknown |  |
| 0×03 | Bytes Read | send every x bytes read by both sides |
| 0×04 | Ping | ping is a stream control message, has subtypes |
| 0×05 | Server BW | the servers downstream bw |
| 0×06 | Client BW | the clients upstream bw |
| 0×07 | Unknown |  |
| 0×08 | Audio Data | packet containing audio |
| 0×09 | Video Data | packet containing video data |
| 0x0A-0x0E | Unknown |  |
| 0x0F | FLEX\_STREAM\_SEND | TYPE\_FLEX\_STREAM\_SEND |
| 0x10 | FLEX\_SHARED\_OBJECT | TYPE\_FLEX\_SHARED\_OBJECT |
| 0x11 | FLEX\_MESSAGE | TYPE\_FLEX\_MESSAGE |
| 0×12 | Notify | an invoke which does not expect a reply |
| 0×13 | Shared Object | has subtypes |
| 0×14 | Invoke | like remoting call, used for stream actions too. |
| 0×16 | StreamData | 这是FMS3出来后新增的数据类型,这种类型数据中包含AudioData和VideoData |

|  |  |
| --- | --- |
| 协议控制消息 | 消息描述 |
| 设置消息块大小 | 设置消息块大小，用来通知对方新的可供利用的最大的消息块的大小。 |
| 中断消息： | 协议控制消息2，中断消息，用来向通知对方是否在等待消息块完成一则消息，然后丢弃部分接收到的消息，放弃消息的处理过程， |
| 确认（ACK）： | 客户端或者服务端当接收到的字节和窗口大小一样时发送确认给对方，窗口大小是发送方发送的在没有从接收方收到确认之前的最大字节数，服务端在应用程序接通后给客户端发送窗口大小，这个消息指定了序号，就是目前为止收到的字节数。 |
| 用户控制 | 客户端或者服务端发送这个消息用来通知对方用户控制事件，这个消息负载事件类型和事件数据，  头两个字节用来标示事件类型，事件数据跟随在事件类型后面，事件数据字段的大小事可变的。 |
| 窗口确认大小 | 客户端或者服务端在发送确认时发送这个消息来告知对方使用的窗口大小。例如：一个服务端期望在服务端发送的字节数和窗口大小一样时每时每刻都收到客户端的确认，服务端在成功处理客户端请求的连接后向客户端更新自己的窗口大小。 |
| 设置带宽 | 客户端或者服务端发送这个消息用来更新对方的输出带宽，输出带宽的值和对方大小一样。如果自己当前窗口大小和消息中接收到得不一样，对方发送“窗口确认大小“。 |

*//可以负载4个字节的消息负载。消息块的大小可以被设置成一个默认的值，但是客户端或者服务端可以改变这个值，可以给对方更新。例如：假设一个客户端想要发送131字节的音频数据，消息块的大小为128字节，在这种情况下，客户端可以发送这个协议控制消息给服务端以通知消息块的大小被设置成131字节，那么客户端就可以用一个消息块发送音频数据。*

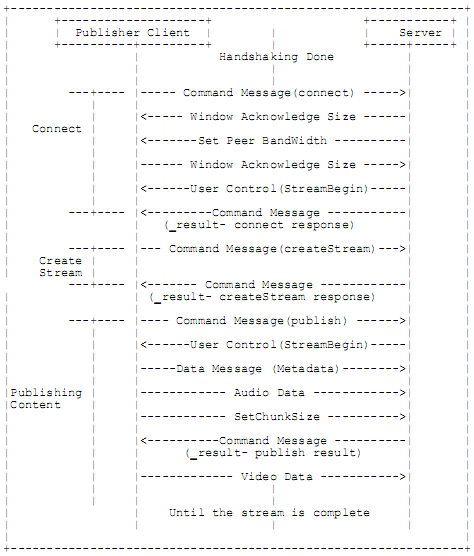
*最大的消息块大小可以是65536个字节，消息块的大小在服务端与客户端的通信以及客户端到服务端的通信是维持不变的。*

# 四：消息交换的例子

这有一些例子解释使用RTMP的消息交换。

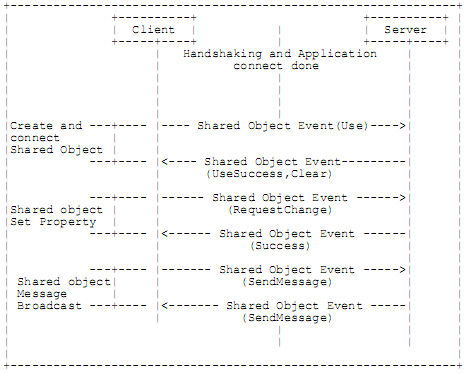
发布录制的视频

这个例子图示一个发布方如何发布一个流然后如何在服务端把视频变为视频流，其他的客户端可以订阅这个发布的流和播放视频。



广播一个共享对象消息。

这个例子图示了在创造和改变共享对象的过程中消息的交换，也是共享对象广播的过程。



从录制的流中发布元数据

这个例子表述了发布元数据时的消息交换。

