C/C++Linux服务器开发

高级架构师课程

三年课程沉淀 | 五次精益升级 | 十年行业积累 | 百个实战项目 | 十万内容受众

讲师:darren/326873713

扫一扫 升职加薪

班主任:柚子/2690491738

讲师介绍--专业来自专注和实力

King老师

系统架构师，曾供职著名创业公司系统架构师，微软亚洲研究院、创维集团全球研发中心。国内第一代商业Paas平台开发者。著有多个软件专利，参与多个开源软件维护。在全球化，高可用的物联网云平台架构与智能硬件设计方面有丰富的研发与实战经验。

Lee老师

曾供职于华为和诺基亚通信长达7年时间，曾在某上市公司担任技术总监。 10年（C/C++）开发经验和产品管理经验，研究过过多款C/C++优秀开源软件的框架，开发过大型音频广播云平台，深入理解需求分析、架构分析和产品管理，精通敏捷开发流程和项目管理。

Darren老师

曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

课题：RTMP流媒体实战2

2020年3月5日 20:05正式上课

1.RTMP协议抓包分析

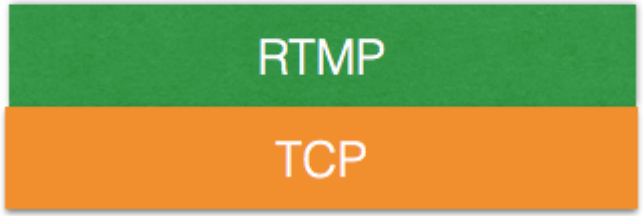
2.RTMP分包策略

3.RTMP实战

RTMP简介

RTMP(Real Time Messaging Protocol)是一个应用层协议，主要用于在Flash player和服务器之间传输视频、音频、控制命令等内容。
该协议的突出优点是：**低延时**。

RTMP协议栈



RTMP基于TCP，**默认使用端口1935**。

RTMP 名词解析

详细见：rtmp_specification中的3.名词解释(3. Definitions)

RTMP推流拉流

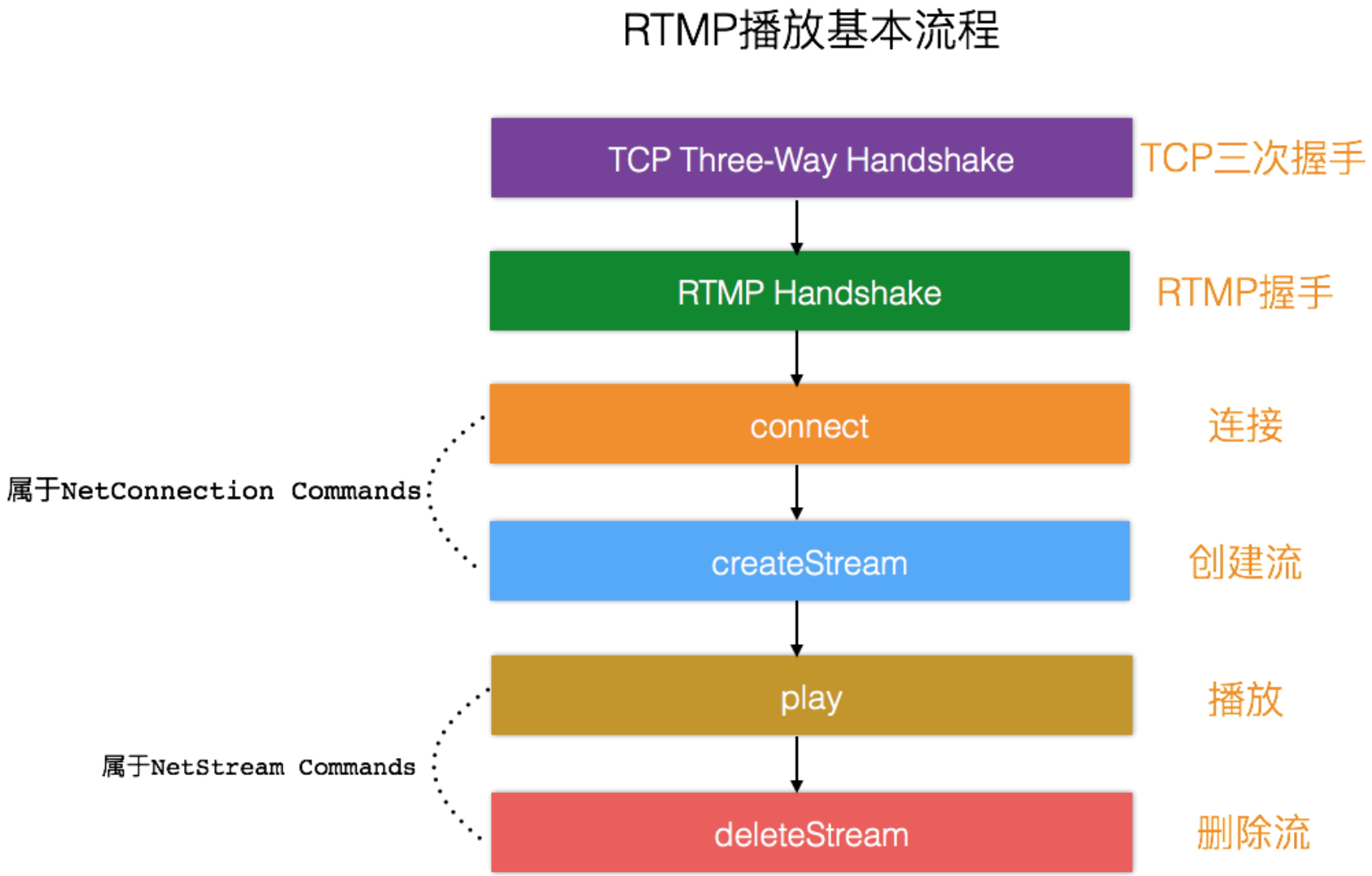
FFMPEG推流+FFPLAY播放

推流:ffmpeg -re -i /mnt/hgfs/linux/vod/35.mp4 -c copy -f flv rtmp://192.168.100.41/live/35
拉流:ffplay rtmp://192.168.100.41/live/35

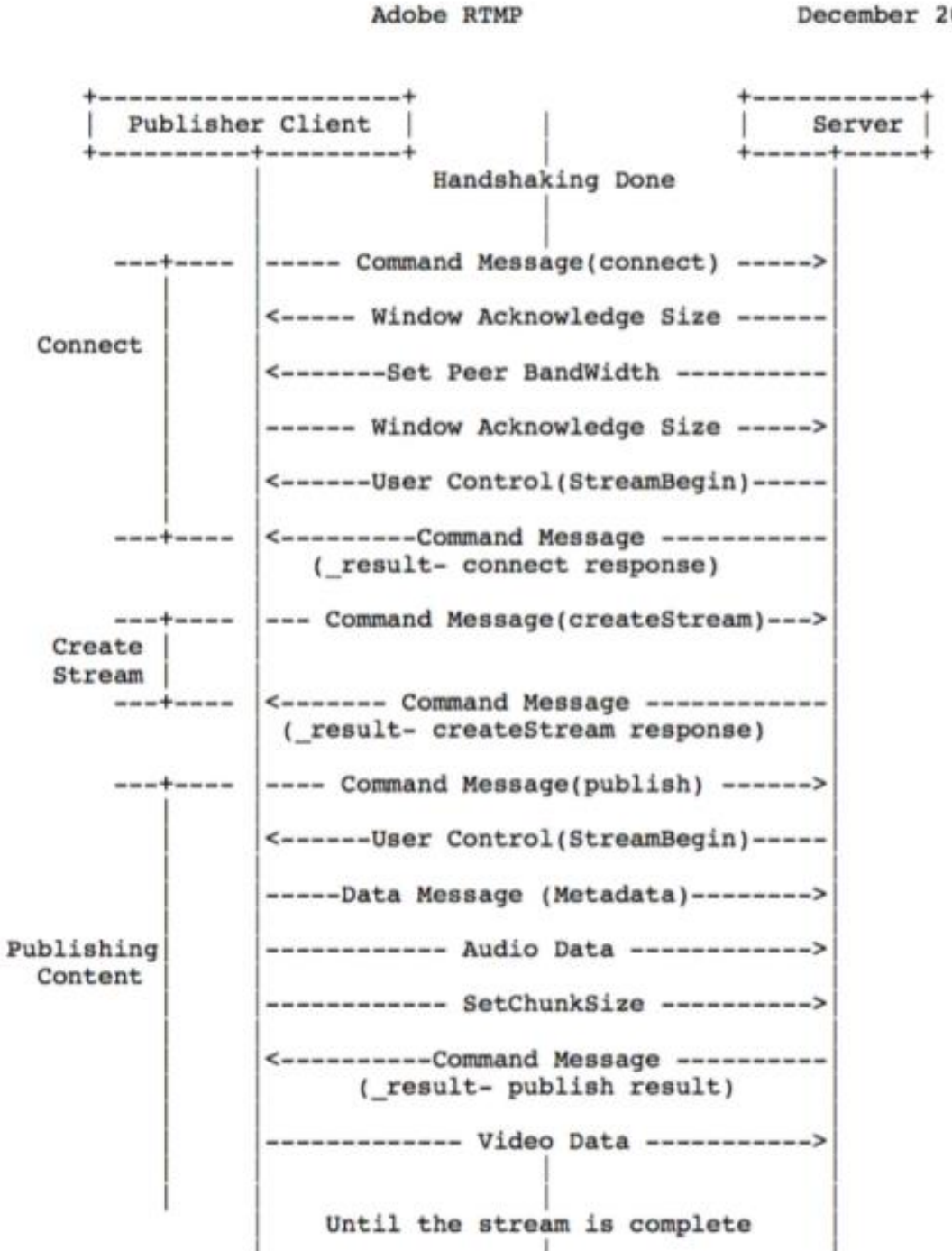
FFPLAY播放

拉流:ffplay rtmp://202.69.69.180:443/webcast/bshdlive-pc

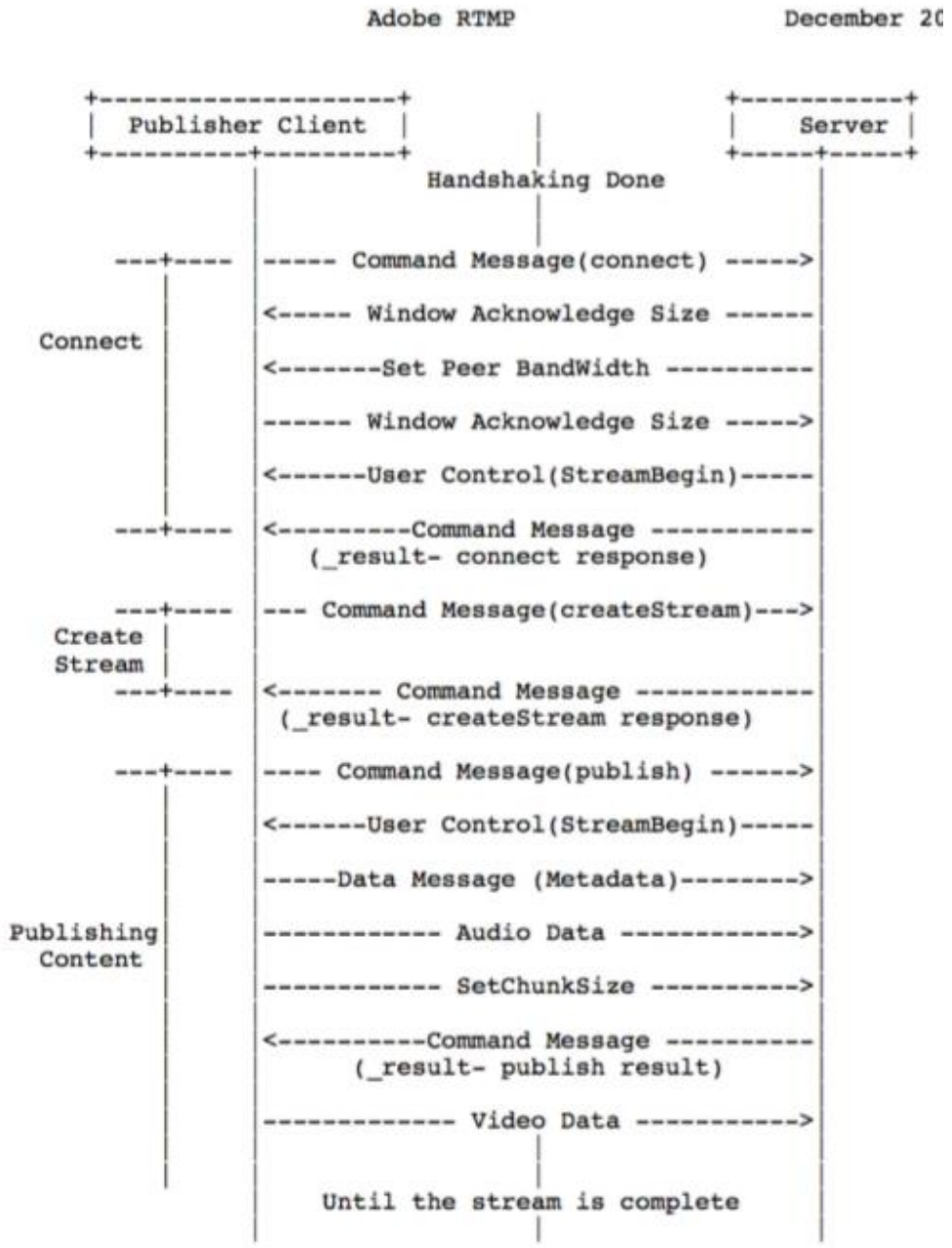
RTMP播放基本流程


RTMP播放基本流程
TCP Three-Way Handshake
TCP三次握手
RTMP Handshake
RTMP握手
connect
连接
属于NetConnection Commands
createStream
创建流
play
播放
属于NetStream Commands
deleteStream
删除流

推流流程

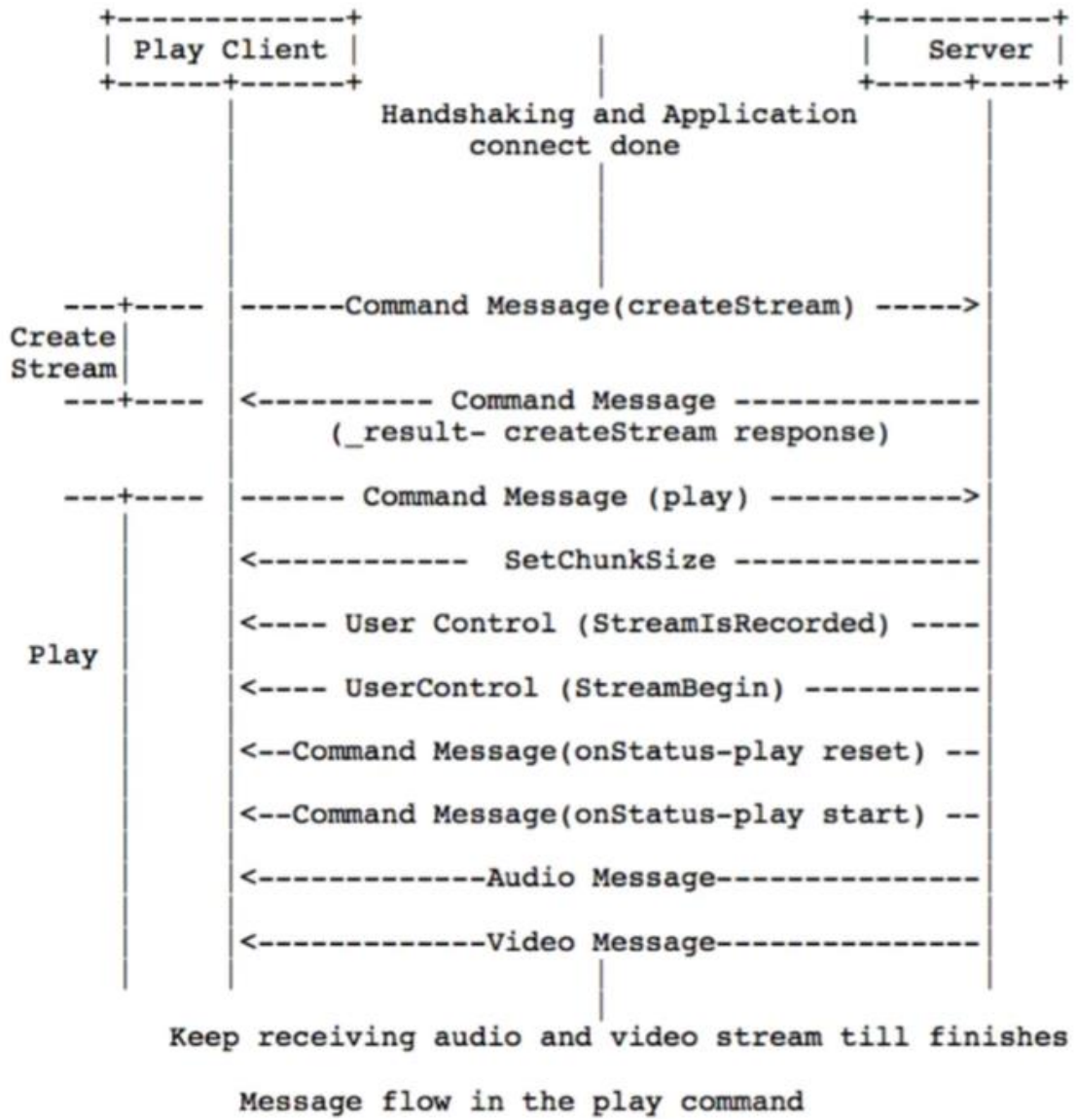

Adobe RTMP
December 20
Publisher Client
Server
Handshaking Done
Command Message(connect)
Window Acknowledge Size
Connect
Set Peer BandWidth
Window Acknowledge Size
User Control(StreamBegin)
Command Message
(_result- connect response)
Command Message(createStream)
Create Stream
Command Message
(_result- createStream response)
Command Message(publish)
User Control(StreamBegin)
Data Message (Metadata)
Audio Data
Publishing Content
SetChunkSize
Command Message
(_result- publish result)
Video Data
Until the stream is complete

推流流程



Message flow in publishing a video stream

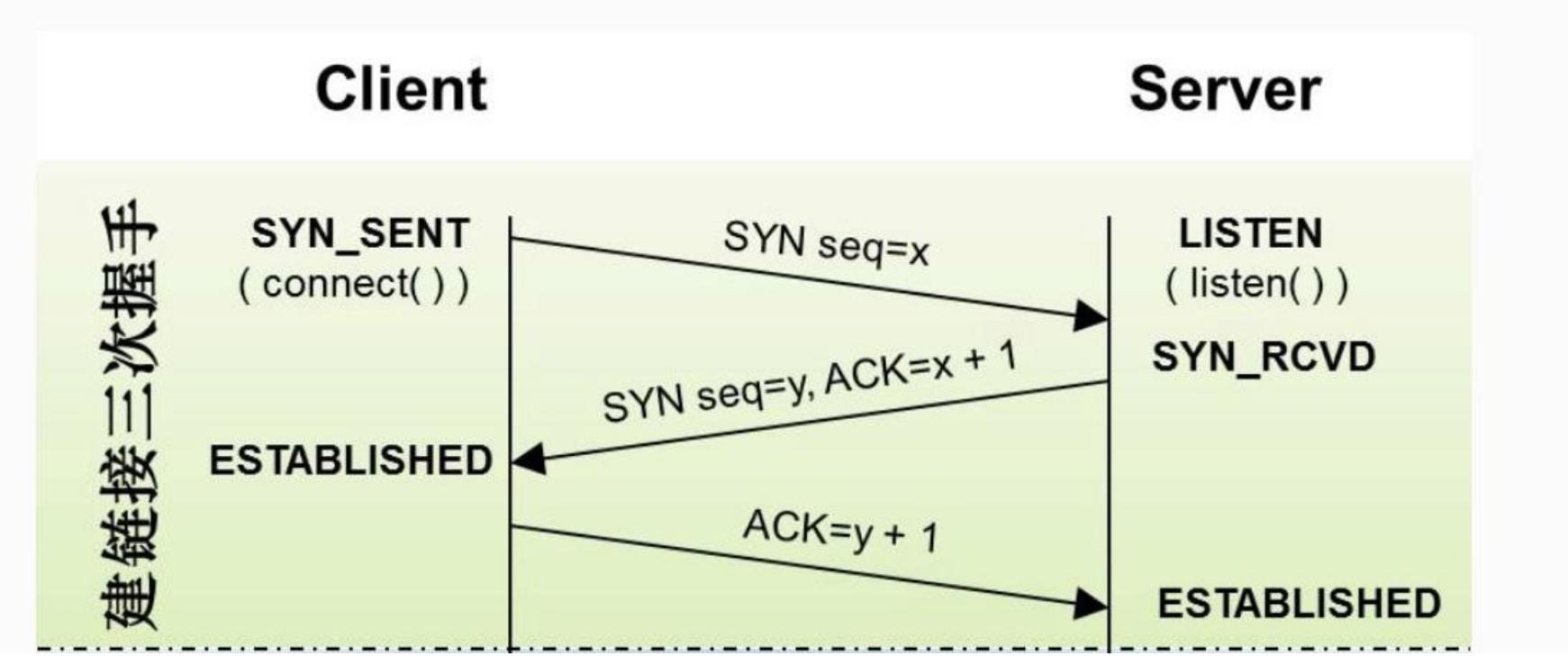
播流流程



Message flow in the play command

Step 1: TCP三次握手 - 修高速公路

RTMP是基于TCP的应用层协议。
通过TCP三次握手，可实现RTMP客户端与RTMP服务器的指定端口(默认端口为1935)建立一个可靠的网络连接。
这里的网络连接才是真正的物理连接。
完成了三次握手，客户端和服务器端就可以开始传送数据。



Step 1: TCP三次握手 - 修高速公路

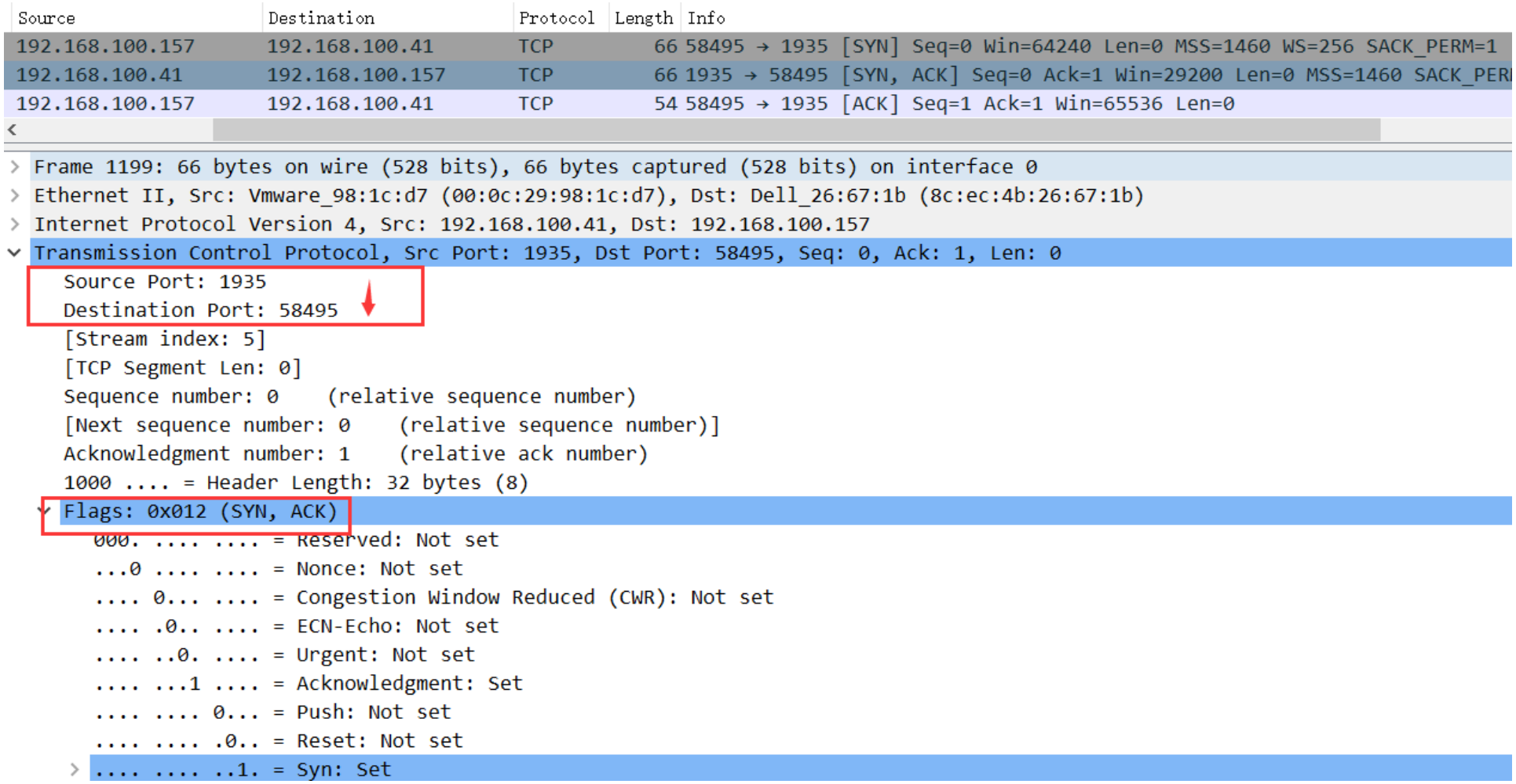
Source	Destination	Protocol	Length	Info
192.168.100.157	192.168.100.41	TCP	66	58495 → 1935 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
192.168.100.41	192.168.100.157	TCP	66	1935 → 58495 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
192.168.100.157	192.168.100.41	TCP	54	58495 → 1935 [ACK] Seq=1 Ack=1 Win=65536 Len=0

RTMP播放的第一步：TCP三次握手

```
> Frame 1198: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: Dell_26:67:1b (8c:ec:4b:26:67:1b), Dst: Vmware_98:1c:d7 (00:0c:29:98:1c:d7)
> Internet Protocol Version 4, Src: 192.168.100.157, Dst: 192.168.100.41
v Transmission Control Protocol, Src Port: 58495, Dst Port: 1935, Seq: 0, Len: 0
    Source Port: 58495
    Destination Port: 1935
    [Stream index: 5]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
  v Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    > .... .... ..1. = Syn: Set
      .... .... ...0 = Fin: Not set
```

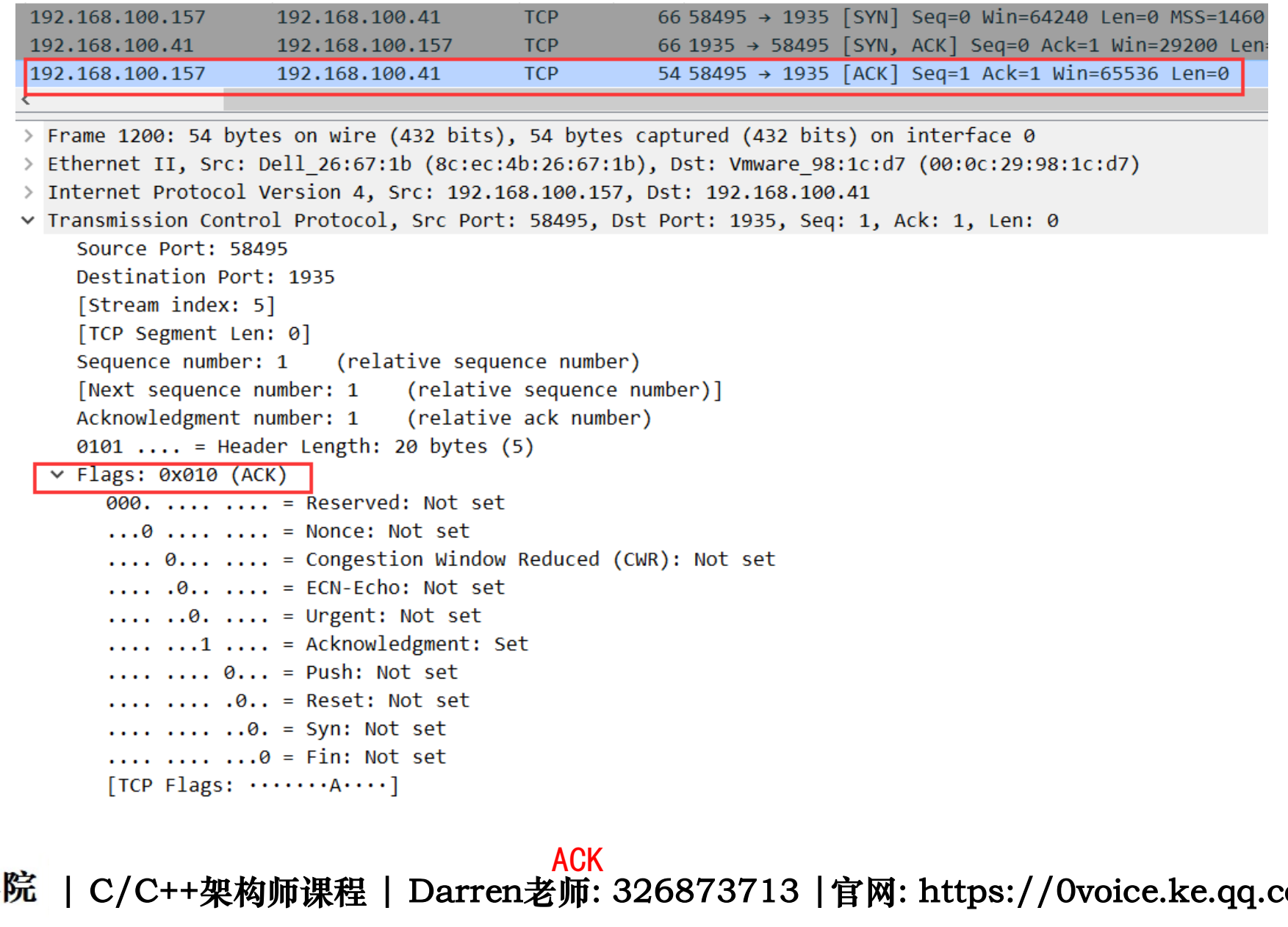
SYN

Step 1: TCP三次握手 - 修高速公路



SYN, ACK

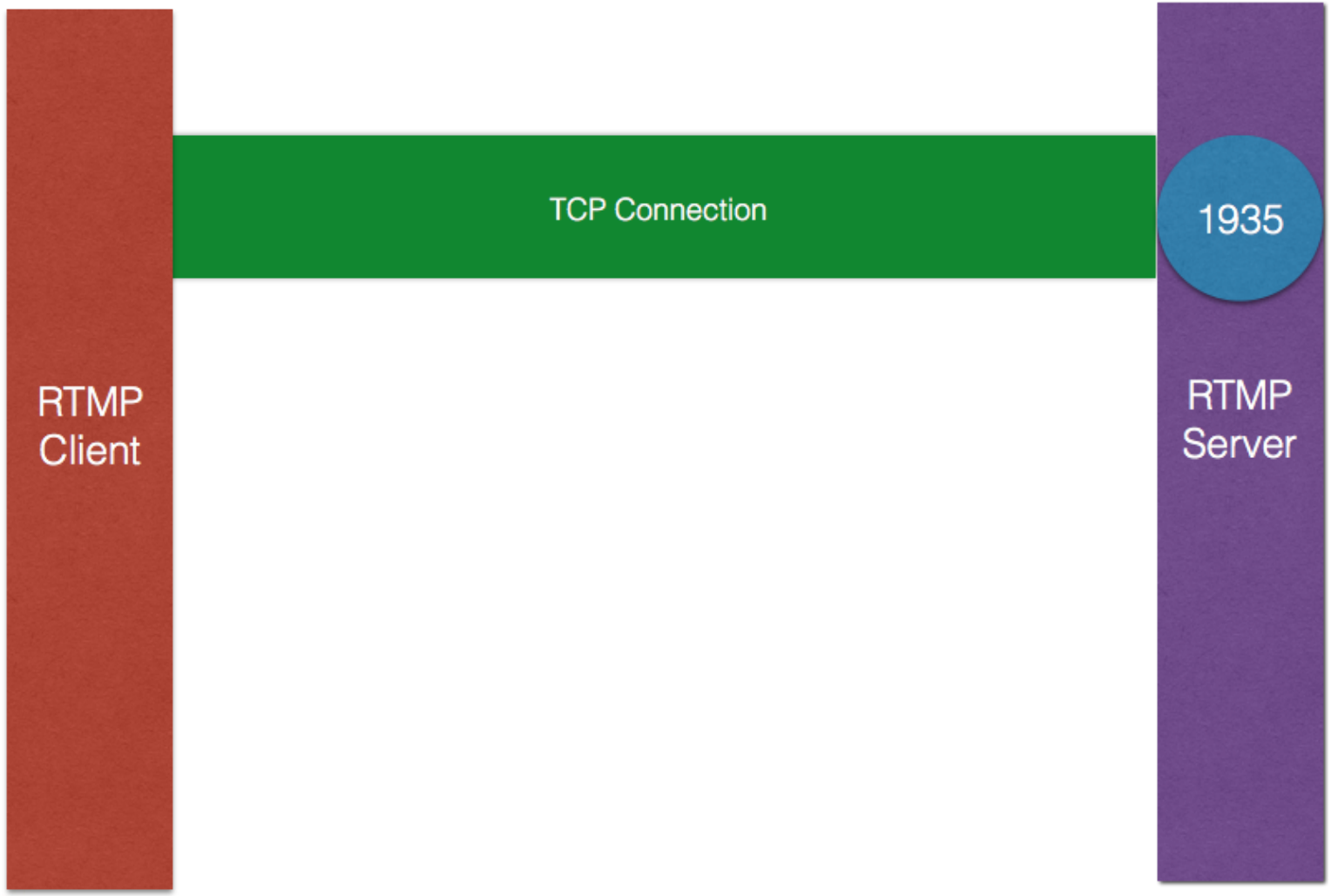
Step 1: TCP三次握手 - 修高速公路



ACK

Step 1: TCP三次握手 - 修高速公路

经过三次握手，客户端与服务器端1935端口建立了TCP Connection。



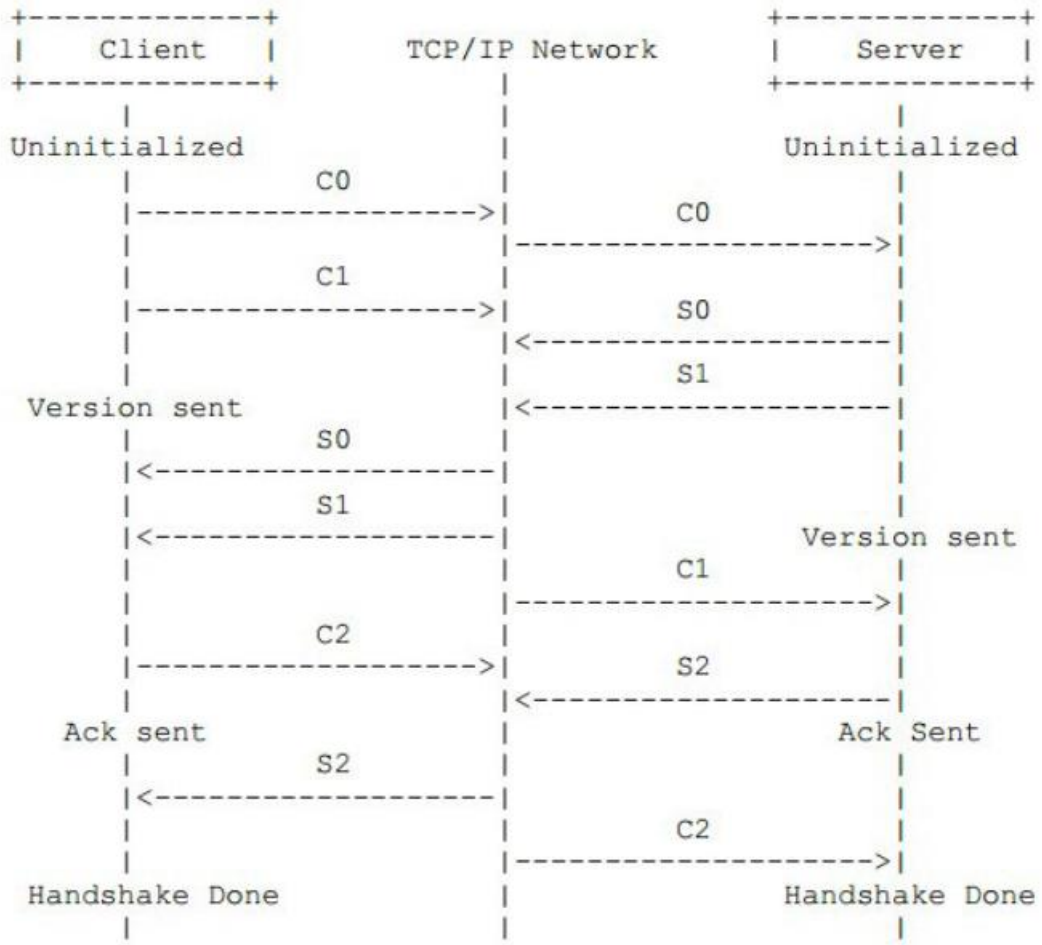
Step 2: RTMP握手 - 安检

与其叫RTMP握手，其实实质上起到的是验证的作用。
RTMP握手的基本流程:

192.168.100.157	192.168.100.41	RTMP	131	Handshake C0+C1
192.168.100.41	192.168.100.157	RTMP	130	Handshake S0+S1+S2
192.168.100.157	192.168.100.41	RTMP	130	Handshake C2

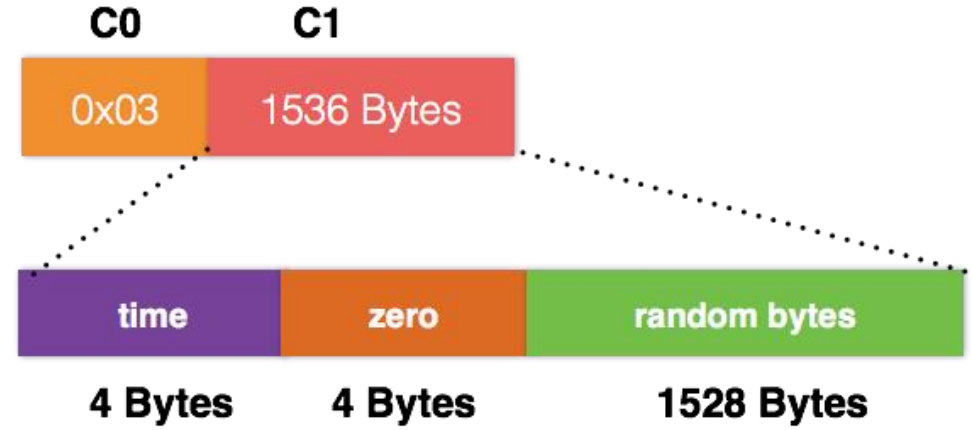
RTMP握手主要分为: **简单握手**和**复杂握手**。

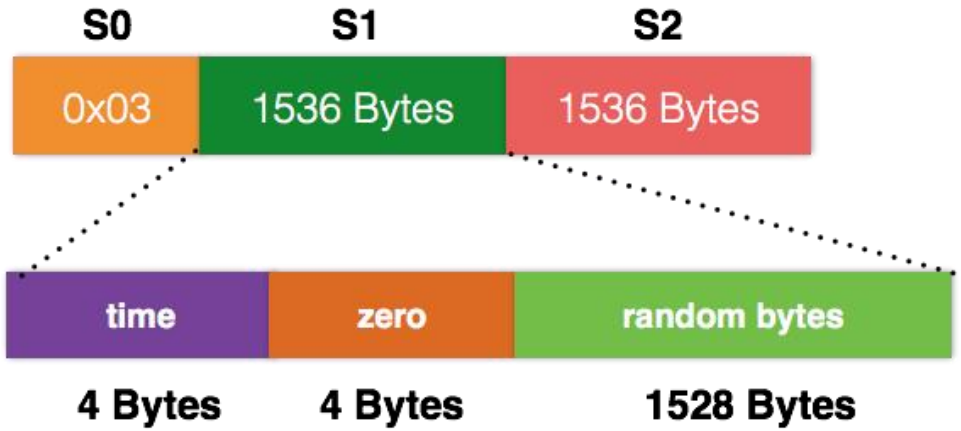
第三章 Step 2: RTMP握手 - 简单握手



Pictorial Representation of Handshake

Step 2: RTMP握手 - 简单握手





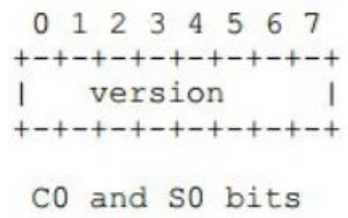
简单握手中C1和S1从第9个字节开始都是随机数。

S2是C1的复制。

C2是S1的复制。

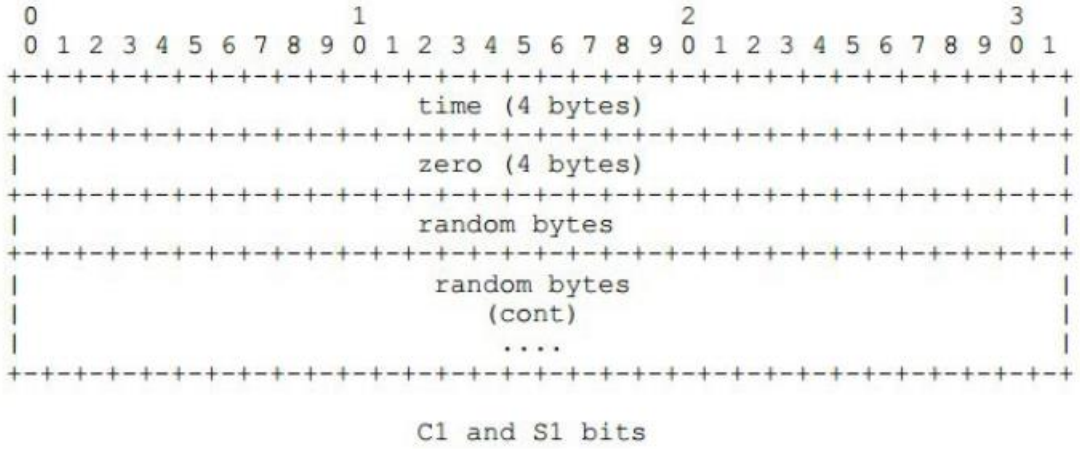
C2 is echo of S1
S2 is echo of C1

Step 2: RTMP握手 - 简单握手



协议版本号（8bit）
C0：客户端版本
S0：服务器版本
目前版本为3，（0，1，2已经废弃）

C1和S1数据包的长度都是1536字节

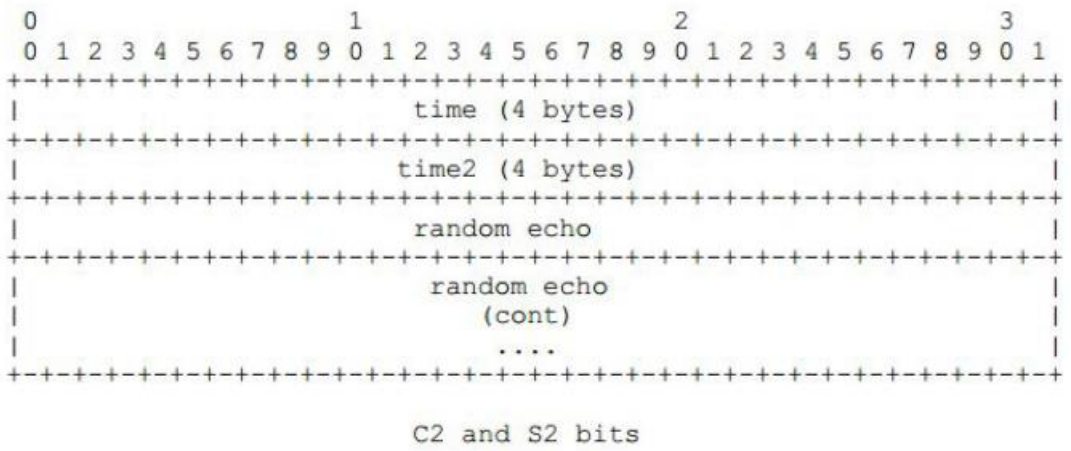


03(C0) **00**(C1开始) **00 00 00 09 00 7c 02** f7 78 55 1e ce ab 8e

S1片段： **00 90 65 30 0d 0e 0a 0d** 64 84 1c ad 1e 7f 0c

Step 2: RTMP握手 - 简单握手

C2 和 S2 数据包长度都是 1536 字节，基本就是 S1 和 C1 的副本。



C2 and S2 bits

S2片段

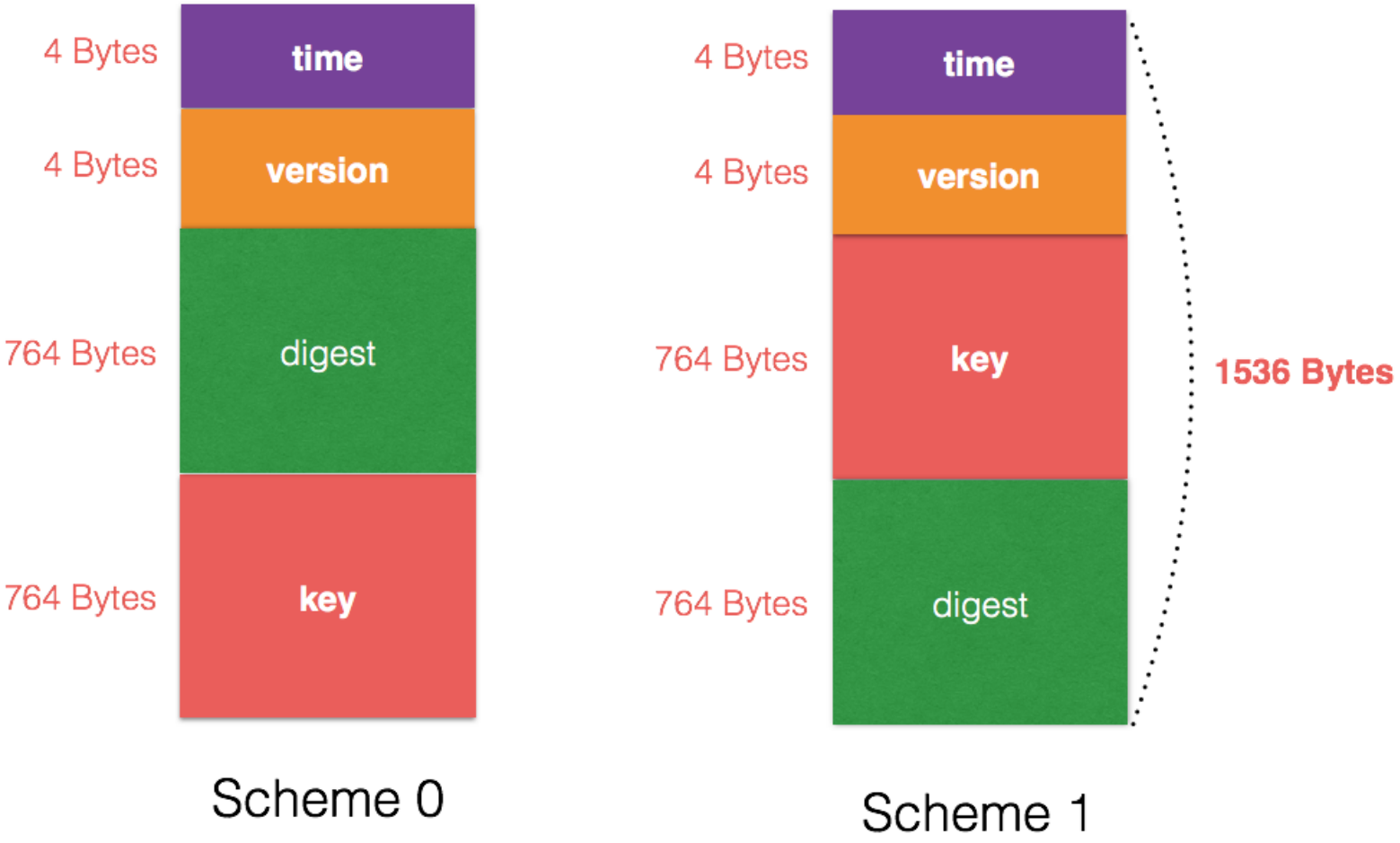
```
4c2608213654b24c58d29
```

```
Handshake C2
   Handshake data: e25c15c22caf72d0986fbd3eda0d7151973e19083e0abbef.
```

Step 2: RTMP握手 – 复杂握手

相对于简单握手，复杂握手主要是增加了更严格的验证。
主要是将简单握手中1528Bytes随机数的部分平均分成两部分，一部分764Bytes存储public key(公共密钥)，另一部分764Bytes存储digest(密文，32字节)。
另外，**复杂握手还有一个明显的特征就是：Version部分不为0，服务器端可根据这个来判断是否简单握手或复杂握手。**

Step 2: RTMP握手 – 复杂握手



Step 3: connect(连接)

这里也叫连接，连接的是什么呢?
这里必须明白RTMP中一个很重要的概念: Application Instance。

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

不同的 Application Instance可根据功能等进行区分，比如直播可以用live来表示，点播回放可以用vod来表示。

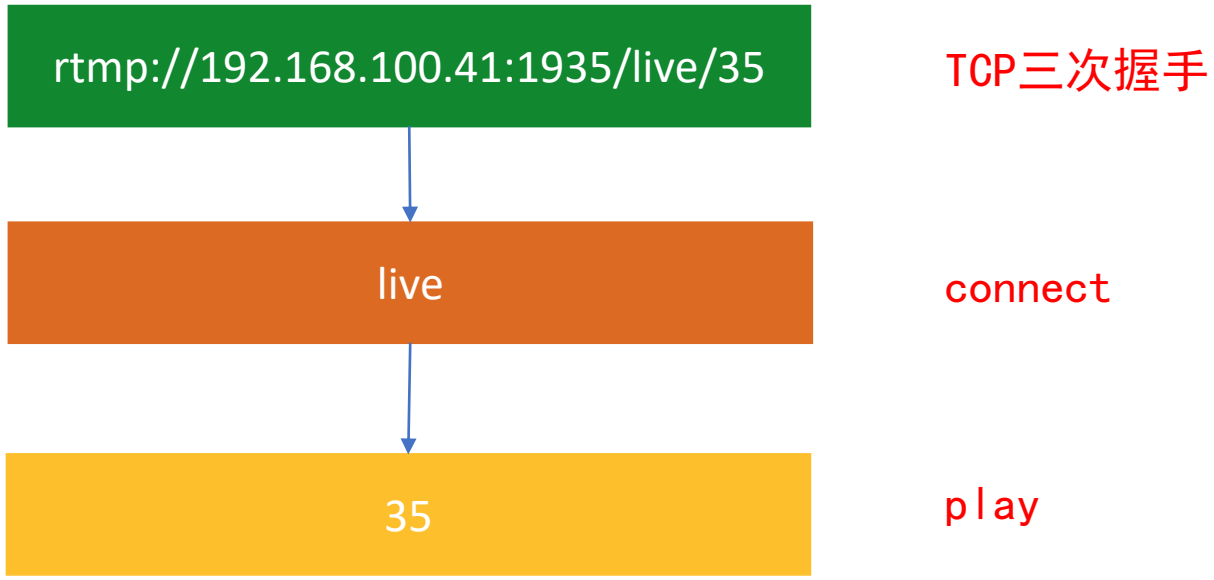
rtmp://192.168.100.41/live/36
其中live就是Application Instance（sport, music）
播放该流时，connect的地址就是
rtmp://192.168.100.41/live/36

Step 3: connect(连接)

```
638 10.677062     192.168.100.157      192.168.100.41       RTMP      127 connect('live')

Real Time Messaging Protocol (AMF0 Command connect('live'))
    Response to this call in frame: 645
  RTMP Header
      00.. .... = Format: 0
      ..00 0011 = Chunk Stream ID: 3
      Timestamp: 0
      Body size: 201
      Type ID: AMF0 Command (0x14)
      Stream ID: 0
  RTMP Body
    String 'connect'
        AMF0 type: String (0x02)
        String length: 7
        String: connect
    Number 1
        AMF0 type: Number (0x00)
        Number: 1
    Object (8 items)
        AMF0 type: Object (0x03)
      Property 'app' String 'live'
          Name: app
        String 'live'
            AMF0 type: String (0x02)
            String length: 4
            String: live
      Property 'flashVer' String 'LNX 9,0,124,2'
      Property 'tcUrl' String 'rtmp://192.168.100.41:1935/live'
          Name: tcUrl
        String 'rtmp://192.168.100.41:1935/live'
```

Step 3: connect(连接)



连接Application Instance举例

Step 4: createStream(创建流) - 创建逻辑通道

7.2.1.3. 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.

createStream命令用于**创建逻辑通道**，该通道用于传输视频、音频、metadata。
在服务器的响应报文中会返回**Stream ID**，用于**唯一的标示该Stream**。

注：**Message ID和Stream ID的区别**

Step 4: createStream(创建流) - 创建逻辑通道

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.

Step 4: createStream(创建流) - 创建逻辑通道

```
649 10.760938    192.168.100.157    192.168.100.41     RTMP    91 Window Acknowledgement Size 5000000|createStream()
651 10.761142    192.168.100.41     192.168.100.157    RTMP    95 _result()
656 10.804979    192.168.100.157    192.168.100.41     RTMP   148 getStreamLength()|play('35')|Set Buffer Length 1,300
```

```
Frame 651: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface 0
Ethernet II, Src: Vmware_98:1c:d7 (00:0c:29:98:1c:d7), Dst: Dell_26:67:1b (8c:ec:4b:26:67:1b)
Internet Protocol Version 4, Src: 192.168.100.41, Dst: 192.168.100.157
Transmission Control Protocol, Src Port: 1935, Dst Port: 59244, Seq: 3325, Ack: 3337, Len: 41
Real Time Messaging Protocol (AMF0 Command _result())
  Call for this response in frame: 649
▾ RTMP Header
    00.. .... = Format: 0
    ..00 0011 = Chunk Stream ID: 3
    Timestamp: 0
    Body size: 29
    Type ID: AMF0 Command (0x14)
    Stream ID: 0
▾ RTMP Body
  ▾ String '_result'
      AMF0 type: String (0x02)
      String length: 7
      String: _result
  ▾ Number 2
      AMF0 type: Number (0x00)
      Number: 2
  ▾ Null
      AMF0 type: Null (0x05)
  ▾ Number 1
      AMF0 type: Number (0x00)
      Number: 1
```

可看出返回的Stream ID为1。
后续的视频或音频的Stream ID就是1。

Step 4: createStream(创建流) - 创建逻辑通道

```
Real Time Messaging Protocol (Audio Data)
RTMP Header
    00.. .... = Format: 0
    ..00 0110 = Chunk Stream ID: 6
    Timestamp: 141675
    Body size: 4
    Type ID: Audio Data (0x08)
    Stream ID: 1
RTMP Body
  Control: 0xaf (HE-AAC 44 kHz 16 bit stereo)
      1010 .... = Format: HE-AAC (10)
```

```
250  00 00 00 00 09 06 02 29  6b 00 00 04 08 01 00 00   .......) k.......
260  00 af 00 11 90 46 00 00  15 00 00 fe 08 af 01 21   .....F.. ........!
```

```
Real Time Messaging Protocol (Video Data)
RTMP Header
    00.. .... = Format: 0
    ..00 0111 = Chunk Stream ID: 7
    Timestamp: 141880
    Body size: 50
    Type ID: Video Data (0x09)
    Stream ID: 1
RTMP Body
  > Control: 0x17 (keyframe H.264)
    Video data: 0000000001640028ffe1001c67640028acc85
```

```
030  01 2b 91 70 00 00 07 02  2a 38 00 00 32 09 01 00
040  00 00 17 00 00 00 00 01  64 00 28 ff e1 00 1c 67
```

Step 5: play(播放)

7.2.2.1. 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.

客户端发送play命令来播放指定流。开始传输音视频数据。
如果发送play命令后想要立即播放，需要清空play队列中的其它流，并将reset置为true。

Step 6: deleteStream(删除流)

7.2.2.3. 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.

删除指定Stream ID的流。
服务器不用对这条命令发送响应报文。

RTMP层次

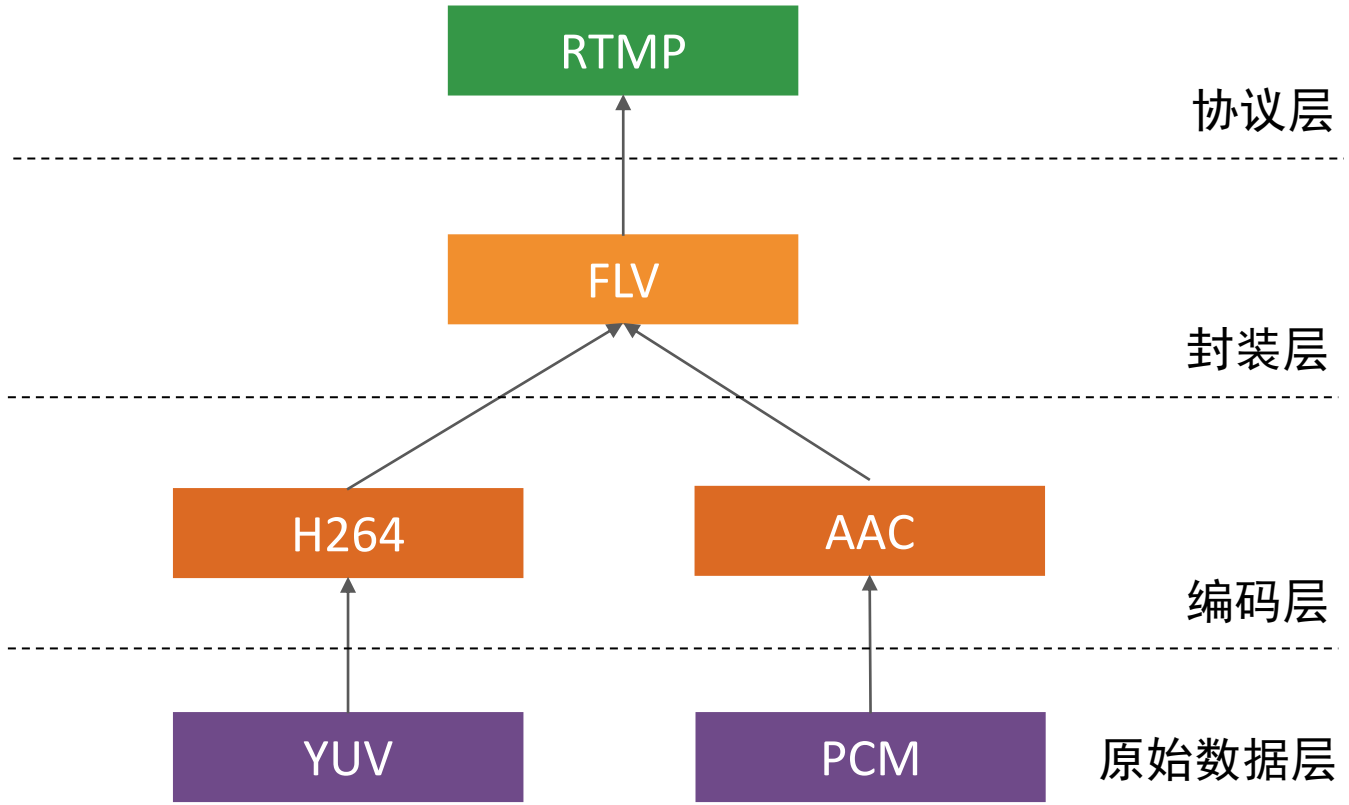
RTMP层次（数据发送角度）

RTMP层次（数据接收角度）

RTMP层次（协议角度）

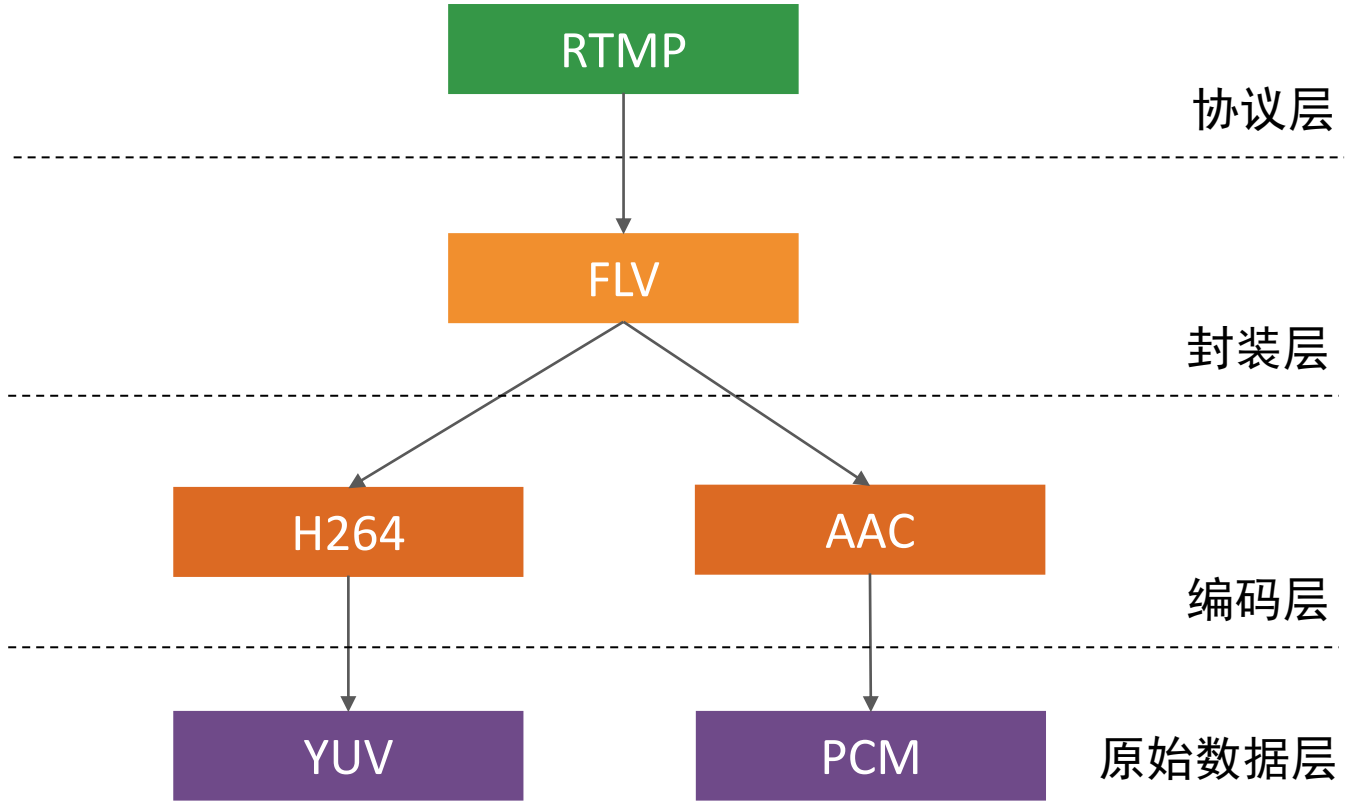
RTMP层次 (数据角度)

RTMP层次（数据**发送**角度）

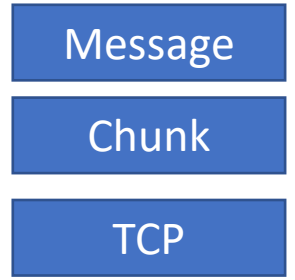


RTMP层次 (数据角度)

RTMP层次 （数据接收角度）

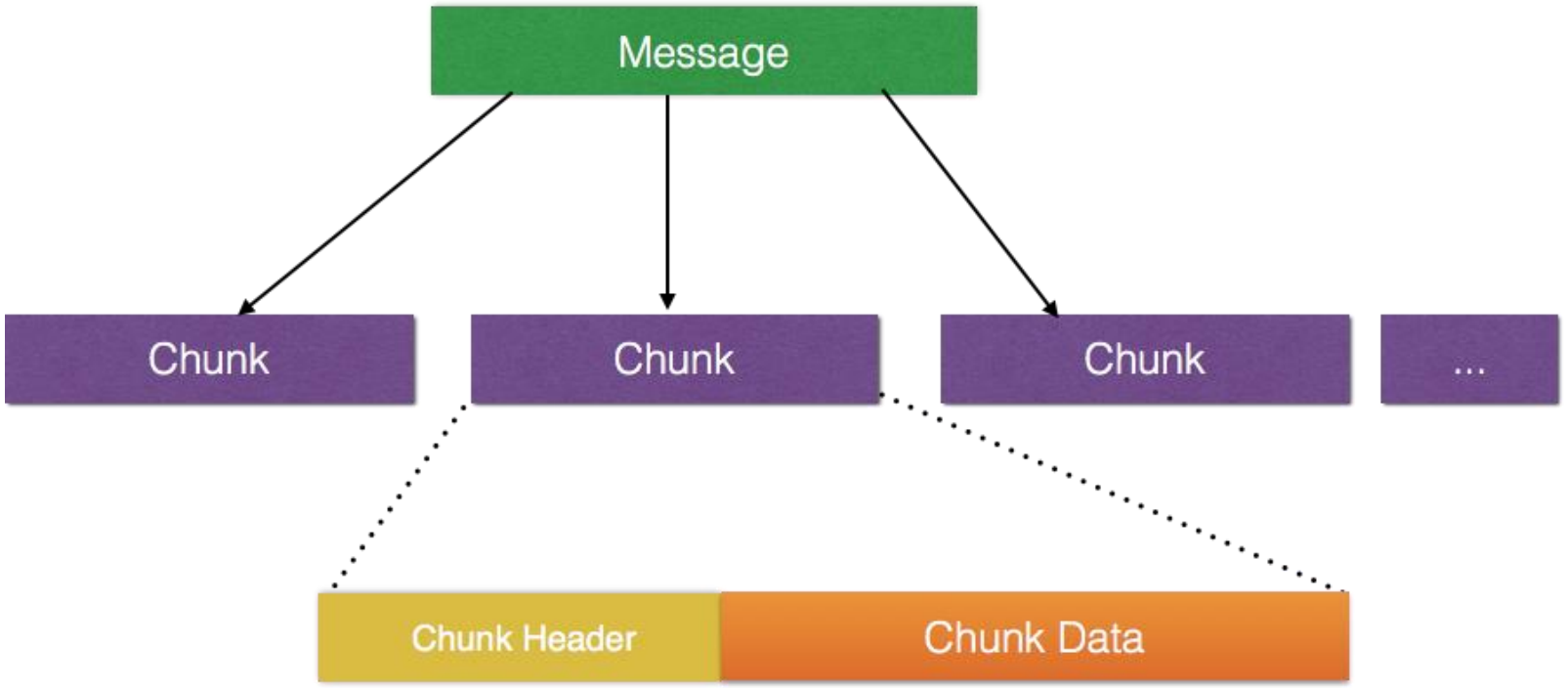


RTMP关键结构



Message 和 Chunk

Message & Chunk

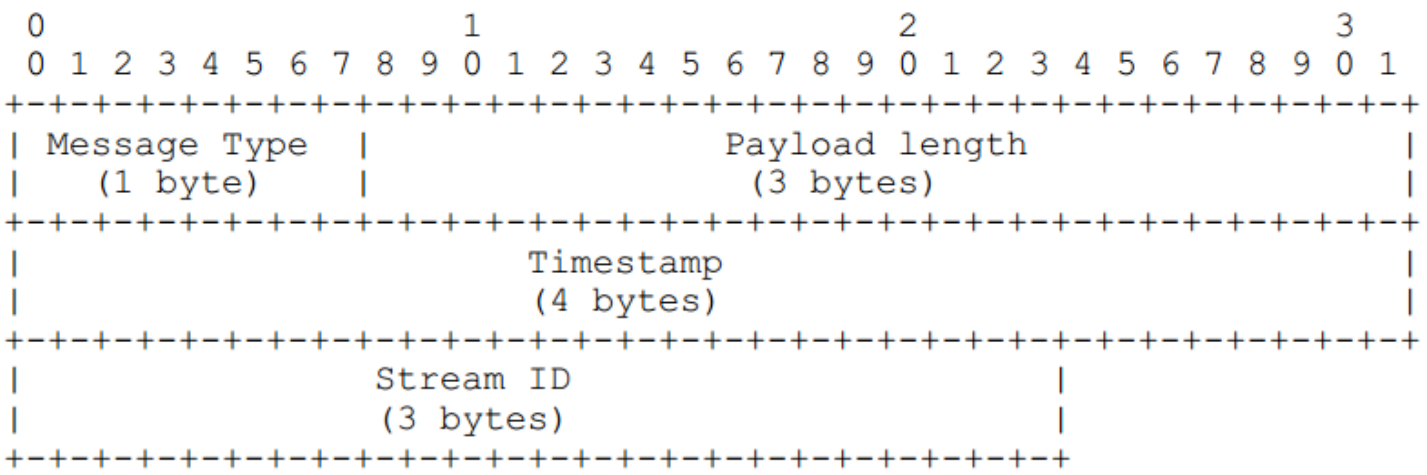


Message & Chunk

Message RTMP中一个重要的概念就是消息。

21: 08分继续

RTMP Message



Message Header

注: 6.1.1. Message Header

消息分类 – Message Type

消息主要分为三类: 协议控制消息、数据消息、命令消息等。

协议控制消息
Message Type ID = 1 2 3 5 6和Message Type ID = 4两大类，主要用于协议内的控制，此部分后续将详细分析。

数据消息
Message Type ID = 8 9 18
8: Audio 音频数据
9: Video 视频数据
18: Metadata 包括音视频编码、视频宽高等信息。

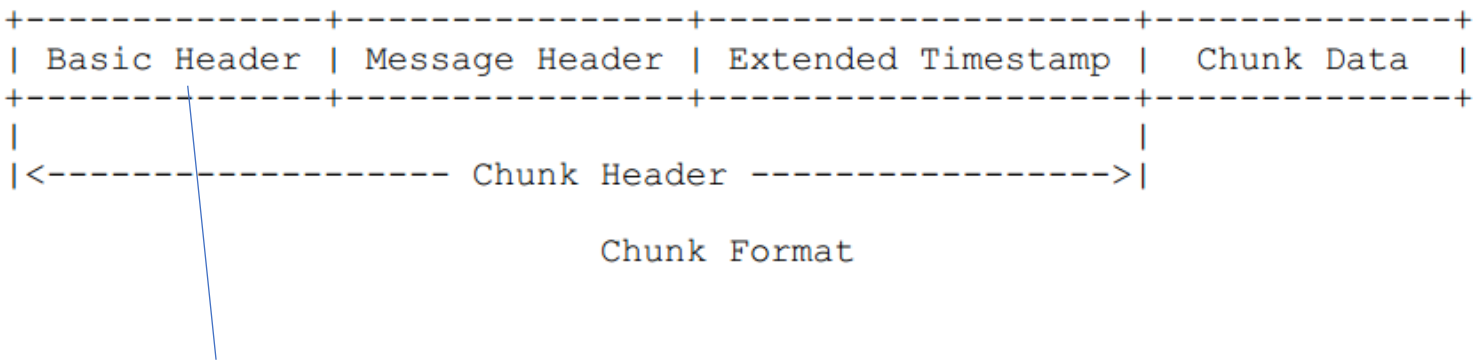
命令消息 Command Message (20, 17)
此类型消息主要有NetConnection和NetStream两个类，两个类分别有多个函数，该消息的调用，可理解为远程函数调用。

消息分类 – Stream ID

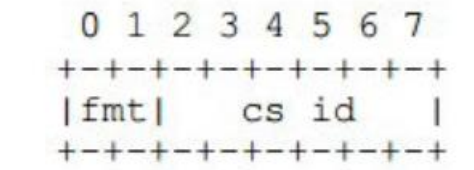
Message StreamID是音视频流的唯一ID，一路流如果既有音频包又有视频包，那么这路流音频包的StreamID和他视频包的StreamID相同。

Message & Chunk

Chunk 网络中实际发送的内容。



Chunk Format



Chunk basic header 1

cs id即是Chunk Stream ID

Chunk Stream ID

Each chunk that is created has a unique ID associated with it called chunk stream ID。(5.3. Chunking)

问题：因为一个流当中可以交错传输多种消息类型的Chunk，那么多个Chunk怎么标记同属于同一类Message的呢？

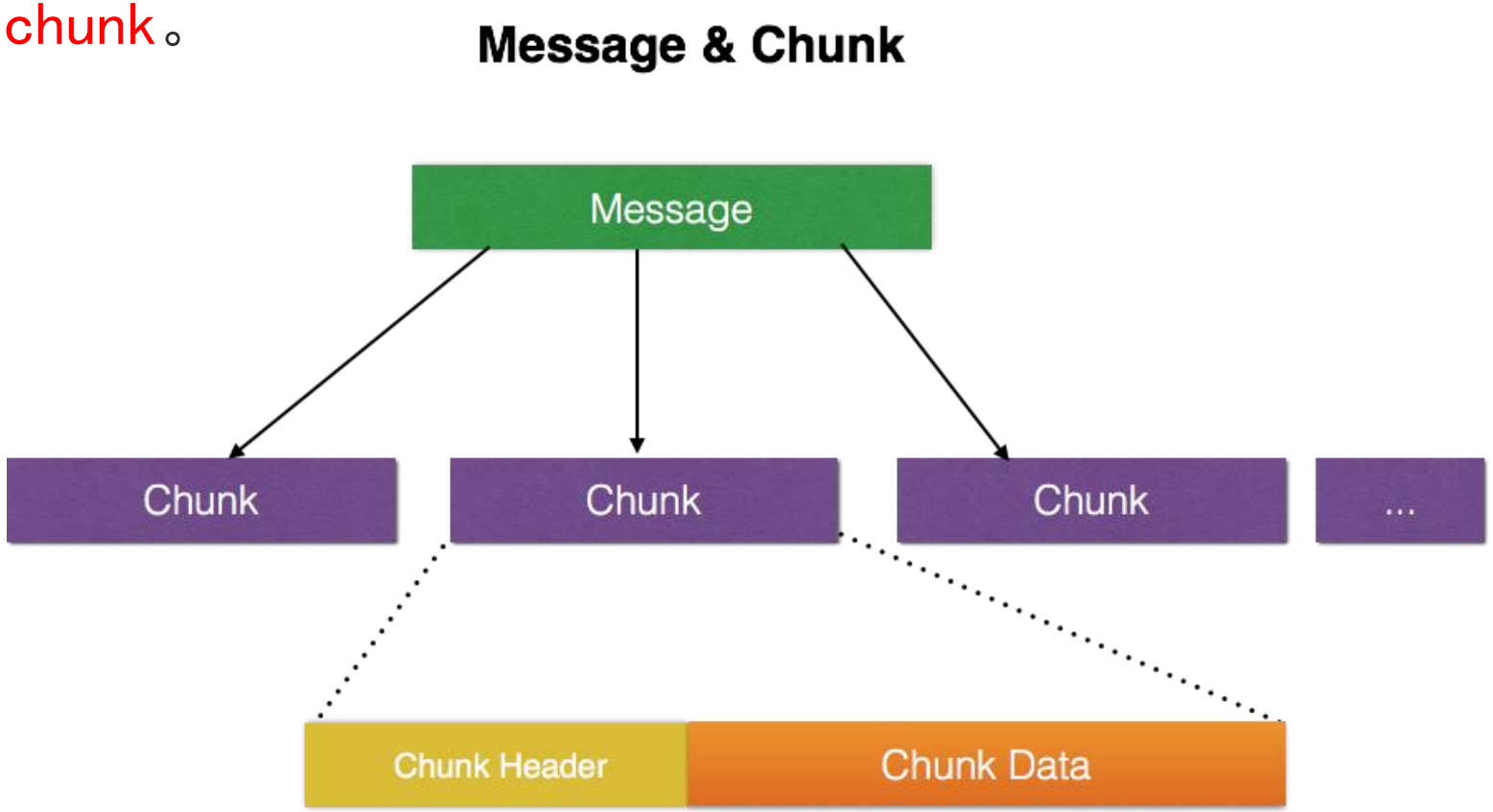
答案：是通过Chunk Stream ID区分的，同一个Chunk Stream ID必然属于同一个Message

RTMP流中视频和音频拥有单独的Chunk Stream ID
比如音频的cs id=20，视频的cs id=21。
接收端接收到Chunk之后，根据cs id分别将音频和视频“拼成消息”。

Message & Chunk

Message被切割成一个或多个Chunk，然后在网络上进行发送。

当发送时，一个chunk发送完毕后才可以发送下一个chunk。

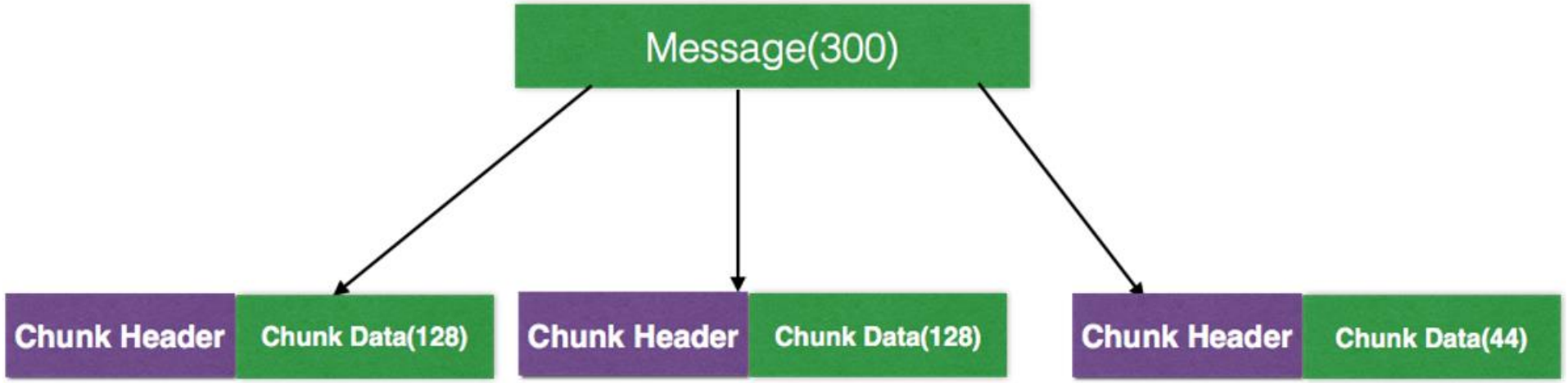


Message & Chunk

拆分的时候，**默认的Chunk Size是128字节**，以Message大小为300字节举例，进行拆分。

300 = 128 + 128 + 44

```
Set Chunk Size 60000
```



Message拆分成Chunk举例

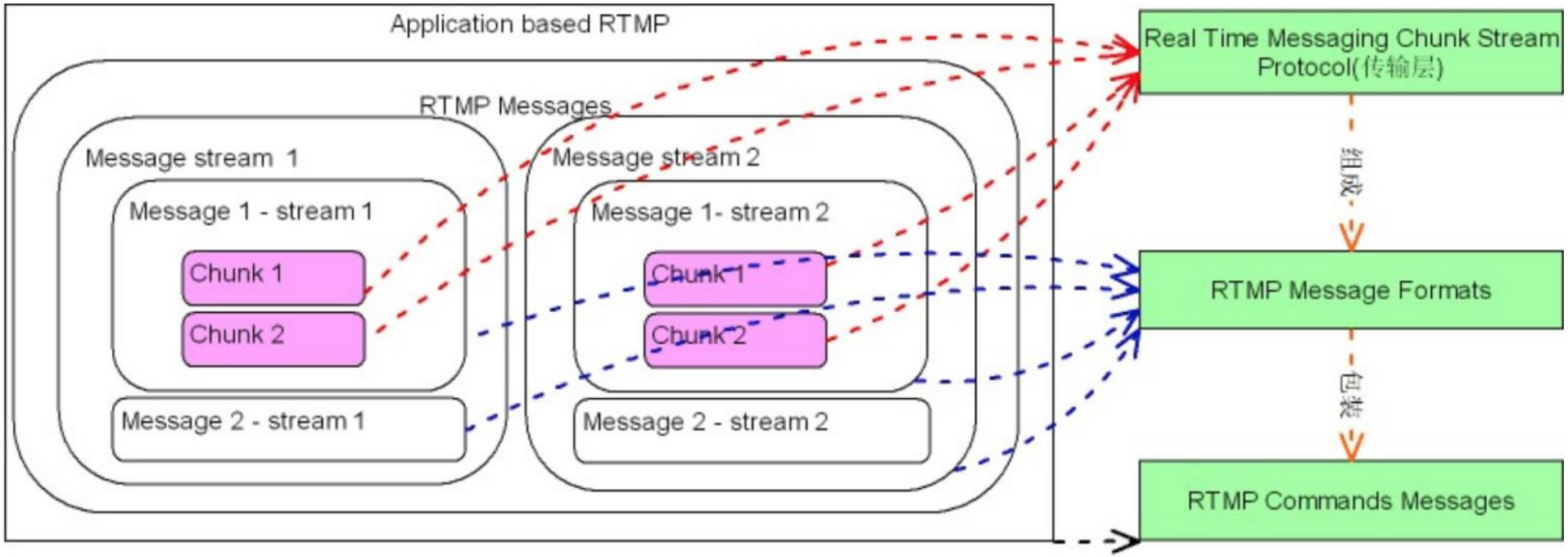
RTMP实质

•发送端

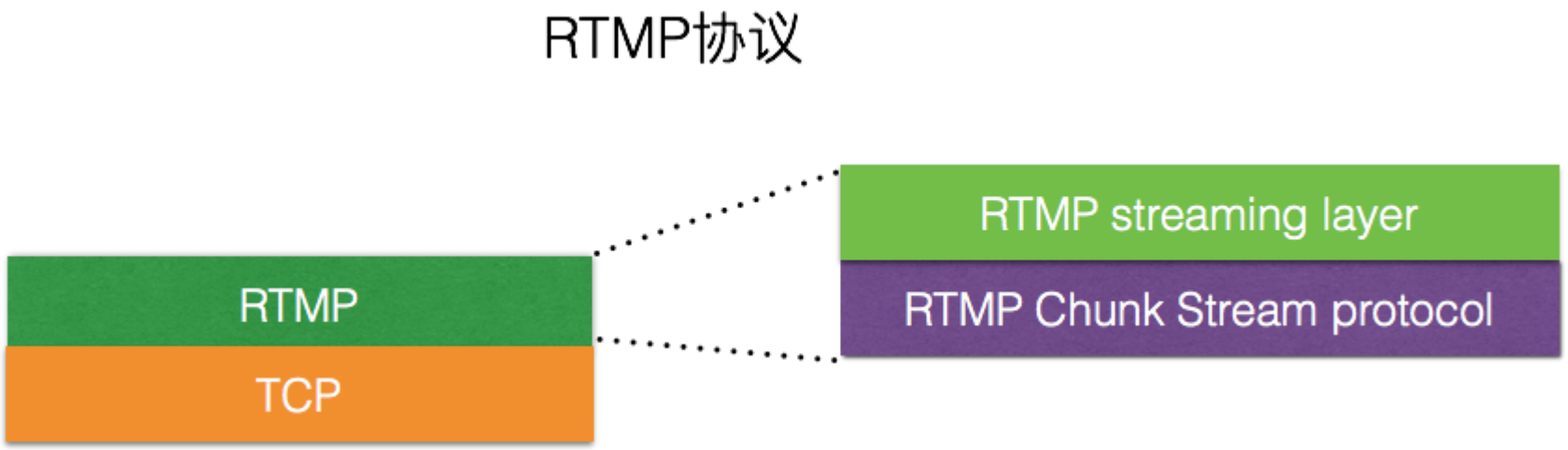
首先将数据加工成消息(中间物)，然后再将消息分割成Chunk(加上Chunk Header)，然后将Chunk通过网络发送出去。

•接收端

接收端将接收到的Chunk组装成消息。



RTMP层次 (协议角度)


RTMP协议
RTMP streaming layer
RTMP Chunk Stream protocol
RTMP
TCP

RTMP Chunk Header

RTMP Chunk Header的长度不是固定的，分为:
12 Bytes、8 Bytes、4 Bytes、1 Byte四种，由RTMP Chunk Header前2位决定。

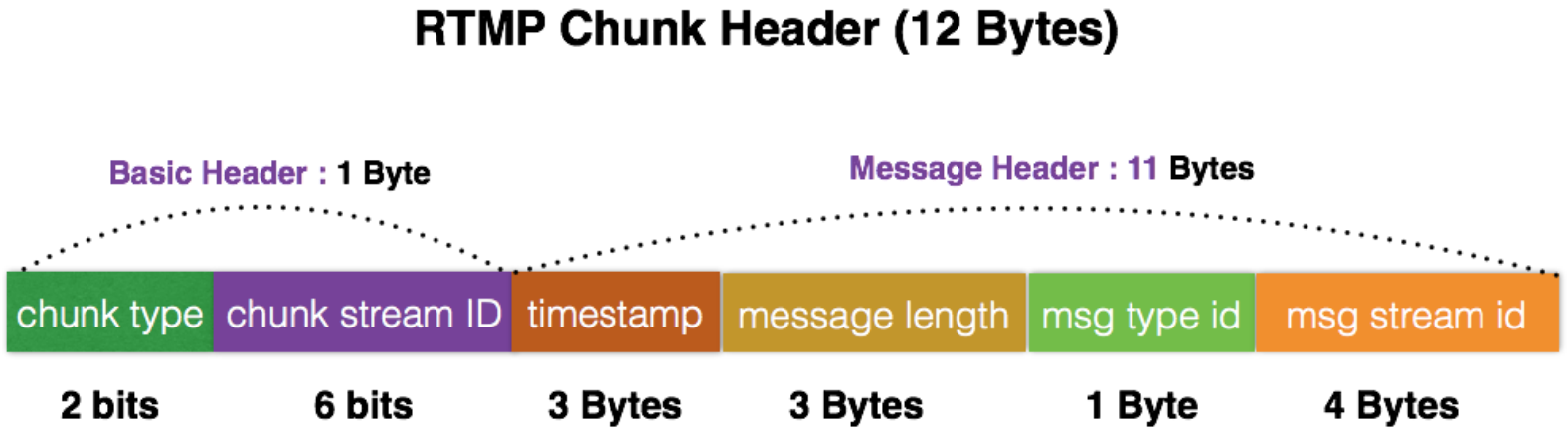
前2 Bits	Header Length	说明
00	12 bytes	onMetaData 流刚开始的绝对时间戳 视频帧 流刚开始的绝对时间戳 音频帧 控制消息 如connect
01	8 bytes	大多数都是这种类型
10	4 bytes	比较少见
11	1 byte	偶尔会出现 频率远远低于8 Bytes

chunk type 与 chunk header length的对应关系

4种type对比

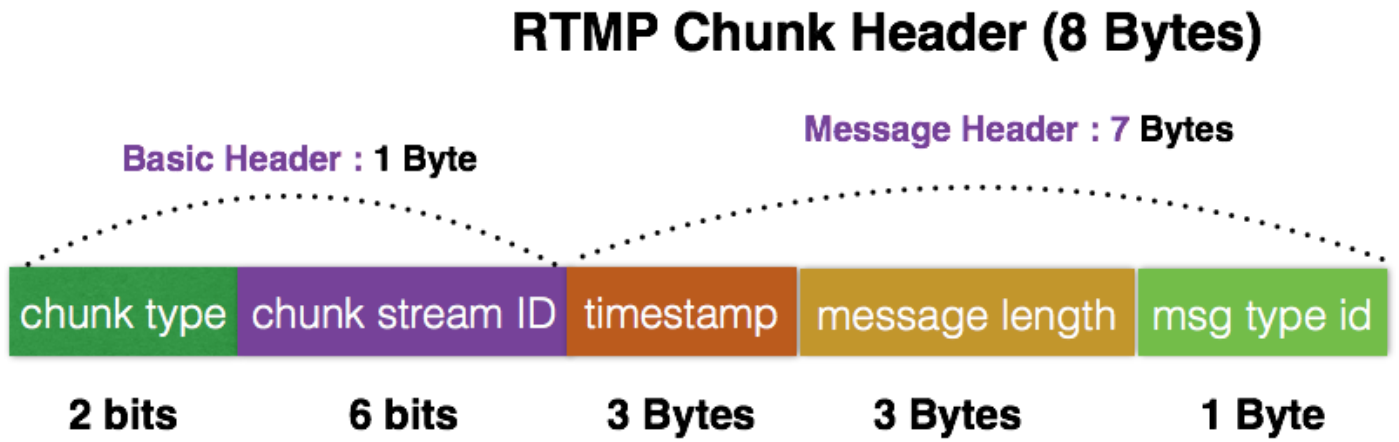

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
timestamp
message length
message length (cont)
message type id
msg stream id
message stream id (cont)
Chunk Message Header - Type 0
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
timestamp delta
message length
message length (cont)
message type id
Chunk Message Header - Type 1
0 1 2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
timestamp delta
Chunk Message Header - Type 2

RTMP Chunk Header



00

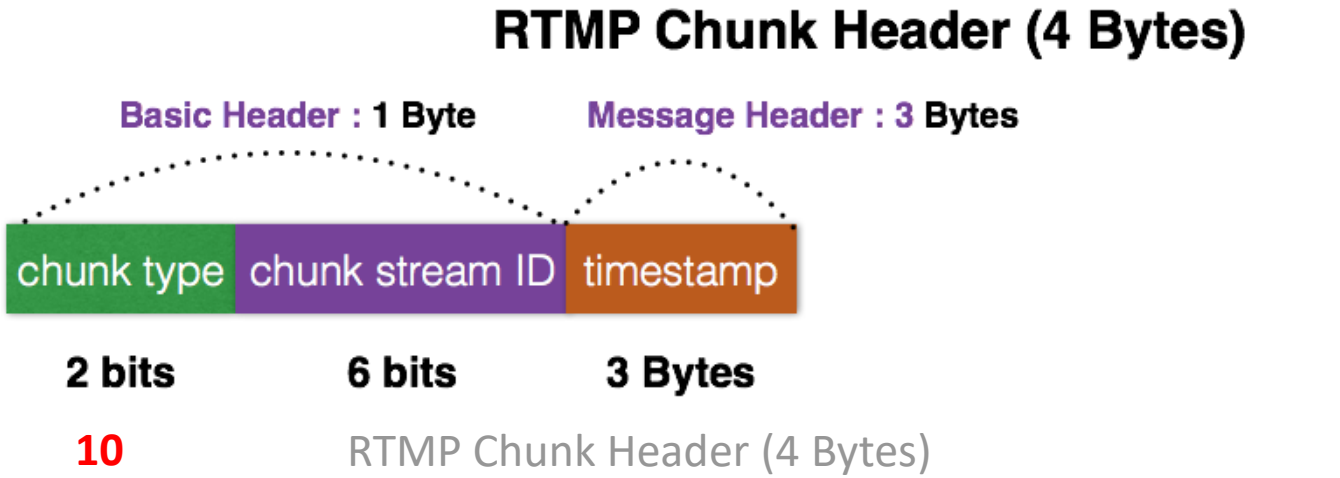
RTMP Chunk Header (12 Bytes)

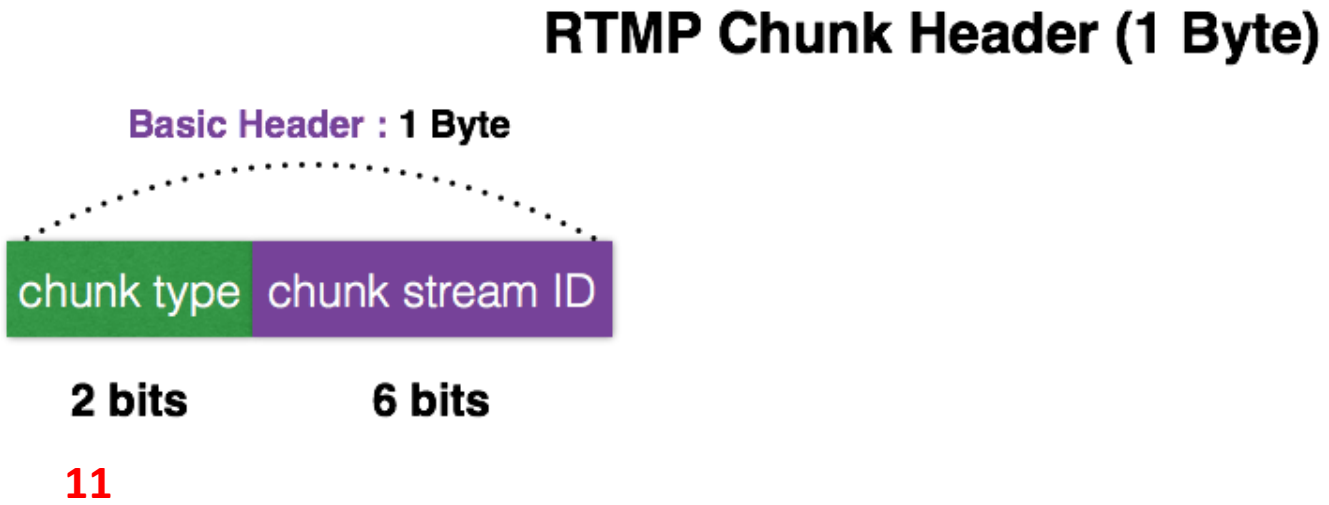


01

RTMP Chunk Header (8 Bytes)

RTMP Chunk Header





RTMP Chunk Header-为什么存在不同长度?

一般情况下，msg stream id是不会变的，所以针对视频或音频， 除了第一个RTMP Chunk Header是12Bytes的，**后续即可采用8Bytes的。（5.3. Chunking）**

如果消息的长度(message length)和类型(msg type id, 如视频为9或音频为8)又相同，即可将这两部分也省去，RTMP Chunk Header采用4Bytes类型的。

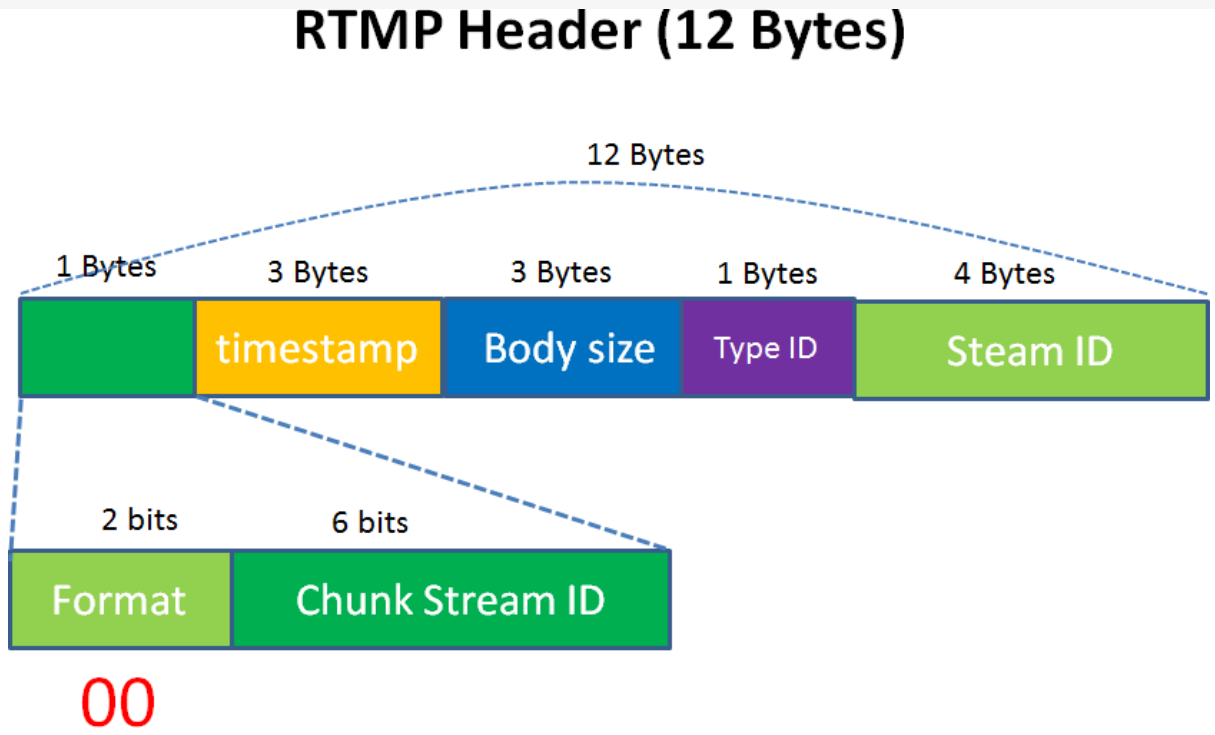
如果当前Chunk与之前的Chunk相比，msg stream id相同，msg type id相同，message length相同，而且都属于同一个消息(由同一个Message切割成)，这类Chunk的时间戳(timestamp)也是相同的，故后续的也可以省去，RTMP Chunk Header采用1 Byte类型的。

当Chunk Size足够大时(一般不这么干)，此时所有的Message都只能相应切割成一个Chunk，该Chunk仅msg stream id相同。此时基本上除了第一个Chunk的Header是12Bytes外，其它所有Chunk的Header都是8Bytes。

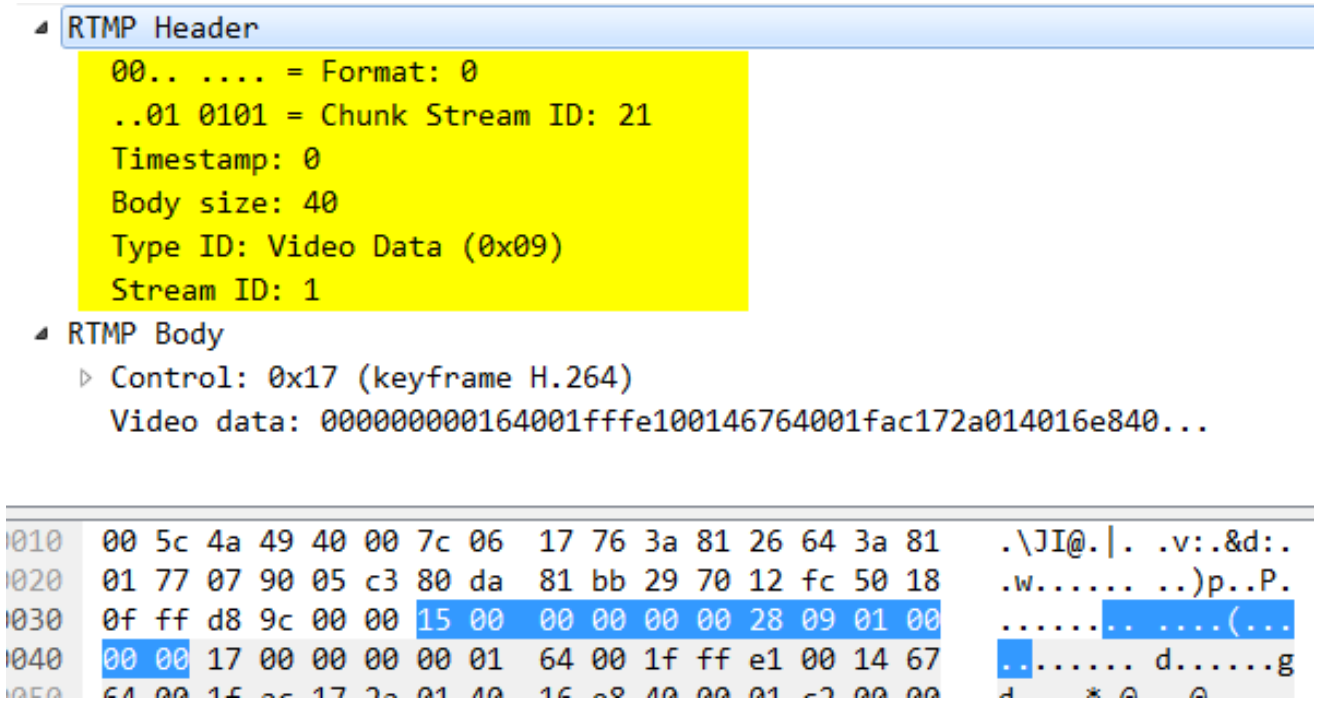
RTMP Chunk Header举例（12 Bytes）

RTMP Header (12 Bytes)

一般只有rtmp流刚开始的metadata、绝对时间戳的视频或音频是12Bytes。



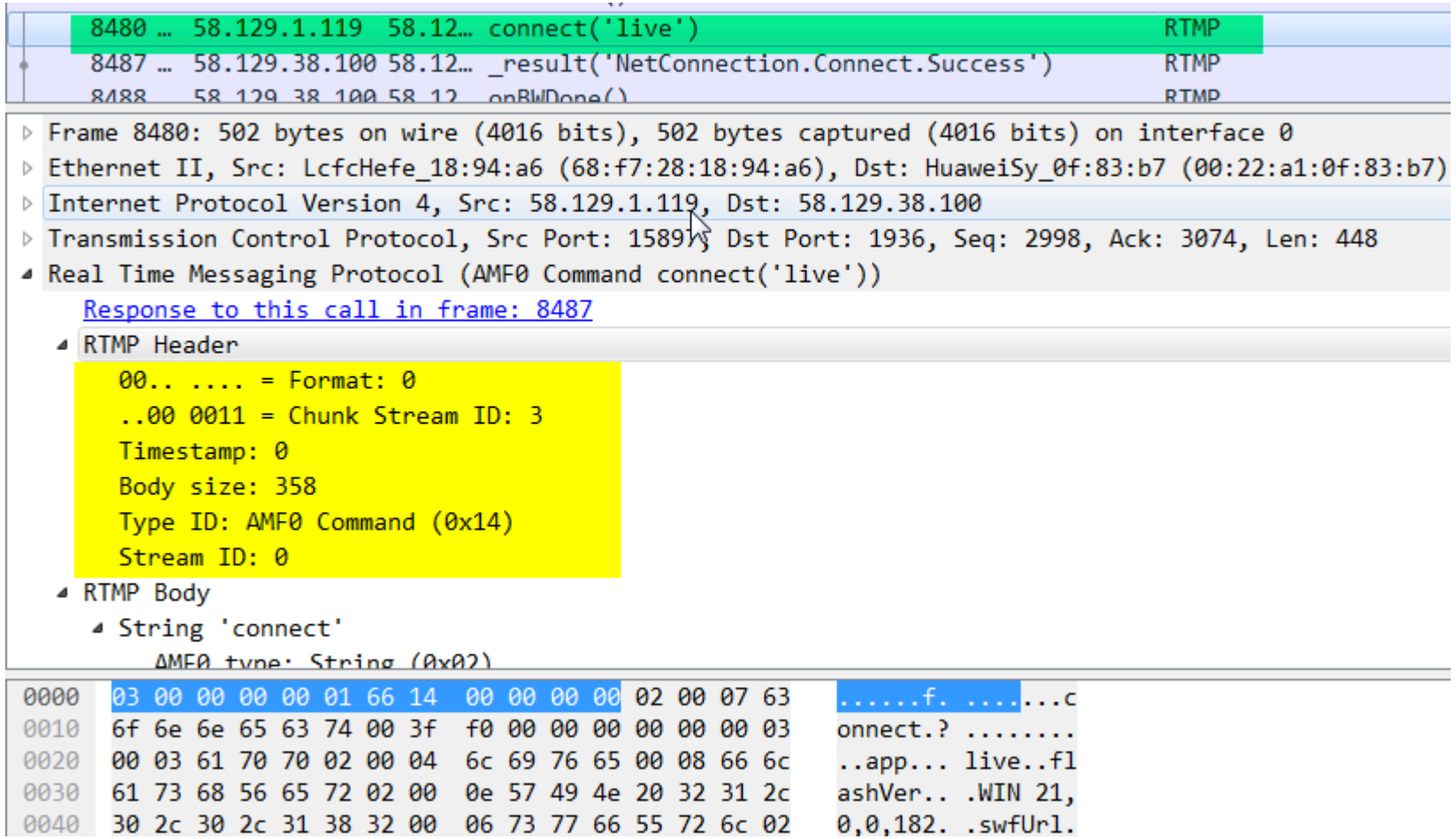
RTMP Chunk Header举例（12 Bytes）



RTMP Chunk Header (12 Bytes) 举例1 - video

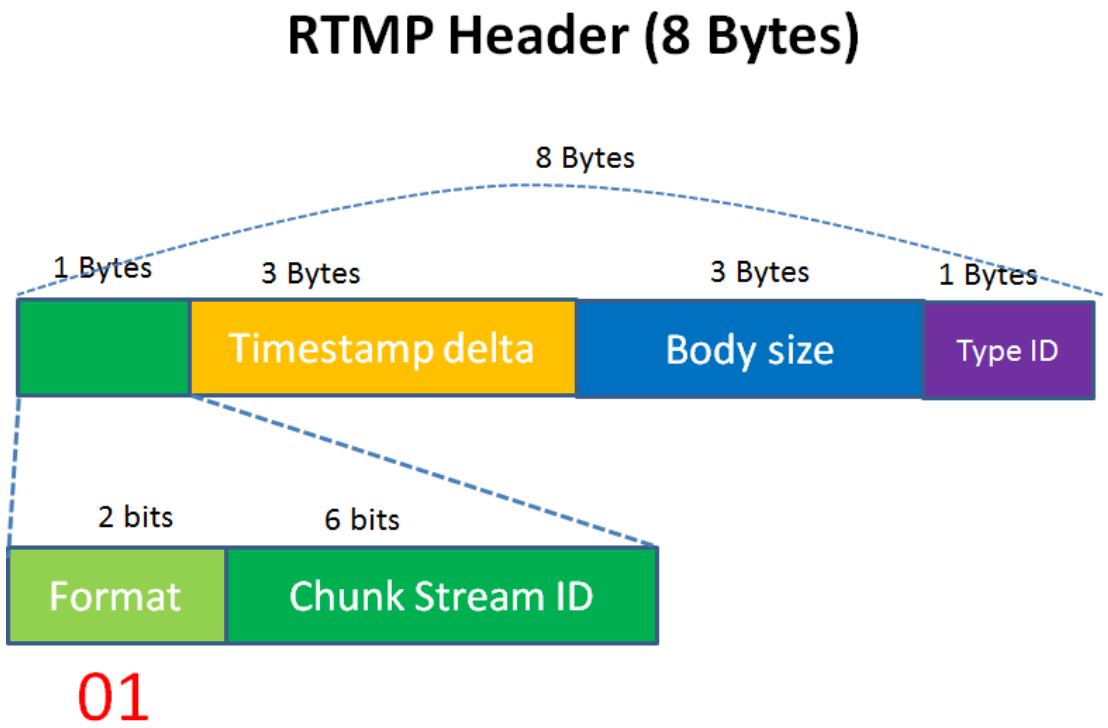
RTMP Chunk Header举例（12 Bytes）

有些控制消息也是12 Bytes，比如connect。



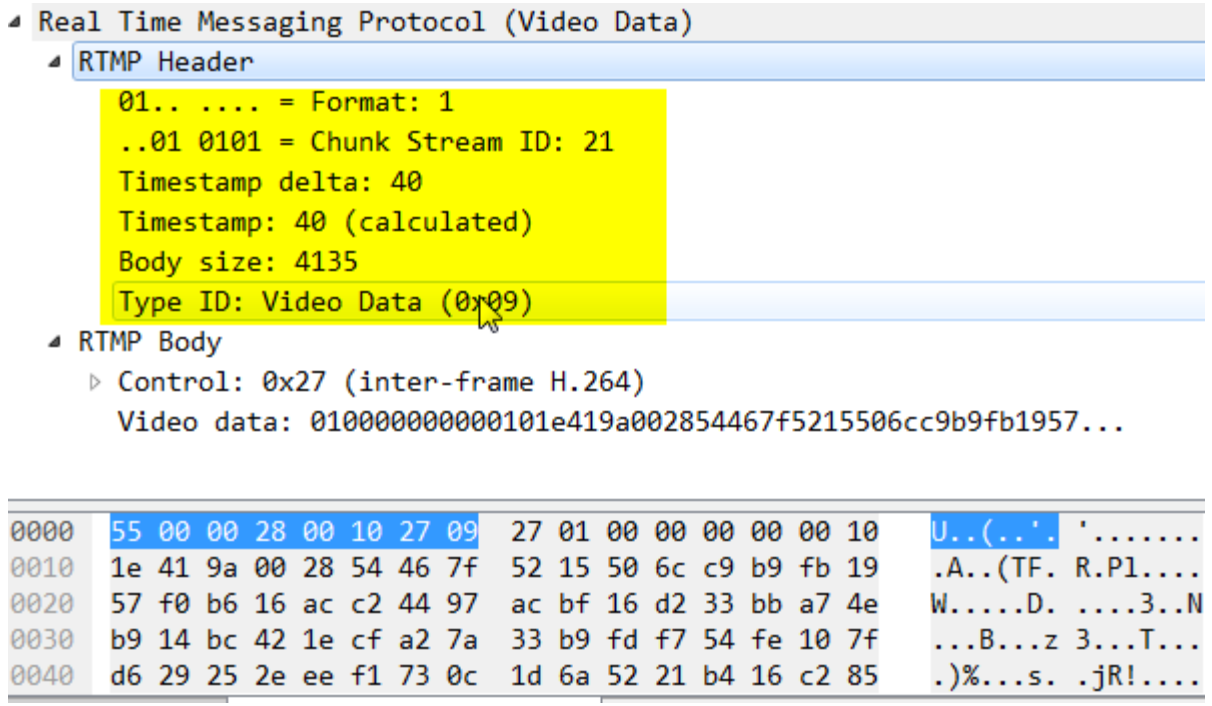
RTMP Chunk Header (12 Bytes) 举例3 - connect

RTMP Chunk Header举例（8 Bytes）



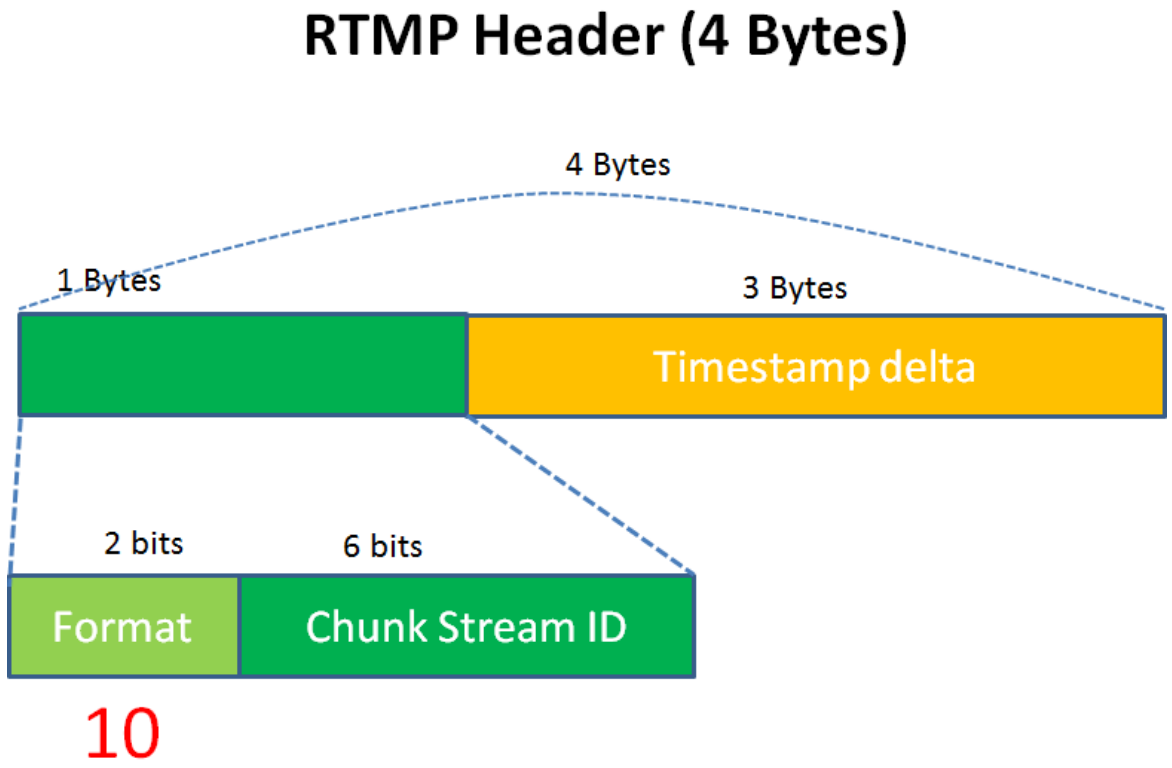
RTMP Chunk Header (8 Bytes)

RTMP Chunk Header举例（8 Bytes）



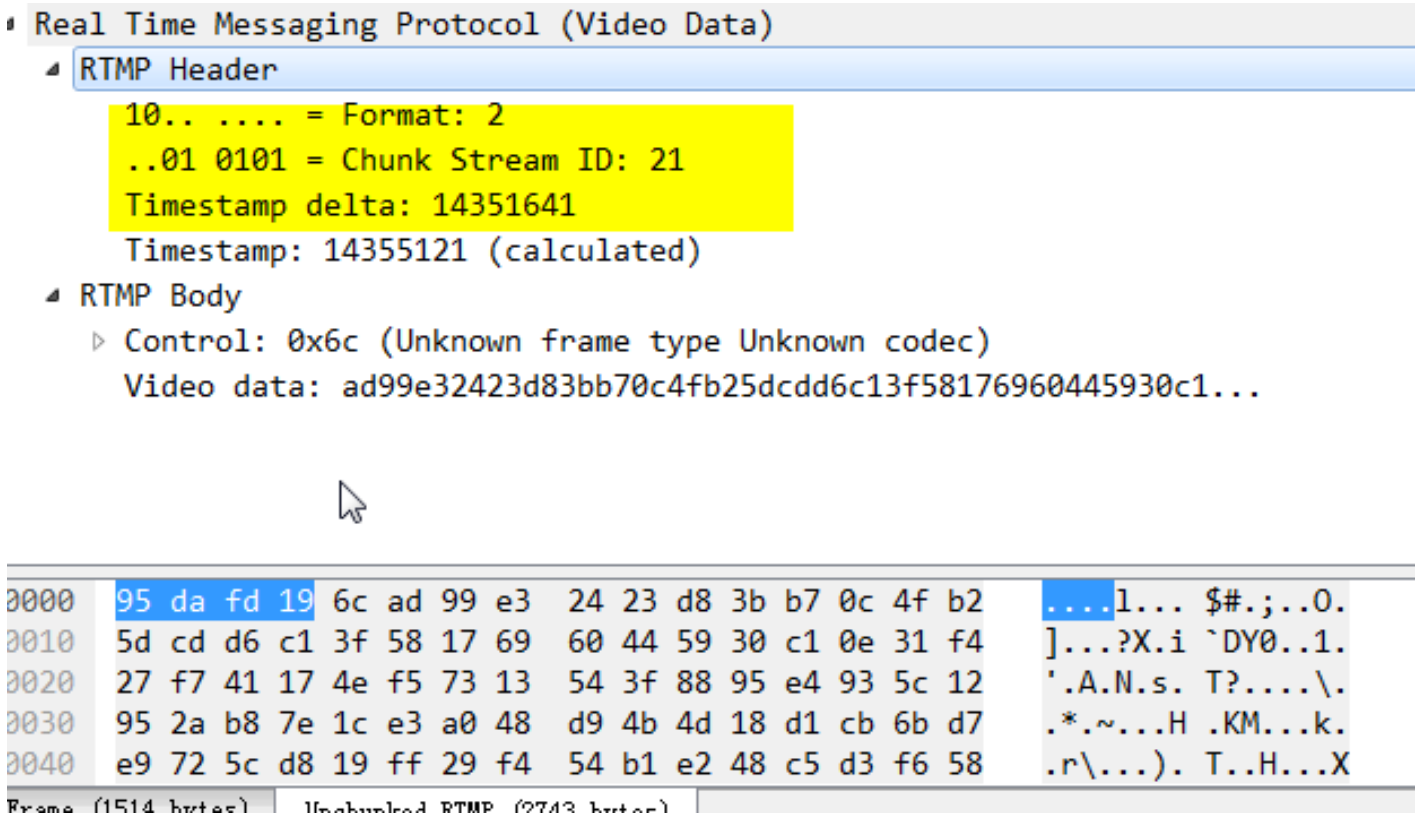
RTMP Chunk Header (8 Bytes) 举例

RTMP Chunk Header举例（4 Bytes）



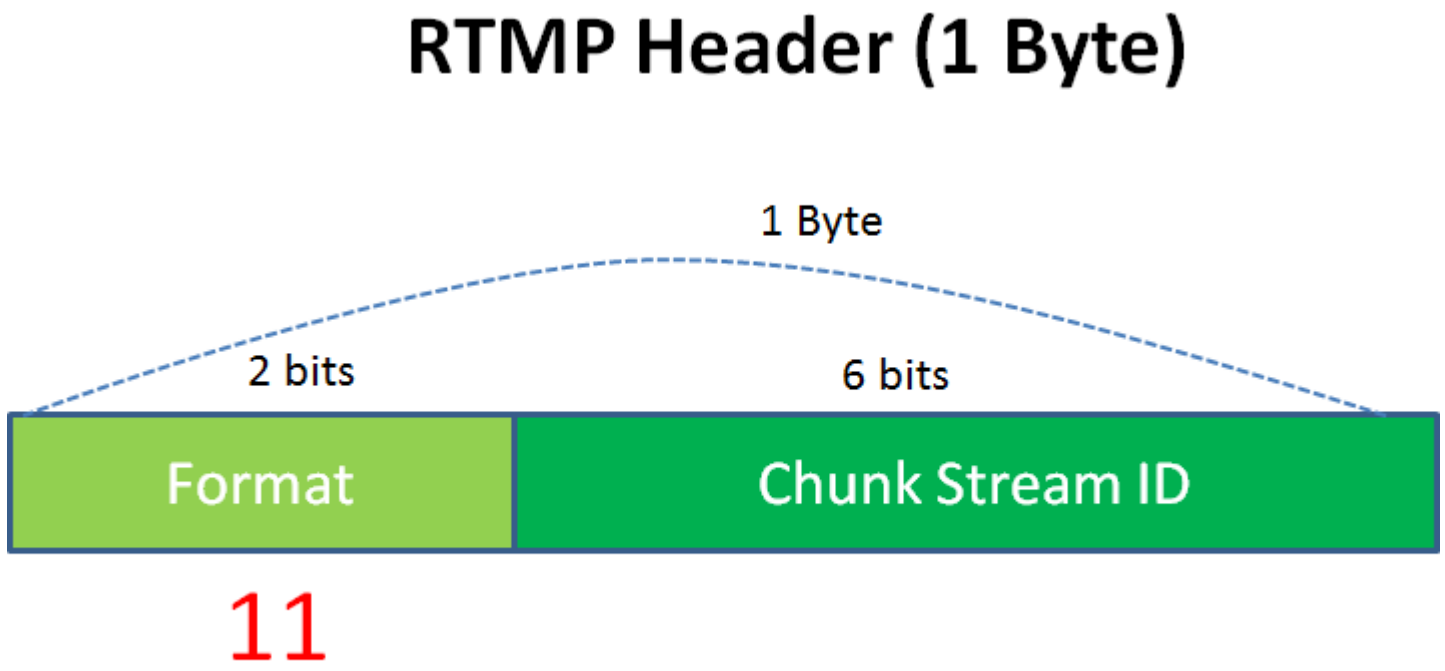
RTMP Chunk Header (4 Bytes)

RTMP Chunk Header举例（4 Bytes）



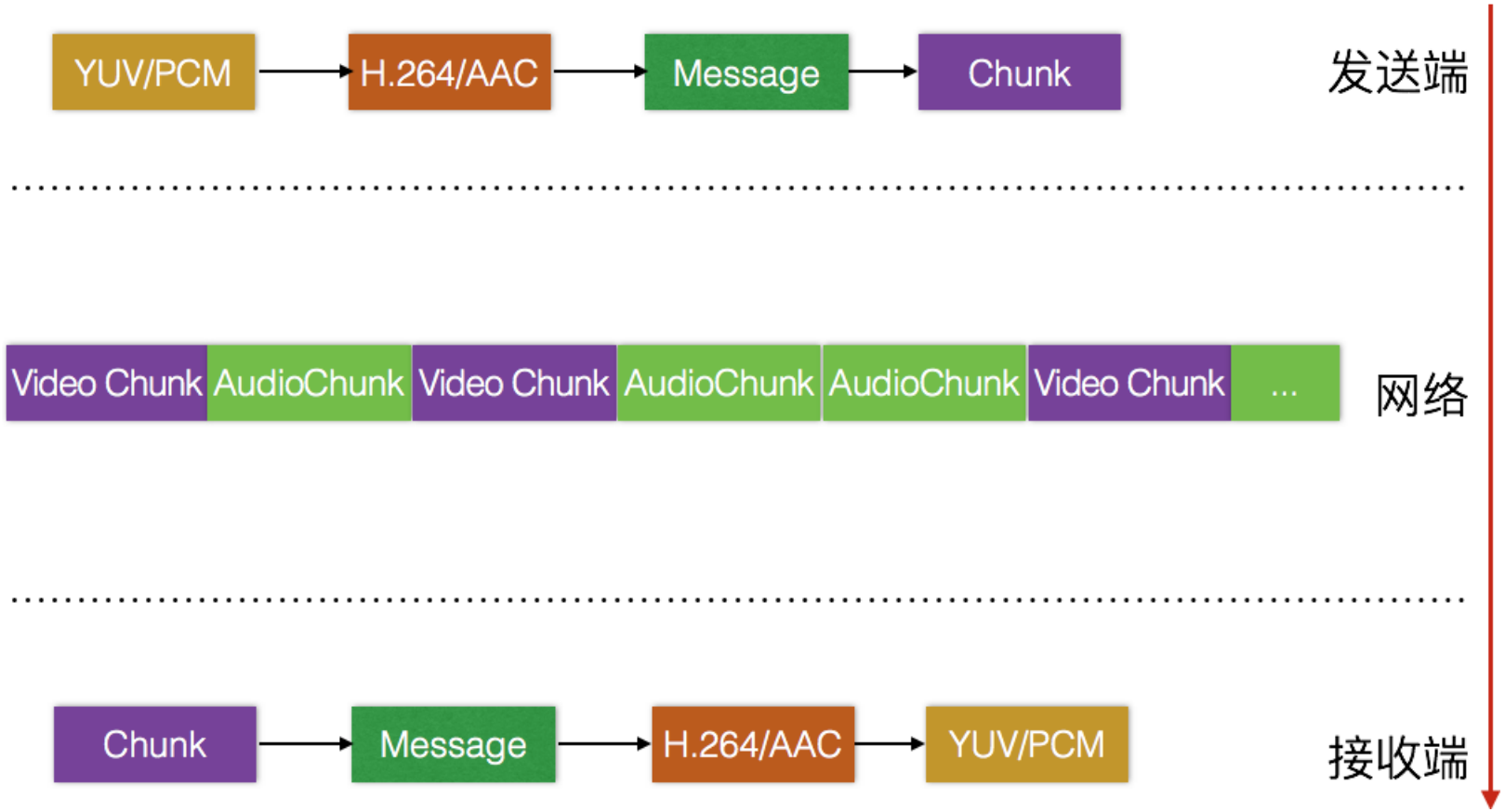
RTMP Chunk Header (4 Bytes) 举例

RTMP Chunk Header举例 （1 Byte）



RTMP Chunk Header (1 Byte)

RTMP传输基本流程



RTMP音视频数据流程

RTMP传输基本流程

发送端

Step 1：把数据封装成消息(Message)。

Step 2：把消息分割成消息块(Chunk，网络中实际传输的内容)。

Step 3：将分割后的消息块(Chunk)通过TCP协议发送出去。

接收端：

Step 1：在通过TCP协议收到数据后，先将消息块重新组合成消息(Message)。

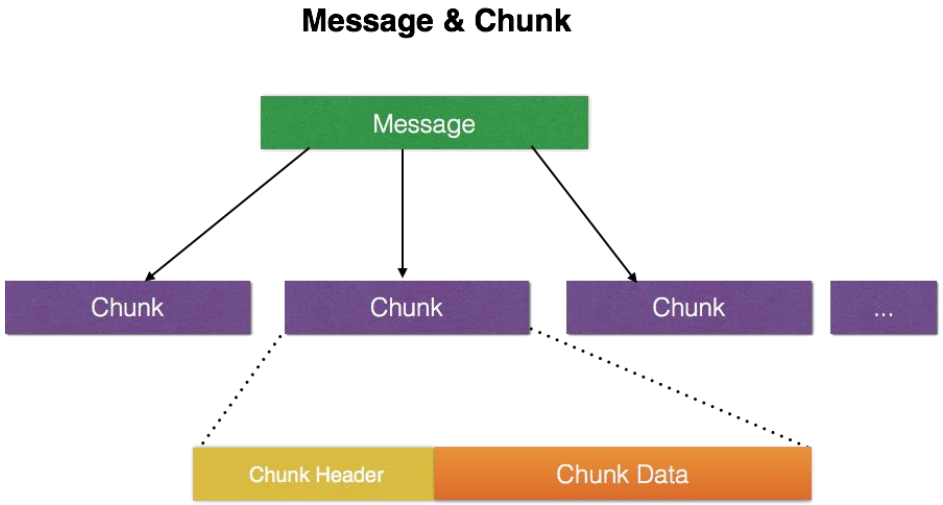
Step 2：通过对消息进行解封装处理就可以恢复出数据。

RTMP为什么要将Message划分Chunk?

在互联网中传输数据时，消息(Message)会被拆分成更小的单元，称为消息块(Chunk).

大的Message被切割成利于在网络上传输的小Chunk

切成小块，可**防止大的数据块(如视频数据)阻塞小的数据块(如音频数据或控制信息)**。



如果一帧1080P I帧数据量为244KBytes，假设带宽为10Mbit/s，传输一帧的耗时为:
244*1024*8/(10*1024*1024) = 0.190625秒=190毫秒
假如要实时传输25帧的I帧文件，即允许每帧传输最大耗时为40毫秒，需要带宽47.65625Mbit，这还没包括传输中的ACK数据。

RTMP消息优先级

在RTMP中，消息(Message)主要分为两大类：控制消息和数据消息。

数据消息中由包括Video消息和Audio消息等。

消息都是怎么进行管理的？

通路只有一条（RTMP是单通路），到底谁先走呢，谁后走呢?

答案是：**分优先级，优先级高的先行**。优先级低的不能阻塞优先级高的。

RTMP消息优先级

delivery of all messages, across multiple streams. RTMP Chunk Stream does not provide any prioritization or similar forms of control, but can be used by higher-level protocols to provide such prioritization. For example, a live video server might choose to drop video messages for a slow client to ensure that audio messages are received in a timely fashion, based on either the time to send or the time to acknowledge each message.

RTMP Chunk Stream层级没有优先级的划分，而是在高层次Message stream提供优先级的划分。

不同类型的消息会被分配不同的优先级，当网络传输能力受限时，优先级用来控制消息在网络底层的排队顺序。

比如当客户端网络不佳时，流媒体服务器可能会选择丢弃视频消息，以保证音频消息可及时送达客户端。

注：5. RTMP Chunk Stream

RTMP消息优先级

Chunking allows large messages at the higher-level protocol to be broken into smaller messages, for example to prevent large low-priority messages (such as video) from blocking smaller high-priority messages (such as audio or control).

RTMP Chunk Stream层级允许在Message stream层次，将大消息切割成小消息，这样可以避免大的低优先级的消息(如视频消息)阻塞小的高优先级的消息(如音频消息或控制消息)。

注：5.3. Chunking

RTMP消息优先等级

RTMP消息优先等级

RTMP消息优先级 协议控制消息

5.4. Protocol Control Messages

RTMP Chunk Stream uses message type IDs 1, 2, 3, 5, and 6 for protocol control messages. These messages contain information needed by the RTMP Chunk Stream protocol.

These protocol control messages MUST have message stream ID 0 (known as the control stream) and be sent in chunk stream ID 2. Protocol control messages take effect as soon as they are received; their

Protocol Control Messages属于RTMP Chunk Stream层级的控制消息，用于该协议的内部控制。

RTMP消息优先级 用户控制消息

Protocol Control Messages属于RTMP Chunk Stream层级的控制消息，用于该协议的内部控制。

RTMP消息优先级 用户控制消息

5.4. Protocol Control Messages

RTMP Chunk Stream uses message type IDs 1, 2, 3, 5, and 6 for protocol control messages. These messages contain information needed by the RTMP Chunk Stream protocol.

These protocol control messages MUST have message stream ID 0 (known as the control stream) and be sent in chunk stream ID 2. Protocol control messages take effect as soon as they are received; their

User Control Messages (4)是RTMP streaming layer(即Message stream层次)的消息。

RTMP消息优先级 总结

协议先行

协议控制消息(Protocol Control Messages)和用户控制消息(User Control Messages)应该包含消息流ID 0(控制流）和块流ID 2，并且**有最高的发送优先级**。

数据次之

数据消息(音频信息、音频消息)比控制信息的优先级低。

另外，一般情况下，音频消息比视频数据优先级高。

RTMP协议 - 时间戳

基本介绍

- RTMP中时间戳的单位为**毫秒(ms)**
- 时间戳为**相对于某个时间点的相对值**
- 时间戳的长度为**32bit**，不考虑回滚的话，最大可表示**49天17小时2分钟47.296秒**
- Timestamp delta单位也是毫秒，为相对于前一个时间戳的一个无符号整数； 可能为24bit或32bit

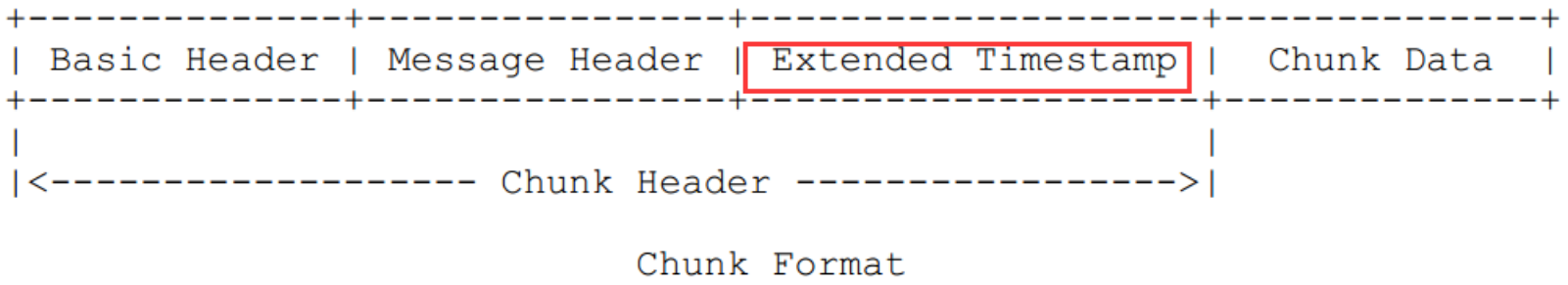
Message时间戳

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

- RTMP Message的时间戳 4个字节
- 大端存储

Chunk时间戳

Chunk Format



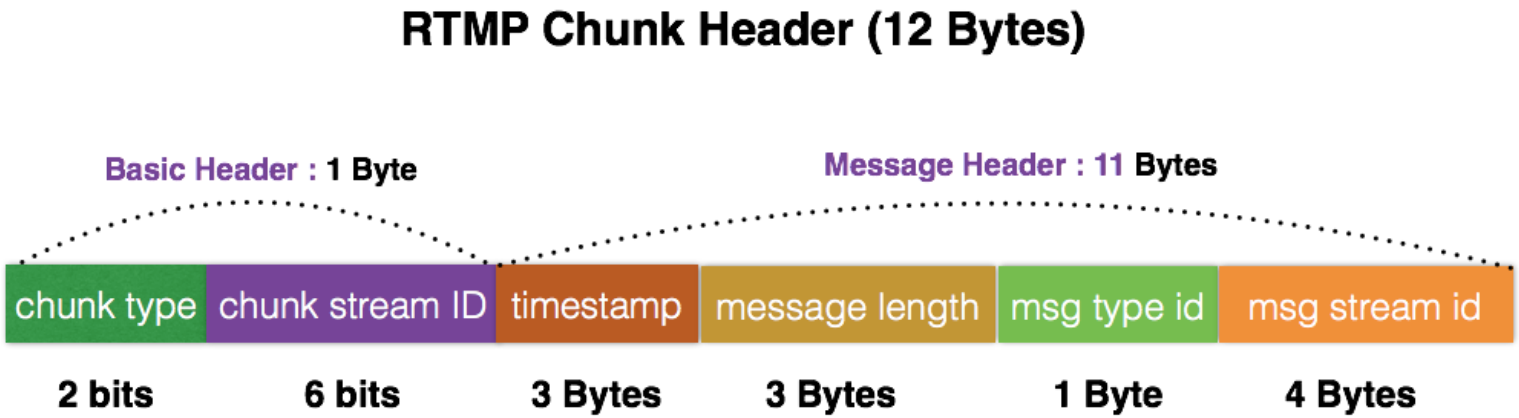
Chunk Format

用wireshark转包分析发现，**rtmp流的chunk视频流(或音频流)除第一个视频时间戳为绝对时间戳外，后续的时间戳均为timestamp delta，即当前时间戳与上一个时间戳的差值。**

比如帧率为25帧/秒的视频流，timestamp delta基本上都为40ms。

Chunk时间戳

通常情况下，Chunk的时间戳(包括绝对时间戳和Timestamp delta)是3个字节。
但时间戳值超过0xFFFFFF时，启用Extended Timestamp(4个字节)来表示时间戳。
通常情况下 -- 3字节



三字节的timestamp可能为绝对**timestamp**或**timestamp delta**。

Chunk时间戳

timestamp delta (3 bytes): For a type-1 or type-2 chunk, the difference between the previous chunk's timestamp and the current chunk's timestamp is sent here.
If the delta is greater than or equal to 16777215 (hexadecimal 0xFFFFFF), this field MUST be 16777215, indicating the presence of the Extended Timestamp field to encode the full 32 bit delta. Otherwise, this field SHOULD be the actual delta.

timestamp delta的值超过16777215（即16进制的0xFFFFFF)时，这时候这三个字节必须被置为：**0xFFFFFF**，以此来标示Extended Timestamp(4字节)将会存在，由Extended Timestamp来表示时间戳。

FLV+H264+AAC

见：
5-AAC ADTS格式分析.pdf
6-H264 NALU分析.pdf
7-FLV格式分析.pdf

零声学院

www.0voice.com

一切只为渴望更优秀的你！

为您的职业
添砖加瓦
升职加薪

努力方向

系统提升

项目实战

全职指导