05-SRS流媒体服务器-RTMP协议

课程链接： https://ke.qq.com/course/468797?tuin=137bb271

开发环境

推流：Windows通过FFmpeg命令行推流
服务器：使用ubuntu上的srs流媒体服务器
协议交互分析：使用wireshark抓包分析
srs服务器细节：使用gdb进行分析
tcpdump：
sudo tcpdump –i any port 1935 –XX and dst 175.0.54.116 抓取服务器回应的数据

具体步骤分析：
（1）以前台的方式运行srs流媒体服务器
srs.3.0–20200720/trunk$ ./objs/srs –c ./conf/srs.conf
以及使用gdb去调试
（2）启动wireshark，过滤rtmp协议，设置的是rtmpt关键字
（3）使用ffmpeg推流，推流20秒后断开
ffmpeg –re –i source.200kbps.768x320.flv –t 20 –vcodec copy –acodec copy –f flv –y rtmp://111.229.231.225/live/livestream
ffmpeg –re –i source.200kbps.768x320.flv –vcodec copy –acodec copy –f flv –y rtmp://111.229.231.225/live/livestream
（4）分析wireshark抓的包
（5）分析srs流媒体服务器的log

ubuntu 16.04 直接安装ffmpeg命令，如果已经安装过了不用安装，命令安装的版本比较老，主要是用来测试命令。

```
sudo add-apt-repository ppa:djcj/hybrid
sudo apt-get update
sudo apt-get -y install ffmpeg
```

srs.3.0–20200720/trunk/doc目录下有source.200kbps.768x320.flv

srs流媒体log

[2020–07–31 19:48:19.032][Trace][1761][412] RTMP client ip=175.0.54.116, fd=10

[2020-07-31 19:48:19.058][Trace][1761][412] complex handshake success
[2020-07-31 19:48:19.079][Trace][1761][412] connect app, tcUrl=rtmp://111.229.231.225:1935/live, pageUrl=, swfUrl=, schema=rtmp, vhost=111.229.231.225, port=1935, app=live, args=null
[2020-07-31 19:48:19.079][Trace][1761][412] protocol in.buffer=0, in.ack=0, out.ack=0, in.chunk=128, out.chunk=128
[2020-07-31 19:48:19.221][Trace][1761][412] client identified, type=fmle-publish, vhost=111.229.231.225, app=live, stream=livestream, param=, duration=0ms
[2020-07-31 19:48:19.221][Trace][1761][412] connected stream, tcUrl=rtmp://111.229.231.225:1935/live, pageUrl=, swfUrl=, schema=rtmp, vhost=__defaultVhost__, port=1935, app=live, stream=livestream, param=, args=null
[2020-07-31 19:48:19.221][Trace][1761][412] source url=/live/livestream, ip=175.0.54.116, cache=1, is_edge=0, source_id=-1[-1]
[2020-07-31 19:48:19.361][Trace][1761][412] hls: win=60000ms, frag=10000ms, prefix=, path=./objs/nginx/html, m3u8=[app]/[stream].m3u8, ts=[app]/[stream]-[seq].ts, aof=2.00, floor=0, clean=1, waitk=1, dispose=0ms, dts_directly=1
[2020-07-31 19:48:19.361][Trace][1761][412] ignore disabled exec for vhost=__defaultVhost__
[2020-07-31 19:48:19.361][Trace][1761][412] start publish mr=0/350, p1stpt=20000, pnt=5000, tcp_nodelay=0
[2020-07-31 19:48:19.501][Trace][1761][412] got metadata, width=1920, height=832, vcodec=7, acodec=10
[2020-07-31 19:48:19.501][Trace][1761][412] 50B video sh, codec(7, profile=High, level=4, 1920x832, 0kbps, 0.0fps, 0.0s)
[2020-07-31 19:48:19.501][Trace][1761][412] 4B audio sh, codec(10, profile=LC, 2channels, 0kbps, 48000HZ), flv(16bits, 2channels, 44100HZ)
[2020-07-31 19:48:19.514][Trace][1761][412] -> HLS time=76056186ms, sno=2, ts=livestream-1.ts, dur=0.00, dva=0p
[2020-07-31 19:48:28.567][Trace][1761][412] -> HLS time=86059973ms, sno=2, ts=livestream-1.ts, dur=0.00, dva=9120p
[2020-07-31 19:48:38.606][Trace][1761][412] -> HLS time=96062310ms, sno=3, ts=livestream-2.ts, dur=0.00, dva=6320p
[2020-07-31 19:48:39.449][Warn][1761][412][11] VIDEO: stream not monotonically increase, please open mix_correct.
[2020-07-31 19:48:39.450][Trace][1761][412] cleanup when unpublish
[2020-07-31 19:48:39.450][Warn][1761][412][104] client disconnect peer. ret=1007

wireshark

推流

Source	Destination	Protocol	Length	Info
192.168.2.223	111.229.231.225	RTMP	167	Handshake C0+C1
111.229.231.225	192.168.2.223	RTMP	727	Handshake S0+S1+S2
192.168.2.223	111.229.231.225	RTMP	166	Handshake C2
192.168.2.223	111.229.231.225	RTMP	212	connect('live')
111.229.231.225	192.168.2.223	RTMP	70	Window Acknowledgement Size 2500000
111.229.231.225	192.168.2.223	RTMP	584	Set Peer Bandwidth 2500000,Dynamic|Set Chunk Size 60000
192.168.2.223	111.229.231.225	RTMP	210	Set Chunk Size 60000|releaseStream('livestream')|FCPublish('livestream')|createStream()|_checkbw()
111.229.231.225	192.168.2.223	RTMP	87	_result()
111.229.231.225	192.168.2.223	RTMP	128	_result()|_result()
192.168.2.223	111.229.231.225	RTMP	94	publish('livestream')
111.229.231.225	192.168.2.223	RTMP	168	onFCPublish()
192.168.2.223	111.229.231.225	RTMP	1478	Unknown (0x0)|Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1393	Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1381	Audio Data|Set Chunk Size -2080374761|Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	300	Audio Data|Audio Data|Video Data
192.168.2.223	111.229.231.225	RTMP	237	Audio Data|Audio Data
192.168.2.223	111.229.231.225	RTMP	1478	Video Data|Audio Data|Audio Data|Video Data|Audio Data|Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	365	Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	375	Audio Data
192.168.2.223	111.229.231.225	RTMP	770	Audio Data
192.168.2.223	111.229.231.225	RTMP	325	Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1478	Unknown (0x0)|Unknown (0x0)|Unknown (0x0)|Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1478	Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1043	Audio Data|Audio Data|Unknown (0x0)|Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	90	Video Data

192.168.2.223	111.229.231.225	RTMP	1449	Audio Data|Audio Data|Unknown (0x0)|Unknown (0x0)|Unknown (0x0)|Audio Data
192.168.2.223	111.229.231.225	RTMP	1478	Audio Data|Audio Data
192.168.2.223	111.229.231.225	RTMP	190	Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1329	Unknown (0x0)|Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	1478	Audio Data|Audio Data
192.168.2.223	111.229.231.225	RTMP	142	Audio Data
192.168.2.223	111.229.231.225	RTMP	961	Unknown (0x0)
192.168.2.223	111.229.231.225	RTMP	336	Audio Data|Audio Data|Video Data
192.168.2.223	111.229.231.225	RTMP	840	Audio Data|Audio Data
192.168.2.223	111.229.231.225	RTMP	163	Video Data
192.168.2.223	111.229.231.225	RTMP	350	Audio Data|Audio Data|Video Data|FCUnpublish()|deleteStream()
111.229.231.225	192.168.2.223	RTMP	171	onFCUnpublish()
111.229.231.225	192.168.2.223	RTMP	271	_result()

stream id何时由服务器返回来：客户端发送createStream后返回stream id

拉流

192.168.2.223	111.229.231.225	RTMP	167	Handshake C0+C1
111.229.231.225	192.168.2.223	RTMP	727	Handshake S0+S1+S2
192.168.2.223	111.229.231.225	RTMP	166	Handshake C2
192.168.2.223	111.229.231.225	RTMP	269	connect('live')
111.229.231.225	192.168.2.223	RTMP	70	Window Acknowledgement Size 2500000
111.229.231.225	192.168.2.223	RTMP	584	Set Peer Bandwidth 2500000,Dynamic|Set Chunk Size 60000
192.168.2.223	111.229.231.225	RTMP	120	Window Acknowledgement Size 2500000|createStream()|_checkbw()
111.229.231.225	192.168.2.223	RTMP	95	_result()
192.168.2.223	111.229.231.225	RTMP	164	getStreamLength()|play('livestream')|Set Buffer Length 1,3000ms
111.229.231.225	192.168.2.223	RTMP	72	Stream Begin 1
111.229.231.225	192.168.2.223	RTMP	1254	Unknown (0x0)|Unknown (0x0)
111.229.231.225	192.168.2.223	RTMP	1254	onStatus('NetStream.Play.Reset')
111.229.231.225	192.168.2.223	RTMP	1254	Unknown (0x0)
111.229.231.225	192.168.2.223	RTMP	1254	Unknown (0x0)|Unknown (0x0)|Unknown (0x0)
111.229.231.225	192.168.2.223	RTMP	1254	Unknown (0x0)
111.229.231.225	192.168.2.223	RTMP	1254	Unknown (0x0)|Unknown (0x0)|Unknown (0x0)

SRS常用类说明

tcp –> chunk – message – packet

SrsCommonMessage

封装rtmp消息

SrsPacket基类

解读取到的数据先暂存到包里。

对应的派生类：
SrsConnectAppPacket
SrsConnectAppResPacket
SrsCallPacket
SrsCallResPacket
SrsCreateStreamPacket
SrsCreateStreamResPacket
SrsCloseStreamPacket
SrsFMLEStartPacket
SrsFMLEStartResPacket
SrsPublishPacket
SrsPausePacket
SrsPlayPacket
SrsPlayResPacket
SrsOnBWDonePacket
SrsOnStatusCallPacket
SrsBandwidthPacket
SrsOnStatusDataPacket
SrsSampleAccessPacket
SrsOnMetaDataPacket
SrsSetWindowAckSizePacket
SrsAcknowledgementPacket
SrsSetChunkSizePacket
SrsSetPeerBandwidthPacket
SrsUserControlPacket

C->S S->C handshake握手

解决RTMP版本一致性的问题

33	8.103754	192.168.0.106	111.229.231.225	RTMP	1591	Handshake C0+C1
38	8.130839	111.229.231.225	192.168.0.106	RTMP	727	Handshake S0+S1+S2
40	8.130930	192.168.0.106	111.229.231.225	RTMP	1590	Handshake C2

C->S connect客户端连接服务器端

客户端连接服务器端

42	8.151838	192.168.0.106	111.229.231.225	RTMP	212 connect('live')

```
Real Time Messaging Protocol (AMF0 Command connect('live'))
  RTMP Header
      00.. .... = Format: 0
      ..00 0011 = Chunk Stream ID: 3
      Timestamp: 0
      Body size: 145
      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 (4 items)
        AMF0 type: Object (0x03)
      > Property 'app' String 'live'
      > Property 'type' String 'nonprivate'
      > Property 'flashVer' String 'FMLE/3.0 (compatible; Lavf58.29.100)'
      > Property 'tcUrl' String 'rtmp://111.229.231.225:1935/live'
        End Of Object Marker
```

- Chunk stream Id：接收端根据相同的chunk stream id拼装出message， **这里是3，对应ffmpeg的枚举RTMP_SYSTEM_CHANNEL**

- Type ID （即是message type id） 比如8音频，9视频，20命令消息，**这里是20命令消息，对应ffmpeg的枚举**RTMP_PT_INVOKE
- Stream ID：

这里属于connect，用于客户端向服务器发送连接请求

握手之后先发送一个connect 命令消息，这些信息是以AMF格式发送的,消息的结构如下：

字段	类型	说明
Command Name(命令名字)	String	命令的名字，如"connect"
Transaction ID(事务ID)	Number	恒为1
Command Object(命令包含的参数对象)	Object	键值对集合表示的命令参数
Optional User Arguments（额外的用户参数)	Object	用户自定义的额外信息

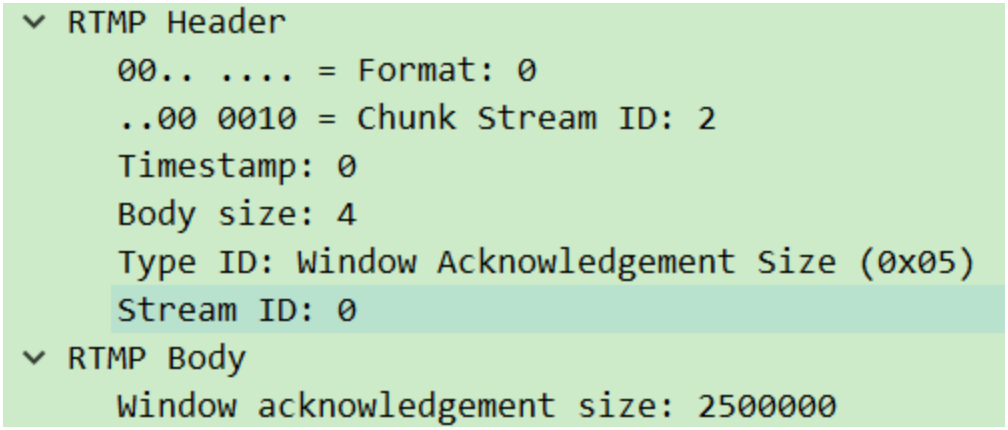
第三个字段中的Command Object中会涉及到很多键值对，使用时可以参考协议的官方文档。
消息的回应有两种，_result表示接受连接，_error表示连接失败。
以下是连接命令对象中使用的名称–值对的描述：

属性	**类型**	**描述**	**范例**
app	字符串	客户端连接到的服务器端应用的名字。	live
flashver	字符串	Flash Player 版本号。和 ApplicationScript getversion() 方法返回的是同一个字符串。	FMLE/3.0(compatible; Lavf58.29.100)
swfUrl	字符串	进行当前连接的 SWF 文件源地址。	file://C:/FlvPlayer.swf
type	字符	判断类型	nonprivate
tcUrl	字符串	服务器 URL。具有以下格式：protocol://servername:port/appName/appInstance	rtmp://localhost:1935/live
fpad	布尔	如果使用了代理就是 true。	true 或者 false。
audioCodecs	数字	表明客户端所支持的音频编码。	SUPPORT_SND_MP3
videoCodecs	数字	表明支持的视频编码。	SUPPORT_VID_SORENSON
videoFunction	数字	表明所支持的特殊视频方法。	SUPPORT_VID_CLIENT_SEEK

pageUrl	字符串	SWF 文件所加载的网页 URL。	http://somehost/sample.html
objectEncoding	数字	AMF 编码方法。	AMF3

S->C Window Acknowledgement Size

通知对方收到 一定的字节数后需要回应发送方。



Type Id：0x5 对应ffmepg RTMP_PT_WINDOW_ACK_SIZE
Window Acknowledgement Size用于设置窗口确认大小，比如这里是服务器发送给客户端的，当客户端收到每次该size就要Acknowledgement (0x3)。
对于ffmpeg，在接收到Window Acknowledgement Size的一半后发送确认包（Acknowledgement），以确保对等方可以继续发送而不等待确认。

```
1  static int handle_window_ack_size(URLContext *s, RTMPPacket *pkt)
2  {
3      RTMPContext *rt = s->priv_data;
4
5      if (pkt->size < 4) {
6          av_log(s, AV_LOG_ERROR,
7                 "Too short window acknowledgement size packet (%d)\n"
   ,
8                 pkt->size);
9          return AVERROR_INVALIDDATA;
10     }
11
12     rt->receive_report_size = AV_RB32(pkt->data);
```

```
13      if (rt->receive_report_size <= 0) {
14          av_log(s, AV_LOG_ERROR, "Incorrect window acknowledgement si
    ze %d\n",
15                 rt->receive_report_size);
16          return AVERROR_INVALIDDATA;
17      }
18      av_log(s, AV_LOG_DEBUG, "Window acknowledgement size = %d\n", rt
    ->receive_report_size);
19      // Send an Acknowledgement packet after receiving half the maxim
    um
20      // size, to make sure the peer can keep on sending without waiti
    ng
21      // for acknowledgements.
22      rt->receive_report_size >>= 1;       // 收到一半大小就确认
23
24      return 0;
25  }
```

ffmpeg在命名的Acknowledgement 的时候使用枚举RTMP_PT_BYTES_READ。

注：

- 客户端作为推流端时，一般即使没有收到服务器的ack，客户端也不会停止码流的推送。
- 当客户端作为拉流端时，一般即使拉流端不回应ack，服务器也不会停止码流的发送。
- 但彼此如果作为接收方时，收到1/2Windows size的数据后对会ack对方。

S->C Set Peer Bandwidth

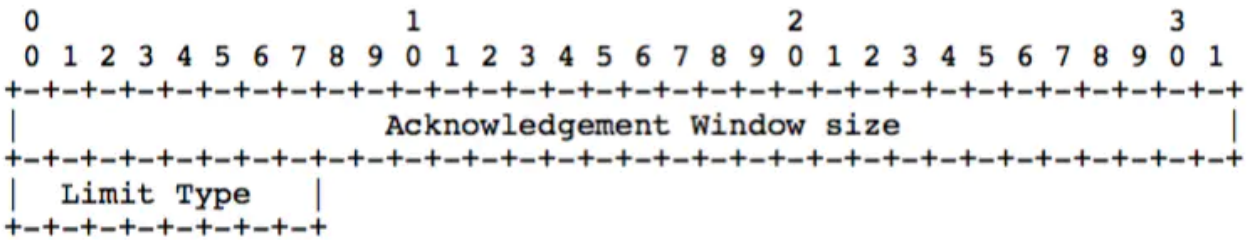
客户端或服务器端发送此消息更新对端（谁发送谁就是这里的对端）的输出带宽。
和Window Acknowledgement Size相比，重点是更新。

50	8.234584	111.229.231.225	192.168.0.106	RTMP	584	Set Peer Bandwidth 2500000,Dynamic|Set Chunk Size 60000

```
RTMP Header
    00.. .... = Format: 0
    ..00 0010 = Chunk Stream ID: 2
    Timestamp: 0
    Body size: 5
    Type ID: Set Peer Bandwidth (0x06)
    Stream ID: 0
RTMP Body
    Window acknowledgement size: 2500000
    Limit type: Dynamic (2)
```

Type ID：0x6 对应ffmpeg RTMP_PT_SET_PEER_BW,

Set Peer Bandwidth(Message Type ID=6):限制对端的输出带宽。接收端接收到该消息后会通过设置消息中的Window ACK Size来限制已发送但未接受到反馈的消息的大小来限制发送端的发送带宽。如果消息中的Window ACK Size与上一次发送给发送端的size不同的话要回馈一个Window Acknowledgement Size的控制消息。



Payload for the 'Set Peer Bandwidth' protocol message

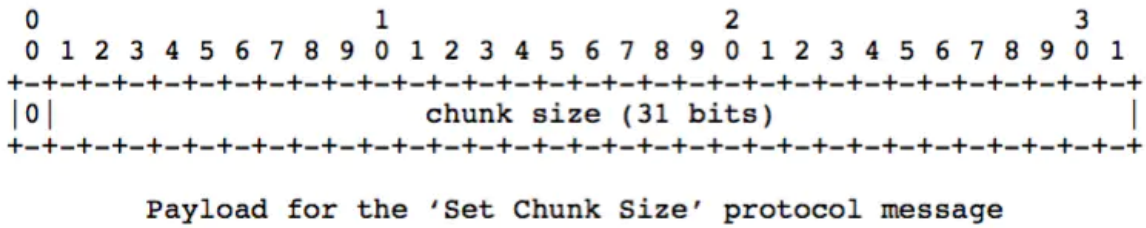
- Hard(Limit Type＝0):接收端应该将Window Ack Size设置为消息中的值
- Soft(Limit Type=1):接收端可以讲Window Ack Size设为消息中的值，也可以保存原来的值（前提是原来的Size小与该控制消息中的Window Ack Size）
- Dynamic(Limit Type=2):如果上次的Set Peer Bandwidth消息中的Limit Type为0，本次也按Hard处理，否则忽略本消息，不去设置Window Ack Size。

实际是用来告诉对端，当你如果不ack时，收到的最大size。但其重在更新。

S->C Set Chunk Size

```
Real Time Messaging Protocol (Set Chunk Size 60000)
  RTMP Header
      00.. .... = Format: 0
      ..00 0010 = Chunk Stream ID: 2
      Timestamp: 0
      Body size: 4
      Type ID: Set Chunk Size (0x01)
      Stream ID: 0
  RTMP Body
      Chunk size: 60000
```

Set Chunk Size(Message Type ID=1):设置chunk中Data字段所能承载的最大字节数，默认为128B，通信过程中可以通过发送该消息来设置chunk Size的大小（不得小于128B），而且通信双方会各自维护一个chunkSize，两端的chunkSize是独立的。比如当A想向B发送一个200B的Message，但默认的chunkSize是128B，因此就要将该消息拆分为Data分别为128B和72B的两个chunk发送，如果此时先发送一个设置chunkSize为256B的消息，再发送Data为200B的chunk，本地不再划分Message，B接受到Set Chunk Size的协议控制消息时会调整的接受的chunk的Data的大小，也不用再将两个chunk组成为一个Message。在实际写代码的时候一般会把chunk size设置的很大，有的会设置为4096，FFMPEG推流的时候设置的是 60*1000，这样设置的好处是避免了频繁的拆包组包，占用过多的CPU。
以下为代表Set Chunk Size消息的chunk的Data：



Payload for the 'Set Chunk Size' protocol message

其中第一位必须为0，chunk Size占31个位，最大可代表2147483647＝0x7FFFFFFF＝2^{31}–1，但实际上所有大于16777215=0xFFFFFF的值都用不上，因为chunk size不能大于Message的长度，表示Message的长度字段是用3个字节表示的，最大只能为0xFFFFFF。

C->S Set Chunk Size

```
Real Time Messaging Protocol (Set Chunk Size 60000)
  RTMP Header
      00.. .... = Format: 0
      ..00 0010 = Chunk Stream ID: 2
      Timestamp: 0
      Body size: 4
      Type ID: Set Chunk Size (0x01)
      Stream ID: 0
  RTMP Body
      Chunk size: 60000
```

以下：客户端需要发送 releaseStream，FCPublish 和 CreateStream 消息，具体可以参考ffmpeg handle_invoke_result函数

C->S releaseStream

```
Real Time Messaging Protocol (AMF0 Command releaseStream('livestream'))
  Response to this call in frame: 55
  RTMP Header
    01.. .... = Format: 1
    ..00 0011 = Chunk Stream ID: 3
    Timestamp delta: 0
    Timestamp: 0 (calculated)
    Body size: 39
    Type ID: AMF0 Command (0x14)
  RTMP Body
    String 'releaseStream'
      AMF0 type: String (0x02)
      String length: 13
      String: releaseStream
    Number 2
      AMF0 type: Number (0x00)
      Number: 2
    Null
      AMF0 type: Null (0x05)
    String 'livestream'
      AMF0 type: String (0x02)
      String length: 10
      String: livestream
```

对应的值 liveStream是要处理的流

C->S FCPublish

```
Real Time Messaging Protocol (AMF0 Command FCPublish('livestream'))
    Response to this call in frame: 60
  RTMP Header
      01.. .... = Format: 1
      ..00 0011 = Chunk Stream ID: 3
      Timestamp delta: 0
      Timestamp: 0 (calculated)
      Body size: 35
      Type ID: AMF0 Command (0x14)
  RTMP Body
    String 'FCPublish'
        AMF0 type: String (0x02)
        String length: 9
        String: FCPublish
    Number 3
        AMF0 type: Number (0x00)
        Number: 3
    Null
        AMF0 type: Null (0x05)
    String 'livestream'
        AMF0 type: String (0x02)
        String length: 10
        String: livestream
```

C->S createStream

```
Real Time Messaging Protocol (AMF0 Command createStream())
   Response to this call in frame: 60
 ⌄ RTMP Header
      01.. .... = Format: 1
      ..00 0011 = Chunk Stream ID: 3
      Timestamp delta: 0
      Timestamp: 0 (calculated)
      Body size: 25
      Type ID: AMF0 Command (0x14)
 ⌄ RTMP Body
   ⌄ String 'createStream'
        AMF0 type: String (0x02)
        String length: 12
        String: createStream
   ⌄ Number 4
        AMF0 type: Number (0x00)
        Number: 4
   ⌄ Null
        AMF0 type: Null (0x05)
```

FFmpeg

Create Stream：创建传递具体信息的通道，从而可以在这个流中传递具体信息，传输信息单元为Chunk。

当发送完createStream消息之后，解析服务器返回的消息会得到一个stream ID，这个ID也就是以后和服务器通信的 message stream ID，一般返回的是1，不固定。

字段	类型	说明
Command Name(命令名)	String	"createStream"
TransactionID	Number	上面接收到的命令消息中的TransactionID
Command Object	Object	命令参数
Optional Arguments	Object	用户自定义参数

C->S _checkbw

```
Real Time Messaging Protocol (AMF0 Command _checkbw())
▾ RTMP Header
     01.. .... = Format: 1
     ..00 0011 = Chunk Stream ID: 3
     Timestamp delta: 0
     Timestamp: 0 (calculated)
     Body size: 21
     Type ID: AMF0 Command (0x14)
▾ RTMP Body
   ▾ String '_checkbw'
        AMF0 type: String (0x02)
        String length: 8
        String: _checkbw
   ▾ Number 5
        AMF0 type: Number (0x00)
        Number: 5
   ▾ Null
        AMF0 type: Null (0x05)
```

服务器并没有做任何处理。

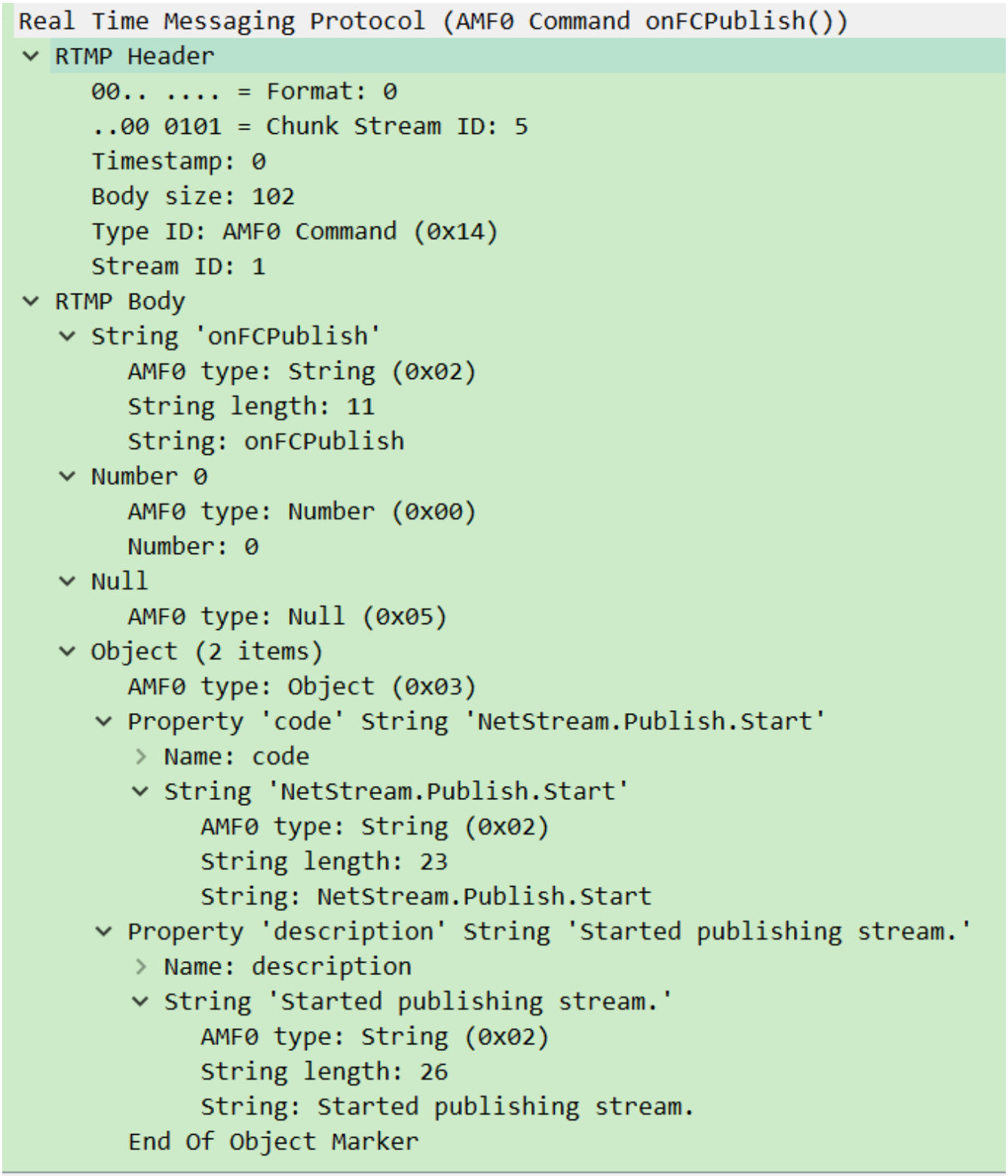
S->C _result

```
111.229.231.225      192.168.2.223        RTMP        87 _result()
111.229.231.225      192.168.2.223        RTMP        87 _result()
111.229.231.225      192.168.2.223        RTMP        95 _result()
```

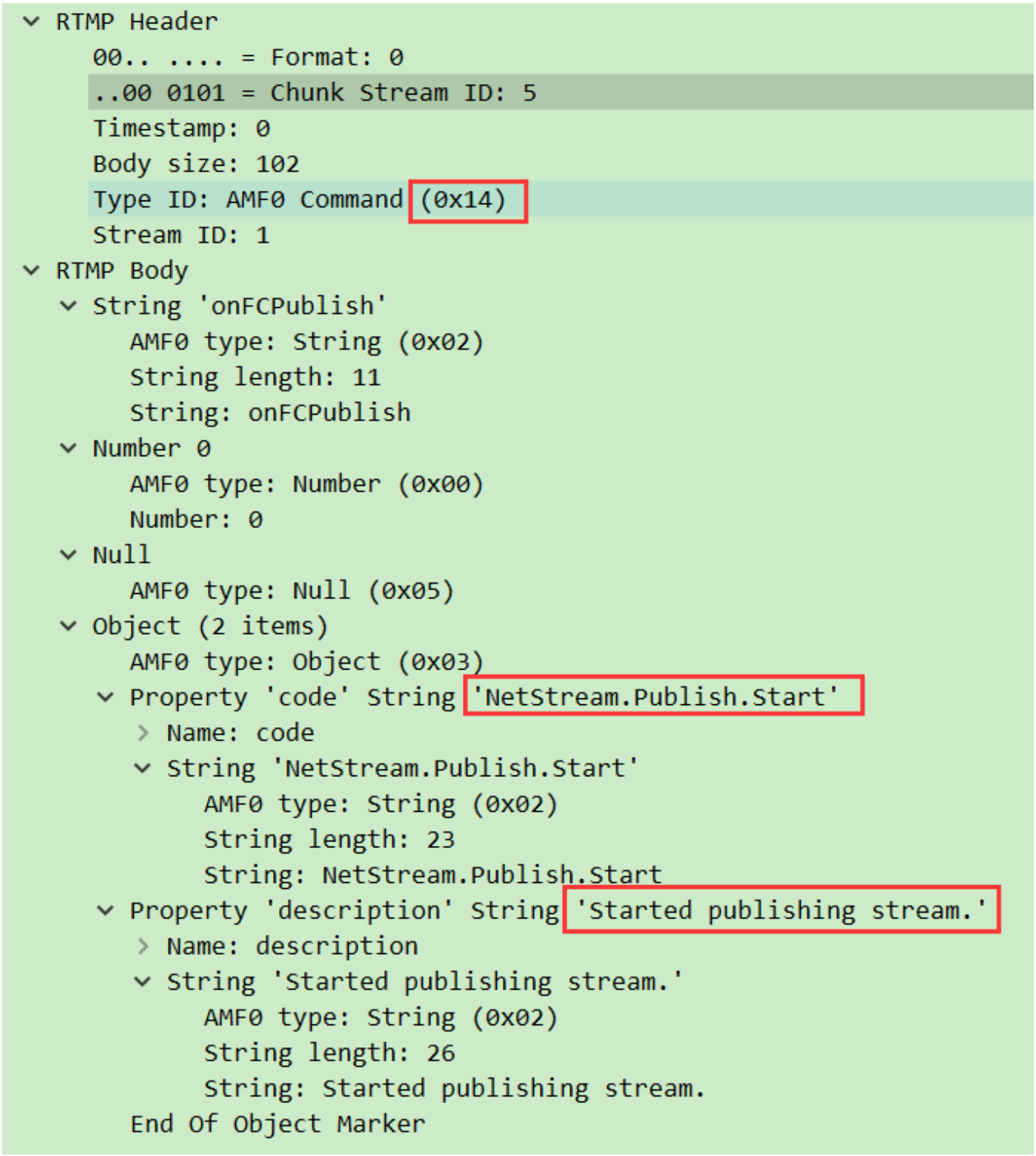
C->S publish

```
Real Time Messaging Protocol (AMF0 Command publish('livestream'))
▾ RTMP Header
    00.. .... = Format: 0
    ..00 1000 = Chunk Stream ID: 8
    Timestamp: 0
    Body size: 40
    Type ID: AMF0 Command (0x14)
    Stream ID: 1
▾ RTMP Body
  ▾ String 'publish'
      AMF0 type: String (0x02)
      String length: 7
      String: publish
  ▾ Number 6
      AMF0 type: Number (0x00)
      Number: 6
  ▾ Null
      AMF0 type: Null (0x05)
  ▾ String 'livestream'
      AMF0 type: String (0x02)
      String length: 10
      String: livestream
  ▾ String 'live'
      AMF0 type: String (0x02)
      String length: 4
      String: live
```

S->C onFCPublish



S->C onFCUnpublish


RTMP Header
00.. .... = Format: 0
..00 0101 = Chunk Stream ID: 5
Timestamp: 0
Body size: 102
Type ID: AMF0 Command (0x14)
Stream ID: 1
RTMP Body
String 'onFCPublish'
AMF0 type: String (0x02)
String length: 11
String: onFCPublish
Number 0
AMF0 type: Number (0x00)
Number: 0
Null
AMF0 type: Null (0x05)
Object (2 items)
AMF0 type: Object (0x03)
Property 'code' String 'NetStream.Publish.Start'
Name: code
String 'NetStream.Publish.Start'
AMF0 type: String (0x02)
String length: 23
String: NetStream.Publish.Start
Property 'description' String 'Started publishing stream.'
Name: description
String 'Started publishing stream.'
AMF0 type: String (0x02)
String length: 26
String: Started publishing stream.
End Of Object Marker

S->C _result