我有一个梦想，陶醉在绿水青山的故乡

趁着年轻

世界如此的熙熙攘攘，让年轻的心找不着方向 。但，这些人是不能小看的啊！如果，他们开始敲打自己的命令行！

请输入关键字...

librtmp实时消息传输协议(RTMP)库代码浅析

2013年11月30日 kulv 发表评论 阅读评论 9811次阅读

没事碰到了librtmp库，这个库是ffmpeg的依赖库，用来接收，发布RTMP协议格式的数据。

代码在这里：git clone git://git.ffmpeg.org/rtmpdump

先看一段通过librtmp.so库下载RTMP源发布的数据的例子，从rtmpdump中抽取出来。使用的大体流程如下：

1. RTMP_Init主要就初始化了一下RTMP*rtmp变量的成员。
2. RTMP_SetupURL 函数将rtmp源地址的端口，app，等url参数进行解析，设置到rtmp变量中。比如这样的地址： rtmp://host[:port]/path swfUrl=url tcUrl=url 。
3. RTMP_SetBufferMS 函数设置一下缓冲大小；
4. RTMP_Connect函数完成了连接的建立，一级RTMP协议层的应用握手，待会介绍。
5. RTMP_ConnectStream总的来说，完成了一个流的创建，以及打开，触发服务端发送数据过来，返回后，服务端应该就开始发送数据了。
6. Download 其实是RTMP_Read函数的封装，后者读取服务端的数据返回。

```
1 RTMP_Init(&rtmp);//初始化RTMP参数
2 //指定了-i 参数，直接设置URL
```

```
3  if (RTMP_SetupURL(&rtmp, fullUrl.av_val) == FALSE) {
4      RTMP_Log(RTMP_LOGERROR, "Couldn't parse URL: %s", fullUrl.av_val);
5      return RD_FAILED;
6  }
7
8  rtmp.Link.timeout = timeout ;
9  /* Try to keep the stream moving if it pauses on us */
10 if (!bLiveStream )
11     rtmp.Link.lFlags |= RTMP_LF_BUFX;
12
13 while (!RTMP_ctrlC)
14 {
15     RTMP_Log(RTMP_LOGDEBUG, "Setting buffer time to: %dms", DEF_BUFTIME);
16     RTMP_SetBufferMS(&rtmp, DEF_BUFTIME);//告诉服务器帮我缓存多久
17
18     RTMP_LogPrintf("Connecting ...\n");
19     if (!RTMP_Connect(&rtmp, NULL)) {//建立连接,发送"connect"
20         nStatus = RD_NO_CONNECT;
21         break;
22     }
23     RTMP_Log(RTMP_LOGINFO, "Connected...");
24
25     //处理服务端返回的各种控制消息包，比如收到connect的result后就进行createStream，以及发送
   play(test)消息
26     if (!RTMP_ConnectStream(&rtmp, 0)) {//一旦返回，表示服务端开始发送数据了.可以play了
27         nStatus = RD_FAILED;
28         break;
29     }
30
31     nStatus = Download(&rtmp, file, bStdoutMode, bLiveStream );
32
33     if (nStatus != RD_INCOMPLETE || !RTMP_IsTimedout(&rtmp) || bLiveStream)
34         break;
35 }
```

为了方便理解，贴一个本文的数据包图：

dst port	Protocol	Length	Info
macromedia-fc:	TCP	74	45204 > macromedia-fcs [SYN] Seq=0 Win=32768 Len=0 MSS=16396 SACK_PERM=1 TSv
45204	TCP	74	macromedia-fcs > 45204 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=16396 SACK
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=1 Ack=1 Win=32768 Len=0 TSval=3571914256 TS
macromedia-fc:	RTMP	1603	Handshake C0+C1
45204	TCP	66	macromedia-fcs > 45204 [ACK] Seq=1 Ack=1538 Win=35968 Len=0 TSval=3571914256
45204	TCP	1603	macromedia-fcs > 45204 [PSH, ACK] Seq=1 Ack=1538 Win=35968 Len=1537 TSval=35
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=1538 Ack=1538 Win=35968 Len=0 TSval=3571914
45204	RTMP	1602	Handshake S0+S1+S2
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=1538 Ack=3074 Win=39040 Len=0 TSval=3571914
macromedia-fc:	RTMP	1602	Handshake C2
macromedia-fc:	TCP	206	45204 > macromedia-fcs [PSH, ACK] Seq=3074 Ack=3074 Win=39040 Len=140 TSval=
45204	TCP	66	macromedia-fcs > 45204 [ACK] Seq=3074 Ack=3214 Win=42112 Len=0 TSval=3571914
macromedia-fc:	RTMP	108	connect('live')
45204	RTMP	82	Window Acknowledgement Size 5000000
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=3256 Ack=3090 Win=39040 Len=0 TSval=3571914
45204	RTMP	301	Set Peer Bandwidth 5000000,Dynamic | Set Chunk Size 2048 | _result('NetConne
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=3256 Ack=3325 Win=42112 Len=0 TSval=3571914
macromedia-fc:	RTMP	82	Window Acknowledgement Size 5000000
macromedia-fc:	RTMP	84	Set Buffer Length 0,300ms
macromedia-fc:	RTMP	99	createStream()
45204	TCP	66	macromedia-fcs > 45204 [ACK] Seq=3325 Ack=3290 Win=42112 Len=0 TSval=3571914
45204	RTMP	107	_result()
macromedia-fc:	RTMP	114	play('hwtest1')
macromedia-fc:	RTMP	77	Set Buffer Length 1,36000000ms
45204	TCP	66	macromedia-fcs > 45204 [ACK] Seq=3366 Ack=3382 Win=42112 Len=0 TSval=3571914
45204	RTMP	84	Stream Begin 1
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=3382 Ack=3384 Win=42112 Len=0 TSval=3571914
45204	RTMP	1249	onStatus('NetStream.Play.Start') | |RtmpSampleAccess() | onMetaData() | Audi
macromedia-fc:	TCP	66	45204 > macromedia-fcs [ACK] Seq=3382 Ack=4567 Win=45184 Len=0 TSval=3571914
45204	RTMP	702	Audio Data

一、建立协议连接

下面来详细介绍下RTMP_Connect函数的工作。

先看代码，下面RTMP_Connect的工作是连接对端，进行握手，并且发送”connect” 控制消息，附带一些app,tcurl等参数。其实时调用了2个函数完成工作的：RTMP_Connect0， RTMP_Connect1 。

```
int  RTMP_Connect(RTMP *r, RTMPPacket *cp)
{//连接对端，进行握手，并且发送"connect" 控制消息，附带一些app,tcurl等参数
    struct sockaddr_in service;
    if (!r->Link.hostname.av_len)
        return FALSE;

    memset(&service, 0, sizeof(struct sockaddr_in));
    service.sin_family = AF_INET;

    if (r->Link.socksport)
    {
        /* Connect via SOCKS */
        if (!add_addr_info(&service, &r->Link.sockshost, r->Link.socksport))
            return FALSE;
    }
    else
    {
        /* Connect directly */
        if (!add_addr_info(&service, &r->Link.hostname, r->Link.port))
            return FALSE;
    }

    if (!RTMP_Connect0(r, (struct sockaddr *)&service))//建立一个socket连接
        return FALSE;

    r->m_bSendCounter = TRUE;

    return RTMP_Connect1(r, cp);//进行C0-2/S0-2协议握手，发送connect命令
}
```

其中RTMP_Connect0 函数比较简单，标准的socket, conect 流程，另外设置了一下TCP_NODELAY选项，方便小包发送等。以及SO_RCVTIMEO读超时，这部分属于基本的TCP层面的连接；

RTMP_Connect1 函数则完成类似HTTP层面的RTMP协议的连接建立过程。首先是HandShake 握手。RTMP的握手是通过客户端跟服务端互相发送数据包来完成的，每人3个数据包，名之为C0，C1，C2 以及S0，S1， S2。其发送数据有严格的限制的。因为互相依赖。这个在官方文档中有详细的介绍，不多说。

对于librtmp来说，可能的一种流程是：

CLIENT SERVER

C0，C1 —>

<— S0， S1，S2

C2 ->

具体看一下代码，比较长。

```
static int HandShake(RTMP *r, int FP9HandShake)
{//C0,C1 -- S0, S1, S2 -- C2 消息握手协议
    int i;
    uint32_t uptime, suptime;
    int bMatch;
    char type;
    char clientbuf[RTMP_SIG_SIZE + 1], *clientsig = clientbuf + 1;
    char serversig[RTMP_SIG_SIZE];

    clientbuf[0] = 0x03;//C0, 一个字节。03代表协议版本号为3     /* not encrypted */

    uptime = htonl(RTMP_GetTime());//这是一个时间戳，放在C1消息头部
    memcpy(clientsig, &uptime, 4);
    memset(&clientsig[4], 0, 4);//后面放4个字节的空数据然后就是随机数据

    //后面是随机数据，总共1536字节的C0消息
#ifdef _DEBUG
    for (i = 8; i < RTMP_SIG_SIZE; i++)
        clientsig[i] = 0xff;
#else
```

```
    for (i = 8; i < RTMP_SIG_SIZE; i++)
        clientsig[i] = (char)(rand() % 256);//发送C0， C1消息
#endif

    if (!WriteN(r, clientbuf, RTMP_SIG_SIZE + 1))
        return FALSE;

    //下面读一个字节也就是S0消息，看协议是否一样
    if (ReadN(r, &type, 1) != 1)    /* 0x03 or 0x06 */
        return FALSE;
    RTMP_Log(RTMP_LOGDEBUG, "%s: Type Answer   : %02X", __FUNCTION__, type);
    if (type != clientbuf[0])//C/S版本不一致
        RTMP_Log(RTMP_LOGWARNING, "%s: Type mismatch: client sent %d, server answered %d",  __FUNCTION__, clientbuf[0], type);

    //读取S1消息，里面有服务器运行时间
    if (ReadN(r, serversig, RTMP_SIG_SIZE) != RTMP_SIG_SIZE)
        return FALSE;

    /* decode server response */

    memcpy(&suptime, serversig, 4);
    suptime = ntohl(suptime);

    RTMP_Log(RTMP_LOGDEBUG, "%s: Server Uptime : %d", __FUNCTION__, suptime);
    RTMP_Log(RTMP_LOGDEBUG, "%s: FMS Version   : %d.%d.%d.%d", __FUNCTION__, serversig[4], serversig[5], serversig[6], serversig[7]);

    /* 2nd part of handshake */
    if (!WriteN(r, serversig, RTMP_SIG_SIZE))//发送C2消息，内容就等于S1消息的内容。
        return FALSE;

    //读取S2消息
    if (ReadN(r, serversig, RTMP_SIG_SIZE) != RTMP_SIG_SIZE)
        return FALSE;

    bMatch = (memcmp(serversig, clientsig, RTMP_SIG_SIZE) == 0);
    if (!bMatch)//服务端返回的S2消息必须跟C1消息一致才行
    {
        RTMP_Log(RTMP_LOGWARNING, "%s, client signature does not match!", __FUNCTION__);
    }
    return TRUE;
}
```

握手的目的其实是互相沟通一下支持的协议版本号，服务器时间戳等。确保连接的对端真的是RTMP支持的。

发送请求给服务端。

然后就是SendConnectPacket的工作了。总结一句其功能是成一个“connect消息以及其app,tcurl等参数，然后调用RTMP_SendPacket函数将其数据发送出去。

到这里连接建立完成了。

二、准备数据通道

RTMP_ConnectStream完成了通道的建立。其处理服务端返回的各种控制消息包，比如收到connect的result后就进行createStream，以及发送play(test)消息。一旦返回，表示服务端开始发送数据了.可以play了。

函数本身比较简单，就是一个while循环，不断的调用RTMP_ReadPacket读取服务端发送过来的数据包进行相应的处理。直到m_bPlaying变老变为TRUE为止，也就是可以播放的时候为止。数据包的处理函数为RTMP_ClientPacket。

```
int RTMP_ConnectStream(RTMP *r, int seekTime)
{//循环读取服务端发送过来的各种消息，比如window ack**, set peer bandwidth, set chunk size, _result等
    //直到接收到了play
    RTMPPacket packet = { 0 };

    /* seekTime was already set by SetupStream / SetupURL.
     * This is only needed by ReconnectStream.
     */
```

```
    if (seekTime > 0)
        r->Link.seekTime = seekTime;

    r->m_mediaChannel = 0;

    //一个个包的读取，直到服务端告诉我说可以play了为止
    while (!r->m_bPlaying && RTMP_IsConnected(r) && RTMP_ReadPacket(r, &packet))
    {
        if (RTMPPacket_IsReady(&packet))//是否读取完毕。((a)->m_nBytesRead == (a)->m_nBodySize)
        {
            if (!packet.m_nBodySize)
                continue;
            if ((packet.m_packetType == RTMP_PACKET_TYPE_AUDIO) ||
                    (packet.m_packetType == RTMP_PACKET_TYPE_VIDEO) ||
                    (packet.m_packetType == RTMP_PACKET_TYPE_INFO))
            {
                RTMP_Log(RTMP_LOGWARNING, "Received FLV packet before play()! Ignoring.");
                RTMPPacket_Free(&packet);
                continue;
            }

            RTMP_ClientPacket(r, &packet);//处理一下这个数据包，其实里面就是处理服务端发送过来的各种消息等。直到接受到了play/publish
            RTMPPacket_Free(&packet);
        }
    }
    //返回当前是否接收到了play/publish 或者stopd等
    return r->m_bPlaying;
}
```

RTMP_ReadPacket 跟Send类似，函数比较长，基本是处理RTMP数据包RTMPPacket的包头，包体的读写等碎碎代码。真正处理事件的函数为RTMP_ClientPacket。

RTMP_ClientPacket函数是一个很大的数据包分发器。负责将不同类型m_packetType的数据包传递给对应的函数进行处理。比如：

1. RTMP_PACKET_TYPE_CHUNK_SIZE 块大小设置消息 HandleChangeChunkSize;
2. RTMP_PACKET_TYPE_CONTROL 控制消息 HandleCtrl ;
3. RTMP_PACKET_TYPE_AUDIO 音频消息 HandleAudio;
4. RTMP_PACKET_TYPE_INFO 元数据设置消息 HandleMetadata;
5. RTMP_PACKET_TYPE_INVOKE 远程过程调用 HandleInvoke;

其中比较重要的是HandleInvoke 远程过程调用。其里面实际是个状态机。

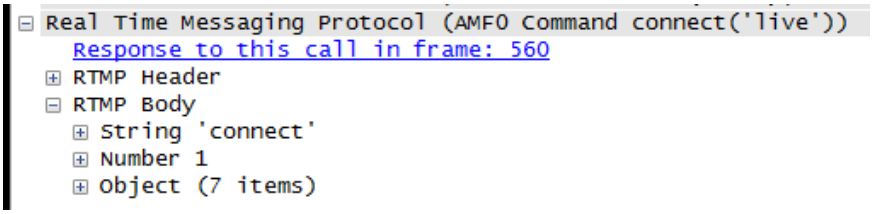
前面说过，建立连接握手的时候，客户端回发送connect字符串以及必要的参数给服务端。然后服务端会返回_result消息。当客户端收到_result消息后，会从消息里面取出其消息号，从而在r->m_methodCalls[i].name 中找到对应发送的消息是什么消息。从而客户端能够确认发送的那条消息被服务端处理了。进而可以进行后续的处理了。来看HandleInvoke开头的代码。

```
static int HandleInvoke(RTMP *r, const char *body, unsigned int nBodySize){
    AMFObject obj;
    AVal method;
    double txn;
    int ret = 0, nRes;

    nRes = AMF_Decode(&obj, body, nBodySize, FALSE);
    if (nRes < 0){
        RTMP_Log(RTMP_LOGERROR, "%s, error decoding invoke packet", __FUNCTION__);
        return 0;
    }

    AMF_Dump(&obj);
    AMFProp_GetString(AMF_GetProp(&obj, NULL, 0), &method);
    txn = AMFProp_GetNumber(AMF_GetProp(&obj, NULL, 1));
    RTMP_Log(RTMP_LOGDEBUG, "%s, server invoking <%s>", __FUNCTION__, method.av_val);

    if (AVMATCH(&method, &av__result))
    {//接收到服务端返回的一个_result包，所以我们需要找到这个包对应的那条命令，从而处理这条命令的对应事件。
```

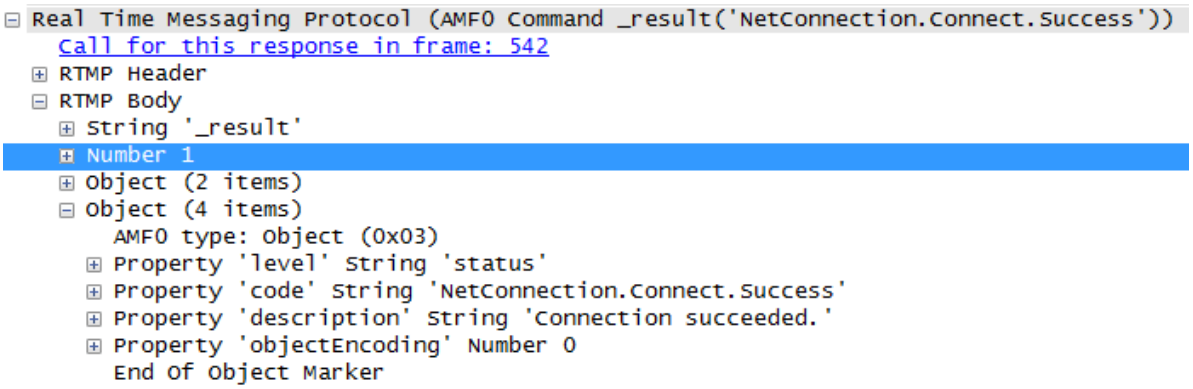
```
        //比如我们之前发送了个connect给服务端，服务端必然会返回_result，然后我们异步收到result后，会调用
        //RTMP_SendServerBW,RTMP_SendCtrl,以及RTMP_SendCreateStream来创建一个stream
        AVal methodInvoked = {0};
        int i;

        for (i=0; i<r->m_numCalls; i++) {//找到这条指令对应的触发的方法
            if (r->m_methodCalls[i].num == (int)txn) {
                methodInvoked = r->m_methodCalls[i].name;
                AV_erase(r->m_methodCalls, &r->m_numCalls, i, FALSE);
                break;
            }
        }
```

上面可以看出，librtmp发送出一条需要得到服务端返回结果的消息的时候，会将消息名称记录在m_methodCalls数组上面，其下标就是告诉服务端的消息id。从而每次收到_result的时候就能知道对那个的是哪条消息methodInvoked。

然后就可以进行对应的处理了，举个例子：在之前发送connect的时候，body部分的第二个元素为一个整数，代表一个唯一ID，这里是1，如下图：



服务端对此数据包的回包会是如下的样子：



注意蓝底的Number 1，他会跟上面的connect(live)消息对应的。因此methodInvoked变量就能等于connect，所以HandleInvoke函数会进入到如下的分支：

```
//下面根据不同的方法进行不同的处理
if (AVMATCH(&methodInvoked, &av_connect))
{
    if (r->Link.protocol & RTMP_FEATURE_WRITE)
    {
        SendReleaseStream(r);
        SendFCPublish(r);
    }
    else
    {//告诉服务端，我们的期望是什么，窗口大小，等
        RTMP_SendServerBW(r);
        RTMP_SendCtrl(r, 3, 0, 300);
    }
    RTMP_SendCreateStream(r);//因为服务端同意了我们的connect，所以这里发送createStream创建一个流
    //创建完成后，会再次进如这个函数从而走到下面的av_createStream分支，从而发送play过去

    if (!(r->Link.protocol & RTMP_FEATURE_WRITE))
    {
        /* Authenticate on Justin.tv legacy servers before sending FCSubscribe */
        if (r->Link.usherToken.av_len)
            SendUsherToken(r, &r->Link.usherToken);
        /* Send the FCSubscribe if live stream or if subscribepath is set */
        if (r->Link.subscribepath.av_len)
```

```
            SendFCSubscribe(r, &r->Link.subscribepath);
        else if (r->Link.lFlags & RTMP_LF_LIVE)
            SendFCSubscribe(r, &r->Link.playpath);
    }
}
else if (AVMATCH(&methodInvoked, &av_createStream))
```

上面的分支在服务端同意客户端的connect请求后，客户端调用。

根据流的配置类型不同，进行不同的处理，比如如果是播放的话，那么就会调用SendReleaseStream，以及SendFCPublish发送publish消息；

否则会调用RTMP_SendServerBW设置缓冲大小，也就是图中的“Window Acknowledgement Size 5000000” 。 然后就是RTMP_SendCtrl设置缓冲时间；

之后就会调用RTMP_SendCreateStream函数，发送注明的流创建过程。发送createStream消息给服务端，创建数据传输通道。当然这里只是发送了数据，什么时候能够确定创建成功呢？答案很简单：当接收到服务端的数据包后，如果其为过程调用，且为_result，并且AVMATCH(&methodInvoked, &av_createStream)的时候，就代表创建成功。看如下代码：

```
    else if (AVMATCH(&methodInvoked, &av_createStream))
    {
        r->m_stream_id = (int)AMFProp_GetNumber(AMF_GetProp(&obj, NULL, 3));

        if (r->Link.protocol & RTMP_FEATURE_WRITE)
        {//如果是要发送，那么高尚服务端，我们要发数据
            SendPublish(r);
        }
        else
        {//否则告诉他我们要接受数据
            if (r->Link.lFlags & RTMP_LF_PLST)
                SendPlaylist(r);
            SendPlay(r);//发送play过去，
            RTMP_SendCtrl(r, 3, r->m_stream_id, r->m_nBufferMS);//以及我们的buf大小
        }
    }
    else if (AVMATCH(&methodInvoked, &av_play) ||
            AVMATCH(&methodInvoked, &av_publish))
    {//接收到了play的回复，那么标记为play
        r->m_bPlaying = TRUE;
    }
    free(methodInvoked.av_val);
}
```

createStream消息确认收到后，客户端就是发送SendPlay 请求开始接收数据，或者SendPublish请求开始发布数据；

此后再经过几次简短的消息传输，比如：onStatus(‘NetStream.Play.Start’) | |RtmpSampleAccess() | onMetaData() 等，真正的数据就能够开始接收了。也就是服务端开始发送数据了。通信的信道已经建立好。

三、读取数据

连接经过漫长的过程建立起来后，数据读取比较简短，只需要调用nRead = RTMP_Read(rtmp, buffer, bufferSize)函数不断的读取数据就行。这些数据就是发送方放入RTMP通道里面的数据了。

所以这部分其实就等于：通道已经建立，读使用RTMP_Read，发送使用RTMP_SendPacket等。

介绍的差不多了，再细致的后续有时间再补上。基本框架就在这里。过段时间看看nginx_rtmp_module模块学习一下。

分类：C/C++, RTMP, TCP/IP 标签：librtmp, rtmp

Related Posts

- 2015 年 10 月 7 日 memcached LRU过期机制lru_crawler指令可能会错误的清理不应该的slabs的bug (0)
- 2015 年 10 月 7 日 Memcached 缓存过期机制源码分析 (0)
- 2016 年 1 月 1 日 你好2016 (5)
- 2016 年 5 月 22 日 阻止一个产品往前的，往往是我们自己的固守和习以为常 (0)