下一代直播 CDN

推流 SDK 方案

V1.2

深圳市网心科技有限公司

版本信息

版本	更改日期	更改要点说明	编制	审核	批准
V1.0	2017-4-10	新建			
V1.1	2017-6-6	修订			
V1.2	2017-6-27	修订			

目录

1. 概述

本文档描述了下一代直播 CDN 推流 SDK 的设计方案，包括 SDK 对应用层的接口以及集成 CDN 厂商私有协议的设计方案和实现规范。

2. 需求背景

下一代直播 CDN 在推流这块仍使用 RTMP 作为上层协议，但对一些细节做了更详细的规定（例如要求加入延时计算所需的信息），因此七牛公开了一个符合上述规范的 librtmp 作为标准实现。而融合各厂商的私有传输协议也是下一代直播 CDN 组织的一个重要目标，因此需要基于上述 librtmp 标准实现发展出一个可以集成私有传输协议的框架。

3. 设计方案

3.1. 整体描述

SDK 整体结构如图。接口层（推流框架）下挂若干传输模块，每个模块对应一种传输协议（例如 RTMP）。接口层对应用层提供标准的 librtmp 接口，经过与 Server 端能力协商选用合适的传输方式进行传输交互。

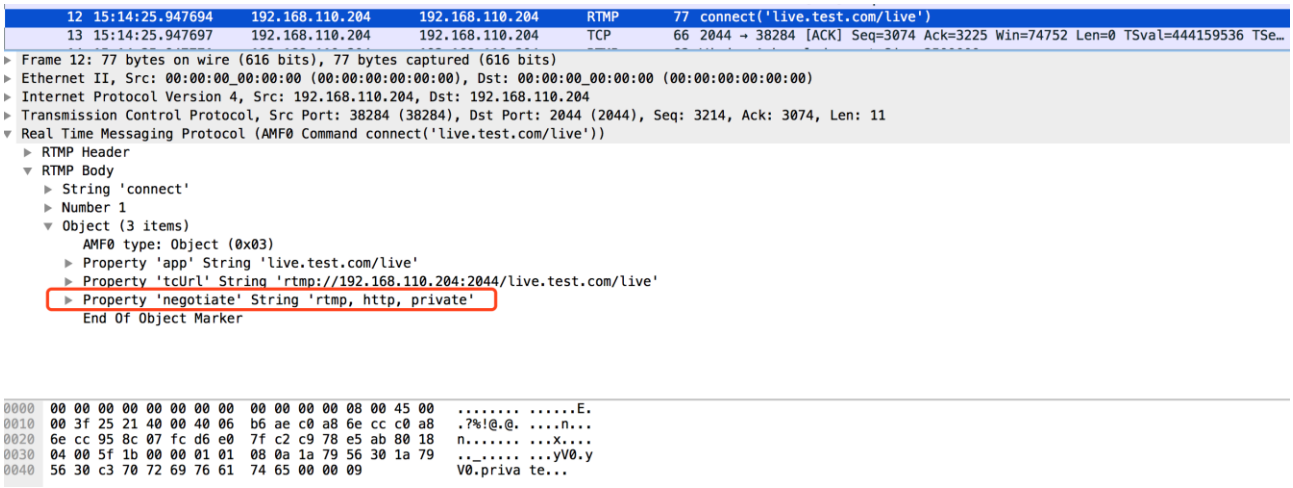
RTMP 模块为默认传输模块，若 Server 端不支持能力协商或不支持客户端的任何非标准传输模块，则使用 RTMP 模块作为保底。

3.2. 交互流程

能力协商包定义

能力协商包为对 rtmp 协议原本 connect 包和_result 包的扩展，connect 和_result 分别添加”negotiate”字段用于能力协商。

Client 在握手成功之后发送带有“negotiate”字段的 connect 包进行协商请求，然后 server 端回复带有“negotiate”字段的“_result”包进行响应。



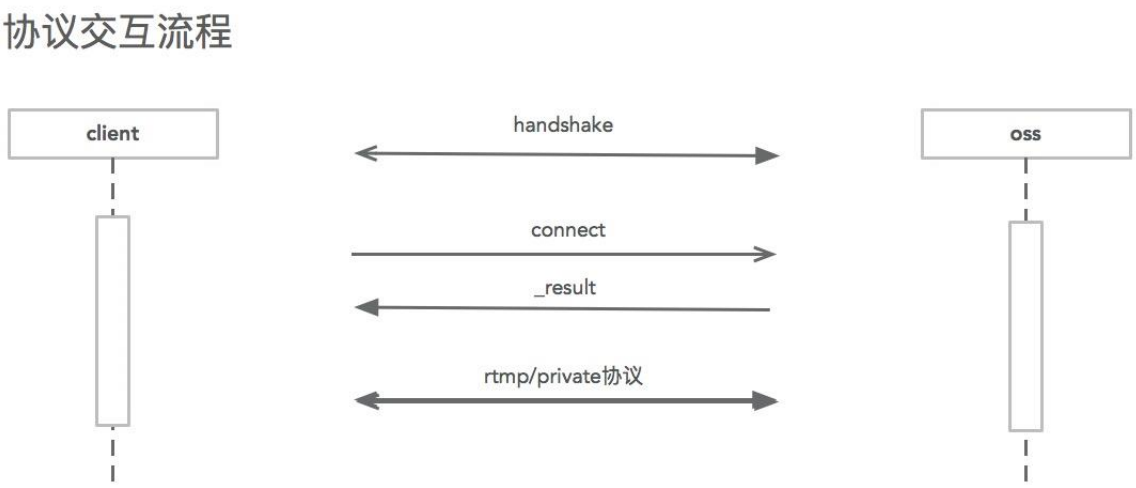
其中“rtmp，http，private”表示客户端支持的协议优先级列表，优先级由高到低排列，支持包含多个协议。“rtmp”优先级最高，“http”次之，“private”优先级最低。协议字段之间以“，”分割。其中“rtmp”为保留字段，不可随意修改。“http”和“private”为各个厂商自定义私有协议名，可由组织统一管理，避免出现冲突。

Server 根据自己支持的协议列表和收到的 client 端协议列表，确定一个协商结果。并发送带有“negotiate”字段的_result 包返回协商结果。

```
380 4.312627631 192.168.110.204  192.168.110.204  2044  34894  RTMP  335 _result('NetConnection.Connect.Success')
381 4.312659507 192.168.110.204  192.168.110.204  2044  34894  RTMP   99 onBWDone()
383 4.312888901 192.168.110.204  192.168.110.204  34894 2044   RTMP  117 releaseStream('livestream')
384 4.312914861 192.168.110.204  192.168.110.204  34894 2044   RTMP  113 FCPublish('livestream')

▷ Frame 380: 335 bytes on wire (2680 bits), 335 bytes captured (2680 bits) on interface 0
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol Version 4, Src: 192.168.110.204 (192.168.110.204), Dst: 192.168.110.204 (192.168.110.204)
▷ Transmission Control Protocol, Src Port: rimsl (2044), Dst Port: 34894 (34894), Seq: 3123, Ack: 3225, Len: 269
▽ Real Time Messaging Protocol (AMF0 Command _result('NetConnection.Connect.Success'))
    Call for this response in frame: 374
  ▷ RTMP Header
  ▽ RTMP Body
    ▷ String '_result'
    ▷ Number 1
    ▽ Object (4 items)
        AMF type: Object (3)
      ▷ Property 'fmsVer' String 'FMS/3,5,3,888'
      ▷ Property 'capabilities' Number 127
      ▷ Property 'mode' Number 1
      ▷ Property 'negotiate' String 'rtmp'
        End Of Object Marker
    ▽ Object (5 items)
        AMF type: Object (3)
      ▷ Property 'level' String 'status'
```

时序图



server 端相关配置：

服务端需要根据 client 端提供的协议优先级列表并结合自身支持的协议优先级列表返回一个最终协商结果。比如： client 发送 connect 包携带协议优先级列表（“rtmp， private”），server 端配置优先级列表（“private，rtmp”），当 client 端优先级高时，_result 返回“rtmp”结果，当 server 端优先级高时，_result 返回“private”结果

server、client 端的具体执行逻辑

1、 client 端和 server 端进行 rtmp 握手。

2、 client 发送包含协商信息（客户端支持的协议列表）的 connect 包到服务端

3、 服务端根据自己支持的协议列表和配置对比客户端支持的协议做一个选优，并发送包含协商结果的 _result 数据包到 client 端

4、 client 端根据返回的_result 消息进行接下来的连接

4.1、如果返回是 rtmp 协议则继续用 rtmp 协议进行接下来的连接交互。

4.2、如果返回是私有协议或其他协议则断开 rtmp 连接，进行私有协议或其他协议的连接。

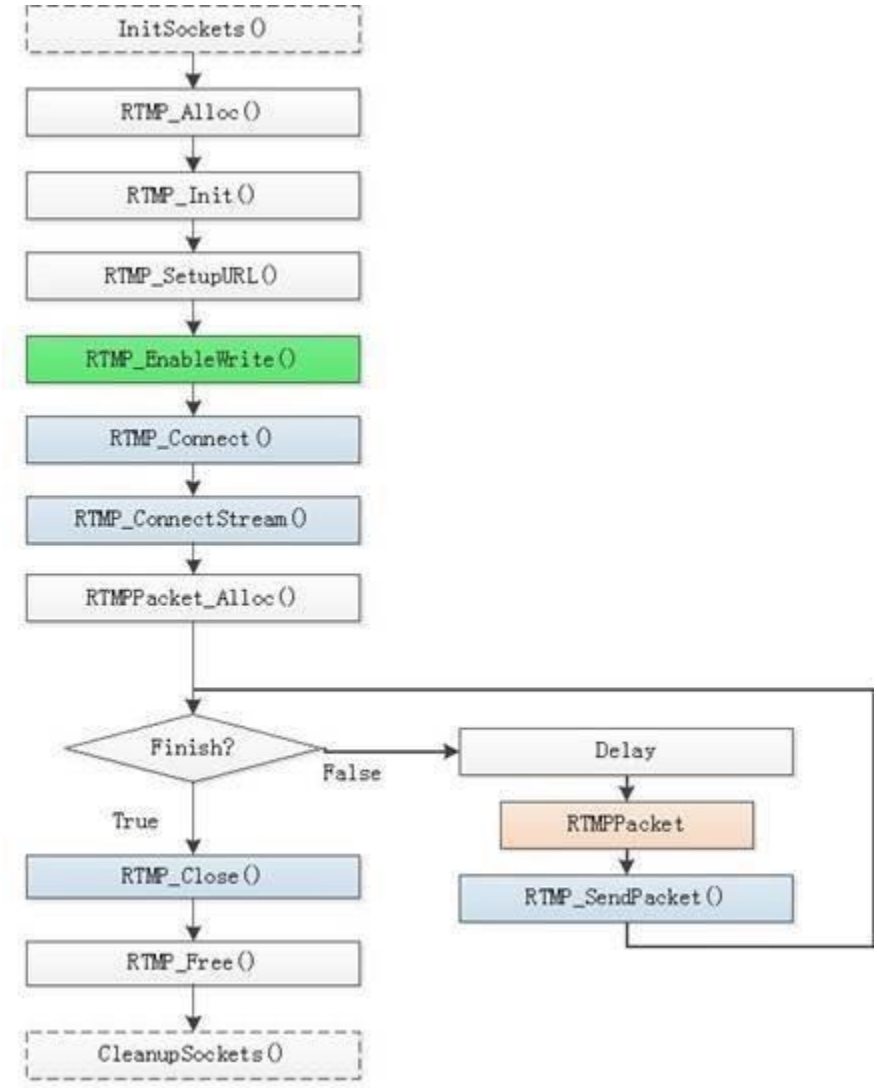
5、音视频数据交互。

向下兼容逻辑

1、 如果 server 端不识别发送的 connect 包的协商标识。则在返回的_result 数据包中不会携带“negotiate”相关字段，client 继续用 rtmp 连接。

2、 如果 client 端发送的 connect 包不包含协商标识 则 server 端在返回的_result 数据包中也不包含“negotiate”相关字段，client 继续用 rtmp 连接。

3.3. 应用层接口



以上是应用 RTMP_ReadPacket 所需要调用到的接口与流程图

InitSockets()：初始化 Socket

RTMP_Alloc()：为结构体“RTMP”分配内存。

RTMP_Init()：初始化结构体“RTMP”中的成员变量。

RTMP_SetupURL()：设置输入的 RTMP 连接的 URL。

RTMP_EnableWrite()：发布流的时候必须要使用。如果不使用则代表接收流。

RTMP_Connect()：建立 RTMP 连接，创建一个 RTMP 协议规范中的 NetConnection。

RTMP_ConnectStream()：创建一个 RTMP 协议规范中的 NetStream。

Delay：发布流过程中的延时，保证按正常播放速度发送数据。

RTMP_SendPacket()：发送一个 RTMP 数据 RTMPPacket。

RTMP_Close()：关闭 RTMP 连接。

RTMP_Free()：释放结构体“RTMP”。

CleanupSockets()：关闭 Socket。

调用 RTMP_Write 的接口与 RTMP_ReadPacket 基本一致

3.4. 传输模块接口

通用的传输模块结构体，开发者可以定义自己的传输模块

结构体定义见 PushModule.h

```
typedef struct push_module_s {
    const char *module_name;

    /* (PILI_RTMP*, RTMP_Error*) */
    int (*init)(void*, void*);

    /* PILI_RTMP* */
    int (*release)(void*);

    /* PILI_RTMP*, const char*, int, RTMP_Error* */
    int (*push_message_push)(void *rtmp, void *buf, uint32_t size, void *err);
} push_module_t;
```

3.4.1. 回调说明

int (*init)(void*);

模块初始化，建立连接的过程应该包含在此函数内部;

传入参数为 RTMP*

int(*release)(void*);

模块的析构函数，推流结束时调用，内部应实现释放文件描述符清理内存的相关操作

传入参数为 RTMP*

int (*push_message_push)(void *rtmp, void *buf, uint32_t size, void *err);

数据推送接口，传入 flv tag 数据。模块内部为私有协议的主体部分，通过私有协议实现数据的发送。

传入参数为 RTMP*, const char*（实际数据）, uint32_t（数据长度）, void*（接收 error 的回调）。

3.4.2. 传输模块注册

见 **PushModule.c**

```
push_module_t *global_modules[] = {
    &rtmppush_module,
    &examplepush_module
    /* 新增模块加在这里即可 */
};
```

global_modules[]定义的全局变量，存储所有定义的模块，优先级从上至下依次减小。

3.4.3. 模块开发示例

1. 实例化模块结构体：

```
push_module_t xypush_module =
{
    "XYPushModule",   //此字串将出现在negotiate中代表该模块
    xypush_module_init,
    xypush_module_release,
    xypush_module_push
};
```

2. 实现接口函数：

```
int xypush_module_init(void *arg);
int xypush_module_release(void *arg);
int xypush_module_push (void*, void*, uint32_t, void*c);
```

3. 在**PushModule.c**包含模块头文件并注册模块

```
#include “xypush_module.h”
```

```
extern push_module_t xypush_module;
push_module_t *global_modules[] = {
        &rtmppush_module,
        &examplepush_module,
        &xypush_module,
        /*  新增模块加在这里即可  */
};
```

4. 后续工作