**P2P模块详细设计**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 内容 | 作者 | 时间 |
| 1.0 | RTSP模块详细方案初稿 | 李纯 | 2015/3/11 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 概述 2](#_Toc414023684)

[2 设计思想 2](#_Toc414023685)

[3 示意图 5](#_Toc414023686)

[3.1 RTP格式 5](#_Toc414023687)

[3.2 SDP格式 5](#_Toc414023688)

[3.3 核心模块调用关系图 6](#_Toc414023689)

[4 接口定义 7](#_Toc414023690)

[4.1 RTSP模块 7](#_Toc414023691)

[4.2 RTP/RTCP模块 19](#_Toc414023692)

[4.3 SDP模块 25](#_Toc414023693)

# 概述

作为设备SDK设计的另一方面，可以通过流媒体转发服务器来进行流数据的转发，达到流

从设备端到APP端传输。为了这个模型，我们需要实现一套协议进行协商和流数据传输。本文以下部分将详细阐述此设计模型

# 2 设计思想

RTSP（Real Time Streaming Protocol），RFC2326，实时流传输协议。该协议用于C/S模型，

是一个基于文本的协议，用于在客户端和服务器端建立和协商实时流会话。实时流[协议](http://baike.baidu.com/view/36190.htm)（RTSP）建立并控制一个或几个[时间同步](http://baike.baidu.com/view/100292.htm" \t "_blank)的连续[流媒体](http://baike.baidu.com/view/794.htm)。尽管连续媒体流与[控制流](http://baike.baidu.com/view/1462753.htm)交换是可能的，通常它本身并不发送连续流。网络传输层可以依赖UDP/TCP协议进行通信。主要的通信方法如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法 | 方向 | 对象 | 要求 | 含义 |
| DESCRIBE | C->S | P,S | 推荐 | 检查演示或媒体对象的描述，也允许使用接收头指定用户理解的描述格式。DESCRIBE的答复-响应组成媒体RTSP初始阶段 |
| ANNOUNCE | C->S  S->C | P,S | 可选 | 当从用户发往服务器时，ANNOUNCE将请求URL识别的演示或媒体对象描述发送给服务器；反之，ANNOUNCE实时更新连接描述。如新媒体流加入演示，整个演示描述再次发送，而不仅仅是附加组件，使组件能被删除 |
| GET\_PARAMETER | C->S  S->C | P,S | 可选 | GET\_PARAMETER请求检查RUL指定的演示与媒体的参数值。没有实体体时，GET\_PARAMETER也许能用来测试用户与服务器的连通情况 |
| OPTIONS | C->S  S->C | P,S | 要求 | 可在任意时刻发出OPTIONS请求，如用户打算尝试非标准请求，并不影响服务器状态 |
| PAUSE | C->S | P,S | 推荐 | PAUSE请求引起流发送临时中断。如请求URL命名一个流，仅回放和记录被停止；如请求URL命名一个演示或流组，演示或组中所有当前活动的流发送都停止。恢复回放或记录后，必须维持同步。在SETUP消息中连接头超时参数所指定时段期间被暂停后，尽管服务器可能关闭连接并释放资源，但服务器资源会被预订 |
| PLAY | C->S | P,S | 要求 | PLAY告诉服务器以SETUP指定的机制开始发送数据；直到一些SETUP请求被成功响应，客户端才可发布PLAY请求。PLAY请求将正常播放时间设置在所指定范围的起始处，发送流数据直到范围的结束处。PLAY请求可排成队列，服务器将PLAY请求排成队列，顺序执行 |
| RECORD | C->S | P,S | 可选 | 该方法根据演示描述初始化媒体数据记录范围，时标反映开始和结束时间；如没有给出时间范围，使用演示描述提供的开始和结束时间。如连接已经启动，立即开始记录，服务器数据请求URL或其他URL决定是否存储记录的数据；如服务器没有使用URL请求，响应应为201（创建），并包含描述请求状态和参考新资源的实体与位置头。支持现场演示记录的媒体服务器必须支持时钟范围格式，smpte格式没有意义 |
| REDIRECT | S->C | P,S | 可选 | 重定向请求通知客户端连接到另一服务器地址。它包含强制头地址，指示客户端发布URL请求；也可能包括参数范围，以指明重定向何时生效。若客户端要继续发送或接收URL媒体，客户端必须对当前连接发送TEARDOWN请求，而对指定主执新连接发送SETUP请求 |
| SETUP | C->S | S | 要求 | 对URL的SETUP请求指定用于流媒体的传输机制。客户端对正播放的流发布一个SETUP请求，以改变服务器允许的传输参数。如不允许这样做，响应错误为"455 Method Not Valid In This State”。为了透过防火墙，客户端必须指明传输参数，即使对这些参数没有影响 |
| SET\_PARAMETER | C->S  S->C | P,S | 可选 | 这个方法请求设置演示或URL指定流的参数值。请求仅应包含单个参数，允许客户端决定某个特殊请求为何失败。如请求包含多个参数，所有参数可成功设置，服务器必须只对该请求起作用。服务器必须允许参数可重复设置成同一值，但不让改变参数值。注意：媒体流传输参数必须用SETUP命令设置。将设置传输参数限制为SETUP有利于防火墙。将参数划分成规则排列形式，结果有更多有意义的错误指示 |
| TEARDOWN | C->S | P,S | 要求 | TEARDOWN请求停止给定URL流发送，释放相关资源。如URL是此演示URL，任何RTSP连接标识不再有效。除非全部传输参数是连接描述定义的，SETUP请求必须在连接可再次播放前发布 |

注释：P----演示，S----流，C----用户端，S----[服务器](http://baike.baidu.com/view/899.htm)端

RTP（Real-time Transport Protocol）是实时传输协议，定义在RFC3550中。RTP协议详

细说明了在互联网上传递音频和视频的标准[数据包](http://baike.baidu.com/view/25880.htm" \t "_blank)格式。它一开始被设计为一个[多播](http://baike.baidu.com/view/378050.htm)协议，但后来被用在很多[单播](http://baike.baidu.com/view/492261.htm)应用中。RTP协议常用于[流媒体](http://baike.baidu.com/view/794.htm" \t "_blank)系统（配合RTSP协议），视频会议和一键通（Push to Talk）系统（配合[H.323](http://baike.baidu.com/view/76998.htm" \t "_blank)或SIP），使它成为IP电话产业的技术基础。RTP协议和RTP控制协议RTCP一起使用，而且它是建立在[用户数据报协议](http://baike.baidu.com/view/468464.htm" \t "_blank)上的。

RTP标准定义了两个子协议，RTP和RTCP。

数据传输协议RTP，用于实时传输数据。该协议提供的信息包括：时间戳（用于同步）、序列号（用于丢包和重排序检测）、以及负载格式（用于说明数据的编码格式）。

控制协议RTCP，用于QoS反馈和同步媒体流。相对于RTP来说，RTCP所占的带宽非常小，通常只有5%。

RTCP协议周期性向会话中的成员发送控制包，

主要有4个功能：

1 提供数据传输质量的反馈：这个是通过发送者和接收者反馈一起合作实现。

2为每个传输源固定一个标识符，称为CNAME.可用来同步同一个信息源的音频流和视频流。

3 发送RTCP报文，得知成员数量，可以据此估计发包速度。

4 传输会议控制信息

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RTCP报文的种类:

SR：发送者报告，描述作为活跃发送者成员的发送和接收统计数字；

RR：接收者报告，描述非活跃发送者成员的接收统计数字；

SDES：源描述项，其中包括规范名CNAME。

BYE：表明参与者将结束会话。

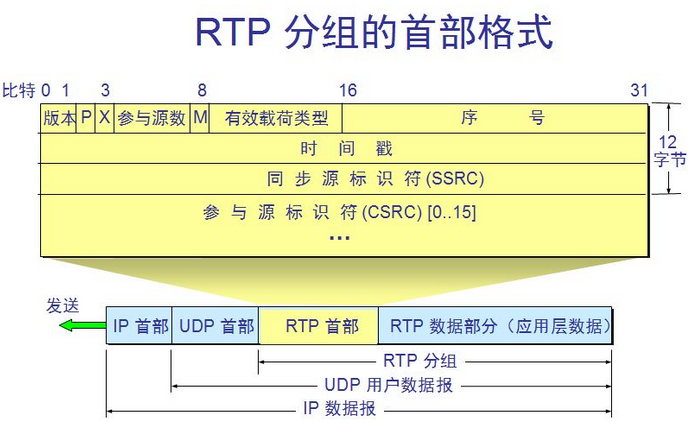
APP：应用描述功能。

SDP 完全是一种会话描述格式 ― 它不属于传输协议 ― 它只使用不同的适当的传输

协议，包括会话通知协议（SAP）、会话初始协议（SIP）、[实时流协议（RTSP）、](http://www.cnblogs.com/qingquan/archive/2011/07/14/2106834.html" \t "_blank)MIME 扩展协议的电子邮件以及超文本传输协议（HTTP）。SDP协议是也是基于文本的协议，这样就能保证协议的可扩展性比较强，这样就使其具有广泛的应用范围。

# 3 示意图

## 3.1 RTP格式

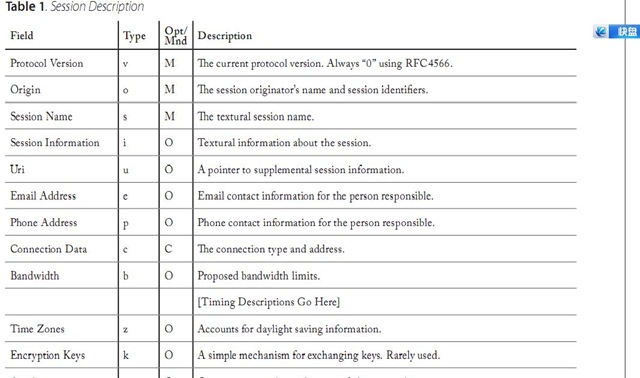


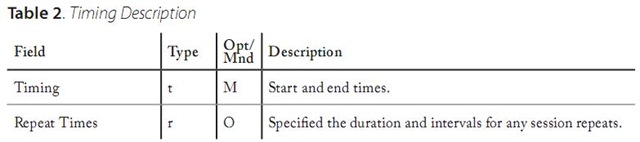
## 3.2 SDP格式

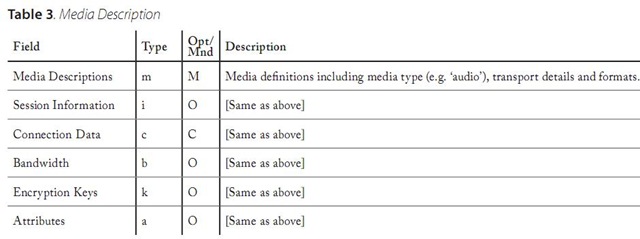
SDP描述由许多文本行组成，文本行的格式为<类型>=<值>，<类型>是一个字母，<值>是结构化的文本串，其格式依<类型>而定。

＜type＞=<value>[CRLF]

常见的fields有：







## 3.3 核心模块调用关系图



# 4 接口定义

## 4.1 RTSP模块

|  |  |
| --- | --- |
| 函数接口定义 | RTSPState\* RtspCreate(char\* url, int ulen);  创建rtsp会话并返回rtsp会话句柄 |
| 函数参数说明 | url: rtsp url 通信地址  ulen: url地址的长度 |
| 返回值 | 返回NULL则失败，其它则成功 |



以上为rtsp会话核心函数的流程图，其中流通道建立还未实现

|  |  |
| --- | --- |
| 函数接口定义 | void RtspDestroy(RTSPState\* rtspcxt);  销毁当前的rtsp会话 |
| 函数参数说明 | rtspcxt: 需要销毁的rtsp会话句柄 |
| 返回值 |  |

相关数据结构请参考

typedef enum

{

UDP\_TRANSPORT\_PROTOCOL,

TCP\_TRANSPORT\_PROTOCOL

}LowTransProtocol;

typedef struct

{

int SockFd;

//remote ip address and port

char sIpAddr[IP\_ADDRESS\_LENGTH+1];

unsigned short uPort;

LowTransProtocol lowtype;

}SocketDataStruct;

enum AVMediaType {

AVMEDIA\_TYPE\_UNKNOWN = -1, ///< Usually treated as AVMEDIA\_TYPE\_DATA

AVMEDIA\_TYPE\_VIDEO,

AVMEDIA\_TYPE\_AUDIO,

AVMEDIA\_TYPE\_DATA, ///< Opaque data information usually continuous

AVMEDIA\_TYPE\_SUBTITLE,

AVMEDIA\_TYPE\_ATTACHMENT, ///< Opaque data information usually sparse

AVMEDIA\_TYPE\_NB

};

/\*\*

\* Network layer over which RTP/etc packet data will be transported.

\*/

enum RTSPLowerTransport {

RTSP\_LOWER\_TRANSPORT\_UDP = 0, /\*\*< UDP/unicast \*/

RTSP\_LOWER\_TRANSPORT\_TCP = 1, /\*\*< TCP; interleaved in RTSP \*/

RTSP\_LOWER\_TRANSPORT\_UDP\_MULTICAST = 2, /\*\*< UDP/multicast \*/

RTSP\_LOWER\_TRANSPORT\_NB,

RTSP\_LOWER\_TRANSPORT\_HTTP = 8, /\*\*< HTTP tunneled - not a proper

transport mode as such,

only for use via AVOptions \*/

RTSP\_LOWER\_TRANSPORT\_CUSTOM = 16, /\*\*< Custom IO - not a public

option for lower\_transport\_mask,

but set in the SDP demuxer based

on a flag. \*/

};

/\*\*

\* Packet profile of the data that we will be receiving. Real servers

\* commonly send RDT (although they can sometimes send RTP as well),

\* whereas most others will send RTP.

\*/

enum RTSPTransport {

RTSP\_TRANSPORT\_RTP, /\*\*< Standards-compliant RTP \*/

RTSP\_TRANSPORT\_RDT, /\*\*< Realmedia Data Transport \*/

RTSP\_TRANSPORT\_RAW, /\*\*< Raw data (over UDP) \*/

RTSP\_TRANSPORT\_NB

};

/\*\*

\* Transport mode for the RTSP data. This may be plain, or

\* tunneled, which is done over HTTP.

\*/

enum RTSPControlTransport {

RTSP\_MODE\_PLAIN, /\*\*< Normal RTSP \*/

RTSP\_MODE\_TUNNEL /\*\*< RTSP over HTTP (tunneling) \*/

};

#define RTSP\_DEFAULT\_PORT 554

#define RTSP\_MAX\_TRANSPORTS 8

#define RTSP\_TCP\_MAX\_PACKET\_SIZE 1472

#define RTSP\_DEFAULT\_NB\_AUDIO\_CHANNELS 1

#define RTSP\_DEFAULT\_AUDIO\_SAMPLERATE 44100

#define RTSP\_RTP\_PORT\_MIN 5000

#define RTSP\_RTP\_PORT\_MAX 65000

/\*\*

\* This describes a single item in the "Transport:" line of one stream as

\* negotiated by the SETUP RTSP command. Multiple transports are comma-

\* separated ("Transport: x-read-rdt/tcp;interleaved=0-1,rtp/avp/udp;

\* client\_port=1000-1001;server\_port=1800-1801") and described in separate

\* RTSPTransportFields.

\*/

typedef struct RTSPTransportField {

/\*\* interleave ids, if TCP transport; each TCP/RTSP data packet starts

\* with a '$', stream length and stream ID. If the stream ID is within

\* the range of this interleaved\_min-max, then the packet belongs to

\* this stream. \*/

int interleaved\_min, interleaved\_max;

/\*\* UDP multicast port range; the ports to which we should connect to

\* receive multicast UDP data. \*/

int port\_min, port\_max;

/\*\* UDP client ports; these should be the local ports of the UDP RTP

\* (and RTCP) sockets over which we receive RTP/RTCP data. \*/

int client\_port\_min, client\_port\_max;

/\*\* UDP unicast server port range; the ports to which we should connect

\* to receive unicast UDP RTP/RTCP data. \*/

int server\_port\_min, server\_port\_max;

/\*\* time-to-live value (required for multicast); the amount of HOPs that

\* packets will be allowed to make before being discarded. \*/

int ttl;

/\*\* transport set to record data \*/

int mode\_record;

//struct sockaddr\_storage destination; /\*\*< destination IP address \*/

char source[INET6\_ADDRSTRLEN + 1]; /\*\*< source IP address \*/

/\*\* data/packet transport protocol; e.g. RTP or RDT \*/

enum RTSPTransport transport;

/\*\* network layer transport protocol; e.g. TCP or UDP uni-/multicast \*/

enum RTSPLowerTransport lower\_transport;

} RTSPTransportField;

/\*\*

\* This describes the server response to each RTSP command.

\*/

typedef struct RTSPMessageHeader {

/\*\* length of the data following this header \*/

int content\_length;

enum RTSPStatusCode status\_code; /\*\*< response code from server \*/

/\*\* number of items in the 'transports' variable below \*/

int nb\_transports;

/\*\* Time range of the streams that the server will stream. In

\* AV\_TIME\_BASE unit, AV\_NOPTS\_VALUE if not used \*/

int64\_t range\_start, range\_end;

/\*\* describes the complete "Transport:" line of the server in response

\* to a SETUP RTSP command by the client \*/

RTSPTransportField transports[RTSP\_MAX\_TRANSPORTS];

int seq; /\*\*< sequence number \*/

/\*\* the "Session:" field. This value is initially set by the server and

\* should be re-transmitted by the client in every RTSP command. \*/

char session\_id[512];

/\*\* the "Location:" field. This value is used to handle redirection.

\*/

char location[4096];

/\*\* the "RealChallenge1:" field from the server \*/

char real\_challenge[64];

/\*\* the "Server: field, which can be used to identify some special-case

\* servers that are not 100% standards-compliant. We use this to identify

\* Windows Media Server, which has a value "WMServer/v.e.r.sion", where

\* version is a sequence of digits (e.g. 9.0.0.3372). Helix/Real servers

\* use something like "Helix [..] Server Version v.e.r.sion (platform)

\* (RealServer compatible)" or "RealServer Version v.e.r.sion (platform)",

\* where platform is the output of $uname -msr | sed 's/ /-/g'. \*/

char server[64];

/\*\* The "timeout" comes as part of the server response to the "SETUP"

\* command, in the "Session: <xyz>[;timeout=<value>]" line. It is the

\* time, in seconds, that the server will go without traffic over the

\* RTSP/TCP connection before it closes the connection. To prevent

\* this, sent dummy requests (e.g. OPTIONS) with intervals smaller

\* than this value. \*/

int timeout;

/\*\* The "Notice" or "X-Notice" field value. See

\* http://tools.ietf.org/html/draft-stiemerling-rtsp-announce-00

\* for a complete list of supported values. \*/

int notice;

/\*\* The "reason" is meant to specify better the meaning of the error code

\* returned

\*/

char reason[256];

/\*\*

\* Content type header

\*/

char content\_type[64];

} RTSPMessageHeader;

/\*\*

\* Client state, i.e. whether we are currently receiving data (PLAYING) or

\* setup-but-not-receiving (PAUSED). State can be changed in applications

\* by calling av\_read\_play/pause().

\*/

enum RTSPClientState {

RTSP\_STATE\_IDLE, /\*\*< not initialized \*/

RTSP\_STATE\_STREAMING, /\*\*< initialized and sending/receiving data \*/

RTSP\_STATE\_PAUSED, /\*\*< initialized, but not receiving data \*/

RTSP\_STATE\_SEEKING, /\*\*< initialized, requesting a seek \*/

};

/\*\*

\* Identify particular servers that require special handling, such as

\* standards-incompliant "Transport:" lines in the SETUP request.

\*/

enum RTSPServerType {

RTSP\_SERVER\_RTP, /\*\*< Standards-compliant RTP-server \*/

RTSP\_SERVER\_REAL, /\*\*< Realmedia-style server \*/

RTSP\_SERVER\_WMS, /\*\*< Windows Media server \*/

RTSP\_SERVER\_NB

};

/\*\*

\* Private data for the RTSP demuxer.

\*

\* @todo Use AVIOContext instead of URLContext

\*/

typedef struct RTSPState {

SocketDataStruct rtsp\_hd; /\* RTSP TCP connection handle \*/

/\*\* number of items in the 'rtsp\_streams' variable \*/

int nb\_rtsp\_streams;

struct RTSPStream \*\*rtsp\_streams; /\*\*< streams in this session \*/

/\*\* indicator of whether we are currently receiving data from the

\* server. Basically this isn't more than a simple cache of the

\* last PLAY/PAUSE command sent to the server, to make sure we don't

\* send 2x the same unexpectedly or commands in the wrong state. \*/

enum RTSPClientState state;

/\*\* the seek value requested when calling av\_seek\_frame(). This value

\* is subsequently used as part of the "Range" parameter when emitting

\* the RTSP PLAY command. If we are currently playing, this command is

\* called instantly. If we are currently paused, this command is called

\* whenever we resume playback. Either way, the value is only used once,

\* see rtsp\_read\_play() and rtsp\_read\_seek(). \*/

int64\_t seek\_timestamp;

int seq; /\*\*< RTSP command sequence number \*/

char RequestMethod[16]; // the RTSP METHOD DEFINED

/\*\* copy of RTSPMessageHeader->session\_id, i.e. the server-provided session

\* identifier that the client should re-transmit in each RTSP command \*/

char session\_id[512];

/\*\* copy of RTSPMessageHeader->timeout, i.e. the time (in seconds) that

\* the server will go without traffic on the RTSP/TCP line before it

\* closes the connection. \*/

int timeout;

/\*\* timestamp of the last RTSP command that we sent to the RTSP server.

\* This is used to calculate when to send dummy commands to keep the

\* connection alive, in conjunction with timeout. \*/

int64\_t last\_cmd\_time;

/\*\* the negotiated data/packet transport protocol; e.g. RTP or RDT \*/

enum RTSPTransport transport;

/\*\* the negotiated network layer transport protocol; e.g. TCP or UDP

\* uni-/multicast \*/

enum RTSPLowerTransport lower\_transport;

/\*\* brand of server that we're talking to; e.g. WMS, REAL or other.

\* Detected based on the value of RTSPMessageHeader->server or the presence

\* of RTSPMessageHeader->real\_challenge \*/

enum RTSPServerType server\_type;

/\*\* the "RealChallenge1:" field from the server \*/

char real\_challenge[64];

/\*\* plaintext authorization line (username:password) \*/

char auth[128];

/\*\* authentication state \*/

//HTTPAuthState auth\_state;

/\*\* The last reply of the server to a RTSP command \*/

char last\_reply[2048]; /\* XXX: allocate ? \*/

/\*\* RTSPStream->transport\_priv of the last stream that we read a

\* packet from \*/

void \*cur\_transport\_priv;

/\*\* The following are used for Real stream selection \*/

//@{

/\*\* whether we need to send a "SET\_PARAMETER Subscribe:" command \*/

int need\_subscription;

/\*\* stream setup during the last frame read. This is used to detect if

\* we need to subscribe or unsubscribe to any new streams. \*/

//enum AVDiscard \*real\_setup\_cache;

/\*\* current stream setup. This is a temporary buffer used to compare

\* current setup to previous frame setup. \*/

//enum AVDiscard \*real\_setup;

/\*\* the last value of the "SET\_PARAMETER Subscribe:" RTSP command.

\* this is used to send the same "Unsubscribe:" if stream setup changed,

\* before sending a new "Subscribe:" command. \*/

char last\_subscription[1024];

//@}

/\*\* The following are used for RTP/ASF streams \*/

//@{

/\*\* ASF demuxer context for the embedded ASF stream from WMS servers \*/

//AVFormatContext \*asf\_ctx;

/\*\* cache for position of the asf demuxer, since we load a new

\* data packet in the bytecontext for each incoming RTSP packet. \*/

unsigned long long int asf\_pb\_pos;

//@}

/\*\* some MS RTSP streams contain a URL in the SDP that we need to use

\* for all subsequent RTSP requests, rather than the input URI; in

\* other cases, this is a copy of AVFormatContext->filename. \*/

char control\_uri[1024];

/\*\* The following are used for parsing raw mpegts in udp \*/

//@{

//struct MpegTSContext \*ts;

int recvbuf\_pos;

int recvbuf\_len;

//@}

/\*\* Additional output handle, used when input and output are done

\* separately, eg for HTTP tunneling. \*/

//URLContext \*rtsp\_hd\_out;

/\*\* RTSP transport mode, such as plain or tunneled. \*/

enum RTSPControlTransport control\_transport;

/\* Number of RTCP BYE packets the RTSP session has received.

\* An EOF is propagated back if nb\_byes == nb\_streams.

\* This is reset after a seek. \*/

int nb\_byes;

/\*\* Reusable buffer for receiving packets \*/

unsigned char\* recvbuf;

/\*\*

\* A mask with all requested transport methods

\*/

int lower\_transport\_mask;

/\*\*

\* The number of returned packets

\*/

unsigned long long int packets;

/\*\*

\* Polling array for udp

\*/

//struct pollfd \*p;

/\*\*

\* Whether the server supports the GET\_PARAMETER method.

\*/

int get\_parameter\_supported;

/\*\*

\* Do not begin to play the stream immediately.

\*/

int initial\_pause;

/\*\*

\* Option flags for the chained RTP muxer.

\*/

int rtp\_muxer\_flags;

/\*\* Whether the server accepts the x-Dynamic-Rate header \*/

int accept\_dynamic\_rate;

/\*\*

\* Various option flags for the RTSP muxer/demuxer.

\*/

int rtsp\_flags;

/\*\*

\* Mask of all requested media types

\*/

int media\_type\_mask;

/\*\*

\* Minimum and maximum local UDP ports.

\*/

int rtp\_port\_min, rtp\_port\_max;

/\*\*

\* Timeout to wait for incoming connections.

\*/

int initial\_timeout;

/\*\*

\* timeout of socket i/o operations.

\*/

int stimeout;

/\*\*

\* Size of RTP packet reordering queue.

\*/

int reordering\_queue\_size;

/\*\*

\* User-Agent string

\*/

char \*user\_agent;

pthread\_t m\_thread;

} RTSPState;

#define RTSP\_FLAG\_FILTER\_SRC 0x1 /\*\*< Filter incoming UDP packets -

receive packets only from the right

source address and port. \*/

#define RTSP\_FLAG\_LISTEN 0x2 /\*\*< Wait for incoming connections. \*/

#define RTSP\_FLAG\_CUSTOM\_IO 0x4 /\*\*< Do all IO via the AVIOContext. \*/

#define RTSP\_FLAG\_RTCP\_TO\_SOURCE 0x8 /\*\*< Send RTCP packets to the source

address of received packets. \*/

#define RTSP\_FLAG\_PREFER\_TCP 0x10 /\*\*< Try RTP via TCP first if possible. \*/

typedef struct RTSPSource {

char addr[128]; /\*\*< Source-specific multicast include source IP address (from SDP content) \*/

} RTSPSource;

struct sockaddr\_storage\_t {

#if HAVE\_STRUCT\_SOCKADDR\_SA\_LEN

unsigned char ss\_len;

unsigned char ss\_family;

#else

unsigned short ss\_family;

#endif /\* HAVE\_STRUCT\_SOCKADDR\_SA\_LEN \*/

char ss\_pad1[6];

long long int ss\_align;

char ss\_pad2[112];

};

struct addrinfo\_t {

int ai\_flags;

int ai\_family;

int ai\_socktype;

int ai\_protocol;

int ai\_addrlen;

struct sockaddr \*ai\_addr;

char \*ai\_canonname;

struct addrinfo\_t \*ai\_next;

};

/\*\*

\* Describe a single stream, as identified by a single m= line block in the

\* SDP content. In the case of RDT, one RTSPStream can represent multiple

\* AVStreams. In this case, each AVStream in this set has similar content

\* (but different codec/bitrate).

\*/

typedef struct RTSPStream {

SocketDataStruct rtp\_handle; /\*\*< RTP stream handle (if UDP) \*/

void \*transport\_priv; /\*\*< RTP/RDT parse context if input, RTP AVFormatContext if output \*/

enum AVMediaType mediaType;

/\*\* corresponding stream index, if any. -1 if none (MPEG2TS case) \*/

int stream\_index;

/\*\* interleave IDs; copies of RTSPTransportField->interleaved\_min/max

\* for the selected transport. Only used for TCP. \*/

int interleaved\_min, interleaved\_max;

int server\_port\_min, server\_port\_max;

char control\_url[1024]; /\*\*< url for this stream (from SDP) \*/

/\*\* The following are used only in SDP, not RTSP \*/

//@{

int sdp\_port; /\*\*< port (from SDP content) \*/

struct sockaddr\_storage\_t sdp\_ip; /\*\*< IP address (from SDP content) \*/

//struct sockaddr\_storage sdp\_ip; /\*\*< IP address (from SDP content) \*/

int nb\_include\_source\_addrs; /\*\*< Number of source-specific multicast include source IP addresses (from SDP content) \*/

struct RTSPSource \*\*include\_source\_addrs; /\*\*< Source-specific multicast include source IP addresses (from SDP content) \*/

int nb\_exclude\_source\_addrs; /\*\*< Number of source-specific multicast exclude source IP addresses (from SDP content) \*/

struct RTSPSource \*\*exclude\_source\_addrs; /\*\*< Source-specific multicast exclude source IP addresses (from SDP content) \*/

int sdp\_ttl; /\*\*< IP Time-To-Live (from SDP content) \*/

int sdp\_payload\_type; /\*\*< payload type \*/

//@}

/\*\* The following are used for dynamic protocols (rtpdec\_\*.c/rdt.c) \*/

//@{

/\*\* handler structure \*/

//RTPDynamicProtocolHandler \*dynamic\_handler;

/\*\* private data associated with the dynamic protocol \*/

//PayloadContext \*dynamic\_protocol\_context;

//@}

/\*\* Enable sending RTCP feedback messages according to RFC 4585 \*/

int feedback;

char crypto\_suite[40];

char crypto\_params[100];

} RTSPStream;

## 4.2 RTP/RTCP模块

RTP协议包含了RTCP的定义，目前有很多开源项目可以作为参考，以下模型设计参考中的RTP模块设计实现。

|  |  |
| --- | --- |
| 函数接口定义 | int RtpAlloc(struct rtp\_sock \*\*rtp);  创建rtp 会话 |
| 函数参数说明 | rtp: 需要存储的rtp会话指针 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtpDestroy(struct rtp\_sock \*rs);  销毁rtp 会话 |
| 函数参数说明 | rs: 需要销毁的rtp会话指针 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtpEncode(struct rtp\_sock \*rs, bool marker, unsigned char pt,  unsigned int ts, struct mbuf \*mb);  rtp头编码 |
| 函数参数说明 | rs: rtp会话句柄  marker:　是否设置标志位，1表示设置，0则不设置 pt:　rtp包的类型  ts: rtp包时间戳  mb:需要编码的数据结构 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtpSendH264(unsigned char\* pVdoData, int nVdoData);  对H264进行rtp封包并发送 |
| 函数参数说明 | pVdoData: 输入一帧视频数据  nVdoData: 输入数据的大小 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtpSendAAC(unsigned char \*buff, int size);  对AAC进行rtp封包并发送 |
| 函数参数说明 | buff: 输入一包音频数据  size: 输入数据的大小 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtcpSessAlloc(struct rtcp\_sess \*\*sessp, struct rtp\_sock \*rs);  rtcp 会话创建 |
| 函数参数说明 | sessp: 需要设置的rtcp会话指针  rs: rtp会话句柄 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | void SessDestructor(struct rtcp\_sess \*sessp);  rtcp 会话销毁 |
| 函数参数说明 | sessp: 需要销毁的rtcp会话指针 |
| 返回值 |  |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | void RtcpStart(struct rtp\_sock \*rs, const char \*cname,  const struct sa \*peer);  开始rtcp会话 |
| 函数参数说明 | rs: rtp会话句柄  cname: 输入一个规范化的名字  peer: rtcp对端地址信息 |
| 返回值 |  |
| 备注 | 内部实现一个定时器，定时更新rtcp会话信息并发送rtcp报文。函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtcpSendApp(struct rtp\_sock \*rs, const char name[4],  const uint8\_t \*data, size\_t len);  发送rtcp自定义的APP数据包 |
| 函数参数说明 | rs: rtp会话句柄  name: 4个八位的字符串  data: 自定义app数据  len: app数据的长度 |
| 返回值 | 返回已发送的数据长度 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtcpSendFIR(struct rtp\_sock \*rs, uint32\_t ssrc);  发送Full INTRA-frame 包 |
| 函数参数说明 | rs: rtp会话句柄  ssrc: 源唯一识别码 |
| 返回值 | 返回已发送的数据长度 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtcpSendNACK(struct rtp\_sock \*rs, uint16\_t fsn, uint16\_t blp);  发送NACK包 |
| 函数参数说明 | rs: rtp会话句柄  fsn: 第一个rtp丢包的序列号  blp: 丢包的位掩码 |
| 返回值 | 返回已发送的数据长度 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int RtcpSendPLI(struct rtp\_sock \*rs, uint32\_t fb\_ssrc);  发送 Picture Loss Indication 包 |
| 函数参数说明 | rs: rtp会话句柄  fb\_ssrc: 反馈的SSRC |
| 返回值 | 返回已发送的数据长度 |
| 备注 | 函数还未实现 |

相关数据结构请参考以下

/\*\* Defines a Socket Address \*/

struct sa {

union {

struct sockaddr sa;

struct sockaddr\_in in;

uint8\_t padding[28];

} u;

int len;

};

/\*\* Defines the RTP header \*/

struct rtp\_header {

unsigned char ver; /\*\*< RTP version number \*/

int pad; /\*\*< Padding bit \*/

int ext; /\*\*< Extension bit \*/

unsigned char cc; /\*\*< CSRC count \*/

int m; /\*\*< Marker bit \*/

unsigned char pt; /\*\*< Payload type \*/

unsigned short int seq; /\*\*< Sequence number \*/

unsigned int ts; /\*\*< Timestamp \*/

unsigned int ssrc; /\*\*< Synchronization source \*/

unsigned int csrc[16]; /\*\*< Contributing sources \*/

struct {

unsigned short int type; /\*\*< Defined by profile \*/

unsigned short int len; /\*\*< Number of 32-bit words \*/

} x;

};

/\*\* Defines a memory buffer \*/

struct mbuf {

unsigned char \*buf; /\*\*< Buffer memory \*/

size\_t size; /\*\*< Size of buffer \*/

size\_t pos; /\*\*< Position in buffer \*/

size\_t end; /\*\*< End of buffer \*/

};

/\*\* SDES Types \*/

enum rtcp\_sdes\_type {

RTCP\_SDES\_END = 0, /\*\*< End of SDES list \*/

RTCP\_SDES\_CNAME = 1, /\*\*< Canonical name \*/

RTCP\_SDES\_NAME = 2, /\*\*< User name \*/

RTCP\_SDES\_EMAIL = 3, /\*\*< User's electronic mail address \*/

RTCP\_SDES\_PHONE = 4, /\*\*< User's phone number \*/

RTCP\_SDES\_LOC = 5, /\*\*< Geographic user location \*/

RTCP\_SDES\_TOOL = 6, /\*\*< Name of application or tool \*/

RTCP\_SDES\_NOTE = 7, /\*\*< Notice about the source \*/

RTCP\_SDES\_PRIV = 8 /\*\*< Private extension \*/

};

/\*\* SDES item \*/

struct rtcp\_sdes\_item {

enum rtcp\_sdes\_type type; /\*\*< Type of item (enum rtcp\_sdes\_type) \*/

unsigned char length; /\*\*< Length of item (in octets) \*/

char \*data; /\*\*< Text, not null-terminated \*/

};

/\*\* Reception report block \*/

struct rtcp\_rr {

unsigned int ssrc; /\*\*< Data source being reported \*/

unsigned int fraction:8; /\*\*< Fraction lost since last SR/RR \*/

int lost:24; /\*\*< Cumul. no. pkts lost (signed!) \*/

unsigned int last\_seq; /\*\*< Extended last seq. no. received \*/

unsigned int jitter; /\*\*< Interarrival jitter \*/

unsigned int lsr; /\*\*< Last SR packet from this source \*/

unsigned int dlsr; /\*\*< Delay since last SR packet \*/

};

/\*\* One RTCP Message \*/

struct rtcp\_msg {

/\*\* RTCP Header \*/

struct rtcp\_hdr {

unsigned int version:2; /\*\*< Protocol version \*/

unsigned int p:1; /\*\*< Padding flag \*/

unsigned int count:5; /\*\*< Varies by packet type \*/

unsigned int pt:8; /\*\*< RTCP packet type \*/

unsigned short int length; /\*\*< Packet length in words \*/

} hdr;

union {

/\*\* Sender report (SR) \*/

struct {

unsigned int ssrc; /\*\*< Sender generating report \*/

unsigned int ntp\_sec; /\*\*< NTP timestamp - seconds \*/

unsigned int ntp\_frac; /\*\*< NTP timestamp - fractions \*/

unsigned int rtp\_ts; /\*\*< RTP timestamp \*/

unsigned int psent; /\*\*< RTP packets sent \*/

unsigned int osent; /\*\*< RTP octets sent \*/

struct rtcp\_rr \*rrv; /\*\*< Reception report blocks \*/

} sr;

/\*\* Reception report (RR) \*/

struct {

unsigned int ssrc; /\*\*< Receiver generating report\*/

struct rtcp\_rr \*rrv; /\*\*< Reception report blocks \*/

} rr;

/\*\* Source Description (SDES) \*/

struct rtcp\_sdes {

unsigned int src; /\*\*< First SSRC/CSRC \*/

struct rtcp\_sdes\_item \*itemv; /\*\*< SDES items \*/

unsigned int n; /\*\*< Number of SDES items \*/

} \*sdesv;

/\*\* BYE \*/

struct {

unsigned int \*srcv; /\*\*< List of sources \*/

char \*reason; /\*\*< Reason for leaving (opt.) \*/

} bye;

/\*\* Application-defined (APP) \*/

struct {

unsigned int src; /\*\*< SSRC/CSRC \*/

char name[4]; /\*\*< Name (ASCII) \*/

unsigned char \*data; /\*\*< Application data (32 bits) \*/

size\_t data\_len; /\*\*< Number of data bytes \*/

} app;

/\*\* Full INTRA-frame Request (FIR) packet \*/

struct {

unsigned int ssrc; /\*\*< SSRC for sender of this packet \*/

} fir;

/\*\* Negative ACKnowledgements (NACK) packet \*/

struct {

unsigned int ssrc; /\*\*< SSRC for sender of this packet \*/

unsigned short int fsn; /\*\*< First Sequence Number lost \*/

unsigned short int blp; /\*\*< Bitmask of lost packets \*/

} nack;

/\*\* Feedback (RTPFB or PSFB) packet \*/

struct {

unsigned int ssrc\_packet;

unsigned int ssrc\_media;

unsigned int n;

/\*\* Feedback Control Information (FCI) \*/

union {

struct gnack {

unsigned short int pid;

unsigned short int blp;

} \*gnackv;

struct sli {

unsigned short int first;

unsigned short int number;

unsigned char picid;

} \*sliv;

struct mbuf \*afb;

void \*p;

} fci;

} fb;

} r;

};

typedef void (\*rtp\_recv\_h)(const struct sa \*src, const struct rtp\_header \*hdr,

struct mbuf \*mb, void \*arg);

typedef void (\*rtcp\_recv\_h)(const struct sa \*src, struct rtcp\_msg \*msg,

void \*arg);

/\*\* Defines an RTP Socket \*/

struct rtp\_sock {

/\*\* Encode data \*/

struct {

unsigned short int seq; /\*\*< Sequence number \*/

unsigned int ssrc; /\*\*< Synchronizing source \*/

} enc;

int proto; /\*\*< Transport Protocol \*/

void \*sock\_rtp; /\*\*< RTP Socket \*/

void \*sock\_rtcp; /\*\*< RTCP Socket \*/

struct sa local; /\*\*< Local RTP Address \*/

struct sa rtcp\_peer; /\*\*< RTCP address of Peer \*/

rtp\_recv\_h recvh; /\*\*< RTP Receive handler \*/

rtcp\_recv\_h rtcph; /\*\*< RTCP Receive handler \*/

void \*arg; /\*\*< Handler argument \*/

struct rtcp\_sess \*rtcp; /\*\*< RTCP Session \*/

int rtcp\_mux; /\*\*< RTP/RTCP multiplexing \*/

pthread\_t udp\_rtp\_TID;

pthread\_t udp\_rtcp\_TID;

};

#define bool int

## 4.3 SDP模块

|  |  |
| --- | --- |
| 函数接口定义 | int SdpPackage(\_sdp\_construct \*ssdp, const char \*content);  封装媒体sdp信息 |
| 函数参数说明 | ssdp: 传入封装SDP信息的媒体结构  content: 存储SDP的字符串，需要能存放至少1024字节 |
| 返回值 | 返回0则成功，其它则失败 |
| 备注 | 函数还未实现 |

|  |  |
| --- | --- |
| 函数接口定义 | int SdpParse(RTSPState \*rt, const char \*content);  解析媒体sdp信息 |
| 函数参数说明 | rt: rtsp上下文句柄  content: 媒体sdp内容 |
| 返回值 | 返回0则成功，其它则失败 |

相关数据结构定义如下

/\*需要组装的SDP信息结构体\*/

typedef struct {

char videoSPSPPS[128]; // H264 SPS PPS information

char videoProfileID[16]; //H264 video profile level id

int videoPktMode; // video package mode

int audioRate; // audio rate

int audioChannel; //audio channel

char audioConfig[16]; //audio AAC config

}\_sdp\_construct;

形成的SDP字符串如下格式，对应以上结构体填充实际对应的参数(红色字段)

v=0\r\n

o=- 0 0 IN IP4 127.0.0.1\r\n

s=No Name\r\n

c=IN IP4 0.0.0.0\r\n"\

t=0 0\r\n

a=tool:libavformat 56.15.100\r\n

m=video 0 RTP/AVP 96\r\n

b=AS:100\r\n

a=rtpmap:96 H264/90000\r\n

a=fmtp:96 packetization-mode=1; sprop-parameter-sets=videoSPSPPS; profile-level-id=videoProfileID\r\n

a=control:streamid=0\r\n

m=audio 0 RTP/AVP 97\r\n

b=AS:48\r\n

a=rtpmap:97 MPEG4-GENERIC/audioRate/audioChannel\r\n

a=fmtp:97 profile-level-id=1;mode=AAC-hbr;sizelength=13;indexlength=3;indexdeltalength=3; config=audioConfig\r\n

a=control:streamid=1\r\n