RTP(Real-time Transport Protocol)是用于Internet上针对多媒体数据流的一种传输协议

RTP被定义为在一对一或一对多的传输情况下工作

RTP通常使用UDP来传送数据。但RTP也可以在TCP或ATM等其他协议之上工作

RTCP(Real-time Transport Control Protocol) 提供流量控制和拥塞控制。

在RTP会话期间,各参与者周期性地传送RTCP包.

RTCP包中含有已发送的数据包的数量、丢失的数据包的数量等统计资料

服务器可以利用这些信息动态地改变传输速率，甚至改变有效载荷类型

开始一个RTP会话时将使用两个端口:一个给RTP一个给 RTCP

实时流协议RTSP(Real-time Streaming Protocol 一对多 它使用TCP或RTP完成数据传输

RTSP在体系结构上位于RTP和RTCP之上

HTTP请求由客户机发出，服务器作出响应；使用RTSP时，客户机和服务器都可以发出请求，即RTSP可以是双向的

RTSP协议是有状态的协议，而HTTP是无状态的协议 RTSP通过维护一个session来维护其状态的转换

RTSP协议的默认端口是554，默认的承载协议为TCP

控制流和数据流是分开的

rfc2326（rtsp）；rfc2327（sdp）

//音频是什么格式的。。 比如LATM还是ADTS的的

语法和运作跟HTTP 1.1类似

RTSP协议分析

http://blog.chinaunix.net/uid-25968088-id-3518900.html

RTSP 协议格式：

请求命令的格式为：

**METHOD** **URL** CR LF

**Field1:value** CR LF

Field2:value CR LF

......

Fieldn:value CR LF

CR LF

应答的格式为：

**RTSP**/major\_version.minor\_version **status** CR LF

Field1:value CR LF

Field2:value CR LF

......

Fieldn:value CR LF

CR LF

1.C->S:OPTION request //询问S 有哪些方法可用

1.S->C:OPTION response //S 回应信息中包括提供的所有可用方法

2.C->S:DESCRIBE request //要求得到S 提供的媒体初始化描述信息

2.S->C:DESCRIBE response //S 回应媒体初始化描述信息，主要是sdp

3.C->S:SETUP request //设置会话的属性，以及传输模式，提醒S 建立会

话

3.S->C:SETUP response //S 建立会话，返回会话标识符，以及会话相关信息

4.C->S:PLAY request //C 请求播放

4.S->C:PLAY response //S 回应该请求的信息

S->C:发送流媒体数据

5.C->S:TEARDOWN request //C 请求关闭会话

5.S->C:TEARDOWN response //S 回应该请求

第一步，只要服务器客户端约定好，有哪些方法可用，则option 请求可以不要。

第二步，如果我们有其他途径得到媒体初始化描述信息（比如http 请求等等），则我们也不需要通过rtsp 中的describe 请求来完成。

第五步，可以根据系统需求的设计来决定是否需要

第3和4步是必需的

命令 与 状态转换

DESCRIBE(协商用于描述媒体信息协议 ) -- > SETUP(协商传输用的协议) --> PLAY --> PLAY(scale=-32) --> PLAY(scale=1.0) --> GET\_PARAMETER --> TEARDOWN

rtsp交互包括options，describe，setup，play，teardown，pause 6个部分

DESCRIBE **rtsp://**118.122.89.27:**554**/**live**/ch10083121594790060557**.sdp**?playtype=1&boid=001&backupagent=118.122.89.27:554&clienttype=1&time=20100929182111+08&life=172800&ifpricereqsnd=1&vcdnid=001&userid=123&mediaid=ch10083121594790060557&ctype=2&TSTVTimeLife=1800&contname=&authid=0&UserLiveType=1&nodelevel=3 RTSP/1.0   
// 媒体URL   
**Accept: application/sdp   
//协商用于描述媒体信息协议**  
CSeq: 1

User-Agent:ZTE Ltd.co RTSP protocal verion 1.0 guid-2.1.1.100/B519D290-C0EC-EE35-7368-893BE4C0B347   
//User Agnet信息，显示是中兴的服务器，1.0的版本。如果是Helix服务器的话会有Helix 服务器的标识。   
x-NAT:2.1.1.100:20081   
//主要用于NAT穿透   
x-zmssRtxSdp: yes

RTSP/1.0 200 OK

//应答编号

Server: ZXUSS100 1.0

Cache-Control: no-cache

Content-Base: rtsp://118.122.89.27:554/live/ch10083121594790060557.sdp/

Content-Length: 320

//内容长度信息

Content-Type: application/sdp

**//描述内容信息所用的协议 使用sdp协议**

CSeq: 1

Date: Wed, 29 Sep 2010 10:20:38 GMT

Expires: Wed, 29 Sep 2010 10:20:38 GMT

**// SDP 描述信息 前面都是RTSP 的 field:value**

v=0

o=- 296874273 1 IN IP4 118.122.89.27

s=envivio

c=IN IP4 0.0.0.0

b=AS:1500

t=0 0

a=**range:clock**=20100929T095038.00Z-20100929T102038.00Z

m=video 5140 RTP/AVPF 33 96

a=**control:trackID**=2

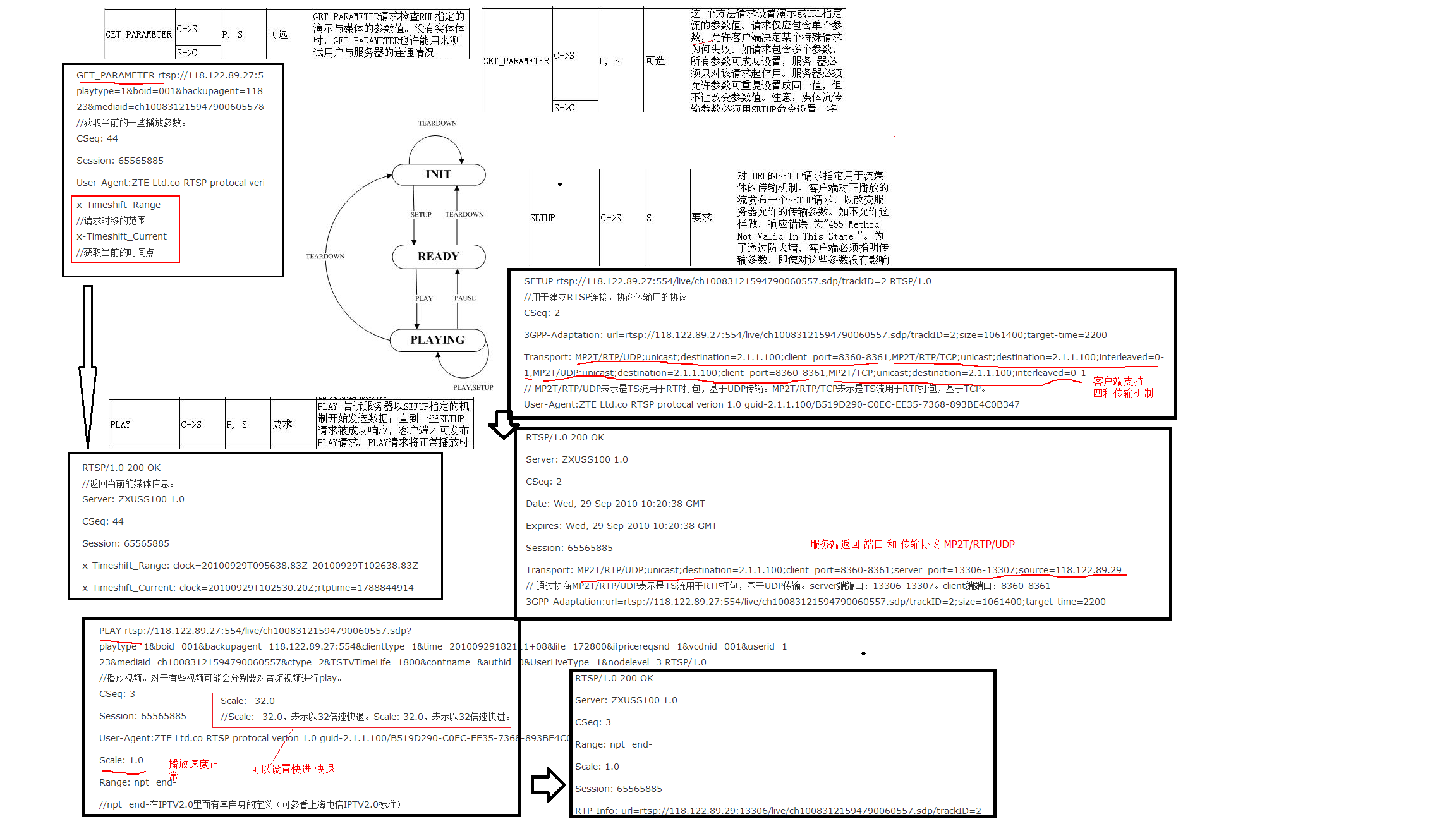
a=rtpmap:33 MP2T/90000

a=3GPP-Adaptation-Support:5

a=rtcp-fb:33 nack

a=rtpmap:96 rtx/90000

a=fmtp:96 apt=33;rtx-time=0



**RTSP状态码**

还有很多）

"200"; OK

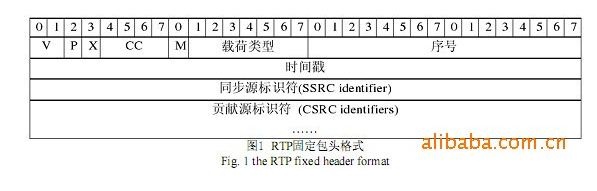
"400" ; Bad Request

"403" ; Forbidden

"404" ; Not Found

RTP 协议实际上是由实时传输协议RTP（Real-time Transport Protocol）和实时传输控制协议RTCP（Real-time Transport Control Protocol）两部分组成。RTP 协议基于多播或单播网络为用户提供连续媒体数据的实时传输服务；RTCP 协议是 RTP 协议的控制部分，用于实时监控数据传输质量，为系统提供拥塞控制和流控制。

每一个 RTP 数据包都由固定包头（Header ）和载荷（Payload）两个部分组成，其中包头前12个字节的含义是固定的，而载荷则可以是音频或视频数据。



（12个字节 上面每行4个字节 rtph[0] = 10 00 00 00 B rtph[1] = 96 rtph[2~3] = 序号 rtph[4~7] = 时间戳 rtph[8~11] = 同步 源 标识符 CSRC不属于包头，而是由CC决定数量，用于混合器，目前为0）

其中比较关键的参数设置解释如下：

（1）标示位（M ）：1 位，该标示位的含义一般由具体的媒体应用框架（profile ）定义， 目的在于标记处RTP 流中的重要事件。

（2）载荷类型（PT）：7 位，用来指出RTP负载的具体格式。

在RFC3551中，对常用的音视频格式的RTP 传输载荷类型做了默认的取值规定，例如，类型2 表明该RTP数据包中承载的是用ITU G.721 算法编码的语音数据，采用频率为 8000HZ，并且采用单声道。

（3）序号:16 位，每发送一个 RTP 数据包，序号加 1。接受者可以用它来检测分组丢失和恢复分组顺序。

（4）时间戳：32 位，时间戳表示了 RTP 数据分组中第一个字节的采样时间，反映出各RTP 包相对于时间戳初始值的偏差。对于RTP 发送端而言，采样时间必须来源于一个线性单调递增的时钟。

从 RTP 数据包的格式不难看出，它包含了传输媒体的类型、格式、序列号、时间戳以及是否有附加数据等信息。这些都为实时的流媒体传输提供了相应的基础。而 传输控制协议RTCP为 RTP传输提供了拥塞控制和流控制，它的具体包结构和各字段的含义可参考RFC3550，此处不再赘述。

根据RFC3984[3]，这里详细给出各个位的具体设置。

V：版本号，2 位。根据RFC3984，目前使用的RTP 版本号应设为0x10。

P：填充位，1 位。当前不使用特殊的加密算法，因此该位设为 0。

X：扩展位，1 位。当前固定头后面不跟随头扩展，因此该位也为 0。

CC：CSRC 计数，4 位。表示跟在 RTP 固定包头后面CSRC 的数目，对于本文所要实现的基本的流媒体服务器来说，没有用到混合器，该位也设为 0x0。

M：标示位，1 位。如果当前 NALU为一个接入单元最后的那个NALU，那么将M位置 1；或者当前RTP 数据包为一个NALU 的最后的那个分片时（NALU 的分片在后面讲述），M位置 1。其余情况下M 位保持为 0。

PT：载荷类型，7 位。对于H.264 视频格式，当前并没有规定一个默认的PT 值。因此选用大于 95 的值可以。此处设为0x60（十进制96）。（spydroid用的都是 rtph[1] = 96'D 60'H 0110 0000 '）

SQ：序号，16 位。序号的起始值为随机值，此处设为 0，每发送一个RTP 数据包，序号值加 1。

TS：时间戳，32 位。同序号一样，时间戳的起始值也为随机值，此处设为0。根据RFC3984， 与时间戳相应的时钟频率必须为90000HZ。

SSRC：同步源标示，32 位。SSRC应该被随机生成，以使在同一个RTP会话期中没有任何两个同步源具有相同的SSRC 识别符。(spyandroid中每个RtpSocket实例对应随机生成的一个值)此处仅有一个同步源，因此将其设为0x12345678。

RTP协议的H.264视频传输

NALU H.264 的基本流（elementary stream,ES）的结构分为两层，包括视频编码层（VCL）和网络适配层（NAL）。

视频编码层负责高效的视频内容表示，而网络适配层负责以网络所要 求的恰当的方式对数据进行打包和传送。

H.264 的基本流由一系列NALU （Network Abstraction Layer Unit ）组成，不同的NALU数据量各不相同。

在每个NALU 前添加起始码：0x000001(起始码不属于NALU，但在NALU header之前，用来隔开每个NALU)，用来指示一个 NALU的起始和终止位置。

每个 NALU单元由一个字节的 NALU头（NALU Header）和若干个字节的载荷数据（RBSP）组成

(上面是一个字节 0 对应的是byteBuffer的最高位 header[4]&0x80 )

( spyDroid中，会根据 MediaCodec::dequeueOutputBuffer buffer[0,1,2,3] 4个字节 确认是否一个NAL开始 00 00 00 01

然后根据 第5个字节 buffer[4] && 0x1F 取出 NALU的类型 ，如果是7/8 代表负载数据是 sps pps 等 )

F：forbidden\_zero\_bit.1 位，如果有语法冲突，则为 1。当网络识别此单元存在比特错误时，可将其设为 1，以便接收方丢掉该单元。

NRI：nal\_ref\_idc.2 位，用来指示该NALU 的重要性等级。值越大，表示当前NALU越重要。具体大于0 时取何值，没有具体规定。

Type：5 位，指出NALU 的类型。具体如表1 所示：



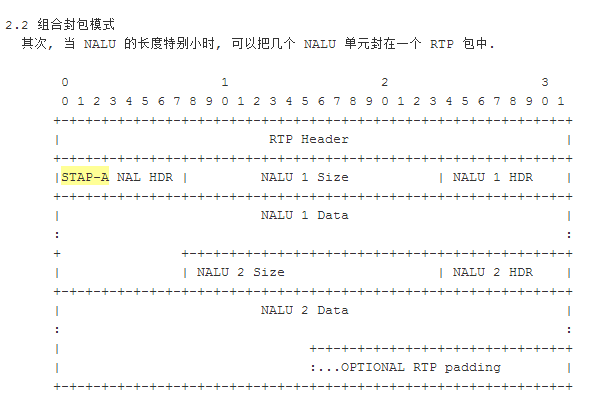
Type: 5 个比特.  
  nal\_unit\_type. 这个 NALU 单元的类型. 简述如下:

  0     没有定义  
  1-23  NAL单元  单个 NAL 单元包.  
  24    STAP-A   单一时间的组合包  
  25    STAP-B   单一时间的组合包  
  26    MTAP16   多个时间的组合包  
  27    MTAP24   多个时间的组合包  
  28    FU-A     分片的单元  
  29    FU-B     分片的单元  
  30-31 没有定义

 可能的结构类型分别有:

  1. 单一 NAL 单元模式  
     即一个 RTP 包仅由一个完整的 NALU 组成. 这种情况下 RTP NAL 头类型字段和原始的 H.264的  
  NALU 头类型字段是一样的.

  2. 组合封包模式  
    即可能是由多个 NAL 单元组成一个 RTP 包. 分别有4种组合方式: STAP-A, STAP-B, MTAP16, MTAP24.  
  那么这里的类型值分别是 24, 25, 26 以及 27.



  3. 分片封包模式  
    用于把一个 NALU 单元封装成多个 RTP 包. 存在两种类型 FU-A 和 FU-B. 类型值分别是 28 和 29.

H.264 视频 RTP 负载格式 <http://www.cppblog.com/czanyou/archive/2008/11/26/67940.html>

NRI 值为 7 和 8 的NALU 分别为序列参数集（sps）和图像参数集（pps）。参数集是一组很少改变的，为大量VCL NALU 提供解码信息的数据。

其中序列参数集作用于一系列连续的编码图像，

而图像参数集作用于编码视频序列中一个或多个独立的图像。

如果\*没能正确接收到这两个参 数集，那么其他NALU 也是无法解码的。因此它们一般在发送其它 NALU 之前发送(SpyDroid中使用RTSP先发送)，并且使用不同的信道或者更加可靠的传输协议（如TCP）进行传输，也可以重复传输。

H.264/AVC视频编码标准中，整个系统框架被分为了两个层面：视频编码层面（VCL）和网络抽象层面（NAL）

其中，前者负责有效表示视频数据的内容，而后者则负责格式化数据并提供头信息，以保证数据适合各种信道和存储介质上的传输。

平时的每帧数据就是一个NAL单元（SPS与PPS除外）。

在实际的H264数据帧中，往往帧前面带有00 00 00 01 或 00 00 01分隔符，一般来说编码器编出的首帧数据为PPS与SPS，接着为I帧…



  NALU类型是我们判断帧类型的利器

以00 00 00 01分割之后的下一个字节就是NALU类型，将其转为二进制数据后，解读顺序为从左往右算，如下:

（1）第1位禁止位，值为1表示语法出错

（2）第2~3位为参考级别

（3）第4~8为是nal单元类型

其中0x67的二进制码为：

0110 0111

4-8为00111，转为十进制7，参考第二幅图：7对应序列参数集SPS

其中0x68的二进制码为：

0110 1000

4-8为01000，转为十进制8，参考第二幅图：8对应图像参数集PPS

其中0x65的二进制码为：

0110 0101

4-8为00101，转为十进制5，参考第二幅图：5对应IDR图像中的片(I帧)

在IP网络中，当要传输的IP 报文大小超过最大传输单元MTU（Maximum Transmission Unit ）时就会产生IP分片情况。在以太网环境中可传输的最大 IP 报文（MTU）的大小为 1500 字节。如果发送的IP数据包大于MTU，数据包就会被拆开来传送，这样就会产生很多数据包碎片，增加丢包率，降低网络速度。对于视频传输而言，若RTP 包大于MTU 而由底层协议任意拆包，可能会导致接收端播放器的延时播放甚至无法正常播放。因此对于大于MTU 的NALU 单元，必须进行拆包处理。

RFC3984 给出了3 中不同的RTP 打包方案：

（1）Single NALU Packet:在一个RTP 包中只封装一个NALU，在本文中对于小于 1400字节的NALU 便采用这种打包方案。

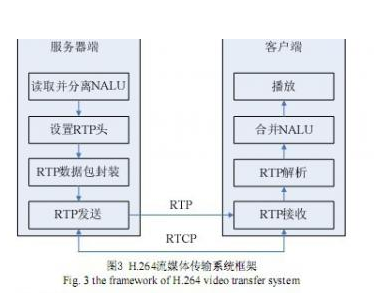
（2）Aggregation Packet:在一个RTP 包中封装多个NALU，对于较小的NALU 可以采用这种打包方案，从而提高传输效率。

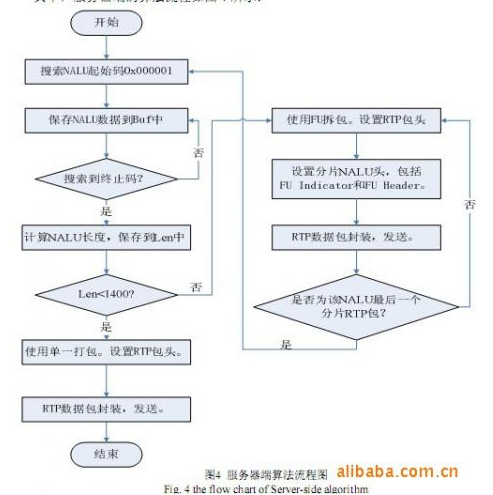
（3）Fragmentation Unit:一个NALU 封装在多个RTP包中，在本文中，对于大于1400字节的NALU 便采用这种方案进行拆包处理。

一 个完整的流媒体传输系统包含服务器端和客户端两个部分[5][6]。

对于服务器端，其主要任务是读取H.264 视频，从码流中分离出每个NALU 单元，分析NALU 的类型，设置相应的 RTP 包头，封装 RTP 数据包并发送。

而对于客户端来说，其主要任务则是接收 RTP数据包，从RTP 包中解析出NALU 单元，然后送至解码播放。该流媒体传输系统的框架如图





在spyDroid中 NALU + rtp header + 2 (NALU头 或者 分片NALU头 FU Indicator/Header )

参考C代码实现：

RTP协议的H.264视频传输 - Do one thing at a time, and do well. - 博客频道 - CSDN.NET.htm

linux rtsp服务器

linux webrtc服务器

VLC 的全名是 Video Lan Client,是一个开源的、跨平台的视频播放器

http://www.videolan.org/vlc/features.html

http://www.videolan.org/vlc/download-android.html

VLC做流媒体服务器的使用方法

<http://wenku.baidu.com/link?url=o_k4wWdOW6kToEk7CnL2z6_zf_W7giVvmwiUcxAYR7cwKZtkcFnrfK-GsRgBMoFr6cY5TznyiLoEDX3cG7iF6nv1xKQMC9YjXvPXM6WAhtG>

vlc的使用：作为服务器/客户端

1.使用 UDP 直接发送 H264+MP3(TS)/ udp://

2. VLC的RTP(基于UDP)发送 H264+MP3(TS)/ rtp://

3. RTSP方式是通过RTP进行流媒体数据的传输的，VLC的实现也是基于UDP的

<http://www.cnblogs.com/MikeZhang/archive/2012/09/09/vlcStreamingServer20120909.html>

[总结]FFMPEG视音频编解码零基础学习方法

http://blog.csdn.net/leixiaohua1020/article/details/15811977/

http://blog.csdn.net/leixiaohua1020/article/details/50535082

http://blog.csdn.net/leixiaohua1020/article/details/50535230

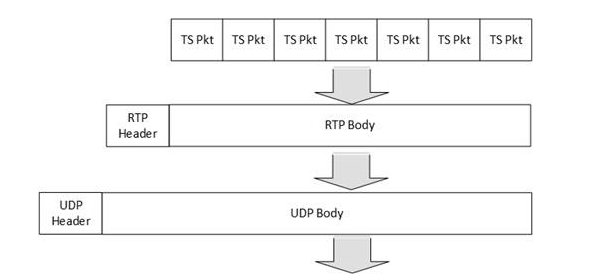
MPEG2-TS（Transport Stream“传输流”；又称TS、TP、MPEG-TS 或 M2T）实时传送的节目，比如实时广播的电视节目。

MPEG2-PS DVD节目中的MPEG2格式，确切地说是MPEG2-PS，全称是Program Stream（程序流）主要应用于存储的具有固定时长的节目，如DVD电影

video/mp2t

MP2T/RTP/TCP表示是TS流用于RTP打包,基于TCP

图中首先每7个MPEG-TS Packet打包为一个RTP，然后每个RTP再打包为一个UDP。其中打包RTP的方法就是在MPEG-TS数据前面加上RTP Header，而打包RTP的方法就是在RTP数据前面加上UDP Header。



spydroid-ipcamera

Android Open Source - Camera stream spydroid-ipcamera

[菜鸟怒读Spydroid源码（一）](http://blog.csdn.net/nmhszyn/article/details/49074027) http://blog.csdn.net/nmhszyn/article/details/49074027

[spydroid源码分析（一）：介绍spydroid每个包的大体功能](http://blog.csdn.net/xiaoliouc/article/details/8493161) http://blog.csdn.net/xiaoliouc/article/details/8493161

[spydroid源码分析（二）：spydroid运行流程](http://blog.csdn.net/xiaoliouc/article/details/8493207) http://blog.csdn.net/xiaoliouc/article/details/8493207

[spydroid源码分析（三）：rtp包头分析以及代码实现](http://blog.csdn.net/xiaoliouc/article/details/8493326) http://blog.csdn.net/xiaoliouc/article/details/8493326

spydroid

spydroid自身内置http服务器， 客户端可以通过在VLC等播放器中输入http://ip：8080/播放

spydroid自身内置rtsp服务器，客户端可以通过在VLC等播放器中输入rtsp://ip：8086/播放

SDP协议介绍

<http://www.cnblogs.com/qingquan/archive/2011/08/02/2125585.html>

SDP协议是也是基于文本的协议

SDP 完全是一种会话描述格式

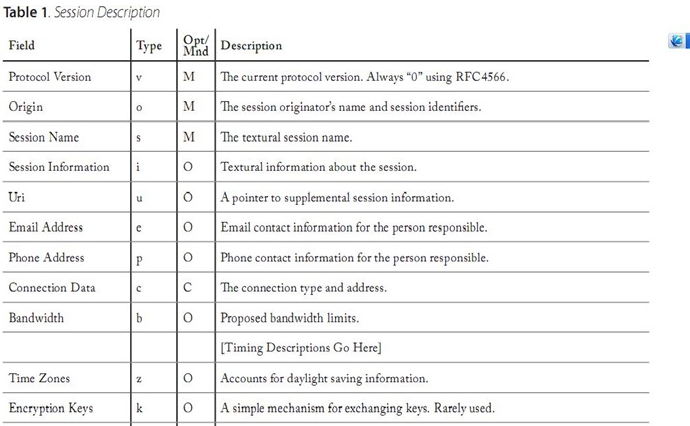
在流媒体中只用来描述媒体信息。媒体协商这一块要用[RTSP](http://www.cnblogs.com/qingquan/archive/2011/07/14/2106834.html)来实现．

SDP协议格式

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

＜type＞=<value>[CRLF]

常见type分类有 如下**1. 会话描述 2. 时序描述 3.媒体描述**



v=<version> 协议版本

o=<username> <session id> <version> <network type> <address type> <address> 所有者/创建者和会话标识符

s=<session name> 会话名称

i=<session description> 会话信息

u=<URI> URI 描述

e=<email address> Email 地址

p=<phone number> 电话号码

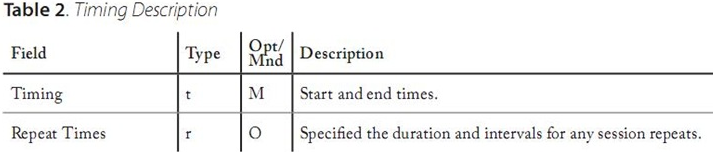
c=<network type> <address type> <connection address> 连接信息

b=<modifier>:<bandwidth-value> 带宽信息

z=<adjustment time> <offset> <adjustment time> <offset> .... 间区域调整

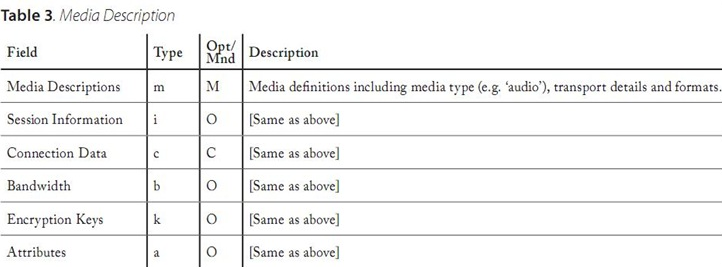
k=<method>

k=<method>:<encryption key> 加密密钥



t=<start time> <stop time> 会话活动时间

r=<repeat interval> <active duration> <list of offsets from start-time> 0或多次重复次数



m=<media> <port> <transport> <fmt list> 媒体名称和传输地址

i = \* （媒体标题）

c = \* （连接信息— 如果包含在会话层则该字段可选）

b = \* （带宽信息）

k=<method>

k=<method>:<encryption key> 加密密钥

a=<attribute>

a=<attribute>:<value> （0 个或多个媒体属性行）

SDP类似 HTML

HTML 是超文本标记语言 描述网页,格式是<html> <header></header><body><//body></html> 通过 http协议传输

浏览器通过请求http://<ip>/index.html apache/nginx服务器

SDP用来描述媒体信息 格式是 ＜type＞=<value>[CRLF] <类型>是一个字母，<值>是结构化的文本串 通过rtsp 协议传输

vlc通过请求 rtsp://darwin流媒体服务器ip:554/test.sdp 的方式实现实时播放。

helix 流媒体服务器的RTSP协议中的SDP协议：

（会话描述）

v=0 //SDP version

// o field定义的**源的**一些信息。其格式为：**o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>**

o=- 1271659412 1271659412 IN IP4 10.56.136.37

s=<No title> // 会话的名字

i=<No author> <No copyright> // 会话的信息

c=IN IP4 0.0.0.0 // 连接信息，分别描述了：网络协议，地址的类型，连接地址。

t=0 0 // 时间信息，分别表示开始的时间和结束的时间，一般在流媒体的**直播的时移**中见的比较多。**t=0 0意味着会话是永久的**

a=SdpplinVersion: 1610641560 // 描述性的信息 **a=属性: 属性类型;属性值**

a=StreamCount: integer;2 // 用来描述媒体流的信息，表示有**两个媒体流**。

a=control: \*

a=DefaultLicenseValue: integer;0 // License信息

a=FileType: string;"MPEG4" // 用来描述媒体流的信息 说明**当前协商的文件**是**mpeg4格式的文件**

a=LicenseKey: string;"license.Summary.Datatypes.RealMPEG4.Enabled"

a=range: npt=0-72.080000 // 用来表示**媒体流的长度**

（多媒体描述(audio)）

m=audio 0 RTP/AVP 96 // 做为媒体描述信息的重要组成部分

// 描述了**媒体信息的详细内容**：表示session的**audio是通过RTP来格式**传送的，其**payload值为96**传送的 端口还没有定

// payload将会出现在 RTP header 中 （spydroid用的都是 rtph[1] = 96'D 60'H 0110 0000 '）

b=as:24 // audio 的bitrate

b=RR:1800

b=RS:600

a=control:streamid=1 // 通过**媒体流1**来发送音频

a=**range**: npt=0-72.080000 // 说明**媒体流的长度**。

a=**length**: npt=72.080000

a=rtpmap: 96 MPEG4-GENERIC/32000/2

// rtpmap的信息，表示**音频为AAC**的 其**sample为32000**

a=fmtp:96 profile-level-id=15; mode=AAC-hbr; sizelength=13; indexlength=3; indexdeltalength=3; config=1210

// **config为AAC的详细格式信息**

a=mimetype: string; "**audio/MPEG4-GENERIC**"

a=Helix-Adaptation-Support: 1

a=**AvgBitRate**: integer; 48000

a=HasOutOfOrderTS: integer; 1

a=**MaxBitRate**: integer; 48000

a=Preroll: integer; 1000

a=OpaqueData: buffer; "A4CAgCIAAAAEgICAFEAVABgAAAC7gAAAu4AFgICAAhKIBoCAgAEC"

a=StreamName: string; "**Audio Track**"

（多媒体描述(video)）

下面是video的信息基本和audio的信息相对称，这里就不再说了。

m=**video** 0 [RTP](http://www.cnblogs.com/qingquan/archive/2011/07/28/2120440.html)/AVP **97** // 或者 m=video **5006** RTP/AVP 96    //video指明是视频信息， 5006是指客户端VLC接收视频信息的**udp端口号**

// 0 代表端口号没有确定 payload值是 97 (刚刚音频的是96)

b=as:150

b=RR:11250

b=RS:3750

a=control:**streamid=2** // 通过**媒体流2**来发送视频

a=range:npt=0-72.080000

a=length:npt=72.080000

a=rtpmap:**97** MP4V-ES/2500 // 或者 a=rtpmap:96 H264/90000          原例子是97

// 这里的96的信息很关键(1100000)

// 因为这个是**rtp的负载类型**,Payload type (**PT)**: 7 bits。

// H264编码，**90000是H264视频传输的默认视频采样频率**，必须是这个值

a=fmtp:97 profile-level-id=1;

// 或者 a=fmtp:96 packetization-mode=1; profile-level-id=42c016; sprop-parameter-sets= ZOLAFukBQHsg , aM4G4g== ;

// sprop-parameter-sets 用 逗号隔开的

// profile，sps，pps这3个数值是从mp4中提取出来的**base64编码**，在net.majorkernelpanic.mp4这个包

// 代码如下：

// "a=fmtp:96;packetization-mode=1;profile-level-id="+mConfig.getProfileLevel()+";**sprop-parameter-sets=**"+mConfig.getB64SPS()+","+mConfig.getB64PPS()+";\r\n";

// **packetization-mode**=1指定**rtp打包模式**

**//** 有3种模式，数值只能为0,1,2,。**0是单NAL单元模式 1是非交互模式 2是交互模式**

a=mimetype: string; "**video/MP4V-ES**"

a=Helix-Adaptation-Support: 1

a=**AvgBitRate**: integer; 300000

a=HasOutOfOrderTS:integer; 1

a=Height: integer; 240 //**影片的长度**

**a=Width: integer; 320 //影片的宽度**

a=**MaxBitRate**: integer; 300000

a=MaxPacketSize: integer; 1400

a=Preroll: integer; 1000

a=OpaqueData: buffer; "AzcAAB8ELyARAbd0AAST4AAEk+AFIAAAAbDzAAABtQ7gQMDPAAABAAAAASAAhED6KFAg8KIfBgEC"

a=StreamName: string; "**Video Track**"

RTSP 点播消息流程实例 （客户端：VLC， RTSP 服务器：LIVE555 Media Server)

**OPTIONS** **rtsp://192.168.1.109/1.mpg** RTSP/1.0

CSeq: 1 // 每个回应消息的cseq 数值和请求消息的cseq 相对应

user-Agent: VLC media player(LIVE555 Streaming Media v2007.02.20)

RTSP/1.0 200 OK

CSeq: 1

Date: wed, Feb 20 2008 07:13:24 GMT

**Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE**  // 服务器提供的可用的方法

**DESCRIBE** **rtsp://192.168.1.109/1.mpg**  RTSP/1.0 // 获得流媒体的信息 用SDP协议描述

CSeq: 2

Accept: **application/sdp**

**User-Agent:** VLC media player(LIVE555 Streaming Media v2007.02.20)

RTSP/1.0 200 OK

CSeq: 2

Date: wed, Feb 20 2008 07:13:25 GMT

Content-Base: **rtsp://192.168.1.109/1.mpg/** // 对应1.mpg流媒体的信息

Content-type: application/sdp

Content-length: 447 // SDP描述流媒体的信息长度

v=0

o =- 2284269756 1 IN IP4 192.168.1.109 // 用户名字 - 会话号 2284269756

s=MPEG-1 or 2 program Stream, streamed by the LIVE555 Media Server // 会话名字

i=1.mpg // 会话信息

t=0 0

a=tool:LIVE555 Streaming Media v2008.02.08 // 工具 Live555

a=type:broadcast // 广播

a=control:\*

a=range:npt=0-66.181 // 时间长度

a=x-qt-text-nam:MPEG-1 or Program Stream, streamed by the LIVE555 Media Server

a=x-qt-text-inf:1.mpg

m=video 0 RTP/AVP 32 // 媒体信息 video payload=32

c=IN IP4 0.0.0.0

a=control:track1 // trackID＝1 表示视频流用的是通道1 后面setup的时候需要指定track，来决定那个端口 传输 哪种stream

m=audio 0 RTP/AVP 14 // 媒体信息 audio payload=14

c=IN IP4 0.0.0.0

a=control:track2 // track1 代表音频 track2代表视频

SETUP  **rtsp://192.168.1.109/1.mpg/track1**  RTSP/1.0 **// URL多了 track1**

CSeq: 3

Transport: RTP/AVP; **unicast**;  **client\_port**=1112-1113 // 在SpyDroid中 DESCRIBE 阶段通过URL来确定是unicast还是mutlicast 以及对应组播或单播地址

User-Agent: VLC media player(LIVE555 Streaming Media v2007.02.20)

RTSP/1.0 200 OK

CSeq: 3

Date: wed, Feb 20 2008 07:13:25 GMT

Transport:

RTP/AVP;unicast;destination=192.168.1.222;source=192.168.1.109;client\_port=1112-1113;serve\_port=6970-6971

// 也可能是：

// Transport: RTP/AVP/TCP;unicast;interleaved=0-1 ？？？？ 音频和视频在同一个rtp socket上传输?? ccsc ???

// uri 中带有trackID＝0，表示对该通道进行设置。Transport 参数设置了传输模式，包的结构。

// 接下来的数据包头部第二个字节位置就是interleaved，它的值是每个通道都不同的，trackID＝0的interleaved 值有两个0或1，0表示rtp 包，1表示rtcp 包，接受端根据interleaved 的值来区别是哪种数据包。

Session: 3

SETUP **rtsp://192.168.1.109/1.mpg/track2** RTSP/1.0

CSeq: 4

Transport: RTP/AVP; unicast; client\_port=1114-1115 // **对于音频track1和视频track2 的rtp和rtcp 都使用不同的端口 (服务端和客户端都一样)**

Session: 3

User-Agent: VLC media player(LIVE555 Streaming Media v2007.02.20)

RTSP/1.0 200 OK

CSeq: 4

Date: wed, Feb 20 2008 07:13:25 GMT

Transport: RTP/AVP; unicast; destination=192.168.1.222; source=192.168.1.109; **client\_port**=1114-1115; **server\_port**=6972-6973

Session: 3

**PLAY**  **rtsp://192.168.1.109/1.mpg/**  RTSP/1.0 // 上面分别SETUP完两个track/stream后 这里开始播放

CSeq: 5

Session: 3

Range: npt=0.000-

User-Agent: VLC media player(LIVE555 Streaming Media v2007.02.20)

RTSP/1.0 200 OK

CSeq: 5

Range: npt=0.000-

Session: 3

**RTP-Info:** **url=**rtsp://192.168.1.109/1.mpg/track1; seq=9200; rtptime=214793785, **url=**rtsp://192.168.1.109/1.mpg/track2; seq=12770; rtptime=31721

(开始传输流媒体...)

[编解码学习笔记（五）：Mpeg系列——AAC音频](http://blog.csdn.net/flowingflying/article/details/5718594) http://blog.csdn.net/flowingflying/article/details/5718594

[H.264 sequence\_parameter\_sets成员值含义学习笔记](http://blog.csdn.net/leesphone/article/details/2159604) http://blog.csdn.net/leesphone/article/details/2159604

使用RTP传输H264的时候,需要用到sdp协议描述

其中有两项::Sequence Parameter Sets (SPS) **序列参数集** 和Picture Parameter Set (PPS) **图像参数集**

SDP中的H.264的**SPS和PPS串**，包含了初始化**H.264解码器**所需要的信息参数，包括编码所用的profile，level，图像的宽和高，deblock滤波器等

SPS: sequence parameter sets

01. **Profile\_idc** 它指的是**码流对应的profile.**

1.1 基线profile(Baseline profile)

遵循基线profile的码流应该遵循以下的约束:

a) 只有I和P切片存在

.....

c) 序列参数集(sps)的frame\_mbs\_only\_fag(之后介绍)应该等于1

d) 图象参数值(psp)的weighted\_pred\_flag和weighted\_bipred\_idc都应该为0

.....

**1**.2 主profile(Main profile)

主profile的码流应该遵循以下的约束:

a) 只有I,P,B切片存在.......

1.3 扩展profile(Extended profile)

.......

02. Constraint\_set0\_flag

当constrained\_set0\_flag值为1的时候，就说明码流**应该遵循基线profile(Baseline profile)的所有约束**.

当constrained\_set0\_flag值为0时，说明码流不一定要遵循基线profile的所有约束。

03. Constraint\_set1\_flag

04. Constraint\_set2\_flag

....

07. **Level\_id:** 12[Level 1.2] 它指的是码流对应的level级. 对于基线profile,主profile和扩展profile的相应的level级约束在H.264的文档中有表可查

....

MulticastSocket多播

单播unicast、广播broadcast和组播multicast

讲组播和普通UDP socket之间的区别在于**必须考虑TTL值**。

这时IP首部中取值 0- 255的一个字节。 它的含义为包**被丢弃前通过的路由数目**。

**每通过一个路由器，其TTL减少1**，有些路由器减少2或更多。当TTL值为0时包就被丢弃

创建：

**MulticastSocket**()throws SocketException:绑定匿名端口

MulticastSocket(int port)throws SocketException:绑定已知端口

MulticastSocket(SocketAddress address)throws IOException:

加入：

public void **joinGroup**(InetAddress address)throws IOException 加入**组播组**

退出：

**leaveGroup**(InetAddress address)

例子：

MulticastSocket ms =new **MulticastSocket**(4000); //新建组播socket

InetAddress ia = InetAddress.getByName("224.2.2.2");//组播地址

ms.**joinGroup**(ia); //加入主播组

byte[] buffer=new byte[512]; //发送数据包

byte[] buf = "Hello,This is a member of multicast!".getBytes();

DategramPacket packet = new **DatagramPacket**( buf, buf.length, ia, 4000);

ms.**send**(packet);//接收数据包

DatagramPacket dp =new **DatagramPacket**(buffer , buffer.length);

ms.**receive**(dp);

String s=new String(dp.getData()).trim(); // 该方法返回一个复制该字符串的开头和结尾的白色空格去掉

System.out.println(s);

ms.leaveGroup(address);

ms.close()

live555

live555 ubuntu下面验证服务端和客户端

http://blog.sina.com.cn/s/blog\_80be40500101evtd.html

[**RTSP监控摄像头web端监控技术**](http://www.cuplayer.com/rtsp/) **http://www.cuplayer.com/player/PlayerCode/live555/**

live555到android上，借助live555里的代码：从本地读取mpg文件推送到darwin，改成从实时流推送到darwin服务器，大致思路是派生一个h264videoSource，从android录制获取流，通过pipe管道发送流，把该流推送到darwin。

网页 android推送实时视频流到darwin流媒体服务器的思路

live555 移植 <http://blog.csdn.net/baby313/article/details/7289489>

liveMedia 项目(http://www.live555.com/)的源代码包括四个基本的库，各种测试代码以及Media Server

Media Server 是一个纯粹的RTSP 服务器。支持多种格式的媒体文件:

\* TS 流文件，扩展名ts。

\* PS 流文件，扩展名mpg。

\* MPEG-4视频基本流文件，扩展名m4e。

\* MP3文件，扩展名mp3。

\* WAV 文件(PCM)，扩展名wav。

\* AMR 音频文件，扩展名.amr。

\* AAC 文件，ADTS 格式，扩展名aac。

https://github.com/spex66/RTSP-Camera-for-Android/issues/1 ，里面有人成功过实现把h264流推送到darwin流媒体服务器

rtspcamera 开源项目

spydroid开源项目

流媒体服务器

Helix Darwin

AAC（Advanced Audio Coding），中文称为“高级音频编码”，出现于1997年， 基于 MPEG-2的音频编码技术。

MPEG-4标准 出现后，AAC 重新集成了其特性，加入了SBR技术和PS技术，为了区别于传统的 MPEG-2 AAC 又称为 MPEG-4 AAC

有损数据压缩

ISO/IEC 13818-7（MPEG-2第7部）, ISO/IEC 14496-3（MPEG-4第3部）

主要扩展名有三种：

AAC - 使用MPEG-2 Audio Transport Stream( ADTS，参见MPEG-2 )容器，区别于使用MPEG-4容器的MP4/M4A格式，属于传统的AAC编码（FAAC默认的封装，但FAAC亦可输出 MPEG-4 封装的AAC）

MP4 - 使用了MPEG-4 Part 14（第14部分）的简化版 即3GPP Media Release 6 Basic (3gp6，参见3GP ) 进行封装的AAC编码（Nero AAC 编码器仅能输出MPEG-4封装的AAC）；

M4A - 为了区别纯音频MP4文件和包含视频的MP4文件而由苹果(Apple)公司使用的扩展名，Apple iTunes 对纯音频MP4文件采用了".M4A"命名。M4A的本质和音频MP4相同，故音频MP4文件亦可直接更改扩展名为M4A。

9 种规格

MPEG-2 AAC LC 低复杂度规格（Low Complexity）

MPEG-2 AAC Main 主规格

MPEG-2 AAC SSR 可变采样率规格（Scaleable Sample Rate）

MPEG-4 AAC LC 低复杂度规格（Low Complexity），现在的手机比较常见的 MP4 文件中的音频部份就包括了该规格音频文件

MPEG-4 AAC Main 主规格

MPEG-4 AAC SSR 可变采样率规格（Scaleable Sample Rate）

MPEG-4 AAC LTP 长时期预测规格（Long Term Predicition）

MPEG-4 AAC LD 低延迟规格（Low Delay）

MPEG-4 AAC HE 高效率规格（High Efficiency）

上述的规格中，主规格（Main）包含了除增益控制之外的全部功能，其音质最好，而低复杂度规格（LC）则是比较简单，没有了增益控制，但提高了编码效率，至‘SSR’对‘LC’规格大体是相同，但是多了增益的控制功能，另外，MPEG-4 AAC/LTP/LD/HE，都是用在低比特率下编码，特别是‘HE’是有 Nero ACC 编码器支持，是近来常用的一种编码器，不过通常来说，Main 规格和 LC 规格的音质相差不大，因此目前使用最多的 AAC 规格多数是‘LC’规格，因为要考虑手机目前的存储器能力未达合理水准。

以上的这两个都是基于局域网的，那么外网怎么解决了？下面我提供几个我自己的灵活方案：

1、vpn，全部都在同一个vpn网络上，不就同一个局域网了吗。

2、花生壳域名映射，现在很多路由器都支持，如果不支持就搞台电脑吧。

3、路由IP映射。

1、rstp协议，网络上有个开源项目，基于Android，且这个项目也是一个服务端，里面也集成了http访问页面，可以通过http或者rstp直连手机：

<https://github.com/fyhertz/spydroid-ipcamera>

2、udp协议，网络上流传的基于udp进行局域网直播方案，采用的是广播：

<https://github.com/bytestar/android-h264-stream-demo>

[http://blog.csdn.net/mirkerson/article/details/38520175](http://images.tuxi.com.cn/blog.csdn.net/mirkerson/article/details/38520175)

udp协议，网络上流传的基于udp进行局域网直播方案，采用的是广播：

[VLC-Android编译与运行](http://blog.csdn.net/dtryl/article/details/50681612)

<git://git.videolan.org/vlc-ports/android.git>

android源码和apk下载

http://www.videolan.org/vlc/download-android.html 或者直接到http://get.videolan.org/vlc-android/2.0.6/ 有不同芯片apk和源码

<http://blog.csdn.net/dtryl/article/details/50681612> 编译

ubuntu 上安装vlc

vlc+ffmpeg+live555\_install

[http://www.doc88.com/p-673129826059.html](%20http:/www.doc88.com/p-673129826059.html)

VLC 播放器

1.live555的效率有些低，建议不使用...

2.编码后，需将码流放在一个缓冲区，然后rtsp再从缓冲区获取再打包传输

3.vlc设置里将延时改成小值

VLC缓存设置为300ms

M-JPEG的主要缺点是，压缩效率是低的，是基于M-JPEG压缩算法的每帧的图像的内容上，而不是基于相邻帧之间的差异，以被压缩，因此而引起的特定的冗余信息被重复存储

不用帧间压缩(mpeg h264等用)

Mjpeg是Motion JPEG 的简称。它的原理是把视频镜头拍成的视频分解成一张张分离的jpg数据发送到客户端。当客户端不断显示图片，即可形成相应的图像

大部分低端的网络摄像头都支持的MJPG[**协议**](http://www.edenw.com/tech/net/col/),较为高级的摄像头支持是H.264.

MJPG的优点是实现简单，客户端实现简单，不会出现马赛克的情况。缺点就是占带宽比较大，因为是一帧一帧按图像来传输的.

MJPG可以在多种传输[**协议**](http://www.edenw.com/tech/net/col/)上传输，比如TCP/UDP，最常见是在HTTP上采用传输。大部分的摄像头也是采用HTTP+MJPG的传输形式的。

mjpg的在http的mime type是"x-mixed-replace"

如果ipcam返回200，表示已经接收的请求，并在返回的头里指明边界字符串,这是在context type的boundary子属性来指明的。

这里的边界字符串是ThisRandomString.

　　GET /xxx.jpg HTTP/1.1

　　HTTP/1.0 200 OK Server: Digital Video Technology PUSH1 Expires: 0 Pragma: no-cache

　　Content-Type: multipart/x-mixed-replace;boundary=ThisRandomString

　　--ThisRandomString

　　Content-Type: image/jpeg

　　Content-Length: 6280

　　...(jpeg数据)

　　--ThisRandomString

[MJPEG协议入门介绍](http://blog.chinaunix.net/uid-22670933-id-1771591.html)

<http://blog.chinaunix.net/uid-22670933-id-1771591.html>

JPEG（英文全称：Joint　Photographic　Experts　Group）压缩技术可以说是所有图像压缩技术的基础。

它适合静态图像的压缩，直接处理整个画面，压缩倍数为20-80倍，分辨率没有选择的余地。所以要等到整个压缩档案传输完成才开始进行解压缩成影像画面，而这样的方式造成传输一个高解析画面时须耗时数十秒甚至数分钟。

高清网络[摄像机](http://dv.zol.com.cn/)视频编码标准主要有 MJPEG和H.264两种。

MJPEG(Motion JPEG)是在JPEG基础发展起来的动态图像压缩技术，它只单独的对某一帧进行压缩，而基本不考虑视频流中不同帧之间的变化 但它的缺陷也非常明显，其一：丢帧现象严重、实时性差，在保证每路都必须是高清晰的前提下，很难完成实时压缩。其二：压缩效率低，存储占用空间较大。由于它在运动性、带宽占用以及存储空间占用方面均有致命缺陷，高清网络摄像机依然以 H.264为主导。

后来又出现了多层式JPEG（ML-JPEG）压缩技术，它采取渐层式技术，先传输低解析的图档，然后再补送更细节的压缩资料，使画面品质改善。这种方式所需的时间虽然与原先的方式一样。但由于可以先看到画面，所以使用者会觉得这种方式较好。

OpenGL ES --> ANativeWindow http://cn.khronos.org/

:OpenMAX DL - 驱动层OpenMAX IL - 整合层OpenMAX AL

OpenMAX 是一套嵌入式的多媒体应用标准, 从驱动层到应用层都有详述的规定

http://blog.csdn.net/bonderwu/article/details/6149332 [OpenMax多媒体引擎](http://blog.csdn.net/bonderwu/article/details/6149332)

### [Android MediaCodec/NdkMediaCodec](http://www.baidu.com/link?url=n78e7DvyCJvpHzt6bsoGfbHu8WHJsFIbacwI6umTzHA7WIuXs20k7EcMCaZ9fzZDCrMqxMhSQDmpPFBdxEHY_a)

./android-21/arch-arm/usr/include/media/NdkMediaCodec.h

使用OpenGL ES 编解码

http://blog.csdn.net/augusdi/article/details/12574679