# **音视频直播——HTTP/RTSP/RTMP协议的区别**

## **共同点：**

1：RTSP RTMP HTTP都是在应用应用层。  
2： 理论上RTSP RTMP HTTP都可以做直播和点播，但一般做直播用RTSP RTMP，做点播用HTTP。做视频会议的时候原来用SIP协议，现在基本上被RTMP协议取代了。

## **区别：**

1. HTTP: 即超文本传送协议(ftp即文件传输协议)。  
   RTSP:（Real Time Streaming Protocol），实时流传输协议。  
   RTMP全称Routing Table Maintenance Protocol（路由选择表维护协议）。
2. HTTP将所有的数据作为文件做处理。http协议不是流媒体协议。  
   RTMP和RTSP协议是流媒体协议。
3. RTMP协议是Adobe的私有协议,未完全公开，RTSP协议和HTTP协议是共有协议，并有专门机构做维护。
4. RTMP协议一般传输的是flv，f4v格式流，RTSP协议一般传输的是ts,mp4格式的流。HTTP没有特定的流。
5. RTSP传输一般需要2-3个通道，命令和数据通道分离，HTTP和RTMP一般在TCP一个通道上传输命令和数据。

## **RTSP、RTCP、RTP区别**

### **1：RTSP实时流协议**

作为一个应用层协议，RTSP提供了一个可供扩展的框架，它的意义在于使得实时流媒体数据的受控和点播变得可能。总的说来，RTSP是一个流媒体表示 协议，主要用来控制具有实时特性的数据发送，但它本身并不传输数据，而是必须依赖于下层传输协议所提供的某些服务。RTSP可以对流媒体提供诸如播放、暂 停、快进等操作，它负责定义具体的控制消息、操作方法、状态码等，此外还描述了与RTP间的交互操作（RFC2326）。

### **2：RTCP控制协议**

RTCP控制协议需要与RTP数据协议一起配合使用，当应用程序启动一个RTP会话时将同时占用两个端口，分别供RTP和RTCP使用。RTP本身并 不能为按序传输数据包提供可靠的保证，也不提供流量控制和拥塞控制，这些都由RTCP来负责完成。通常RTCP会采用与RTP相同的分发机制，向会话中的 所有成员周期性地发送控制信息，应用程序通过接收这些数据，从中获取会话参与者的相关资料，以及网络状况、分组丢失概率等反馈信息，从而能够对服务质量进 行控制或者对网络状况进行诊断。

RTCP协议的功能是通过不同的RTCP数据报来实现的，主要有如下几种类型：

SR：发送端报告，所谓发送端是指发出RTP数据报的应用程序或者终端，发送端同时也可以是接收端。(SERVER定时间发送给CLIENT)。

RR：接收端报告，所谓接收端是指仅接收但不发送RTP数据报的应用程序或者终端。(SERVER接收CLIENT端发送过来的响应)。

SDES：源描述，主要功能是作为会话成员有关标识信息的载体，如用户名、邮件地址、电话号码等，此外还具有向会话成员传达会话控制信息的功能。

BYE：通知离开，主要功能是指示某一个或者几个源不再有效，即通知会话中的其他成员自己将退出会话。

APP：由应用程序自己定义，解决了RTCP的扩展性问题，并且为协议的实现者提供了很大的灵活性。

## **3：RTP数据协议**

RTP数据协议负责对流媒体数据进行封包并实现媒体流的实时传输，每一个RTP数据报都由头部（Header）和负载（Payload）两个部分组成，其中头部前12个字节的含义是固定的，而负载则可以是音频或者视频数据。

RTP用到的地方就是 PLAY ，服务器往客户端传输数据用UDP协议，RTP是在传输数据的前面加了个12字节的头(描述信息)。

RTP载荷封装设计本文的网络传输是基于IP协议，所以最大传输单元(MTU)最大为1500字节，在使用IP／UDP／RTP的协议层次结构的时候，这 其中包括至少20字节的IP头，8字节的UDP头，以及12字节的RTP头。这样，头信息至少要占用40个字节，那么RTP载荷的最大尺寸为1460字 节。以H264 为例，如果一帧数据大于1460，则需要分片打包，然后到接收端再拆包，组合成一帧数据，进行解码播放。

**直播应用中，RTMP和HLS基本上可以覆盖所有客户端观看，**

**HLS主要是延时比较大，RTMP主要优势在于延时低。**

## **一、应用场景**

低延时应用场景包括：

* **互动式直播**：譬如大行其道的美女主播，游戏直播等等各种主播，流媒体分发给用户观看。用户可以文字聊天和主播互动。
* **视频会议**：我们要是有同事出差在外地，就用视频会议开内部会议。其实会议1秒延时无所谓，因为人家讲完话后，其他人需要思考，思考的延时也会在1秒左右。当然如果用视频会议吵架就不行。
* **其他：**监控，直播也有些地方需要对延迟有要求，互联网上RTMP协议的延迟基本上能够满足要求。

## **二、RTMP和延时**

## **1. RTMP的特点如下：**

**1) Adobe支持得很好：**

RTMP实际上是现在编码器输出的工业标准协议，基本上所有的编码器（摄像头之类）都支持RTMP输出。

原因在于PC市场巨大，PC主要是Windows，Windows的浏览器基本上都支持flash，

Flash又支持RTMP支持得非常好。

**2) 适合长时间播放：**

因为RTMP支持的很完善，所以能做到flash播放RTMP流长时间不断流，

当时测试是100万秒，即10天多可以连续播放。

对于商用流媒体应用，客户端的稳定性当然也是必须的，否则最终用户看不了还怎么玩？

我就知道有个教育客户，最初使用播放器播放http流，需要播放不同的文件，结果就总出问题，如果换成服务器端将不同的文件转换成RTMP流，客户端就可以一直播放；

该客户走RTMP方案后，经过CDN分发，没听说客户端出问题了。

**3）延迟较低：**

比起YY的那种UDP私有协议，RTMP算延迟大的（延迟在1-3秒），

比起HTTP流的延时（一般在10秒以上）RTMP算低延时。

一般的直播应用，只要不是电话类对话的那种要求，RTMP延迟是可以接受的。

在一般的视频会议应用中，RTMP延时也能接受，原因是别人在说话的时候我们一般在听，

实际上1秒延时没有关系，我们也要思考（话说有些人的CPU处理速度还没有这么快）。

**4) 有累积延迟：**

技术一定要知道弱点，RTMP有个弱点就是累积误差，原因是RTMP基于TCP不会丢包。

所以当网络状态差时，服务器会将包缓存起来，导致累积的延迟；

待网络状况好了，就一起发给客户端。

这个的对策就是，当客户端的缓冲区很大，就断开重连。

**FFmpeg/WebRTC/RTMP音视频流媒体高级开发学习资料、学习视频和学习路线图如有需要添加学习群获取 （**[学习群资料获取](https://link.zhihu.com/?target=https://jq.qq.com/" \t "https://zhuanlan.zhihu.com/p/_blank)**）**

**学习地址：[https://ke.qq.com/course/3202131](https://link.zhihu.com/?target=https://ke.qq.com/course/3202131" \t "https://zhuanlan.zhihu.com/p/_blank)?flowToken=1031864**



## **2. HLS低延时**

主要有人老是问这个问题，如何降低HLS延迟。  
HLS解决延时，就像是爬到枫树上去捉鱼，奇怪的是还有人喊，看那，有鱼。  
你说是怎么回事?  
  
我只能说你在参与谦哥的魔术表演，错觉罢了。  
如果你真的确信有，请用实际测量的图片来展示出来，参考下面延迟的测量。

## **3. RTMP延迟的测量**

如何测量延时，是个很难的问题，  
不过有个行之有效的方法，就是用手机的秒表，可以比较精确的对比延时。  
  
经过测量发现，在网络状况良好时：

* RTMP延时可以做到0.8秒左右。
* 多级边缘节点不会影响延迟（和SRS同源的某CDN的边缘服务器可以做到）
* Nginx-Rtmp延迟有点大，估计是缓存的处理，多进程通信导致？
* GOP是个硬指标，不过SRS可以关闭GOP的cache来避免这个影响.
* 服务器性能太低，也会导致延迟变大，服务器来不及发送数据。
* 客户端的缓冲区长度也影响延迟。  
  譬如flash客户端的NetStream.bufferTime设置为10秒，那么延迟至少10秒以上。

## **4. GOP-Cache**

什么是GOP？就是视频流中两个I帧的时间距离。  
GOP有什么影响？  
Flash（解码器）只有拿到GOP才能开始解码播放。  
也就是说，服务器一般先给一个I帧给Flash。  
可惜问题来了，假设GOP是10秒，也就是每隔10秒才有关键帧，  
如果用户在第5秒时开始播放，会怎么样？  
第一种方案：等待下一个I帧，  
也就是说，再等5秒才开始给客户端数据。  
这样延迟就很低了，总是实时的流。  
问题是：等待的这5秒，会黑屏，现象就是播放器卡在那里，什么也没有，  
有些用户可能以为死掉了，就会刷新页面。  
总之，某些客户会认为等待关键帧是个不可饶恕的错误，延时有什么关系？  
我就希望能快速启动和播放视频，最好打开就能放！

第二种方案：马上开始放，  
放什么呢？  
你肯定知道了，放前一个I帧。  
也就是说，服务器需要总是cache一个gop，  
这样客户端上来就从前一个I帧开始播放，就可以快速启动了。  
问题是：延迟自然就大了。

有没有好的方案？  
有！至少有两种：  
编码器调低GOP，譬如0.5秒一个GOP，这样延迟也很低，也不用等待。  
坏处是编码器压缩率会降低，图像质量没有那么好。

## **5. 累积延迟**

除了GOP-Cache，还有一个有关系，就是累积延迟。  
服务器可以配置直播队列的长度，服务器会将数据放在直播队列中，  
如果超过这个长度就清空到最后一个I帧：

当然这个不能配置太小，  
譬如GOP是1秒，queue\_length是1秒，这样会导致有1秒数据就清空，会导致跳跃。

有更好的方法？有的。  
延迟基本上就等于客户端的缓冲区长度，因为延迟大多由于网络带宽低，  
服务器缓存后一起发给客户端，现象就是客户端的缓冲区变大了，  
譬如NetStream.BufferLength=5秒，那么说明缓冲区中至少有5秒数据。

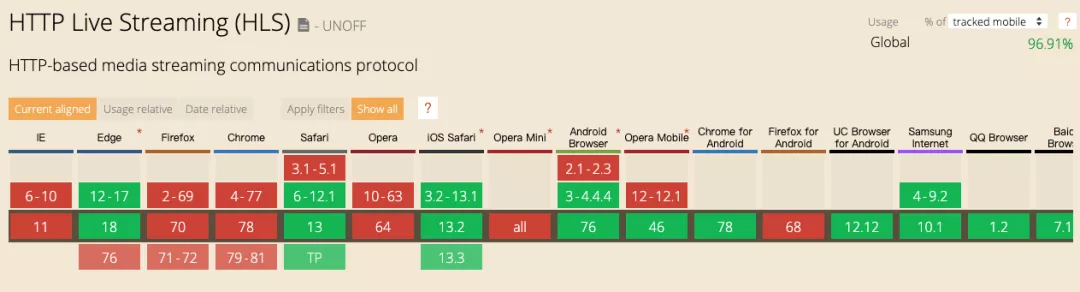
处理累积延迟的最好方法，是客户端检测到缓冲区有很多数据了，如果可以的话，就重连服务器。  
当然如果网络一直不好，那就没有办法了。

# 视频秒开的秘密の为什么HLS能吊打MP4

<https://mp.weixin.qq.com/s?__biz=MzU0OTExNzYwNg==&mid=2247484968&idx=1&sn=0405f45e2ea0fcf70b5ff110c3aa616a>

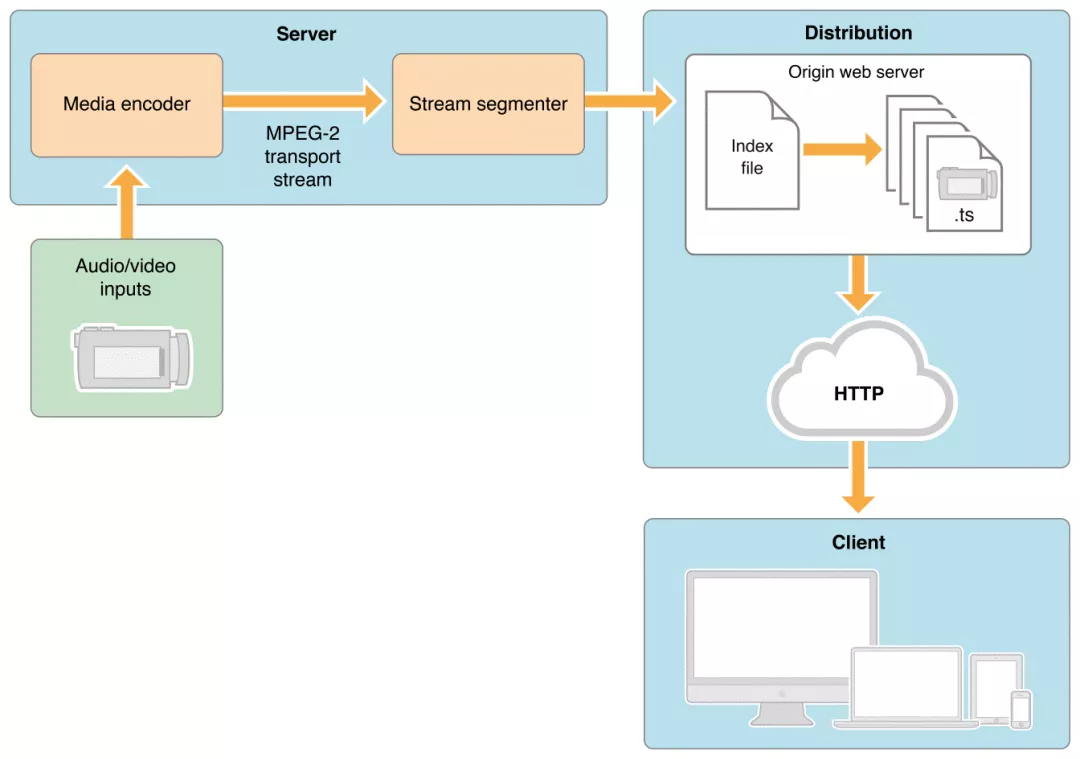
有没有其他针对直播/点播的流协议呢？

答案是肯定（这里先不讨论FLV了），移动端苹果系统（iOS 3.0及更高版本的设备均支持）也给出了HTTP实时流的解决方案-HLS，安卓迫于iOS的淫威也开始系统层兼容HLS了，Can I use 上来看移动端兼容性相当不错。



关于HLS的介绍可以查看官方文档 developer.apple.com 。

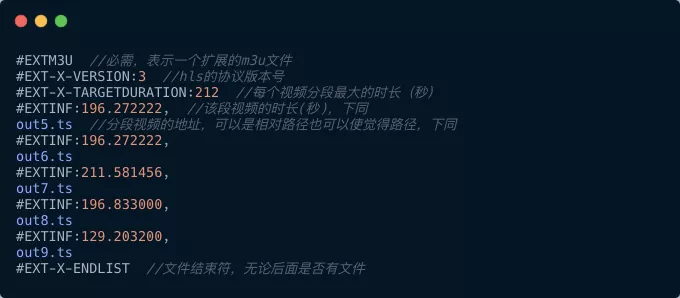
首先服务端要根据源视频文件转换成HLS格式（m3u8和ts两种后缀）的文件，这个转换需要先把码源视频转码到目标编码格式（eg.H264），在进行 Stream segmenter（流分割）对视频切片后生成一个文本类型的xxx.m3u8索引文件和一组 xxxx.ts的分片文件，剩下的客户端获取.m3u8和.ts文件就交给HTTP服务器了（eg.nginx 、 apache）。整个从生成流到拉流的过程如下图：



客户端（H5端）首先会拉取到索引文件xxx.m3u8，解析后根据索引的中播放列表的顺序依序拉取ts文件，如果当前客户端不支持HLS，就需要通过js去手动下载解析这些ts文件，然后解析重组成视频流保存到SourceBuffer 中，在媒体源扩展 API（MSE） 进行播放（现有方案hls.js）。这里的注意的H.264视频解码能力是系统本身提供的，如果系统本身不支持那就gg了。

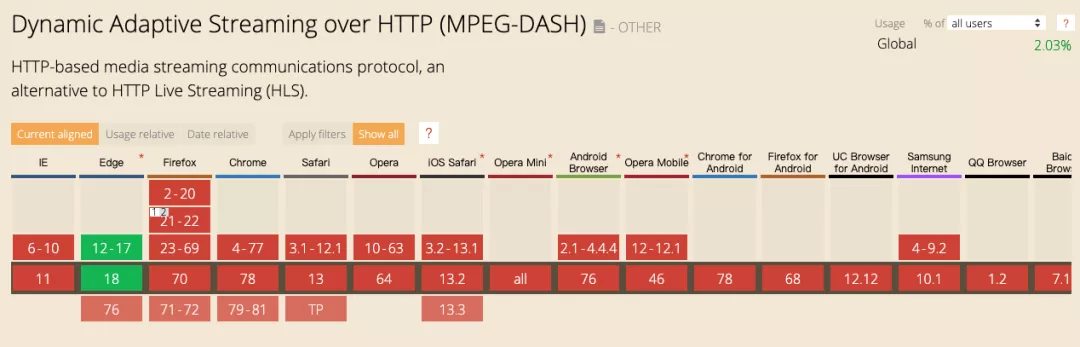
m3u8 文件实质是一个播放列表（playlist），其可能是一个媒体播放列表（Media Playlist），或者是一个主列表（Master Playlist）。但无论是哪种播放列表，其内部文字使用的都是 utf-8 编码。格式示例：

**具体的的某个码流(eg.voide.m3u8)**



更多m3u8的格式信息可以点 ietf.org 查看，目前HLS还是草案，并不是国际标准。

其实关于自适性流技术的有个国际标准，是MPEG-DASH，不过这货在2011年11月才成为国际标准，相对较晚，也不好推，各浏览器基本上都未在系统层支持，总的来说位置比较尴尬。



整体上来对比下这三种流格式的优缺点：

|  |  |  |
| --- | --- | --- |
| **协议格式** | **播放体验** | **流量占用情况** |
| DASH | 对视频进行切片，按切片播放，缓存小起播快；拖动时间轴到任意时间播放时，可以快速定位到对应的切片进行播放，响应快。 | 小 |
| HLS | 对视频进行切片，按切片播放，缓存小起播快；拖动时间轴到任意时间播放时，可以快速定位到对应的切片进行播放，响应快。 | 整体占用小，播放一个切片只下载一个切片内容；对于低码率的视频场景，因封装代价高导致流量占用相对较高 |
| MP4 | 头文件较大，边下边缓存，起播相对HLS和DASH慢一些；拖动时间轴播放时，需要一定的时间缓存；市场上大多数的浏览器客户端均能够播放，播放成功率高。 | 拖动时间轴播放时，仍然需要下载整个头文件，耗费流量大；因流量占用较大，建议用在短视频处理的场景。 |
| HTTP-FLV | 缓存小起播快 | 小 |

从兼容性、性能、体验上来看，HLS是不错的选择（ps.最主要的是服务端转码不支持MPEG-DASH）。