直播点播流媒体技术方案书

如何快速构建低延时，高可用，大规模，跨终端、
跨地域加速的流媒体直播服务

xuli@qiniu.com

2014年11月

引言

流媒体技术介绍

流媒体技术的主要特点是以 “流(Streaming)” 的形式在基于 IP 协议的互联网中进行多媒体数据的实时、 连续传播，客户端在播放前并不需要下载整个媒体文件，而是在将缓存区中已经收到的媒体数据进行播放。同时，媒体流的剩余部分仍持续不断地从服务器递送到客户端，即所谓的“边下载，边播放”。

移动互联网是传统桌面互联网向移动通信网络的延伸。作为移动互联网中具有代表性的典型应用，移动流媒体业务主要是利用互联网和3G、4G通信网络平台，为以手机为主的嵌入式终端设备提供基于音视频的流式多媒体服务。

流媒体协议是支撑流媒体业务运行的关键核心技术之一。在传统桌面互联网时代，常用的流媒体协议主要有 HTTP 渐进式下载和基于 RTSP/RTP 的实时流媒体协议栈等等，这些流媒体协议大多数可以平移到移动流媒体中继续应用。然而由于移动互联网及其终端设备的一些独有特性，传统流媒体协议在移动互联网中的应用在功能、性能的提供和用户体验等方面都会受到不同程度的约束和限制，于是一些新的流媒体协议应运而生。例如，苹果公司的 HTTP Live Streaming 就是其中具有代表性且得到较为广泛应用的一个。

本篇内容主要针对上述几种流媒体协议进行综述，并对这几种协议的优缺点以及适用范围进行较为深入的分析和比较，以辅助工程人员对流媒体实现的技术选型有较为全面的认知。

1. HTTP 渐进式下载流媒体播放

基于 HTTP 的渐进式下载 (Progressive Download) 流媒体播放仅是在完全下载后再播放模式基础上做了一些小的改进。与下载播放模式中必须等待整个文件下载完毕后才能开始播放不同，渐进下载客户端在开始播放之前仅需等待一段较短的时间用于下载和缓冲该媒体文件最前面的一部分数据，之后便可以一边下载一边播放。 在正式开始播放之前的这一小段缓冲应使得后续即使在网络较为拥塞的情况下媒体数据也能够得以不间断地连续播放，通常需要几十秒甚至上百秒的时间。在这种模式下，客户端以自己以及 Web 服务器和网络所能允许的最大速度尽可能快地从服务器索取数据，而不考虑当前所播放压缩码流的实际码率参数。只有满足特定封装条件的媒体文件格式才支持这种类型的渐进下载播放，例如用于初始化解码器的编码参数必须放置在媒体文件的起始部位，音视频数据完全按照时间顺序进行交织等。

渐进下载流媒体播放采用标准 HTTP 协议来在 Web 服务器和客户端之间递送媒体数据，而HTTP 又承载于 TCP 之上。TCP 最初是为非实时性数据传输而设计的，其优化目标在于在保证整个网络总的稳定性和高吞吐量的前提下，最大化数据传输速率。为达到这个目的，TCP 采用了一种称之为慢启动的算法，它首先以一个较低的速率来发送数据，然后再逐渐提高这个速率，直到接收到来自目的方的分组丢失反馈报告。此时 TCP 认为它已达到最高带宽限制或者网络中出现了拥塞，于是重新开始以一个较低速率来发送数据，然后逐渐提高，这个过程不断地重复下去。TCP 通过重传丢失的分组来达到可靠传输的目的。然而，对于流媒体数据来说，TCP 无法保证所有重传的数据能在它们预定的播放时刻之前按时到达客户端。当这种情况出现时，客户端不能跳过这些丢失或迟到的数据直接播放时间上靠后的媒体数据，而必须停下来等待，从而导致播放器画面停顿和断断续续的现象发生。

在 HTTP 渐进下载播放模式中，客户端需要在硬盘上缓存所有前面已经下载的媒体数据，对本地存储空间的需求较大。播放过程中用户只能在前面已经下载媒体数据的时间范围内进行进度条搜索和快进、快退等操作，而无法在整个媒体文件时间范围内执行这些操作。

2. RTSP/RTP/RTMP 流媒体协议

上述基于 HTTP 渐进式下载的流媒体播放仅能支持点播而不能支持直播，媒体流数据到达客户端的速率无法精确控制，客户端仍需维持一个与服务器上媒体文件同样大小的缓冲存储空间，在开始播放之前需要等待一段较长的缓冲时间从而导致实时性较差，播放过程中由于网络带宽的波动或分组丢失可能会导致画面停顿或断续等待，不支持全时间范围的搜索、快进、快退等 VCR 操作。 为克服这些问题，需要引入专门的流媒体服务器以及相应的实时流媒体传输和控制协议来进行支持。

RTSP/RTP 是目前业界最为流行和广为采用的实时流媒体协议。它实际上由一组在 IETF 中标准化的协议所组成，包括RTSP(实时流媒体会话协议)，SDP(会话描述协议)，RTP(实时传输协议)，以及针对不同编解码标准的 RTP 净载格式等，共同协作来构成一个流媒体协议栈。基于该协议栈的扩展已被 ISMA (互联网流媒体联盟) 和 3GPP (第三代合作伙伴计划) 等组织采纳成为互联网和 3G 移动互联网的流媒体标准。

RTSP（Real Time Streaming Protocol）是用来建立和控制一个或多个时间同步的连续音视频媒体流的会话协议。通过在客户机和服务器之间传递 RTSP 会话命令，可以完成诸如请求播放、开始、暂停、查找、快进和快退等VCR控制操作。虽 然 RTSP 会话通常承载于可靠的 TCP 连接之上，但也可以使用 UDP 等无连接协议来传送 RTSP 会话命令。

SDP 协议用来描述多媒体会话。SDP 协议的主要作用在于公告一个多媒体会话中所有媒体流的相关描述信息，以使得接收者能够感知这些描述信息并根据这些描述参与到这个会话中来。SDP 会话描述信息通常是通过 RTSP 命令交互来进行传递的，其中携带的媒体类信息主要包括:

- 媒体的类型(视频，音频等)
- 传输协议(RTP/UDP/IP，RTP/TCP/IP 等)
- 媒体编码格式(H.264 视频，AVS 视频等)
- 流媒体服务器接收媒体流的 IP 地址和端口号

RTP（Real-time Transport Protocol）称为实时传输协议，用于实际承载媒体数据并为具有实时特性的媒体数据交互提供端到端的传输服务，例如净载类型识别、序列号、时间戳和传输监控等。应用程序通常选择在 UDP 之上来运行 RTP 协议，以便利用 UDP 的复用和校验和等功能，并提高网络传输的有效吞吐量。然而 RTP 仍可选择与其它网络传输协议（例如 TCP）一起使用。

RTSP/RTP 流媒体协议栈的使用需要专门的流媒体服务器进行参与。与渐进下载中媒体数据的被动突发递送不同，在有流媒体服务器参与的媒体分发过程中，媒体数据是以与压缩的音视频媒体码率相匹配的速率主动和智能地发送的。在整个媒体递送过程中，服务器与客户端紧密联系，并能够对来自客户端的反馈信息做出响应。

RTP 是真正的实时传输协议，客户端仅需要维持一个很小的解码缓冲区用于缓存视频解码所需的少数参考帧数据，从而大大缩短了起始播放时延，通常可控制在1秒之内。使用 UDP 来承载 RTP 数据包可提高媒体数据传输的实时性和吞吐量。当因为网络拥塞而发生 RTP 丢包时，服务器可以根据媒体编码特性智能地进行选择性重传，故意丢弃一些不重要的数据包；客户端也可以不必等待未按时到达的数据而继续向前播放，从而保证媒体播放的流畅性。

RTMP（Real Time Messaging Protocol）实时消息传送协议是 Adobe Systems 公司为 Flash 播放器和服务器之间音频、视频和数据传输 开发的开放协议。

它有三种变种：

- 工作在 TCP 之上的明文协议，使用端口1935；
- RTMPT 封装在 HTTP 请求之中，可穿越防火墙；
- RTMPS 类似 RTMPT，但使用的是 HTTPS 连接；

RTMP 协议是被 Flash 用于对象、视频和音频的传输。这个协议建立在 TCP 协议或者轮询 HTTP 协议之上。

3. HTTP Live Streaming 协议

HTTP Live Streaming（HLS）最初是苹果公司针对其 iPhone、iPod、iTouch 和 iPad 等移动设备而开发的流媒体协议，后来在支持 HTML5 的浏览器（例如 Safari）以及 Android 包括 Adobe Flash Player 中也得到了支持。 HTTP Live Streaming 允许内容提供者通过普通 Web 服务器向上述客户端提供接近实时的音视频流媒体服务，包括直播和点播。

相对于常见的流媒体直播协议，HLS 直播最大的不同在于，直播客户端获取到的，并不是一个完整的数据流。HLS 协议在服务器端将直播数据流存储为连续的、很短时长的媒体文件（MPEG-TS格式），而客户端则不断的下载并播放这些小文件，因为服务器端总是会将最新的直播数据生成新的小文件，这样客户端只要不停的按顺序播放从服务器获取到的文件，就实现了直播。由此可见，基本上可以认为，HLS 是以点播的技术方式来实现直播。由于数据通过 HTTP 协议传输，所以完全不用考虑防火墙或者代理的问题，而且分段文件的时长很短，客户端可以根据网络带宽的变化在这些不同码率的替换流之间进行智能切换。此外，HTTP Live Streaming 还支持通过媒体加密和用户认证等方式来达到媒体版权保护。

对于 HLS 视频直播，编码器首先将设备端实时采集的音视频数据压缩编码为符合特定标准的音视频基本流（目前支持 H.264 视频和 AAC 音频），然后再复用和封装成为符合 MPEG-2 系统层标准的传输流 （TS）格式进行输出。

流分割器（Stream Segmenter）负责将编码器输出的 MPEG-2 TS 流分割为一系列连续的、长度均等的小TS 文件（后缀名为.ts），并依次发送至内容分发组件中的 Web 服务器进行存储。与此同时，为了跟踪播放过程中媒体文件的可用性和当前位置，流分割器还需创建一个含有指向这些小 TS 文件指针的索引文件，同样放置于 Web 服务器之中。这个索引文件可以看作是一个连续媒体流中的播放列表滑动窗口，每当流分割器生成一个新的 TS 文件时，这个索引文件的内容也被更新，新的文件 URI（统一资源定位符）加入到滑动窗口的末尾，老的文件 URI 则被移去，这样索引文件中将始终包含最新的固定数量的 x 个分段。流分割器还可以对其生成的每个小 TS 文件进行加密，并生成相应的密钥文件。

对于 HLS 视频点播（VOD），文件分割器（File Segmenter）首先将预编码存储的媒体文件转码为 MPEG-2 TS 格式文件（已经封装为 TS 格式的文件则忽略这一步），然后再将该 TS 文件分割成一系列长度均等的小 TS 文件。文件分割器同样也生成一个含有指向这些小文件指针的索引文件。与直播型会话不同的是，这里的索引文件是一个不随时间而更新的静态文件，其中包含一个节目从头到尾所有分段文件的 URI 列表，并以 #EXT-X-ENDLIST 标签结尾。

可以通过下述方法将一个直播事件转换成 VOD 节目源供以后点播：在服务器上不删除已经过期的分段媒体文件，同时在索引文件中也不删除这些文件所对应的 URI 索引项，在直播结束的时候将 #EXT-X-ENDLIST 标签添加至索引文件的末尾。

在基于 HTTP Live Streaming 的流媒体系统中，服务器可以为同一节目源准备多份以不同码率和质量编码的替换流，并为每个替换流都生成一个衍生的索引文件。在主索引文件中通过包含一系列指向其他衍生索引在移动互联网环境下，由于网络覆盖面的不同和信号强弱的变化，移动终端可能随时在不同的无线接入网络（例如 3G, EDGE, GPRS 和 WiFi 等）之间进行切换。 时客户端软件可根据网络和带宽的变化情况随时切换到不同衍生索引文件所指向的替换流进行下载，从而自适应地为用户提供相应网络条件下接近最优的音视频 QoS体验。

4. 分析与比较

作为最简单和原始的流媒体解决方案，HTTP 渐进式下载唯一显著的优点在于它仅需要维护一个标准的 Web 服务器，而这样的服务器基础设施在互联网中已经普遍存在，其安装和维护的工作量和复杂性比起专门的流媒体服务器来说要简单和容易得多。然而其缺点和不足却也很多，首先是仅适用于点播而不支持直播，其次是缺乏灵活的会话控制功能和智能的流量调节机制，再次是客户端需要硬盘空间以缓存整个文件而不适合于移动设备等。

基于 RTSP/RTP 的流媒体系统专门针对大规模流媒体直播和点播等应用而设计，需要专门的流媒体服务器支持，与 HTTP 渐进下载相比主要具有如下优势：

- 流媒体播放的实时性。与 HTTP 渐进下载客户端需要先缓冲一定数量媒体数据才能开始播放不同，基于 RTSP/RTP 的流媒体客户端几乎在接收到第一帧媒体数据的同时就可以启动播放。
- 支持进度条搜索、快进、快退等高级 VCR 控制功能。
- 平滑、流畅的音视频播放体验。

在基于 RTSP 的流媒体会话期间，客户端与服务器之间始终保持会话联系，服务器能够对来自客户端的反馈信息动态做出响应。当因网络拥塞等原因导致可用带宽不足时，服务器可通过适当降低帧率等方式来智能调整发送速率。此 外，UDP 传输协议的使用使得客户端在检测到有丢包发生时，可选择让服务器仅选择性地重传部分重要的数据（如关键帧），而忽略其他优先级较低的数据，从而保证在网络不好的情况下客户端也仍能连续、流畅地进行播放。

尽管如此，基于 RTSP/RTP 的流媒体系统在实际的应用部署特别是移动互联网应用中仍然遇到了不少问题，主要体现在：

- 与 Web 服务器相比，流媒体服务器的安装、配置和维护都较为复杂，特别是对于已经建有 CDN（内容分发网络）等基础设施的运营商来说，重新安装配置支持 RTSP/RTP 的流媒体服务器工作量很大。

- RTSP/RTP 协议栈的逻辑实现较为复杂，与 HTTP 相比支持 RTSP/RTP 的客户端软硬件实现难度较大，特别是对于嵌入式移动设备终端来说。
- RTSP 协议使用的网络端口号(554)可能被部分用户网络中的防火墙和NAT等封堵，导致无法使用。 虽然有些流媒体服务器可通过隧道方式将 RTSP 配置在 HTTP 的 80 端口上承载，但实际部署起来并不是特别方便。

HTTP Live Streaming 正是为了解决这些问题应运而生的，其主要特点是：放弃专门的流媒体服务器，而返回到使用标准的 Web 服务器来递送媒体数据；将容量巨大的连续媒体数据进行分段，分割为数量众多的小文件进行传递，迎合了 Web 服务器的文件传输特性；采用了一个不断更新的轻量级索引文件来控制分割后小媒体文件的下载和播放，可同时支持直播和点播，以及 VCR 类会话控制操作。HTTP 协议的使用降低了 HTTP Live Streaming 系统的部署难度，同时也简化了客户端（特别是嵌入式移动终端）软件的开发复杂度。此外，文件分割和索引文件的引入也使得带宽自适应的流间切换、服务器故障保护和媒体加密等变得更加方便。与 RTSP/RTP 相比，HTTP Live Streaming 的最大缺点在于它并非一个真正的实时流媒体系统，在服务器和客户端都存在一定的起始延迟，对于毫秒级实时的支持程度尚需进一步的探究和验证。

归纳起来，上述几种流媒体协议的综合比较如下所示：

	HTTP 渐进式下载	**RTSP/RTP/RTMP**	**HTTP Live Streaming**
服务器端实现	普通 Web 服务器	流媒体服务器	普通 Web 服务器
客户端实现	容易	较难	容易
支持业务类型	点播	直播、点播	直播、点播
实时性(起始时延)	> 30s	< 2s	< 30s
客户端缓冲区	硬盘，文件大	内存，小	内存，小
VCR 支持(拖动时移)	部分支持	支持	支持
网络带宽适应	不支持	不支持	支持
服务器故障保护	不支持	不支持	支持
DRM 支持(内容加密)	差	较好	好
防火墙穿透性	好	差	好
适用范围	低码率视频(广告,片花)	大规模,低延时流媒体	大规模, 移动流媒体

5. 结论

本篇介绍了 HTTP 渐进下载、RTSP/RTP/RTMP 和 HTTP Live Streaming 这几种流媒体协议，并在此基础上对这几种流媒体协议进行了对比分析。总体来说，HTTP 渐进式下载系统部署起来最为简单，但仅适用于较小规模的短视频点播应用，通常只适合 PC Web 端不适合移动设备端点播（通常用 HTTP-FLV 提供渐进式下载，但移动端例如 iOS 不支持 Flash）。基于 RTSP/RTP/RTMP 的协议栈适合于大规模可扩展的交互式实时流媒体应用，但需要专门流媒体服务器的支持，安装和维护起来都较为复杂，且移动端没有天然的支持实现也较为难。 HTTP Live Streaming 可直接部署于目前拥有广泛运营基础的 Web 服务器网络环境，不需要对网络基础设施进行升级改造，不仅 PC 端浏览器支持良好（HTML5 和 Flash 都支持），且移动端 iOS 和 Android 系统本身天然支持，特别适合移动互联网类的应用进行直播和点播。

霹雳流媒体云

——直播流媒体 *API* 云服务

API based, Live Streaming as-a-Service

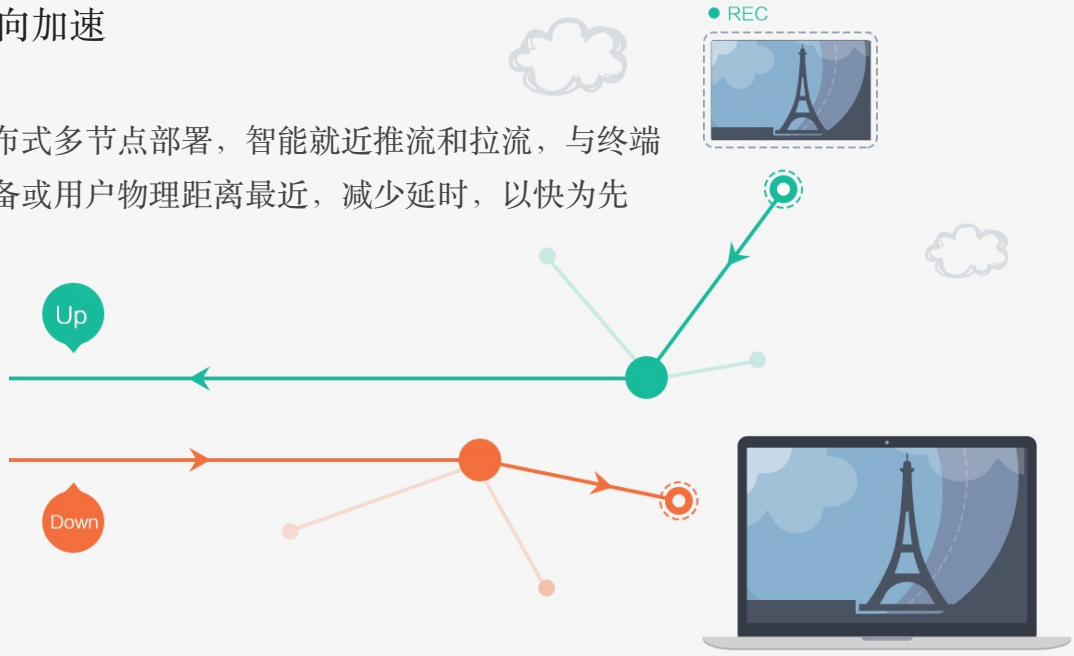
1. 简介

霹雳流媒体云（以下简称“霹雳流”）是七牛研发团队基于七牛分布式多数据中心，充分利用海量网络节点资源打造的一套满足高可用，大规模，低延时，跨终端和全球加速的新型直播流媒体云服务。开发者可以根据霹雳流的 RESTful API，轻松灵活地在数分钟之内为自己的 Web 和 Mobile APP 赋予直播、点播等丰富的流媒体功能属性。

2. 功能特性

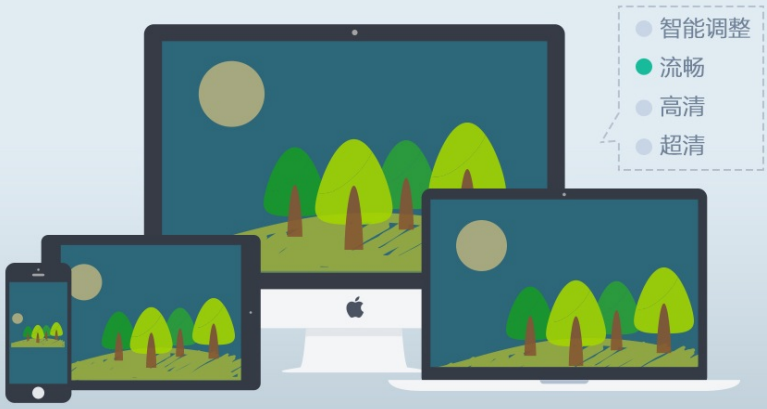
双向加速

分布式多节点部署，智能就近推流和拉流，与终端设备或用户物理距离最近，减少延时，以快为先



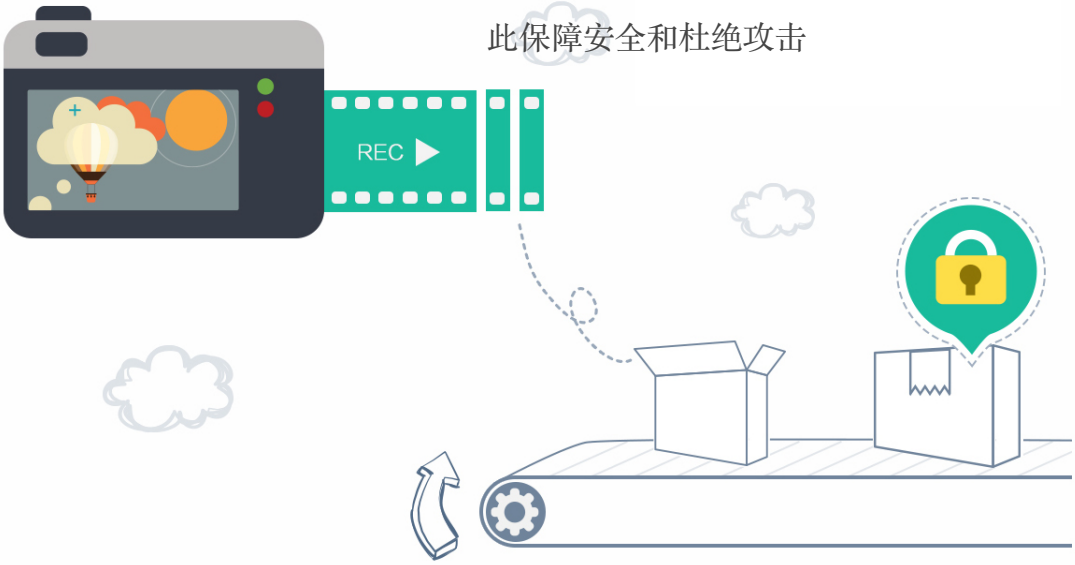
跨平台适配

无论 Android, iOS 或 Web 等其他设备，都支持播放，且支持动态码率均衡，自适应终端用户网络质量，智能调节音质和清晰度



安全加密

支持传输通道加密，支持流媒体内容进行切片加密，支持推流和拉流进行请求授权认证，以此保障安全和杜绝攻击



丰富的开发套件，方便您使用熟悉的编程语言快速集成

没有找到您熟悉的语言包？别担心，还有简洁清晰的 RESTful API 可以直接插入

阅读 API 文档　　寻求帮助

3. 对开发者友好的 RESTful API

如下示例代码，只需简单三步曲，即可快速实现无缝接入。

第一步：新建流（New Streaming）

```
$ curl https://api.pili.qiniu.com/v1/streams
        -H "Authorization: pili {api_key}:{signature}"
        -H "Content-Type: application/json"
        -X "POST"
        --data-binary '{
          "name": " live-streaming-001",
          "stream_key": "a1ecfea7-57b3-495c-b9f1-44a9a8b48313",
          "storage_period": -1,
          "is_private": true,
          "protocol": "rtmp"
        }'

    HTTP/1.1 200 OK
    {
      "id": "54068a9063b906000d000001",
      "name": "live-streaming-001",
      "stream_key": "a1ecfea7-57b3-495c-b9f1-44a9a8b48313",
      "storage_period": -1,
      "is_private": true,
      "protocol": "rtmp",
      "push_url": "rtmp://pili-in.qiniu.com/livestream/4q5cdgn2",
      "play_url": {
        "hls": "http://m3u8.pili.clouddn.com/api/v1/hls/4q5cdgn2.m3u8",
        "rtmp": "rtmp://rtmp.pili.clouddn.com/livestream/4q5cdgn2"
      }
    }
```

第二步：推流（Push Streaming）

```
# 从新建流 API 拿到的推流地址
$ export PUSH_URL="rtmp://pili-in.qiniu.com/livestream/4q5cdgn2"

# nonce为计数每次推流加1,然后再结合流密钥动态签算token以保证不被伪造请求
$ export PUSH_URL="$PUSH_URL?nonce={timestamp++}&token={push_token}"

# 将一个 H.264+AAC 的 MP4 视频转为 RTMP 封包，然后
# 用 FFmpeg 模拟推流，实际上可使用 iOS/Android 等 SDK 实现推流
$ export LOCAL_FILE="/path/to/local.mp4"
$ ffmpeg -re -i $LOCAL_FILE -acodec copy -vcodec copy -f flv $PUSH_URL
```

第三步：拉流（Pull Streaming）

```
# 从之前新建流API拿到的拉流观看直播地址
$ export HLS_PLAY_URL  ="http://m3u8.pili.clouddn.com/api/v1/hls/4q5cdgn2.m3u8"
$ export RTMP_PLAY_URL ="rtmp://rtmp.pili.clouddn.com/livestream/4q5cdgn2"

# 开发调试中，我们用 ffplay 工具模拟播放

# 如果直播的属性为公开，则可匿名观看
$ ffplay $HLS_PLAY_URL
$ ffplay $RTMP_PLAY_URL

# 如果直播的属性为私有，则必须动态签算凭证 token
$ ffplay "$HLS_PLAY_URL?expiry={timestamp}&token={play_token}"
$ ffplay "$RTMP_PLAY_URL?expiry={timestamp}&token={play_token}"
```

Done! That's all.

直播过程中，或直播结束后，都支持按指定时间片段进行回放点播：

```
# 从之前的新建流 API 中拿到流ID
export STREAM_ID="54068a9063b906000d000001"
# 点播 API
export SEGMENTS_URL="https://api.pili.qiniu.com/v1/streams/{$STREAM_ID}/segments/play"

# 调用点播 API
$ curl "$SEGMENTS_URL?starttime={timestamp}&endtime={timestamp}"
    -H "Authorization: pili {api_key}:{signature}"
    -H "Content-Type: application/json"
    -X "GET"

# 请求返回
HTTP/1.1 200 OK
{
    "url": "http://m3u8.pili.clouddn.com/api/v1/hls/jnl617jk.m3u8"
}

# 根据新建流时指定流的隐私属性（公开 / 私有），重复第三步按需拼接播放 URL
```