



技术开启新“视”界
Technology Bring New Vision

三体云视频抗抖动演进之路

踩过的坑和我们做出的应对

- 2005年开始从事音视频技术研究
- 10年移动端音视频研发经验
- 曾任著名视频会议厂商研发经理，负责过多行业100+音视频项目
- 08奥运会TD3G供应商项目主要负责人
- 陌陌、尚德机构等等实时音视频服务
- 平台日处理百万+用户，数千万分钟实时音视频通讯
- 专注实时音视频技术未来发展方向

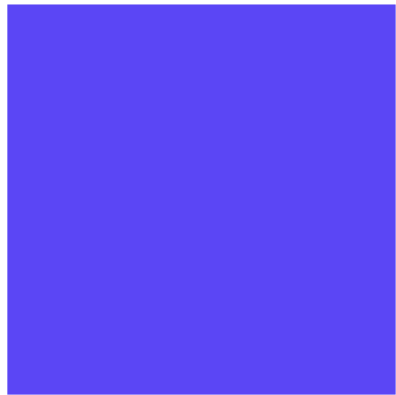


个人介绍

内容大纲



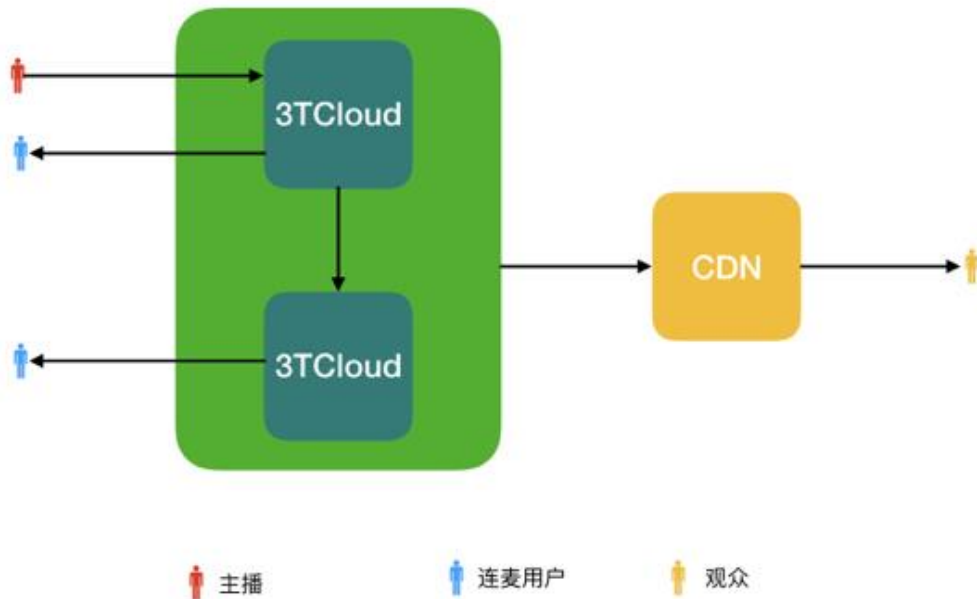
- 抖动是如何产生的？
- 找到系统中可能产生抖动的地方




第一部分



可能引入抖动的地方



可能引入抖动的地方

- 主播端上行
 - 3TCloud内部媒体服务器转发
 - 主播端下行
 - 3TCloud向CDN推流
- 

我们所做过的努力

- 主播上行端的演进
- 媒体服务器间转发的优化
- 主播下行端的策略调整
- 推流CDN的改进



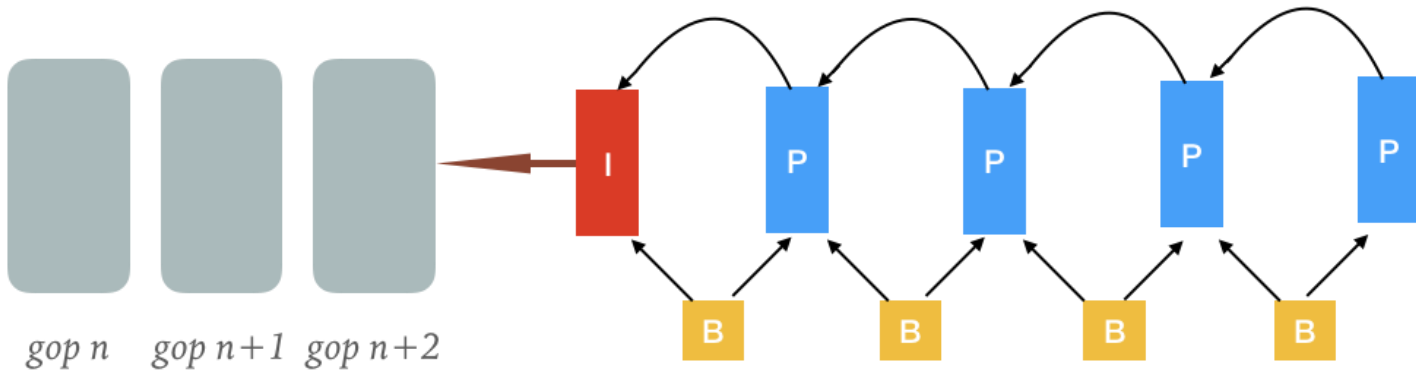
第二部分

主播端上行的演进

- TCP传输
 - KCP传输
 - RTP&RTCP传输
- 

TCP传输阶段

- 没有乱序、丢包，实现简单
- 流控简单-根据发送缓冲区大小以及过去一段时间的网络发送状况



丢包顺序 时间更早的gop, B, P, I



TCP传输阶段

- 延迟较高
- 对网络拥塞的感知过于滞后
- 没有从源头进行控制
- 流控简单粗暴

早期面向企业用户，网络环境比较简单和稳定





KCP传输阶段

- 为什么选择KCP
 - UDX
 - 在移动端上性能消耗较高
 - 在当时并不支持ipv6
 - UDT
 - 网络发生严重抖动时表现较差
 - bug





KCP传输阶段

- 牺牲10%-20%带宽，平均延迟降低30%-40%
- 动态码率调整
 - 根据缓冲区的变化、RTT以及一段时间的数据发送情况估算带宽
 - 对网络拥塞的感知依然较为滞后
 - 带宽估算偏保守
 - 为保证画质，可能导致帧率过低





KCP传输阶段

- 较高丢包率的情况下延迟降低
- 丢包与延迟并存的情况下呢？
 - 30%丢包，500ms+网络延迟，端到端的视频延迟5秒以上
- 一脚踩进了另一个坑里
 - 多人连麦，音频高频率的小数据包导致的过量重传





RTP&RTCP传输阶段

- 基于丢包的拥塞控制
 - 无线传输中，丢包率大增，并不等价于网络拥塞的丢包
 - 网络设备中抗抖动buffer的存在，使得这种塞满链路的带宽评估并不准确
 - 塞满的buffer会进一步恶化延迟





RTP&RTCP传输阶段

- 基于延时的拥塞控制
 - tcp vegas、webrtc gcc、google bbr
 - 存在的问题
 - 如何与基于丢包拥塞控制的流共存，往往在前者塞满网络buffer前就进入拥塞避免阶段





RTP&RTCP传输阶段

- 实时传输的拥塞控制设计目标
 - 低延迟
 - 极端网络情况下端到端延迟控制在1秒以内
 - 尽可能的抢占带宽
 - 保证码率控制后的视频质量



RTP&RTCP传输阶段



➤ 策略

➤ 容许一定程度的丢包

➤ gcc中，低于10%的丢包不会降低估算带宽

➤ 动态调节带宽估算

➤ 独享带宽下，提高估算敏感度，依赖基于延时的控制算法

➤ 网络延迟增加时，适当降低敏感度，尽可能抢占带宽

➤ 延迟恶劣的情况下，退化为基于丢包的控制算法



RTP&RTCP传输阶段



➤ 策略

➤ 重传、fec

- 重传包，fec与调整后的视频码流共享估算的带宽
 - $\text{nack_bw} + \text{fec_bw} + \text{video_bw} = \text{est_bw}$
- 调整后的视频码流不应过低
 - 保证视频质量
- 重传流按时间片限流发送
- 较低丢包且带宽充足时开启fec，根据丢包和延时动态调整fec策略





RTP&RTCP传输阶段

➤ 策略

➤ 视频码流控制

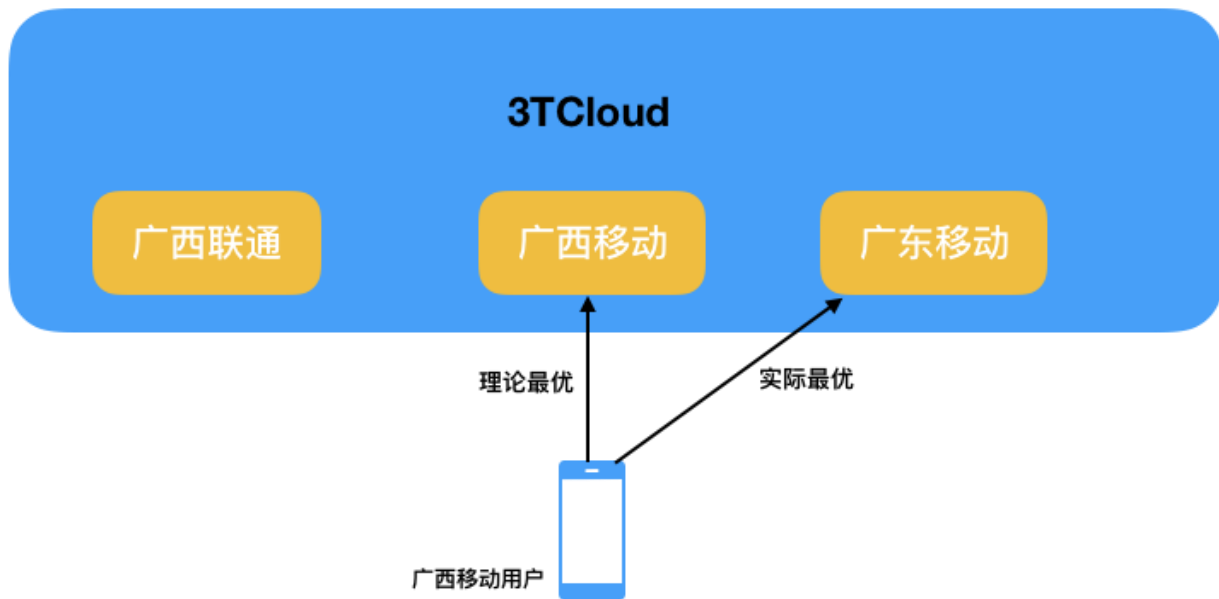
- 丢包较高时，适当降低视频码流，为重传留出足够带宽
- 视频码流较低时，适当降低帧率，保证画质。极端情况下可适当降低分辨率

➤ 结果


在延时，流畅度，画质间找到平衡



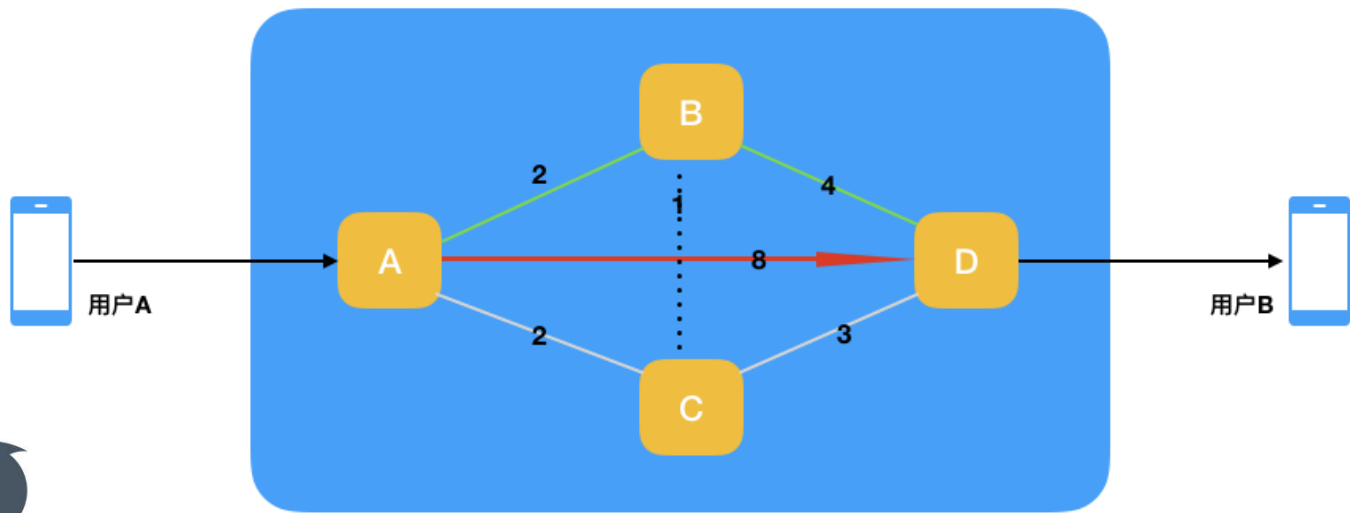
接入媒体服务器的选择



接入媒体服务器的选择

- 大量后台数据分析
 - 向主播端推送最适合媒体服务器
 - 实时网络探测
 - 主播端动态选择最适合的媒体服务器
 - 负载控制
- 

媒体服务器间的转发





服务器间的转发策略

- 面对的问题
 - 直连最简单，根本实现不了
 - 最小的跳数往往并不是网络的最优选择
 - 突发异常的应对
 - 基于运营成本的考量





服务器间的转发策略

- 实时的服务器间网络探测
- 大量数据分析，帮助路径选择
- 路径增加权重，优化路径选择
- 实时路径切换，应对异常情况



主播下行端



- 低延迟与流畅度间的矛盾
- 不同策略的选择
 - 维持低延迟，牺牲部分弱网下的流畅度
 - 允许适当延迟，提高弱网下的流畅度

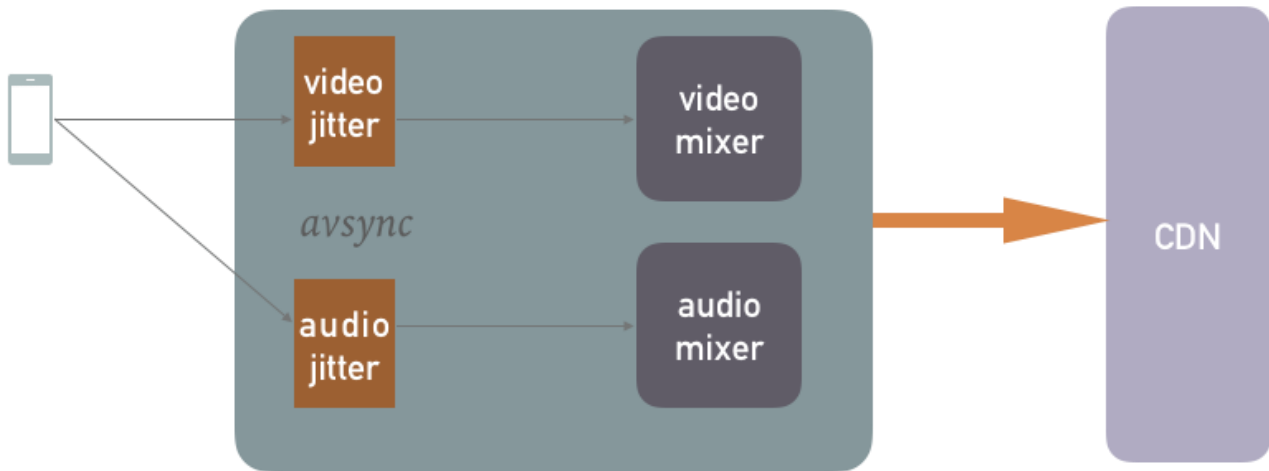


主播下行端



- 动态码流切换
 - 带宽允许情况下对端上行大小两路视频
 - 带宽足够情况下拉取原始视频路播放
 - 带宽不够情况下拉取低码流视频播放

推流CDN





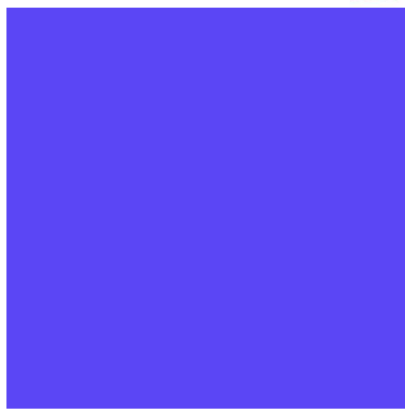
推流CDN

- 假定混音、混屏模块引入固定延迟
- 允许增加一定的延迟对抗可能的抖动
- 音视频经过同步后输出到混音、混屏模块
- 进一步优化
 - 多个用户媒体流间的同步



最终的结果

- 低延迟
- 高流畅度
- 高质量



第三部分

友商对比测试

测试设备	iPhone 8 Plus	iphone 6
系统版本	11.4.1	11.1.1

上行网络限制			三体云		友商	
Bandwidth/kbps	Latency/ms	Loss/%	实际效果	延迟	实际效果	延迟
不限速	0	10	视频流畅, 画质清晰	1.3s	视频流畅, 画质清晰	1.6s
	0	20	视频流畅, 画质清晰	1.3s	视频流畅, 画质清晰	1.6s
	0	30	视频流畅, 画质清晰	1.3s	视频流畅, 画质清晰	1.8s
	300	0	视频流畅, 画质清晰	0.5s	视频流畅, 画质清晰	0.5s
	800	0	视频流畅, 画质清晰	1.2s	视频流畅, 画质清晰	1.7s
	300	30	视频偶尔卡住一下, 整体流畅	2.3s	视频整体卡顿	2.9s
	800	20	视频卡顿频繁	2.9s	视频卡顿频繁	4.6s
上下行网络限制			三体云		友商	
Bandwidth/kbps	Latency/ms	Loss/%	实际效果	延迟	实际效果	延迟
600	0	10	视频流畅, 画质清晰	1.1s	视频流畅, 画面轻微马赛克	1.1s
	0	20	视频偶尔卡住一下, 整体流畅	1.2s	视频整体卡顿严重	1.4s
	300	0	视频流畅, 画质清晰	1.3s	视频慢动作, 轻微马赛克	1.6s
	800	0	视频流畅, 画质清晰	1.8s	视频卡死频繁,	2.2s
	300	30	视频卡死频繁	3.2s	视频卡死频繁	4.7s

Thank you

