LiveVideoStack

精彩内容赏析

它山之石可以攻玉

分享最新技术研究、探索与应用实践

仰望星空，脚踏实地

观察行业热点技术、最新动向以及未来趋势

关注 LiveVideoStack 公众号

浏览更多精彩内容

篇首语

LiveVideoStack 社区

LiveVideoStack 是国内及华人中唯一的专注在音视频领域的全球顶级技术社区之一。通过 LiveVideoStackCon 音视频技术大会，将全球顶尖的音视频技术人才汇聚于此，推动全球范围内的多媒体音视频技术交流。通过技术内容、互联网人才的交互，迅速推动技术在产业中的落地，增强了行业与音视频技术的融合性发展。

LiveVideoStack 已通过技术培训、高质量技术内容及咨询服务，帮助技术人成长，推动了相关开源项目与最佳实践普及和传播，解决了部分企业发展中的技术难点。全球多媒体音视频领域拥有最新技术、最有潜力的公司和组织都已汇聚于此。

LiveVideoStack 精彩内容赏析

5G 网络、AI 人工智能等技术的发展为多媒体技术的应用提供了前所未有的机遇，多媒体技术正在渗透到生活的方方面面。

LiveVideoStack 精彩内容赏析收录往届 LiveVideoStackCon 音视频技术大会、LiveVideoStack 线上分享中部分专家精彩演讲内容，涵盖编解码、网络传输、客户端/服务端架构、QoE、开源技术等音视频技术相关话题。

此外，还包含音视频技术行业最新动向报道、业内资深专家深度采访、热点技术专题策划等系列文章内容。希望能为大家深入了解音视频相关技术、解决当前存在问题提供力所能及的帮助。

目录

它山之石可以攻玉

- 编解码之争
  - AV1 的实时模式
  - 中国 AVS 超高清编码标准体系与生态建设
  - H.265 在花椒直播中的应用与优化
  - 新一代视频编码标准：VVC、AVS3
- 国内之痛：传输协议与网络优化
  - 新一代直播传输协议 SRT
  - B 站 QUIC 实践之路
  - TCP 的困境与解决方案
- 服务端架构演进与实践
  - 视频云下半场 向前走还是向“厚”走?
  - 新浪微博：大规模离线视频处理系统的架构设计
- 跨 Web 与客户端的优化之路
  - 360 视频云 Web 前端 HEVC 播放器实践剖析
  - 微博客户端播放器的演进之路
- QoE：一切为了用户体验
  - 于冰：用户导向的音视频体验优化
  - 提升音视频 QoE 的六条建议
- AI+多媒体，不止于技术升级
  - 编码压缩新思路:面向 QoE 的感知视频编码
  - 实时远程医学影像服务质量保障与网络优化
- 音频技术之美
  - 开源声码器 WORLD 在语音合成中的应用
  - 语音编解码技术演进和应用选型
- 多媒体开源技术栈
  - 开源流媒体服务器：为何一定得再撸个新的
  - 基于 Nginx 的媒体服务器技术
  - 刘岐：FFmpeg Filter 深度应用
  - WebRTC 的拥塞控制和带宽策略

仰望星空，脚踏实地

- 2020 年热点新闻
- 专题内容规划
  - 专访腾讯多媒体实验室商世东：咬咬牙也要跟上新版本的发布
  - 云游戏之真心话：一切才刚刚开始 | 对话南京大学副教授马展
- 从开发者到决策者，你的眼界足够了吗?
  - 码代码，到白头|专访 SRS 创始作者&阿里云 RTC 服务器团队负责人杨成立
  - 李郁韬：短期爆发还是未来趋势？腾讯云海量音视频通信服务背后的技术发展

LiveVideoStackCon 技术专家

LiveVideoStackCon 合作伙伴

结束

它山之石可以攻玉

编解码之争

AV1 的实时模式

不久前，Google Duo 宣布将全面转向 AV1。相比于 VP9，AV1 能够带来更好的编码效率，为用户提供更好的视频体验。本文来自 Google 软件工程师 姜健在 LiveVideoStack 线上分享的内容。

姜健　LiveVideoStack 线上分享　2020-06-10

1.关于 AV1

AV1 编解码技术对视频质量有非常大的提高，尤其是在低码率和网络状态非常差的情况下，AV1 编解码技术相对于 VP9 和 H264 编解码都有一个质的飞跃。

从图中明显可以看出，在码率为 30kbps 情况下，左图是 AV1 编解码技术处理效果，右图是 Duo 目前所使用的编解码技术的处理效果，对比质量差别很大。在印度、巴西等一些发展中国家大多数人的网络接入情况较差，这些网络较差的地区正是谷歌最受欢迎的市场，所以 Google Duo 也在尽力提高低码率的视频质量，因此 AV1 技术对 Duo 来说有非常大的帮助。

RTC Encoder Configuration

- Profile 0
- Real-time mode
- 1 Pass
- 0 lag-in-frame
- CBR
- Speed >= 6

```
aomenc --profile=0 --width=$width --height=$height --kf-max-dist=#key_frame_distance --static-thresh=0 --end-usage=cbr --cpu-used=$speed --passes=1 --usage=1
--lag-in-frames=0 --noise-sensitivity=0 --error-resilient=0 --threads=$threads --tile-columns=#tiles --aq-mode=3 --min-q=2 --max-q=63 --undershoot-pct=50
--overshoot-pct=50 --buf-sz=1000 --buf-initial-sz=500 --buf-optimal-sz=600 --max-intra-rate=300 --deltaq-mode=0 --enable-tpl-model=0 --enable-obmc=0
--enable-warped-motion=0 --mode-cost-upd-freq=2 --coeff-cost-upd-freq=2 --enable-ref-frame-mvs=0 --mv-cost-upd-freq=2 --fps=30/1 --target-bitrate=50 input.yuv -o
output
```

Google

关于 AV1 编码器的设定，图中最下端区域是可执行的命令行，一般使用该命令行测试手机或电脑上的速度，需要强调的设定有以下几点：

- Profile 必须设置为 0；
- Real-time 的模式；
- 在实时视频编解码时必须是 1Pass
- lag-in-frame 必须设定 0，目标码率必须恒定，若目标码率上下波动的起伏很大对网络的要求很高；
- 最后比较重要的是速度设定必须大于等于 6，目前我们有实时模式下 6、7、8 三个速度档，很多编码器的决策设定都需要根据 CPU 参数。

Real-time mode in AV1

- Speed 6,7,8
  - Speed 6 - RD pick-mode
  - Speed 7,8 - Non-RD pick-mode path, faster
- Variance based partition
  - Up to 128x128 for 720p and above, 64x46 for lower
- SVC support
- More NEON optimization on the way

Google

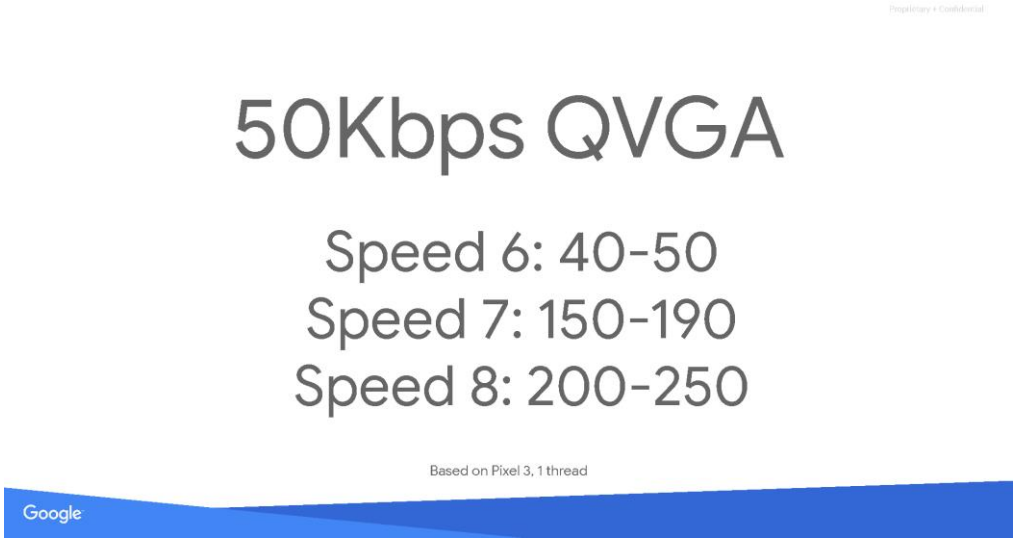
AV1 的实时模式与 VOD 的区别有以下四点：三个速度档：6、7、8。

- 速度档 6 严格来讲与 VP9 的区别非常大，VP9 的实时速度档使用的是 Non-RD 进行决策，因为 RD (rate-distortion)决策会很慢，但是在 AV1 中特意增加了速度档 6，使用了 RD(rate-distortion)优化来进行决策，因此速度档 6 的质量是最好的（速度越小质量越好）。
- 速度档 7、8 使用 Non-RD 进行决策，Non-RD 即使用其他算法进行快速决策。但是速度越快质量越差。
- AV1 对于 720P 以及更高清视频使用了最大 128×128 的区块，低分辨率视频使用 64×64 分区。如果视频分辨率太小使用 128×128 分区后，区块很少，会降低质量，但是对于高清视频来讲，使用 128×128 可以提高编码速度。分区使用 Variance 进行决策，在 VP9 中的基于方差分区（Variance based partition)使用的是机器学习中的 CNN 训练模型，通过计算 Variance 与提前设定的阈值进行比较。
- AV1 中加入了 SVC 支持，并且在 libaom 的代码中放入了 SVC 示例，该 SVC 支持比 VP9 更加灵活。
- AV1 目前的 SIMD 优化还没有完全结束，实时编码速度在未来还有比较大的提升空间，暂时还未对一些函数进行优化，目前正在手机端进行 NEON 的优化。

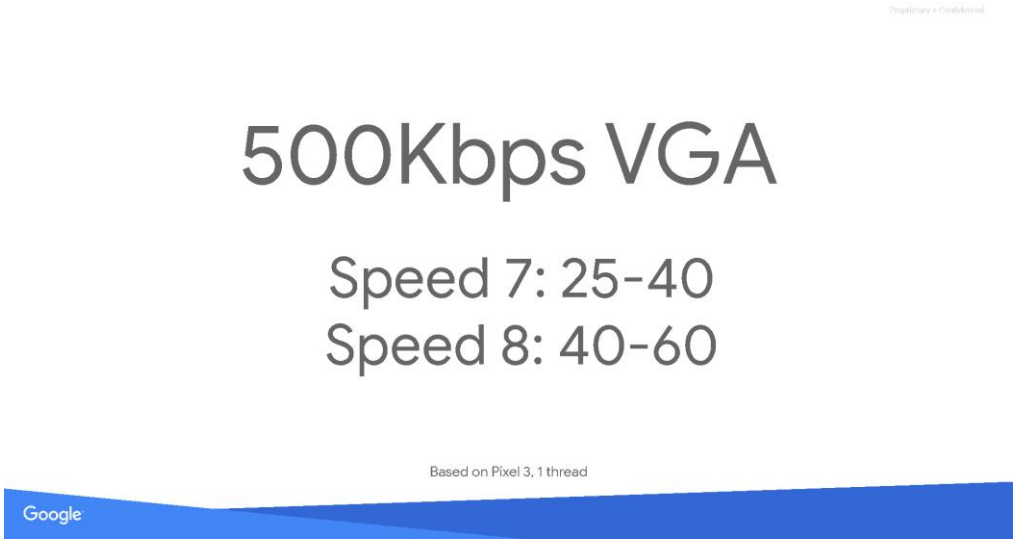
2.测试成果

实时模式的优化主要是速度与质量的平衡。

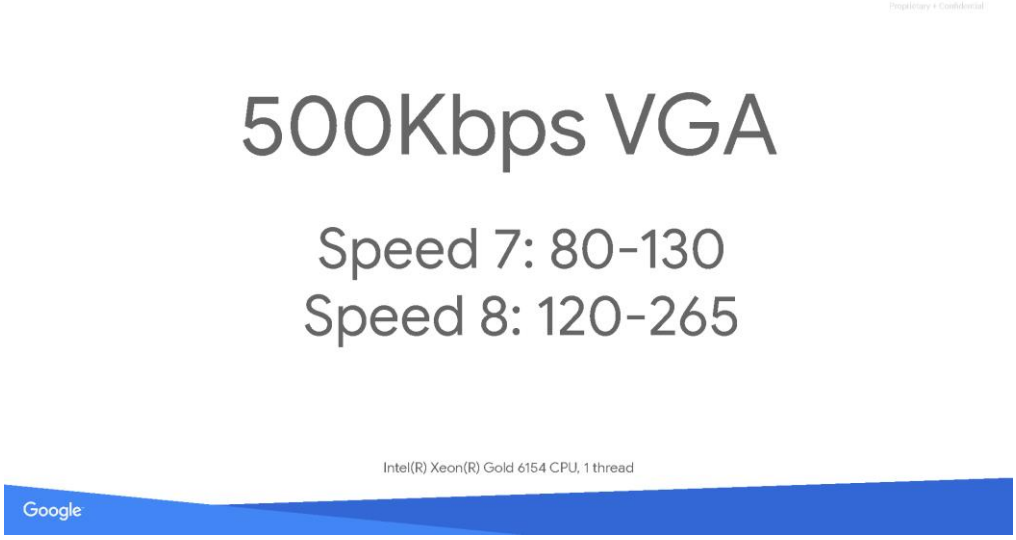
手机端速度测试



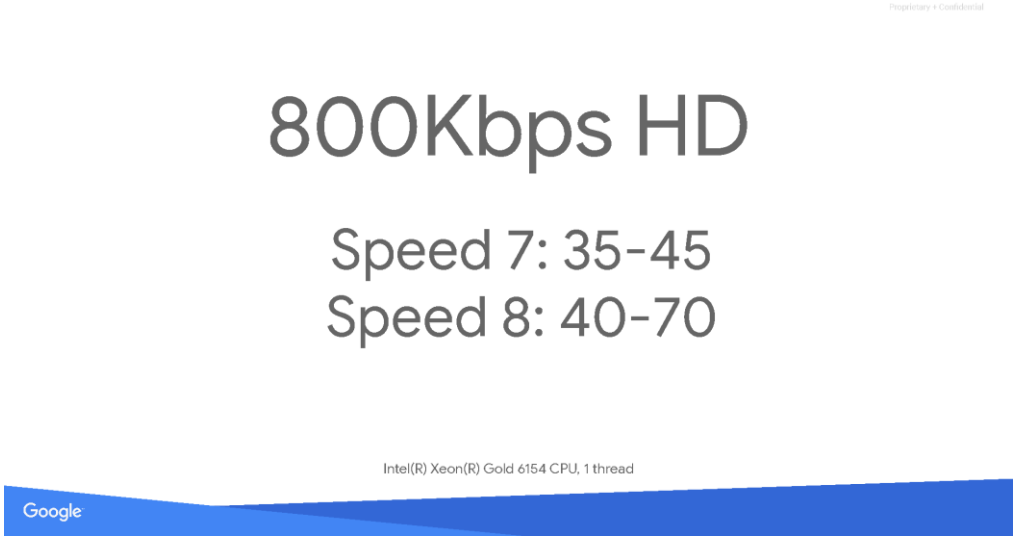
首先进行了低码率 50Kvps QVGA 的测试，320×240 及以下分辨率的速度 6 档可达到 40-50 帧每秒的传输速度，速度 7 档可达到 150-190 帧每秒，速度 8 可达到 200-250 帧每秒，均为单线程测试。



640×360 或 640×480 的视频在 500KbpsVGA 进行测试，结果显示速度 6 档位的传输速度非常慢，速度 7 档可达到 25-40 帧每秒，速度 8 挡可达到 40-60 帧每秒的速度，同样均为单线程测试。由于手机端诸多性能因素使用多线程测试数据不稳定，因此未进行大量多线程测试。台式机的速度测试。

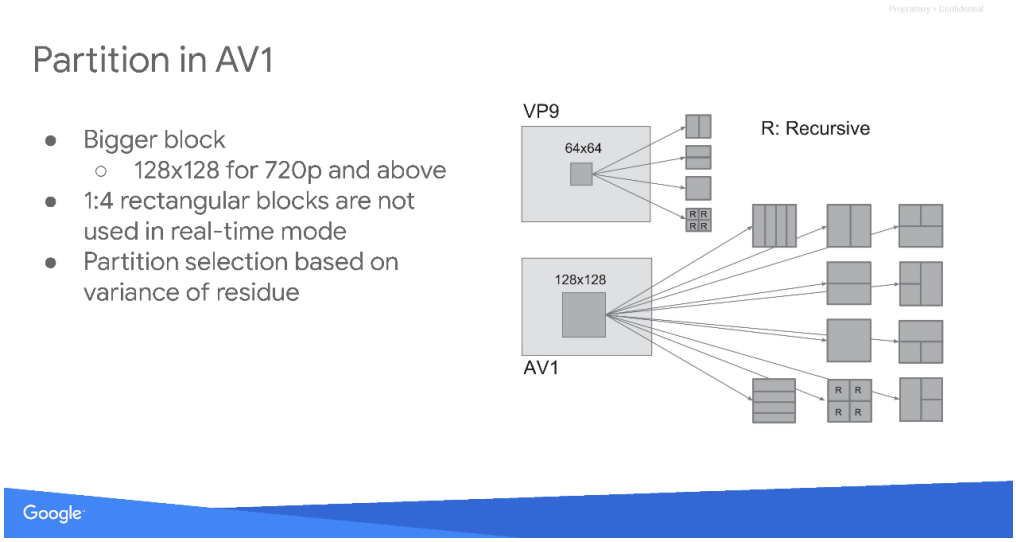


台式机 CPU 参数如图中最下端所见，单线程测试，500Kbps VGA 测试，速度档 7 可达到 80-130 帧每秒的传输速度，速度档 8 可达到 120-265 帧每秒的传输速度。



台式机上测试 800Kbps HD 的 720P 以上，单线程速度 7 挡的测试结果为 35-45 帧每秒，速度 8 的测试结果为 40-70 帧每秒，该速度较 VP9 差，因为 AV1 编码器的复杂程度较高，虽然速度较慢，但是视频质量高很多。

3.AV1 编码器优势



如图为 VP9 与 AV1 区块划分的区别，VP9 最大支持 64×64 区块，而 AV1 最大支持 128×128 区块。AV1 中的区块存在边长比为 1:4 的长方形分块，这种情况很少见，在实时编码器不考虑搜索该区块，比 VP9 灵活很多。AV1 编码器的算法复杂度虽然高但是搜索空间大，视频质量提升很多。

Source Content Analysis

- SAD, variance
- Skip modes
  - new motion vector
  - intra search

Google

在视频编码之前，首先对原视频进行分析，即原视频的 SAD (绝对误差和) 和 variance (方差)，使用这两个数据跳过一些模式的选择，例如在计算运动向量（motion vector）时需要搜索很多模式，使用 SAD 和 varance 两个数据与阈值比较，可以跳过运动向量的一些模式选择。
另外在帧内编码时，帧内模式的一些搜索也可以通过这两个数据简化。手机端对 SAD（绝对误差和）和 variance（方差）的计算进行了优化，因此运算速度很快。

Flexible Multi-Threading

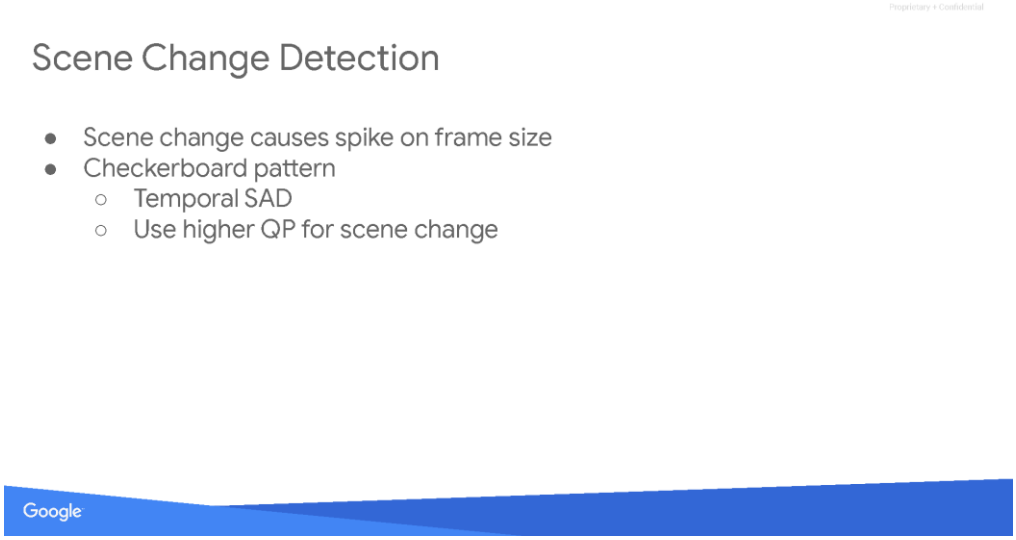
- Tile
  - AV1: Unit of superblock - 64x64 or 128x128
- Row
- 4 tiles, 720p, 4 threads
  - 3x speed up[*]
- Large scale tiles
  - Allow decoder only decode specific section[*]

[*]Chen, Y., Mukherjee, D., Han, J., Grange, A., Xu, Y., Parker, S., . . . Liu, Z. (2020). An Overview of Coding Tools in AV1: The First Video Codec from the Alliance for Open Media. *APSIPA Transactions on Signal and Information Processing*, *9*, E6. doi:10.1017/ATSIP.2020.2

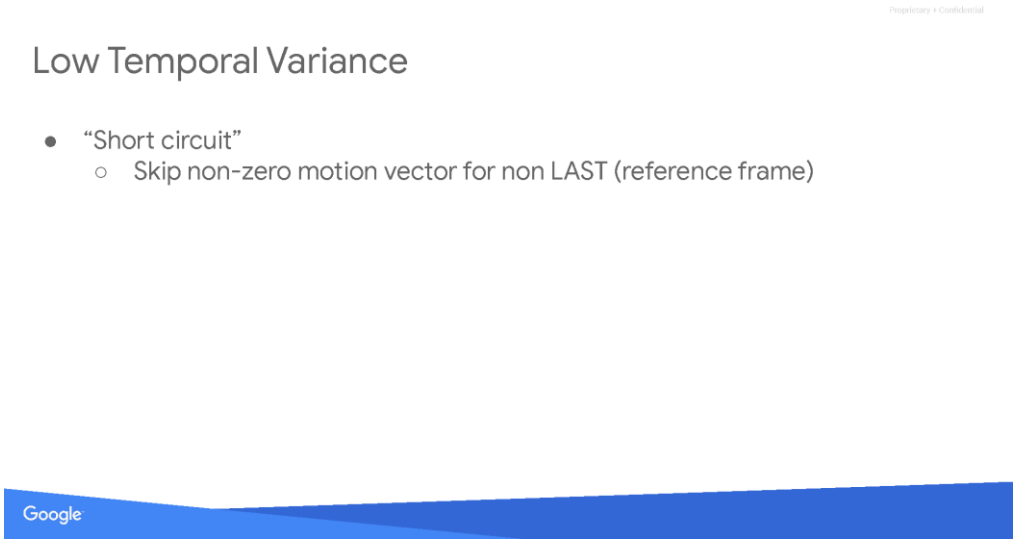
Google

VP9 只支持固定 256 像素宽度的块，当出现分辨率较小的帧例如 640×480 等，最多只能使用两个区块做并行。但是 AV1 在这方面很灵活，720P 分辨率以上的最大区块单元是 128 个像素宽度，720P 以下为 64 像素宽度，因此可以根据情况灵活增加区块，以进行多线程操作。同时，AV1 与 VP9 都支持多行并行处理。

另外 AV1 的解码器可以只解码特定区域，例如某一个区块为 64 像素宽度，可以要求解码器解码其中某一部分，这样当用户对视频的某一部分感兴趣时就可以只解码该部分。图中最下边的文献是对 AV1 编码工具的综述，文中指出 720P 分辨率使用 4 个线程且进行 4 块区分，有 3 倍的速度提升。



当实时视频的前后两帧的变化很大时，码率会突然升高，帧的尺寸会突然变化很大，对网络是很大的压力，导致视频质量突然变差，因此 AV1 编码器会每隔一个最大区块计算时域 SAD 与前一帧作比较，若该时域 SAD 非常高，编码器即认为发生很大变化，就使用更高的 QP，缩小该帧的尺寸，以降低网络压力。



AV1 编码器会通过计算时域的方差简化模式选择

SVC 的 AV1 设定（比 VP9 直观很多）：其中有三个变量：参考帧（设定不同分辨率的参考帧位置）、当前帧刷新哪一个缓存、模式设定，AV1 的设定直接将所有的控制权交给用户，用户需要哪种模式即可设置哪种模式。

4.AV1 编码器的未来

目前 AV1 正在开发 WebRTC 支持，很快会推出。

由于 AV1 编码器的复杂度远远高于 VP9，因此在产品中较大的缺点是编码之后的尺寸是 VP9 的几倍，进行很多优化之后，例如去掉一些用不到的工具等，使编码后尺寸下降 30%。

中国 AVS 超高清编码标准体系与生态建设

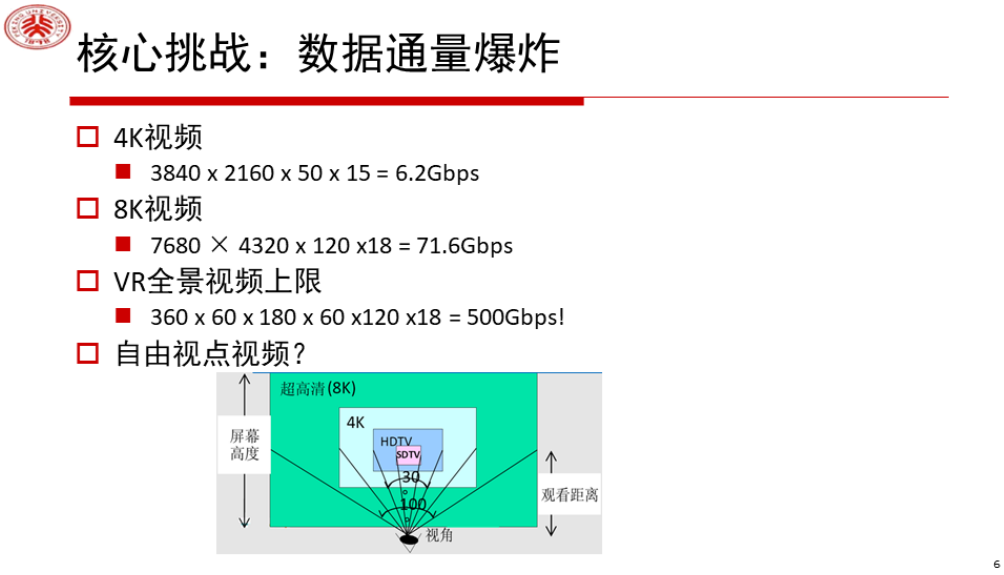
随着 5G 时代的到来，视频压缩方面面临更大的挑战，另外用户对于视频需求的提高使得在视频压缩方面需要做更多的提升。本文来自北京大学的王荣刚在 LiveVideoStackCon 2019 深圳站上的精彩分享，主题是中国 AVS 超高清编码标准体系与生态建设。

王荣刚　LiveVideoStackCon 2019 深圳　2020-03-10

1.中国 AVS 超高清编码标准体系建设背景

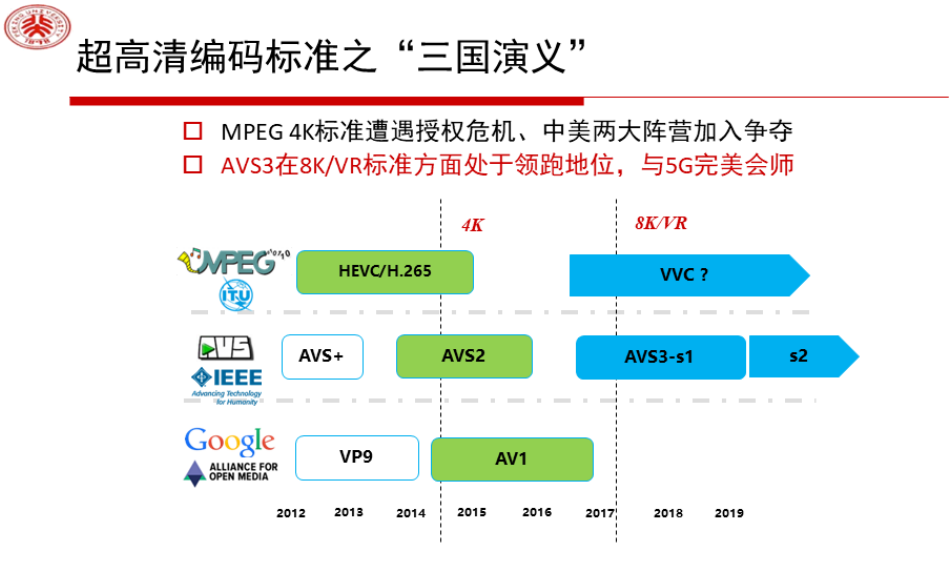
大家好，我是来自北京大学的王荣刚，随着 5G 时代的到来，视频进入以 4K，8K，VR 超高清时代，国家和地方政府也相继出台了很多超高清视频行动计划。

核心挑战：数据通量爆炸



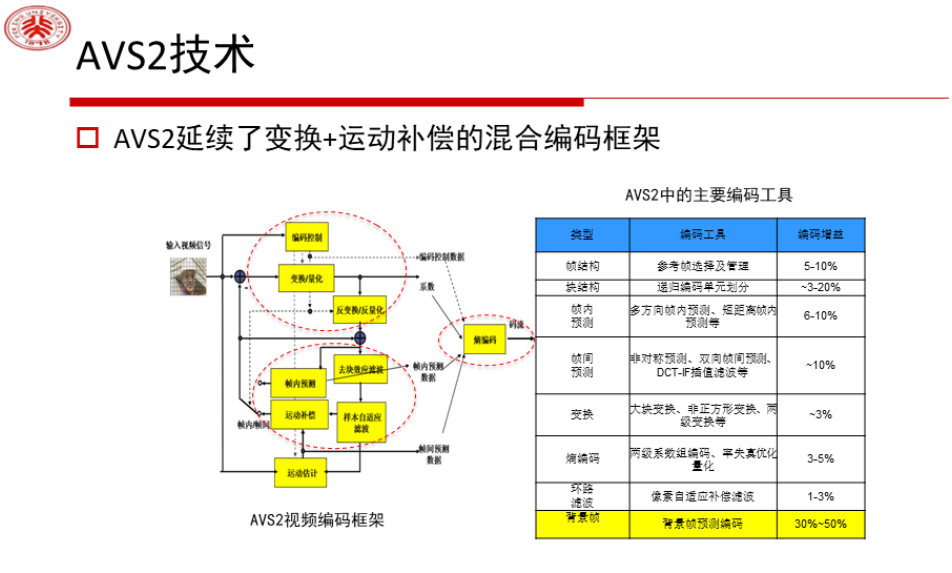
视频压缩方面一直面临着巨大的挑战，在网络带宽提升后，我们对视频需求更高，两者的上升速度并不匹配。4K 视频一秒可达 6Gbps 数据量，8K 视频可达 70Gbps，而 VR 视频上限达 500Gbps，另外还有自由视点视频需求，即使使用 5G 网络，仍然面临极大挑战，因此在视频压缩方面我们依然需要做很多提升。

超高清编码标准之“三国演义”



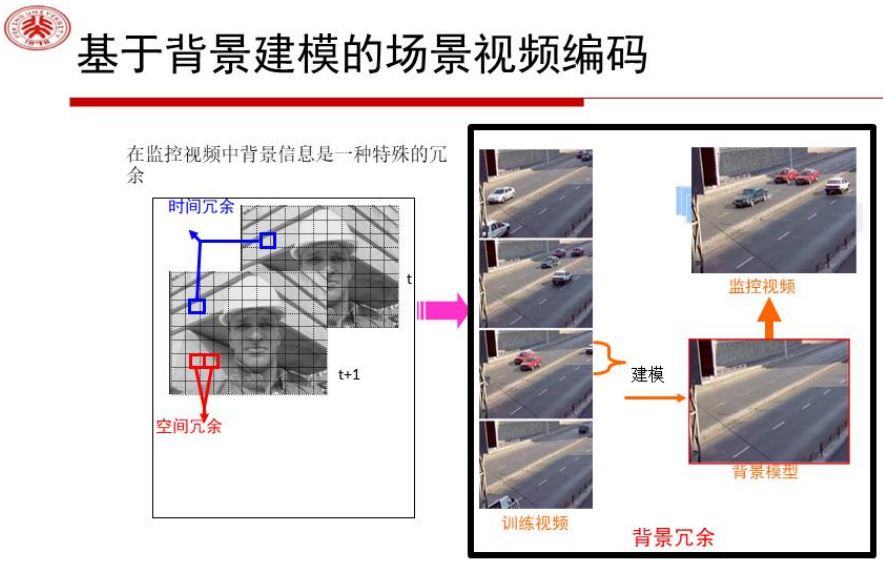
现今编码标准呈“三国演义”态势。国际主流标准为 MPEG 和 ITU 标准。这个标准以针对 4K 视频的 H.265 标准为代表，目前正在研发面向 8K、VR 视频的 VVC 标准。AV1 虽然出现较晚，但备受关注。AVS 标准已存在多年，是国内一个相对平价的标准，而 AVS2 主要面向 4K 视频。AVS3 的发展分为两个阶段，第一阶段主要是面向 8K 与 5G 结合，预计在国内广泛使用；第二阶段目标是超越 VVC 标准压缩效率。

2.AVS2

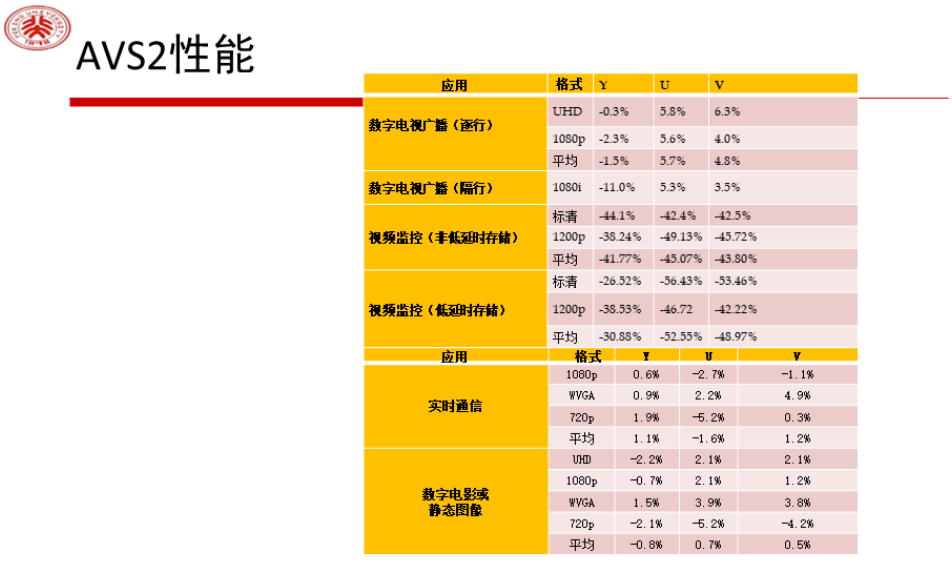


类型	编码工具	编码增益
帧结构	参考帧选择及管理	5-10%
块结构	递归编码单元划分	~3-20%
帧内预测	多方向帧内预测、短距离帧内预测等	6-10%
帧间预测	非对称预测、双向帧间预测、DCT-IF插值滤波等	~10%
变换	大块变换、非正方形变换、两级变换等	~3%
熵编码	两级系数组编码、率失真优化量化	3-5%
环路滤波	像素自适应补偿滤波	1-3%
背景帧	背景帧预测编码	30%~50%

AVS2 框架与 H.265 类似，在编码工具方面有所创新。其中基于背景建模的场景技术可大幅度提升监控场景 30%到 50%编码效率。



AVS2 性能



应用	格式	Y	U	V
数字电视广播（逐行）	UHD	-0.3%	5.8%	6.3%
	1080p	-2.3%	5.6%	4.0%
	平均	-1.5%	5.7%	4.8%
数字电视广播（隔行）	1080i	-11.0%	5.3%	3.5%
视频监控（非低延时存储）	标清	-44.1%	-42.4%	-42.5%
	1200p	-38.24%	-49.13%	-45.72%
	平均	-41.77%	-45.07%	-43.80%
视频监控（低延时存储）	标清	-26.52%	-56.43%	-53.46%
	1200p	-38.53%	-46.72	-42.22%
	平均	-30.88%	-52.55%	-48.97%

应用	格式	Y	U	V
实时通信	1080p	0.6%	-2.7%	-1.1%
	WVGA	0.9%	2.2%	4.9%
	720p	1.9%	-5.2%	0.3%
	平均	1.1%	-1.6%	1.2%
数字电影或静态图像	UHD	-2.2%	2.1%	2.1%
	1080p	-0.7%	2.1%	1.2%
	WVGA	1.5%	3.9%	3.8%
	720p	-2.1%	-5.2%	-4.2%
	平均	-0.8%	0.7%	0.5%

上图所示内容是 AVS2 和 HEVC 的性能对比。在数字广播、电影、实时通讯领域，二者的性能相当。图中的负号表示相对于 H.265 节省的码率，正好则是增加的码率。监控场景可比 265 节省 30%到 40%码率。总体略优于 HEVC。

AVS2 4K 视频编码性能

4K视频编码性能

- 4K@50P 10比特编码

AVS2编码效率比HEVC/H.265平均高7.4%

应用	序列	Y	U	V
全4K视频（央视提供）	UHD_sidai_3840x2160_50_10bit	-3.9%	1.5%	-8.1%
	UHD_veggie_fruits_3840x2160_50_10bit	-14.7%	-15.2%	-15.1%
	UHD_fountain_lady_3840x2160_50_10bit	-3.9%	-7.5%	-7.5%
	UHD_leaves_3840x2160_50_10bit	-11.6%	-8.6%	-6.5%
	UHD_opera_3840x2160_50_10bit	-11.3%	-2.7%	-4.3%
	UHD_dancer_3840x2160_50_10bit	-5.9%	-2.4%	-2.7%
	UHD_SilkSeduction_3840x2160_50_10bit	-6.7%	3.7%	3.9%
	UHD_SpringFestivalGala_3840x2160_50_10bit	-11.3%	-2.2%	-3.2%
	平均	-7.4%	-4.2%	-4.5%

12

上图是针对 4K、10 比特的编码序列测试效果。在最优参考软件上，AVS2 比 HEVC 性能高 7.4%，测试过程中选用的是央视提供的 4K 50P 视频。

AVS2 应用

基于以上测试结果及前期准备，2018 年 10 月，央视开播了中国第一个超高清频道，在同年 11 月，广东电视台也开播了超高清频道，目前这两个频道都使用的是 AVS2 标准。

应用—IPTV

- 2018年3月30日，由中关村视听产业技术创新联盟会同中国智慧家庭产业联盟、中国视频体验联盟、广东4K生态产业联盟及众多相关企业等编写的《IPTV 业务系统AVS2实施指南（2018年）》正式发布
- 中国电信和移动4K机顶盒规范要求支持AVS2标准，支持AVS2标准的机顶盒近5000万

14

另外 AVS2 在 IPTV 业务方面比广播电视发展 4K 更早，中国智慧家庭联盟编写了《IPTV 业务系统 AVS2 实施指南》，这是一个针对 IPTV 系统切换 AVS2 的指导。目前央视 4K 视频还未开放给 IPTV 系统平台，在运营商的推动下，应该很快会开放。

我们在互联网领域也做了很多工作，其中最具代表性的就是腾讯在 2017 年推出了 TPG 图像格式。TPG 格式不仅适用于图像，图像也可利用它进行压缩，可实现比 JPEG 提升 50%码率的效果。目前腾讯云也已上线 TPG。咪咕视频也已经有 AVS2 内容上线。

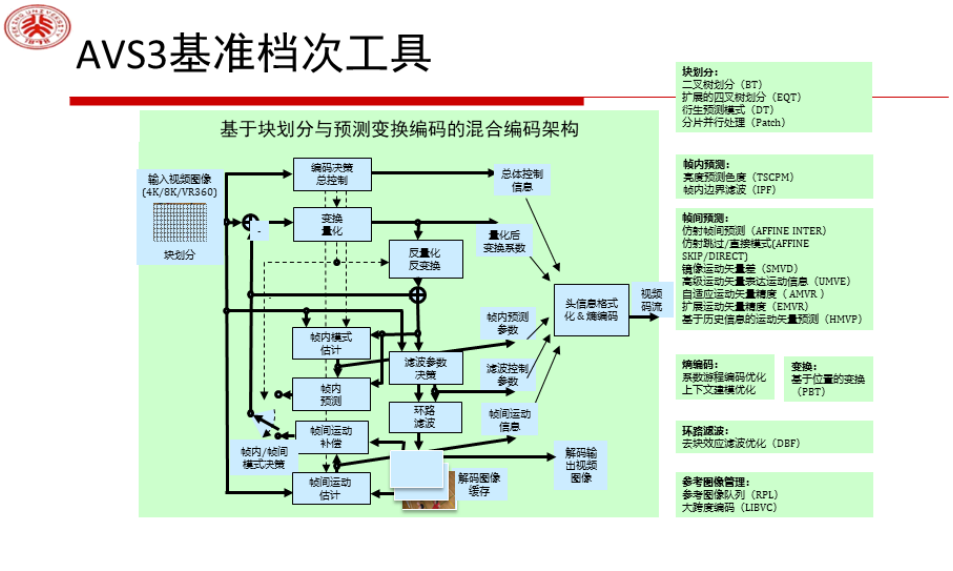
AVS2 国际化

AVS2 国际化工作也取得了重要进展。在 2018 年，UHD Forum 将 AVS2 列为和 H.265 并列的超高清编码标准，同年 11 月，AVS2 被正式颁布为 IEEE 国际标准。

3.AVS3 标准

AVS3 标准主要为下一代做 8K 应用布局，如之前所讲，它主要分为两个阶段，第一阶段为希望我们做产业抢跑，避免出现 4K、H265 那种已经布局机顶盒不便更换的情况。第二阶段目标为实现比 VVC 高 20%码率，通过早发布比上一代国际标准高百分之二三十的新一代标准形成一定国际领先效果。AVS3 的另一个预期是希望能在智能编码方面取得一定进展。

AVS3 基准档次工具



上图展示的是一些 AVS3 的编码工具，它的框架与 AVS2 是基本相同的，在图片右边部分列举了 AVS3 新增的编码工具。通过这些工具可以实现比 HEVC 高 30%的目标。

AVS3 基准档次性能

AVS3基准档次性能

- AVS3比AVS2在8K超高清视频上性能提升超过30%

	Random Access AVS3 HPM2.4 vs. AVS2 RD19.2		
	BDR-Y 码率节省	BDR-U 码率节省	BDR-V 码率节省
8K	-36.9%	-46.9%	-44.5%
4K	-23.2%	-26.4%	-28.7%
1080P	-20.8%	-25.0%	-26. 3%
720P	-20.5%	-25.1%	-24.2%
平均性能	-21.50%	-25.5%	-26.4%

20

上图是 AVS3 和 AVS2 在 8K 视频方面压缩效率的对比情况。AVS3 在压缩效率上已经提升超过了 30%，在 4K、1080P、720P 视频上有 20%多提升。

AVS3基准档次性能

- AVS3比H.265在4K上提升近30%

	Random Access AVS3 HPM2.4 vs. HM16.2				
	BDR-Y 码率节省	BDR-U 码率节省	BDR-V 码率节省	编码 复杂度	解码 复杂度
4K	-28.6%	-16.5%	-20.9%	799%	107%
1080P	-19.7%	-13.0%	-13.8%	841%	96%
720P	-17.8%	-12.3%	-11.3%	593%	84%
平均性能	-22.0%	-13.9%	-15.3%	736%	95%

21

上图是 AVS3 与 H265 在性能上的对比数据。在 4K 视频方面有 28%码率提升，1080P 有 20%左右提升。

AVS3主观性能

22

AVS3 主观效果展示图（1）

AVS3主观性能

23

AVS3 主观效果展示图（2）

上图是 AVS3 相比 AVS2 节省 25%码率情况下的主观性能质量对比，几乎是看不出差别的。

国际上首颗 8K@120P 芯片

上图的内容主要介绍海思发布了第一款 8K 120P 解码芯片，目前 8K 落地项目还很少，可以确定的是 2022 年北京冬奥会将会实现 8K 视频转播，最近已经发布要在 2022 年前开通 8K 频道的消息。

4. AVS 互联网生态推进组

成立背景

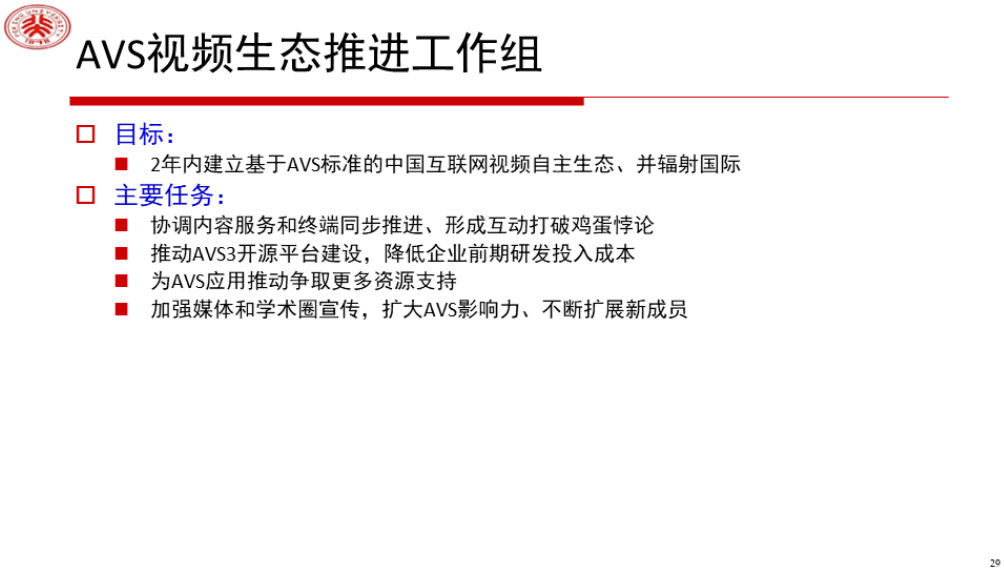
在近两年 ISO 和 ITU 标准授权政策不确定，中国企业在面临技术供应链安全挑战的背景下，为推进 AVS 在互联网应用推广，我们联合国内主要互联网厂商，于 2019 年 12 月 5 号在深圳成立了 AVS 互联网生态推进组。

AVS3 基准档次优势

AVS2 第一阶段基准档次有以下特点：

1. 编码效率高：比 H265 高近 30%。
2. 编码复杂度低：目前 AVS3 与 H265 解码复杂度相当，后面的竞争标准比 H.265 提升了很多。
3. 授权政策可控：AVS 技术只允许 AVS 专利池授权，AVS 专利池一直沿用 AV1 授权政策，对互联网软件免费，芯片硬件上每台设备一块钱。
4. 有相当的实现基础：8K 芯片已经问世。

AVS 视频生态推进工作组



AVS 视频生态推进组的目标是两年内建立基于 AVS 标准的互联网视频自主生态，同时向国外辐射。主要目标是希望打破内容与终端的鸡蛋悖论，另外我们会有开源的项目发布，主要面向 AVS3，降低企业研发成本，快速推进应用。

发起单位

- 鹏城实验室、华为、海思、腾讯、爱奇艺、优酷、阿里、快手、字节跳动、咪咕、蚂蚁金服、数码视讯、MTK、博冠、广东超高清创新中心、北京大学、北京大学深圳研究生院、电子科技大学

30

活动主要发起单位

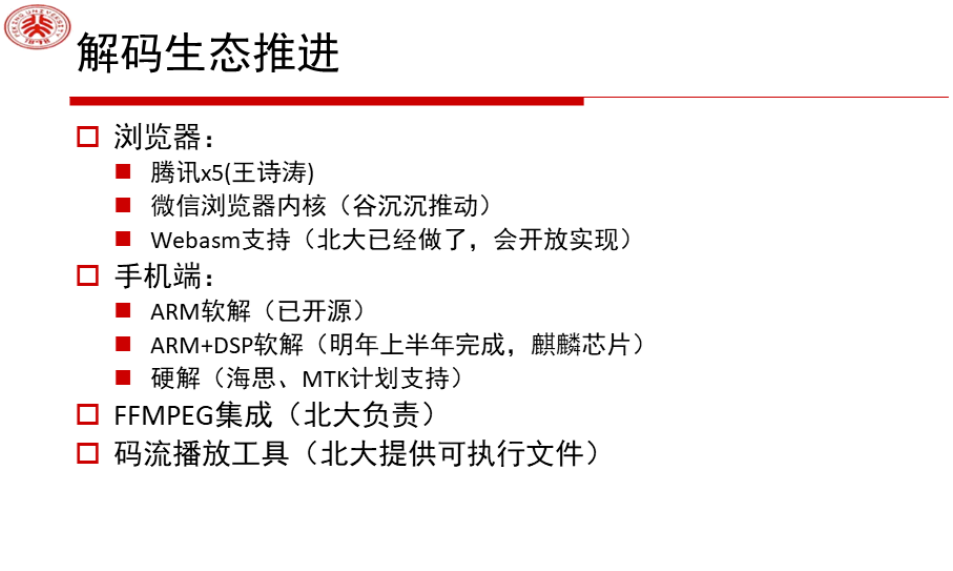
组织架构

- 指导委员会
  - 主任：高文
  - 委员：各公司产品线领导
- 组长：张伟民
  - 联合组长：谷晓军（华为）、朱子荣（腾讯视频）、王荣刚（北大深研院）
- AVS开源小组组长：王苦社
  - 联合组长：王振宇、罗法蕾

31

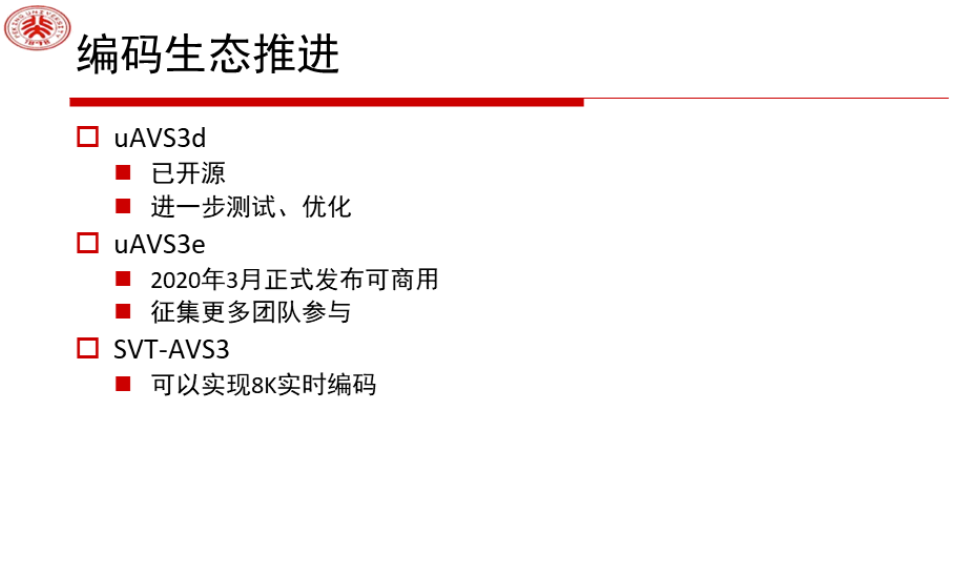
小组主要组织架构

解码生态推进



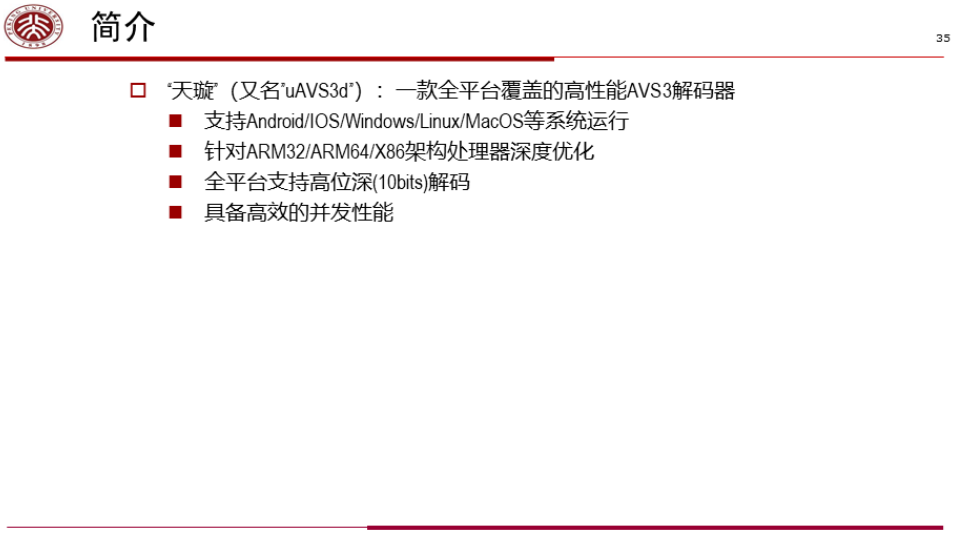
解码器生态目前已在部署，我们希望国内浏览器内核内置，国外浏览器通过 WebASM 方式内置，这个方案已经完成。在手机端软解已经开源，之后我会介绍。除了软解，我们第二阶段会实现 ARM+DSP 软解，这样可以大幅度降低解码功耗。第三阶段是芯片硬解阶段，海思、MTK 已经部署下一代芯片支持 AVS3 硬解，另外还有 FFMPEG 集成以及播放码流工具，我们会在网站开放提供。

编码生态推进



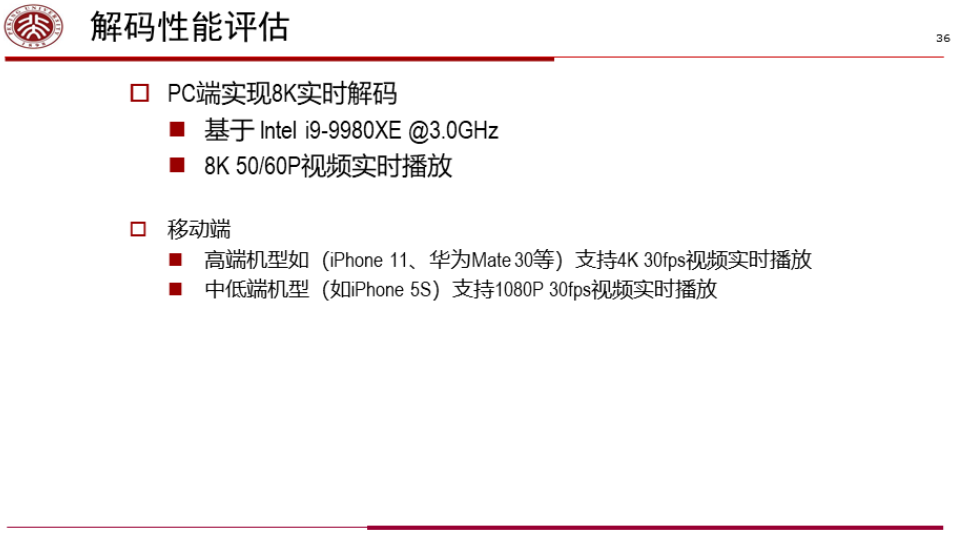
我们在编码生态推进阶段会实现两个编码器，一个由北大和 Intel 合作研发的 SVT-AVS3 编码器，已经可以实现 8K 实时编码；我们在研制另外一个不依赖于 SVT 架构的 AVS3 编码器叫 uAVS3e（“天枢”），预计会在 2020 年 3 月份正式发布。

5. AVS3 开源解码器“天璇”（uAVS3d）



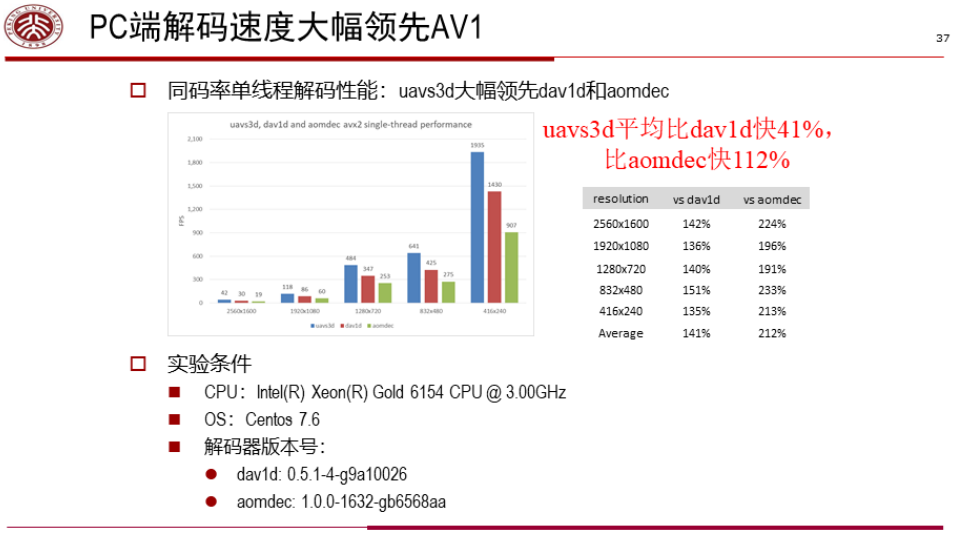
天璇（uAVS3d）是一款 PC、手机全覆盖的全平台解码器，同时支持 8 比特 10 比特解码。

解码性能评估



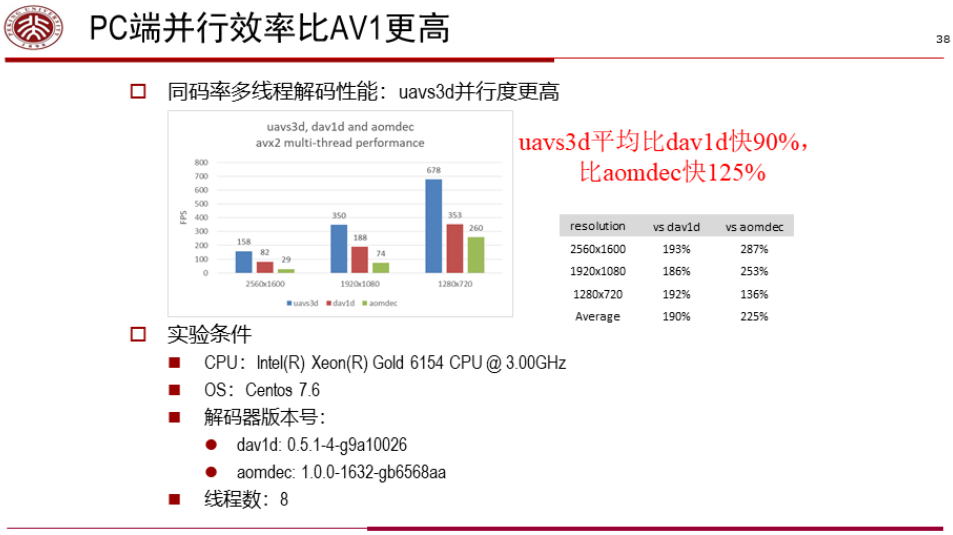
天璇的特点是解码速度快，在通用 Intel i9 上可做 8K 60P 视频实时解码，对移动端例如 iPhone11 单线程可做到 4K 30P 实时解码。对于中低端手机可实现实现实时 1080P 解码。

PC 端解码速度大幅领先 AV1



同时，我们也与其它标准做了对比，由于 VVC 标准还未定稿，我们无法与其做对比，但是 AV1 有我们对标的对象。我们和最快的 dav1d 对比，AVS3 在 PC 端单线程解码快 41%，比 aomdec 快 112%。

PC 端并行效率比 AV1 更高



AVS3 多线程解码比 AV1 快更多，uavs3d 平均比 dav1d 快 90%，比 aomdec 快 125%。

移动端解码性能测试

移动端解码性能测试

39

- 实验条件
  - CPU：海思麒麟990
  - 解码器：

	version	简介
uavs3d	1.0.0	AVS3开源解码器
libhevc	HEVCDEC_production Ver:05.00	Android系统内置hevc软件解码器
dav1d	0.5.1	VideoLAN开源AV1解码器

  - 解码配置：

	single-thread	multi-thread
uavs3d	-i input.avs3 -l 1	-t 8
libhevc	--input input.avs3 --share_display_buf 1 --num_frames 1000	--num_cores 8
dav1d	--cpumask neon --muxer null --filmgrain 0 -i input.ivf	--framethreads 8

上图是海思麒麟芯片团队对比 AV1 和 HEVC 移动端的解码性能。Android 使用原生 LibHEVC 解码器，同样从单线程多线程两个方面对比。

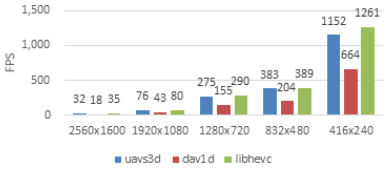
移动端性能 —— 单线程

移动端性能 – 单线程

40

- 同码率单线程解码性能（数据来自华为）
  - uavs3d大幅领先dav1d 80%
  - uavs3d与libhevc基本相当

uavs3d, dav1d and libhevc arm64 single-thread performance



uavs3d平均比dav1d快80%，比libhevc慢6%

resolution	vs dav1d	vs libhevc
2560x1600	182%	91%
1920x1080	177%	96%
1280x720	178%	95%
832x480	188%	99%
416x240	173%	91%
Average	180%	94%

单线程 AVS3 平均比 dav1d 快 80%，比 libhevc 慢 6%。两者性能并没有太大差别。

移动端性能 —— 多线程

移动端性能 – 多线程

41

- 同码率多线程解码性能（数据来自华为）
  - uavs3d大幅领先dav1d近80%
  - uavs3d与libhevc基本相当

resolution	seq	uavs3d	dav1d	libhevc	uavs3d_vs_dav1d	uavs3d_vs_libhevc
2560x1600	Traffic	81.10	48.46	75.17	167.4%	107.9%
	PeopleOnStreet	72.40	30.34	63.00	238.6%	114.9%
	Average	76.75	39.4	69.09	203.0%	111.4%
1920x1080	Kimono	140.80	68.94	138.43	204.2%	101.7%
	ParkScene	118.90	73.51	127.70	161.7%	93.1%
	Cactus	132.20	104.55	164.92	126.4%	80.2%
	BasketballDrive	161.90	79.85	155.03	202.8%	104.4%
	BQTerrace	131.00	89.58	155.20	146.2%	84.4%
	Average	136.96	83.29	148.26	168.3%	92.8%
All average					178.2%	98.1%

多线程解码比 dav1d 快 80%，比 libHEVC 慢 2%。以上是同码率对比，实际上同质量对比的话，AVS3 比 libHEVC 还要快。

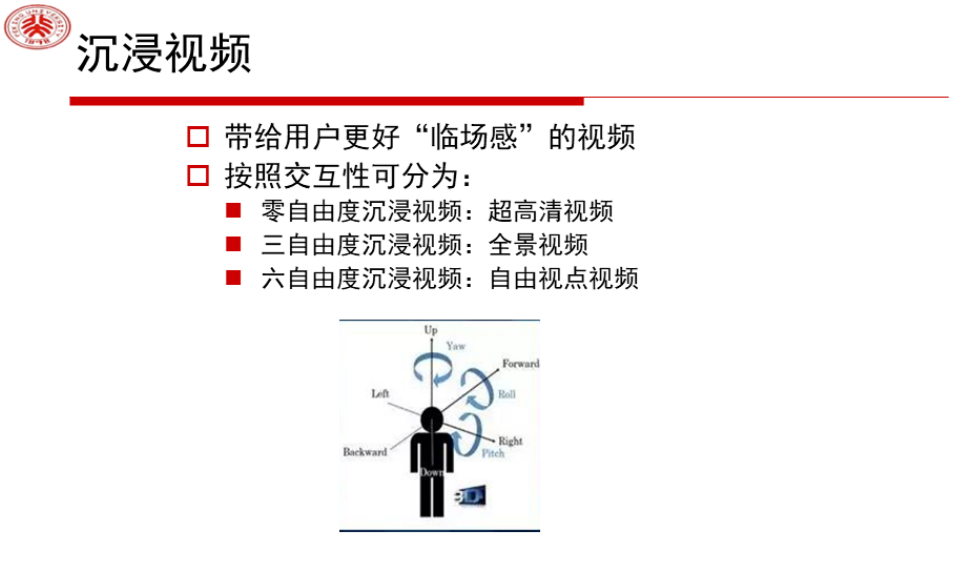
开源地址

42

- 鹏城汇智的开源项目地址：
  http://bggit.ihub.org.cn/p10472598/uavs3d.git
- 同步在启智开源平台OpenI上发布
- 主要贡献单位：
  - 北京大学深圳研究生院
  - 鹏城实验室
  - 广东博华超高清创新中心有限公司
- 鸣谢：麒麟芯片软件团队（王田、高艳征等）的交叉测试！
- 欢迎更多单位参与进一步完善、优化！

在鹏城实验室的开源平台 iHub 和 OpenI 上可以下载源码（可以免费商用，不强制二次开源）

6.AVS3 VR 编码标准

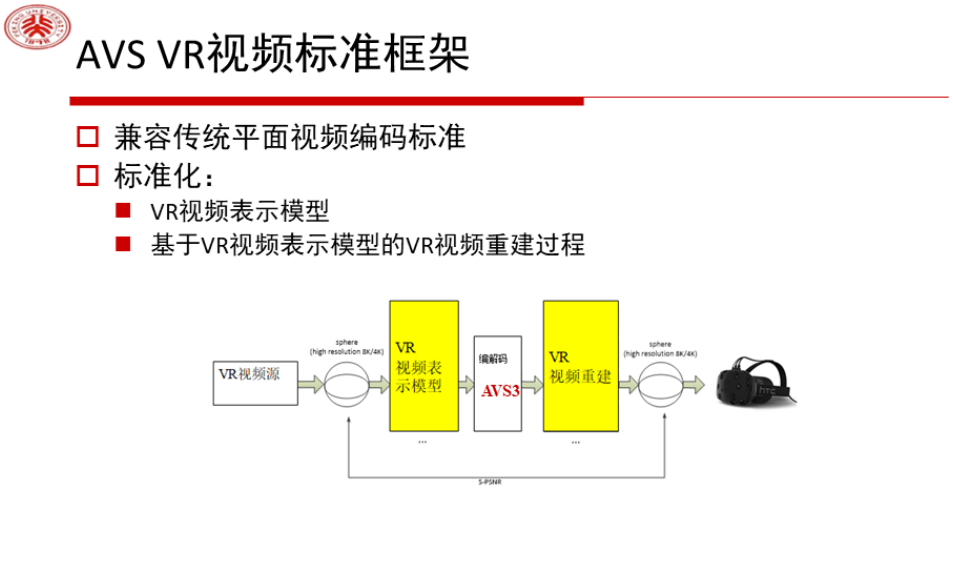


AVS3 在 VR 视频领域也做了相应布局。VR 又叫做沉浸视频，可以给用户更好的临场感。与传统视频相比，VR 视频具有交互性。按照交互性，VR 视频可分为零自由度沉浸视频，三自由度沉浸视频和六自由度沉浸视频三种。零自由度沉浸视频是单视点视频，和以前的视频相比唯一不同就是清晰度不同，我们称为超高清视频。三自由度沉浸视频是全景视频，支持用户在同一地点转头看到不同场景。更高级阶段是六自由度沉浸视频，允许用户自由移动，移动时可以看到如现场看到的场景。

AVS3 DoFVR：全景视频

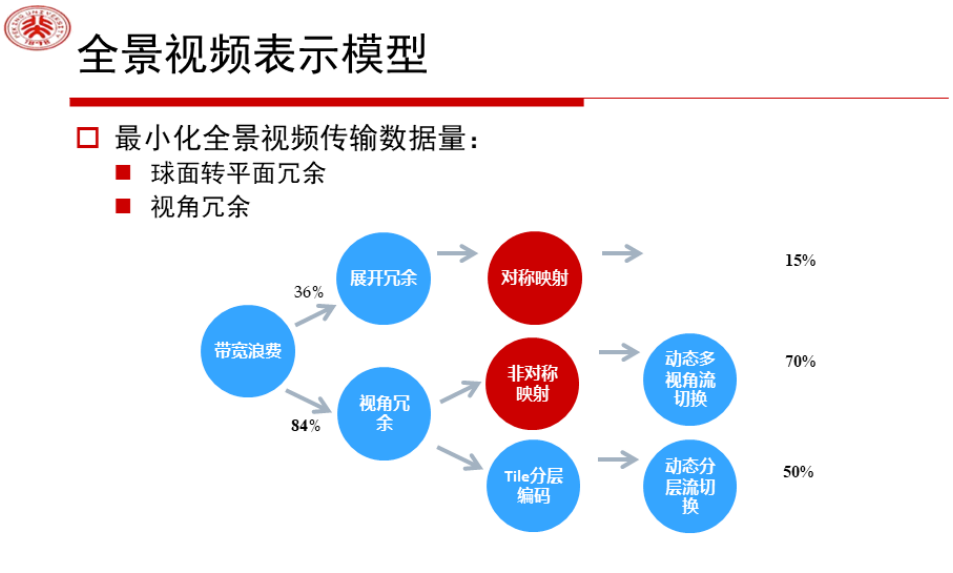
AVS2 已经可以满足超高清视频需求。针对三自由度我们也制定了相应标准，于 2019 年正式立项，称为《信息技术 虚拟现实内容表达 第二部分：视频》。

AVS3 VR 视频标准框架



上图展示了标准名称的由来，实际上 VR 视频与超高清视频有很多共同点，全景视频是一个球面视频，可以映射为平面视频，然后按照平面视频编码方式进行编码。这其中的核心内容就是映射方式，如何高效地将全景视频映射为平面视频是我们需要着重考虑的内容。

全景视频表示模型



上图清晰地分析了全景视频如何压缩、降低码率的过程。其中有两个冗余，一个是从平面映射为球面的展开时产生的展开冗余，另一个是视角冗余，人不能同时看到 360 度全方位图像，只能看到一定范围内容，所以产生视角冗余。其中，视角冗余相对较大，需要依靠动态码流切换来消除。动态码流方案有两种方案，一种是非对称映射方式，另一种是 Tile 分块编码方式。

采纳的表示模型

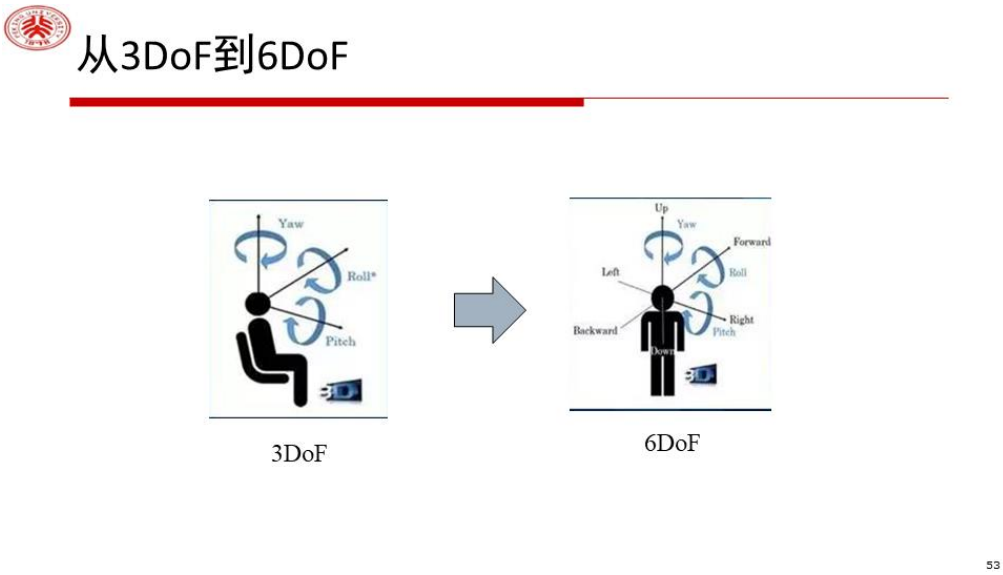
展开冗余用对称映射方式解决，在 AVS 中定义了一系列表示模型，其中一个用于消除展开冗余，可节省 20%到 30%码率。另一个是用于消除视角冗余的非矩形表示模型，可节省 70%的码率。

基于 AVS VR 映射模型的 VR 视频系统已经在阿里云上线。

应用 AVS VR 映射模型前后传输质量对比

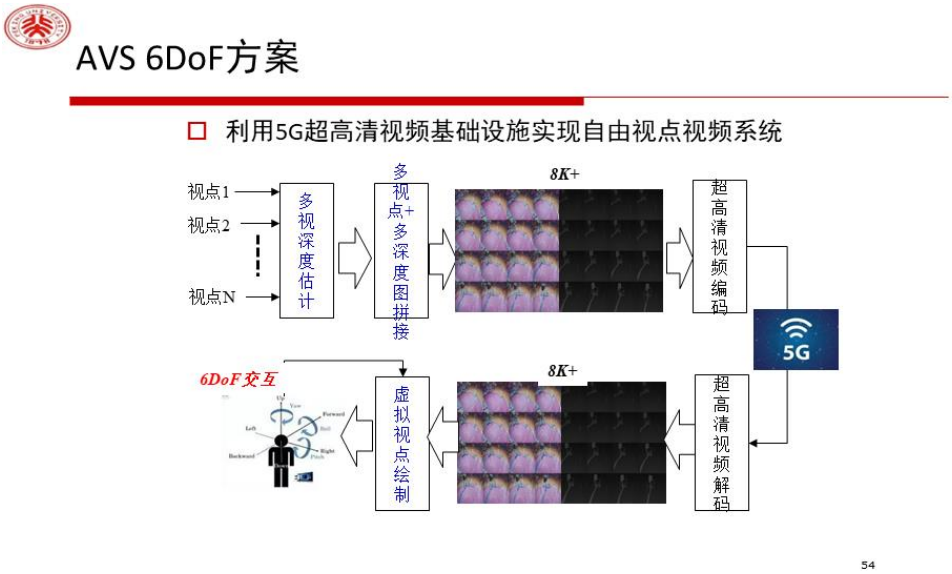
上图是阿里优酷给出的对比图，左边是传统模型，右边是使用 AVS VR 模型的显示效果，可以看到效果提升是很明显的。

7. AVS 6DoFVR：自由视点视频



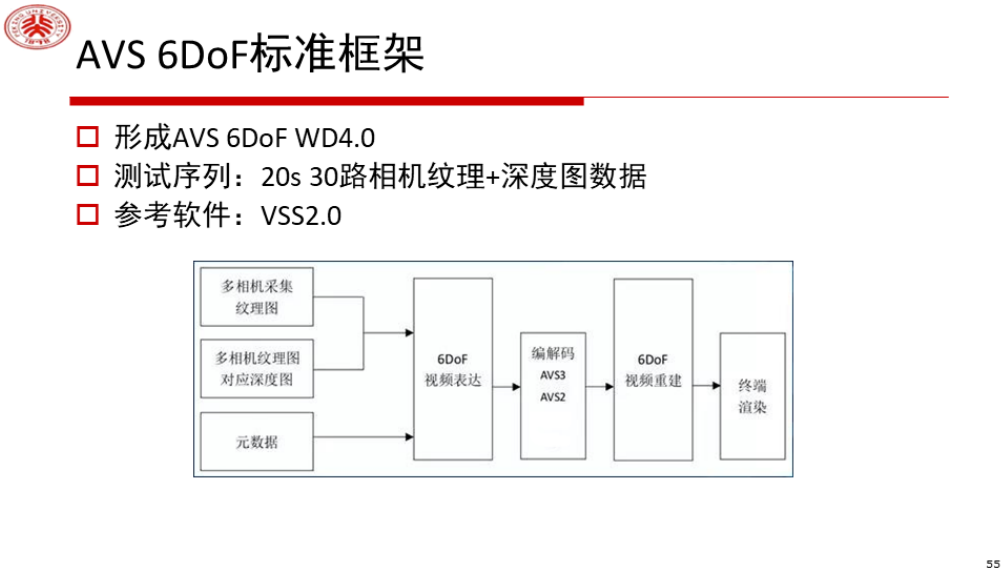
我们目前在做的工作是六自由度沉浸视频，它在三自由度的基础上提高了沉浸度。六自由度除了转动，还可以在三个轴上移动。

AVS 6DoF 方案



实现过程中有多种方案，其中也包括很多挑战。目前我们实现了一套可快速落地的方案——基于 5G 超高清基础设施方案，用超高清方法做编解码。我们通过多视点、多深度方案表达六自由度视频。在头端部署多个相机，基于多个相机做场景深度信息估计。有了多视点信息、场景深度信息之后，将这些信息拼成一个至少 8K 的大图，图越大视点信息越多。拼成大图后即可按照平面视频方式编码，编码完成后通过 5G 发送到解码端，解码端用户在实时位置根据场景深度信息、多视点信息实时绘制，这是目前的一套可落地的方案。

AVS 6DoF 标准框架



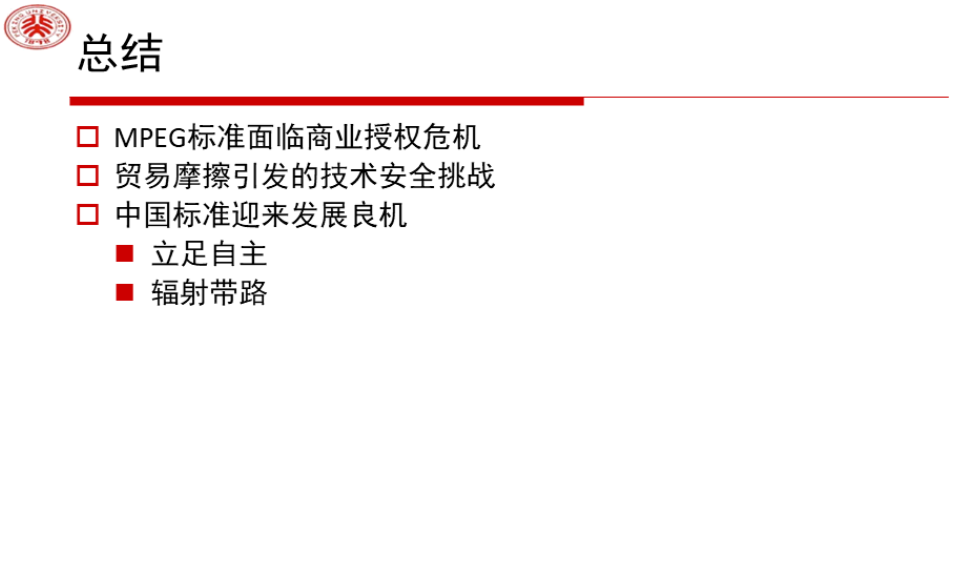
目前，已形成一套工作组草案（WD），其中的核心技术就是 DIBR 以及拼接方法。这是我们的重点工作，目前已形成一套基本方案。

自由视点视频原型

上图是效果展示。我们不再是单视角观看体育比赛，用户端可连续自主选择观看视角，比现场体验感更好。并且，此系统无需依赖头盔，可以在手机等设备上使用，大屏幕滑动体验更好，可以达到身临其境的效果。目前我们已经在推进这套系统在冬奥会测试赛上落地应用。

我们选择了滑雪、速滑、花样滑冰、冰球等作为重点项目使用 5G 网络在“科技冬奥”上展示。

8.总结



我们如今面临的既是机遇也是挑战，最近一系列事件表明技术也是有国界的，中国标准迎来了发展良机，我们需要做到立足自主，用开放的心态辐射国际。

H.265 在花椒直播中的应用与优化

本文由花椒直播技术总监唐赓在 LiveVideoStack 线上交流分享中的演讲内容整理而成。在分享中，唐赓详细介绍了花椒直播在两年多的 H.265 线上实践过程中所积累的一些经验，以及一些成本、效果优化方案。

唐赓　LiveVideoStack 线上分享　2019-04-15

大家好，我是来自花椒直播的唐赓。接下来我将借本次分享与大家一起探索 H.265 在直播中的应用及方案优化。我从 2015 年加入花椒直播开始便一直分管花椒在移动端领域的基础技术与创新技术，H.265 也是其中一个很重要的部分。

1.H.265 相对于 H.264 的主要改进

首先简单介绍一下相对于 H.264，H.265 的主要改进：

- H.265 使用 CU/PU/TU 代替 H.264 上的宏块，使其拥有更多优化选择，提升画质的同时明显降低码率；
- 取消了隔行扫描模式从而进一步简化设计；
- 支持 35 种模式的帧内预测，相对于 H.264 的 9 种有明显的提升；
- 增加 TMVP 时域运动矢量预测功能以提高画质与压缩率；
- 增加了 TILE 与 WPP 等并行工具集以提高编码速度；
- 在 H.264 的 Deblock 基础上增加了可提升画质的 SAO 滤波；

观察上图左侧图我们可以看到，H.264 的宏块大小十分密集且其中包含许多冗余部分，如右下角的暗处画面浪费如此多的宏块对其进行描述，无疑造成编码资源的浪费与带宽利用的低效；H.265 主要面向 4K 甚至 8K 等高分辨率应用场景，高分辨率意味着像素密度更大，其颜色变化趋于平缓。如果仍使用传统 H.264 的宏块策略借助平均而密集的宏块描述画面，显然会造成带宽与码流资源的浪费。

于是 H.265 引入了自适应宏块划分，按照画面分析得出的数据找到更高效的宏块布局方式，在画面细节较少的部分使用较大面积的宏块从而精准模拟画面中物体元素的轮廓，更加高效地利用现有码流与带宽资源。有很多工具可以显示 H.265 码流里的分块划分方案，也有开源版本，可在 github 上找到。此工具可以显示经过自适应宏块划分之后的画面（PPT 上只是示意图，不是实际划分方案）。

得益于以上提及的各种优化措施以及其它众多改进措施，H.265 比 H.264 可节省 30%～50%的带宽消耗。但在实际应用当中 H.265 的终端设备普及程度远不如 H.264，接下来我为大家分享的绝大多数技术优化方案基本上都是为了弥补 H.265 在终端设备普及率上的劣势。

H.265 在花椒直播上的画质表现可以说很直观地显示了 H.265 的技术优势：上图左侧是一个基于 H.264 1500k 进行的直播画面，右侧则是基于 H.265 1100k，似乎两个画面并没有很大区别……

但当我们放大画面之后就可以看到左侧女主播的头发细节相对于右侧女主播的更加模糊，这还是在左侧画面的码率更高的基础之上，可以说 H.265 对于画面质量的提升显而易见。

2.移动端：软解 vs 硬解

从 2017 年初决定上线 H.265 以来，花椒直播一直希望先找到移动端的解码方案。我们在 2016 年底通过测试发现 FFmpeg 的性能已经足够出色，在当时的 iPhone 6S 上仅需 30%左右的双核 CPU 性能就可完成对 H.265 文件的流畅软解，这无疑是令人振奋的。所以 2017 年上半年开始花椒移动端（iOS 和安卓）的播放器陆续开始加入对 H.265 的软解支持，这个阶段同时也开始提供 H.265 的软编支持，因为硬解方案是在 iOS 11 正式推出之后才陆续被支持；Android 平台则是从 5.0 API 24 开始陆续支持 H.265，之后才逐步有厂商开始支持，由于其碎片化特性，硬编支持的普及程度至今还远逊于 iOS。

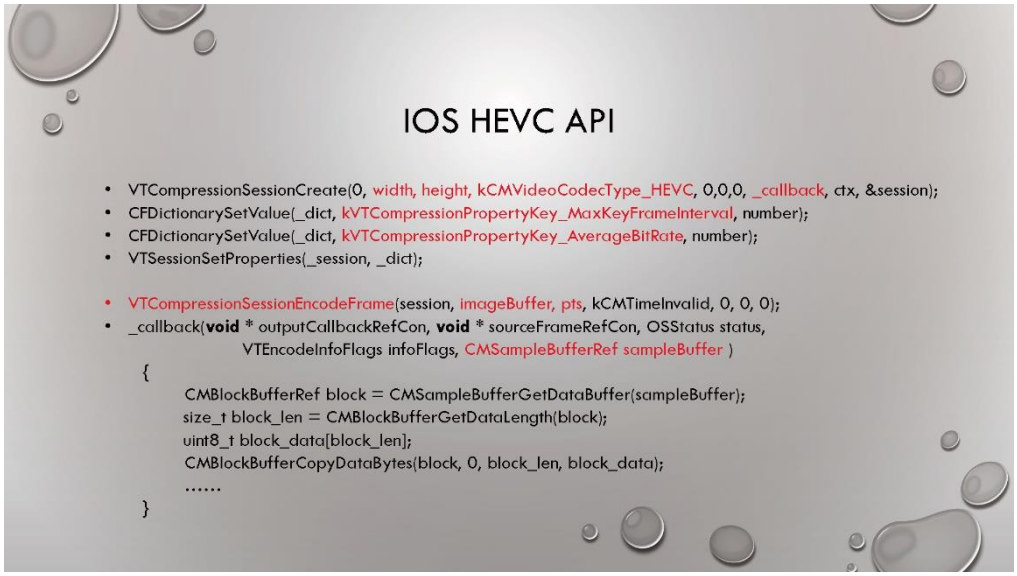
3.移动端：软编 vs 硬编

花椒从 2017 年上半年加入 H.265 的解码支持后，同时上线了 H.265 软编版本（显然没有编码器生成 H.265 内容解码器也就没什么存在价值了）；2017 年初，在对 H.265 软编进行灰度测试时我们发现了一些问题，例如尽管上线之初 H.265 软编无论是 CPU 消耗之少还是编码效果之佳都出乎我们的意料，但随着直播编码时间的延长，手机发热的问题也逐渐凸显，过热的手机温度会导致 CPU 主动降频，算力下降而算力需求不变，软编的 CPU 消耗占比显著提升，直接造成直播画面的卡顿与丢帧等问题。当时我们收到了一些用户关于此问题的负面反馈，不得已只能停止灰度测试。终于在 2017 年 Q4，苹果 WWDC 2017 上介绍过的 iOS11 正式推出，提供了对 H.265 硬件编码的支持，我们认为时机已经成熟于是立即投入开发了 iOS 端的硬件编码功能，并在半个月之后上线正式版；与此同时我们也在逐步增加 Android 平台的 H.265 硬编支持，包括图中标注的几十款手机我们都能够良好兼容其硬件编码功能。

4.各个平台的 H.265 实现方案

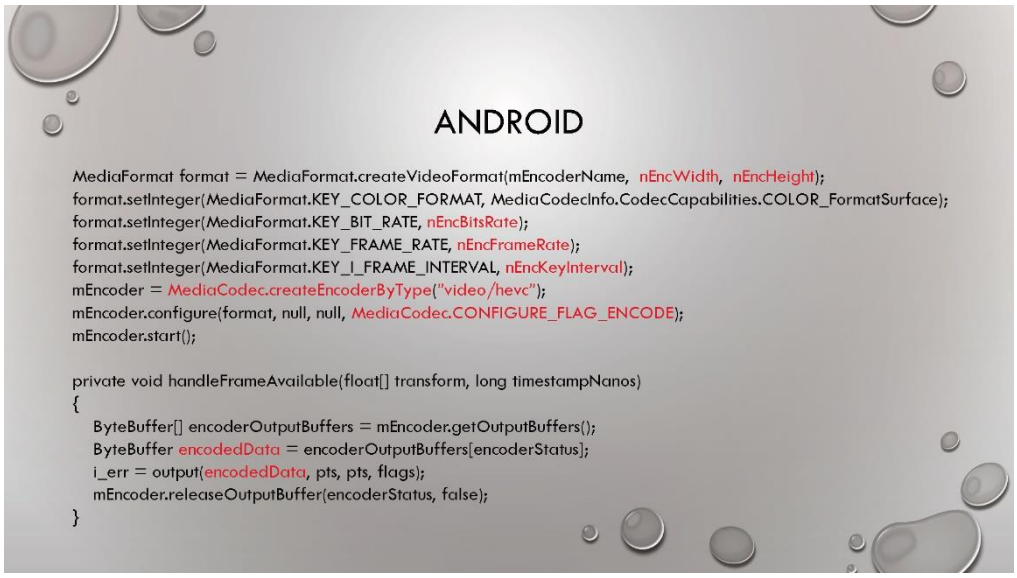
下面按照平台划分简介 H.265 的具体实现方案。iOS 是在它的 VideoToolBox API 中提供了 HEVC 编码支持， 此 API 其实很早就已经包含 H.265 支持，只是一直是苹果私有 API，用于苹果自家的 FaceTime App 中，iOS11 后苹果将其开放给所有开发者；Android 平台则通过 MediaCodec 框架实现 H.265，PC 端则是基于 FFMPEG 或 NVIDIA 生成 H.265 内容。服务端实现 H.265 的方案有很多，如 FFMPEG、NVIVDIA 与英特尔以及其它一些公司的定制硬件等。

4.1iOS HEVC API



上图代码展示了 VideoToolBox 的使用流程，其中红色字体代表关键参数：首先创建一个 CompressionSession 并选定 CodecType（图中选择 HEVC）以及后续的 CallBack 参数，其次通过字典设置 MaxKeyFrameInterval 与 AverageBitRate，这时视频的 GOP 与帧率就设置好了；接下来我们调用 VTCompressionSessionEncodeFrame，其中的最关键参数是 ImageBuffer----每一帧图像数据采集之后都会由 ImageBuffer 承载，这里我们需要设置 PTS 也就是帧的显示时间；设置完成之后我们调用 VTCompressionSessionEncodeFrame 并在其内部编码完毕之后获得回调，所得 CMSampleBufferRef 就包含编码压缩过的数据；最终我们通过 CMBlockBufferRef block=CMSampleBufferGetDataBuffer(sampleBuffer) 得到其 block 长度就可以拷贝得到数据，包含已经完成编码的码流。最后需要我们做的就是将此码流经过一定额外处理之后推送至服务端，整个流程结束。

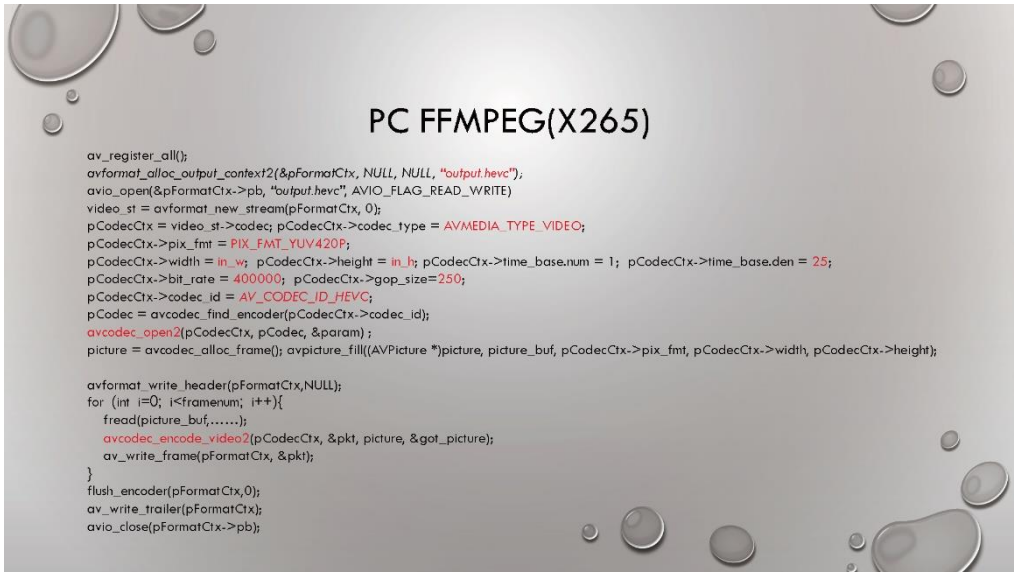
4.2Android



Android 平台的 API 的使用流程类似，MediaFormat 用于封装宽度、高度、编码码率、缓冲格式、帧率、关键帧等参数，随后在 mEncoder = MediaCodec.createEncoderByType 选择“video/hevc”作为编码类型；接下来 mEncoder.configure 通知 Encoder 创建用于编码器 MediaCodec.CONFIGURE_FLAG_ENCODE，之后调用 mEncoder.start()即可开始编码。相机获得的输入在 Texture 里，输入之后， handleFrameAvailable 获得 Encode 结果，getOutputBuffers 可获取 ByteBuffer 数组，数组里头，实际上就是一个个的 ByteBuffer 数据，直接就把这些数据输出就 OK 了，所以这个也是一个比较简单的一个过程。

4.3PC FFmpeg

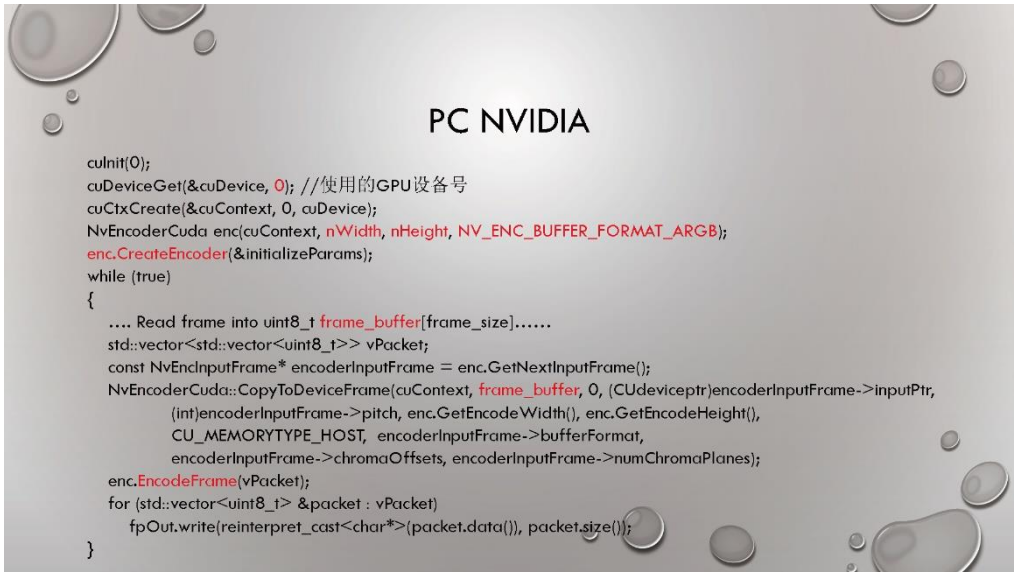
PC 端基于 X265 的 FFMPEG 拥有上图展示的部分主要优势：FFMPEG 的开源特性使其使用成本明显降低，同时也可达到令人满意的生成质量；除此之外，其代码升级较为及时，同时可灵活定制的特性也使其应用更加广泛。当然它也有一些不足与缺陷，如 CPU 处理速度较慢等。



上图代码展示了 PC FFMPEG（X265）的大致流程，仅展示了较为重要的几行，实际可运行的代码可以看到在代码量上相对于 iOS 与 Android 版本有明显提升。首先，avformat_alloc_output_context2 中给定了一个“output.hevc”作为输出，随后再将“output.hevc”给定 avio_open 用以根据扩展名判断应当使用什么编码器；video_st = avformat_new_stream 会创建相应的视频流；接下来的 pCodecCtx 除了指定 Codec（pCodecCtx = videost->codec;）与 Media 类型（pCodecCtx->codectype = AVMEDIA_TYPE_VIDEO）,也会指定如输入像素格式、宽度、高度、帧率、码率、GOP 大小等参数，同时指定 Codec ID 为 HEVC；接下来的 pCodec = avcodec_find_encoder(pCodecCtx->codecid)就会寻找相应的 codec ID 并初始化对应编码器从而获取 pCodec, pCodec = avcodec_find_encoder(pCodecCtx->codecid)，最后 avcodec_open2(pCodecCtx, pCodec, &param) 完成初始化；随后的 picture = avcode_calloc_frame(); 获取到此帧的数据结构之后会填充 Picture 也就是 avpicture_fill，之后通过 av_write_frame 输出为指定容器的文件。这里只是大致描述 FFMPEG（x265）编码的一个过程。

4.4PC NVIDIA

PC 端的 NVIDIA SDK 具有硬件编码性能强悍但价格十分昂贵的特点；家用的游戏卡有限制，只能并发进行两路编码任务；相同画质下编码输出的码率偏高；API 非常简洁，实际使用中其代码量也明显减少。



以 CUDA 版本为例，首先 cuDeviceGet(&cuDevice, 0);用以获取所使用的 GPU 设备，随后的 cuCtxCreate(&cuContext, 0, cuDevice);创建上下文；完成上下文创建之后，NvEncoderCuda 会创建相应对象并输入宽高以及编码格式，编码器初始化完成；接下来就是数据读取、调用、写入、输出等常规流程的循环。

4.5SERVER FPGA/ASIC

除了上述方案，还有一些基于硬件定制的解决方案如 FPGA、ASIC 定制硬件等。通常他们都提供 ffmpeg 的插件，所以开发过程和上面 ffmpeg 的开发过程类似。需要注意的是由于现在相关领域的厂商已经不少，我们需要详细对比不同的解决方案。FPGA 普遍能达到 60～240FPS 1080P 的性能，支持 B FRAME、SCENE CHANGE DECTECTION、自适应量化等算法；同时，Scale 运算的性能也是选择方案的主要参考因素之一。有的解决方案板子上的逻辑电路主要用于编码，Scale 能使用的运算单元不够多导致需要改变分辨率的场合下性能稍差，而如果 Scale 使用过多运算单元那么编码能使用的运算单元也会偏少，此时如何对有限运算资源进行有效分配成为摆在厂商面前一个需要仔细权衡的问题，从客户的角度来说应当根据自己产品使用场景重点考核一下各家的解决方案，进行充分的测试。

5.H.265 高清低码方案

接着再说说我们在 H.265 的高清低码解决方案方面的探索，我们有如下思路

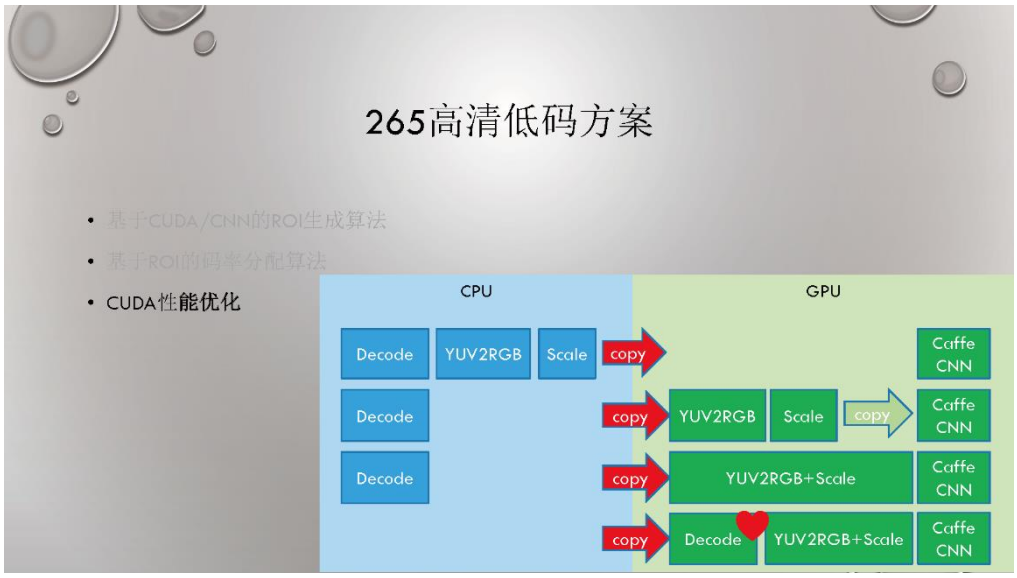
其原理是基于 CUDA/CNN 的 ROI 生成算法，通过我们自己训练的一套卷积神经网络分析生成用户兴趣热点区域并基于此兴趣热点科学分配码率。

如上图输入主播直播画面，输出的是前景轮廓，这就是我们计算出来的用户在观看直播画面时最感兴趣的视觉重点区域。将此结果输出给编码器，编码器就可根据用户视觉重点区域的分布情况合理分配码率，提高前景区域的码率，降低背景码率，从而确保观众能够在视觉核心区域获得最好的观影质量。

上图展示了我们此项探索的成果，尽管画面整体看上去似乎没有多大区别，但如果放大非前景区域画面可以观察到细微的画质损失，而前景区域没有不同，达到保证用户的视觉焦点上画质不降低而整体码率降低的效果。

再简单介绍一下我们的一个 Cuda 优化细节，大家在做类似功能的时候可以借鉴一下类似想法。早期技术 demo 阶段我们主要使用 OpenCV 实现 ROI 前处理，包括 Scale、色彩空间变换等，发现性能远达不到上线要求，急需改进。

进一步优化 CUDA 性能是我们接下来需要重点解决的关键问题



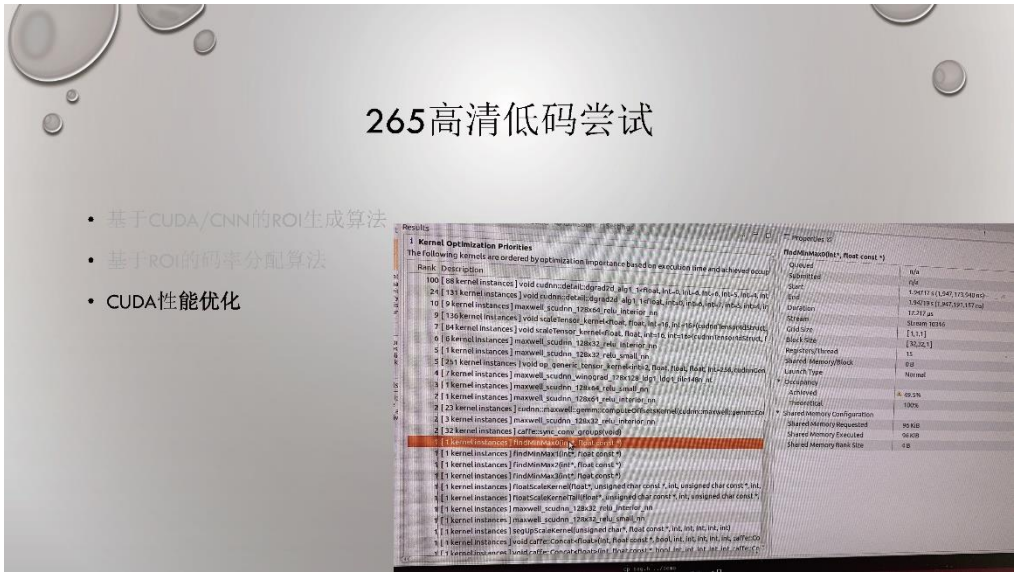
分析发现，完全依赖 CPU 进行计算的 OpenCV 确实存在很多问题：首先 Decode 输出的数据经过 YUVtoRGB 转换后进行 Scale 处理（因为 CNN 对输入数据的大小有限制），这一步非常消耗 CPU 时间；之后数据由 CPU 主内存拷贝至 GPU 显存交给 Caffe CNN 处理，这一步也消耗大量总线传输时间。改进的第一个版本是由 GPU 来完成色彩空间转换与 Scale， 完成处理的数据会进行一次显卡上的数据复制，复制到 Caffe CNN 输入缓冲区，交给 Caffe CNN 处理。

继续研究我们发现，Caffe CNN 本身也是由 CUDA 的 API 编写，可以获得其设备输入缓冲区的指针。于是我们将 YUVtoRGB 与 Scale 两项转换整合成一段代码，并使其输出结果直接写入 Caffe CNN 的输入缓冲区从而减少 GPU 上的一个内存拷贝，通过上述改进获得了明显的处理速度提升，达到了上线要求。

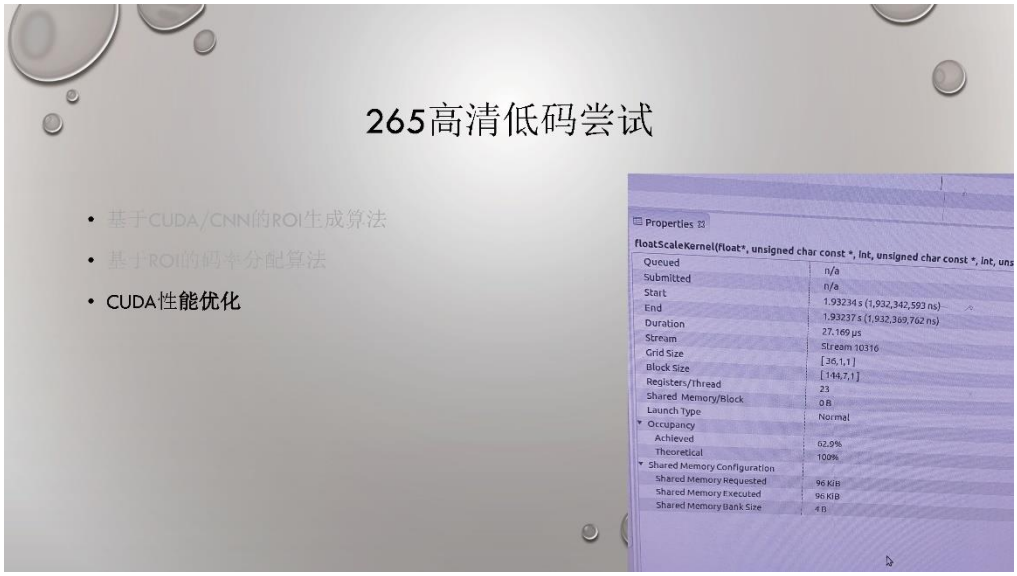
我们期待的优化结果是将尽可能多的处理流程整合并转移至 GPU，如果将 Decode 功能也转移至 GPU，相信会带来质的性能提升。

6.H.265 高清低码尝试

实际开发中我们使用了 NVIDIA VISUAL PROFILER 来进行优化，接下来介绍一下优化过程。



用 Profiler 运行过程序后，生成一个类似上图的视图，列举了许多 kernel 的性能数据，因为我们用了 Caffe，可以看到其中大部分是 Caffe 的函数，也可以找到自己开发的 kernel 函数。通常情况下 YUVtoRGB 会得到 byte 类型的输出，而 Caffe CNN 则要求需要输入浮点数，因此我们将 YUVtoRGB（含 float 转换）与 scale 全部整合至一个 kernel 中。



上图中的 Occupancy->Achieved 利用率为 62.9%，可以说基本符合我们的预期与需求。

7.WEB 端 H.265 播放解决方案

之前说的 H.265 解决方案主要运行在移动端，下面再说说我们在 WEB 端上的尝试。我们以 FLASH 的 H.264 解码器消耗单核 25%的能力作为基准，进行了如下探索。

首先调研 WEB 端 H.265 播放解决方案的原生支持情况，如上图所示，红色代表不支持，黄绿色代表有条件的部分支持，其中的绿色所在 iOS 端的 Safari 浏览器，对 WEB 端 H.265 播放解决方案的支持情况良好。

参考网上的数据，Flash 方案使用 ACTION SCRIPT 加载 FlashCC，所生成的 SWC 可达到 1080P 10FPS；而 JS 方案的相关代码 LIBDE265.JS 可以在网络上找到，使用 LIBDE265.JS 播放 480P 可以维持帧率在 40FPS 以上，CPU 使用率大约为 110%到 140%。

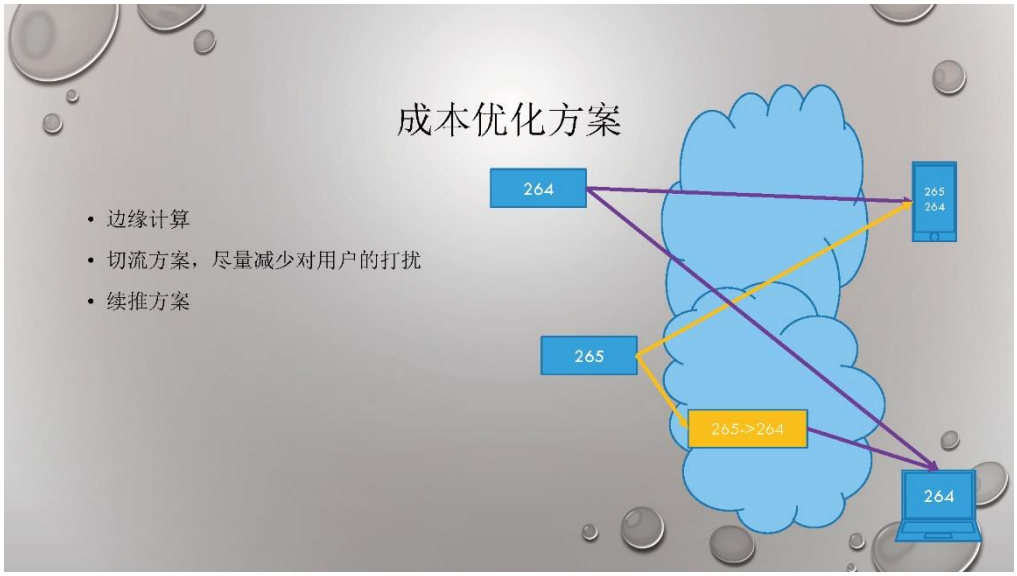
继续测试 WASM 解码器，CPU 单核占用为 50%～60%，解码 720P 直播流可达到 15FPS 的帧率； 厂商宣称 1080P 可以达到 40FPS 的处理速度。因为用到了 WASM，浏览器需要至少不低于上图列举的版本。

综合以上各种评测，我们选择了 WASM 方案，目前我们 WASM 版本的解决方案已经正式上线运行，效果良好。

8.成本考虑与优化方案

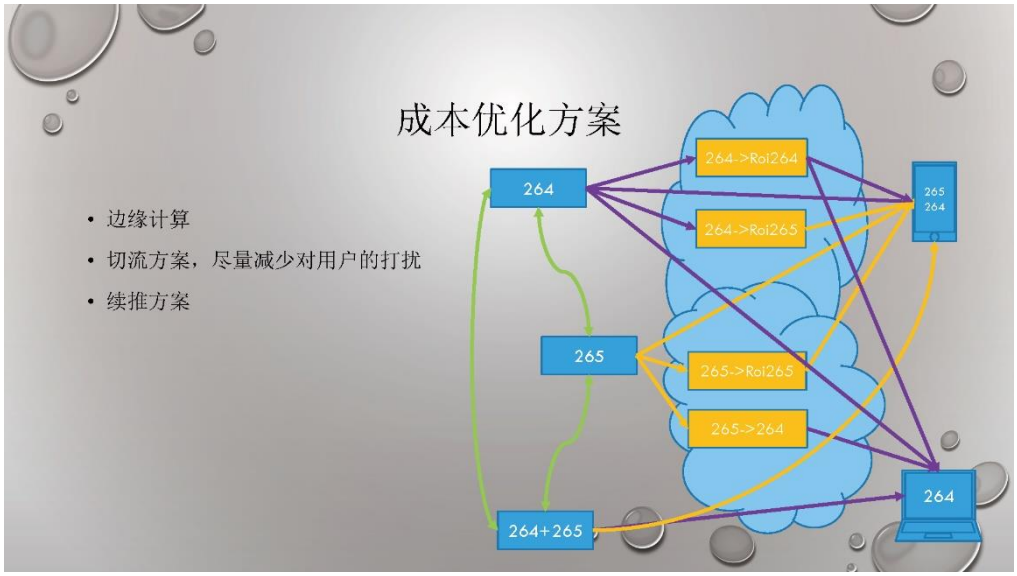
实施 H.265 的最大动机是它能显著节省流量成本。理论上 H.265 使用越多流量成本节省越多，但另一方面来看，企业付出的成本还包括采购或自研编解码器所带来的转码成本以及购买转码服务器、GPU、FPGA/ASIC 在内的硬件成本，这些都是需要企业综合考虑投入产出比、制定合理的流量分配方案。从节省成本的角度来看，自研的采购成本会低许多；同时硬件也不是性能越强越好，而是需要结合应用场景与产品特点综合考量选择最合适的硬件组合以更高效地利用资金并收获理想效果。

接下来我们将着手探索如何优化成本。



传统的成本优化方案如上图展示的那样，引入 265 上行推流，给支持 265 的终端直接提供 265 码流，但为了保证 Web/H5 等只支持 264 的终端正常观看，需要提供对应的 264 码流（服务端实时进行 265 到 264 转码）。

降低成本的第二步，引入高清低码方案，在服务端进行 264/265 的高清低码转码。将 H.264 转为 ROI H.264 或 ROI H.265 可明显节省成本。



如果 PC 推流端性能足够强大，例如配置了专业显卡，可以执行更多的边缘计算任务，可以尝试直接输出 H.264 与 H.265 两路流，从而不需要在云端转码。

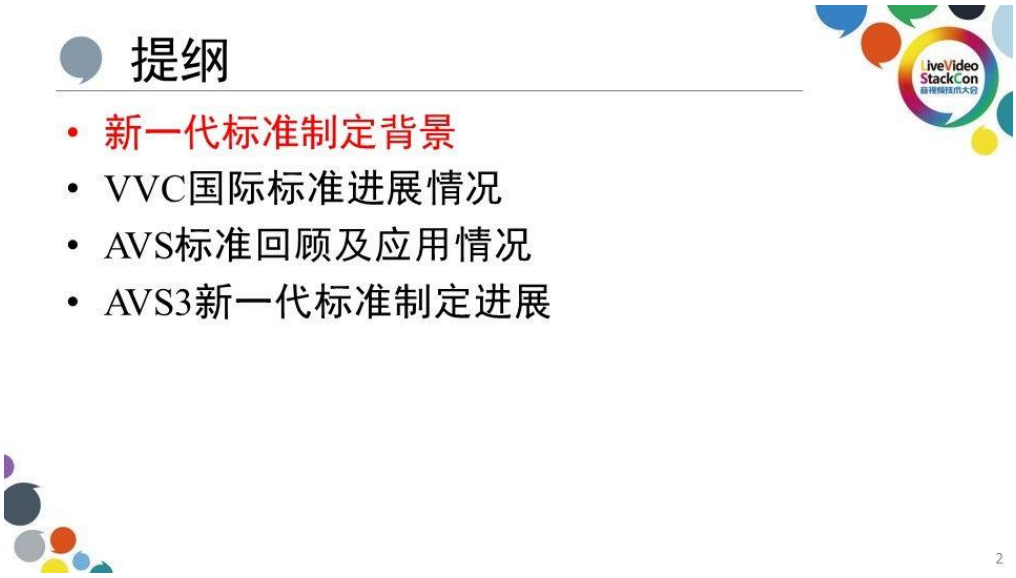
更激进的方案是在以上几种上行推流状态中动态切换。通过实时统计一路流的拉流数目，选择性价比最高的方案。当然此策略需要播放器业务端的良好支持，流切换一定需要科学高效，避免卡顿。同时切换也不能过于频繁，以避免对用户观看体验造成不良影响。

新一代视频编码标准：VVC、AVS3

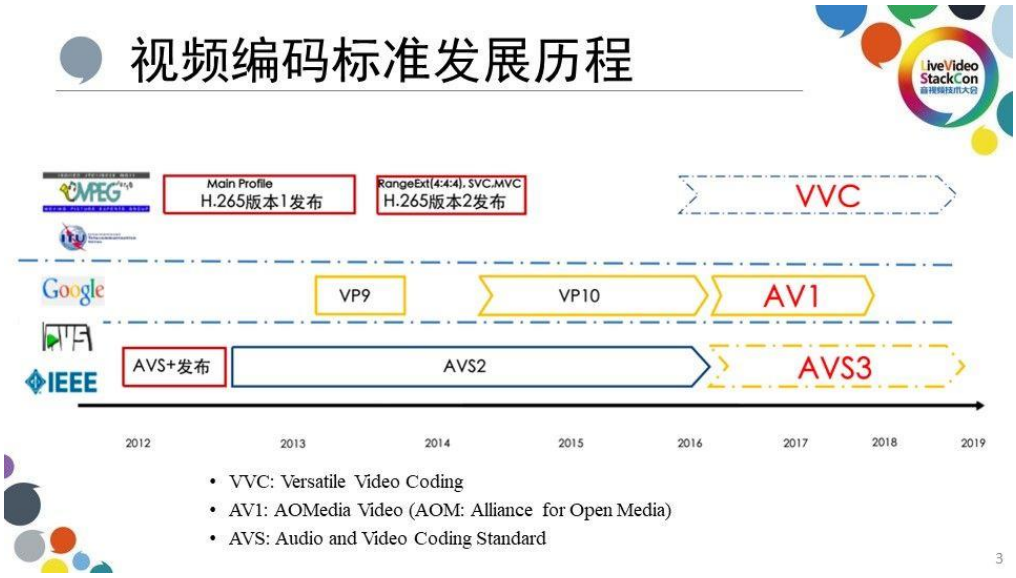
本文来自北京大学信息科学技术学院教授马思伟在 LiveVideoStackCon 2018 大会上的演讲，详细介绍了最新一代 VVC 和 AVS3 视频编码标准进展，包括关键技术特色以及未来的应用展望。

马思伟　LiveVideoStackCon 2018 北京　2019-01-10

大家好，我是来自北京大学的马思伟。本次我将从以下四个方面为大家分享有关最新一代 VVC 与 AVS3 视频编码标准的进展及关键技术特色等内容，希望可以为大家带来有价值的帮助。

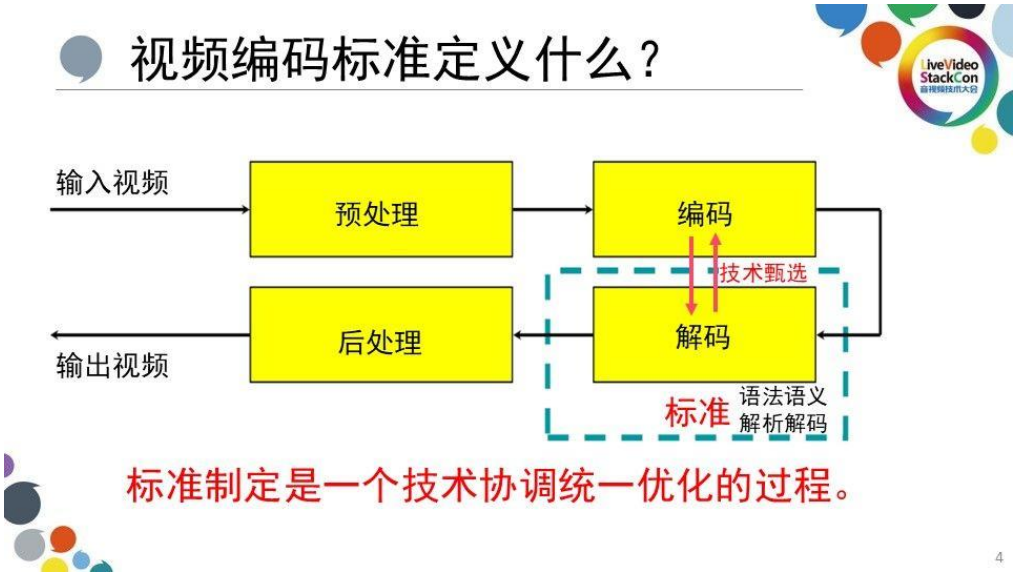


1. 视频编码标准发展历程



上图展示了视频编码国际标准的发展历程，我们可以看到目前主要是以 VVC、AV1 与 AVS3 为主流发展趋势。

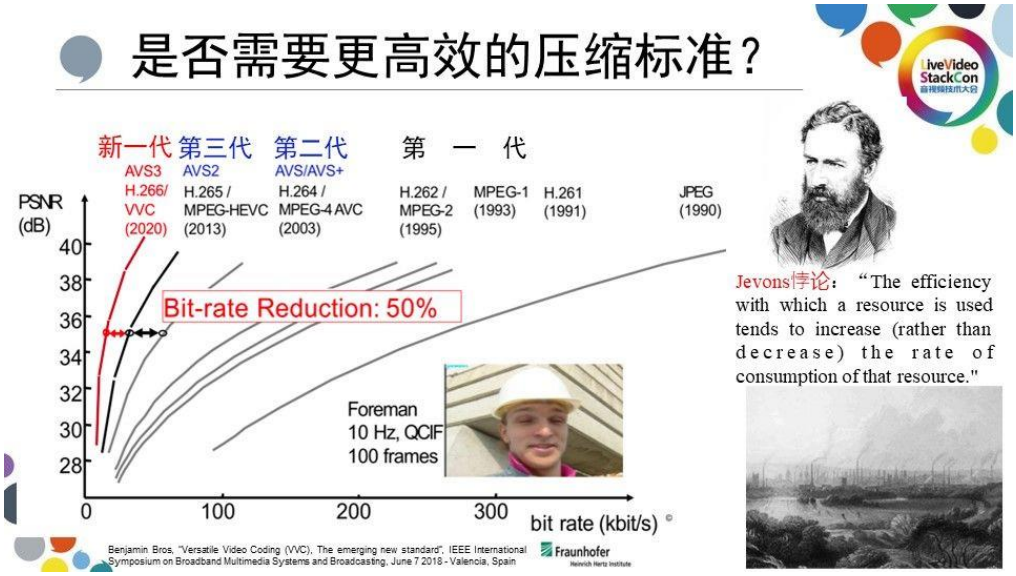
1.1 视频编码标准的定义



我们可以看到上图展示的视频编码流程，视频数据输入至编码系统首先会进行预处理工作，预处理后进行编码。编码完成后的数据会传输至解码模块并在完成解码之后进行预处理，最后输出目标视频。这里需要注意的是我们对视频编码标准的定义是对其中解码部分的定义，也就是定义码流的语法语义和解析解码过程。

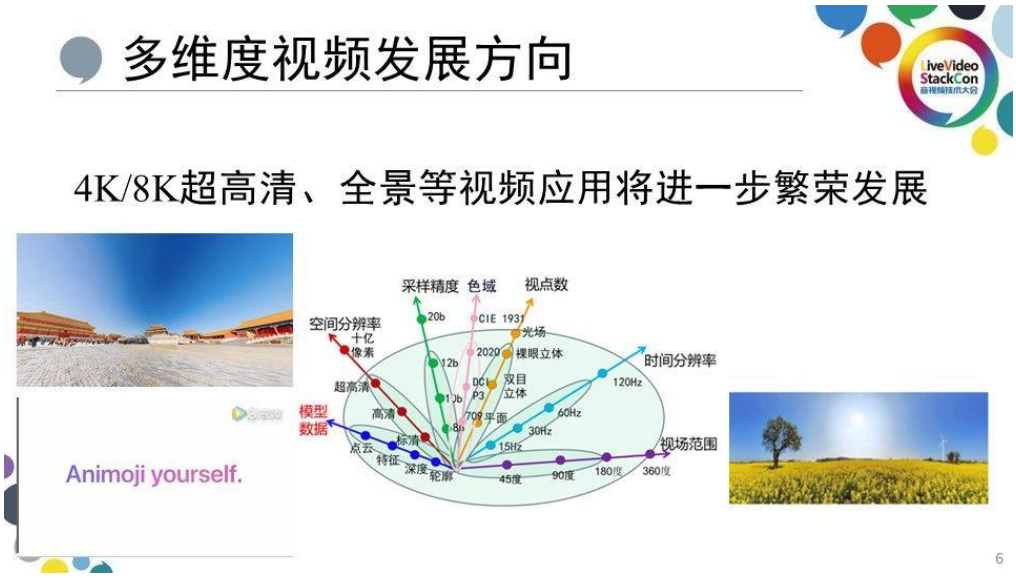
我们需要在制定标准的过程中通过技术筛选择优选择技术加入标准中从而避免解码过程的冗杂与低效，降低解码过程的复杂程度。也就是说可以在编码流程等其他环节完成的工作就不应当被加入解码流程中，如滤波等作为后处理可提高主客观性能水平。标准制定是一个技术协调统一优化的过程，无论是国际标准的制定还是我们以学术为目的的讨论都会引入一种技术竞争机制，选择最好的技术放入标准之中。

1.2 高效压缩标准的意义



那么现在我们需要更加高效地压缩标准吗？答案是毋庸置疑的。按照提升规律，编码效率每提升 50%以上就会成为新一代标准，如早前的 H.261、HAV1 等其性能基本持平所以我们将其称为第一代，而像 AVS 和 H.264 等比前一代提升 50%就可称其为第二代，至于 AVS2、H.265 等相对于第二代的提升达到 50%我们就将其称为第三代。现在我们正在探索的 AV3/VVC/H.266 则为最新一代的压缩标准。我们需要更高效压缩标准原因可以通过 Jevons 悖论来解释：业界对节约煤炭研究不会降低煤炭需求，反而会因为提升了煤炭使用效率而加大对煤炭的需求。我们将这个理论过渡到压缩标准上来说，虽然我们尝试通过更好的压缩标准提高压缩效率降低带宽压力，实际上压缩效率的提高会进一步刺激视频用的需求不断增长，整体来看就是更高效的压缩标准推动了视频行业的持续发展。

1.3 多位度视频发展方向



我相信在未来 4K/8K 超高清、全景视频等先进视频应用会得到进一步发展。随着视频应用的不断演进，更高效的视频编码解决方案也会不断被提出。图中展示的全景视频、3D 面部捕捉等都会帮助我们进一步提升视频应用的用户体验。我相信视频行业会得到长足发展，现在的标准在未来也会被更佳先进的标准所取代。

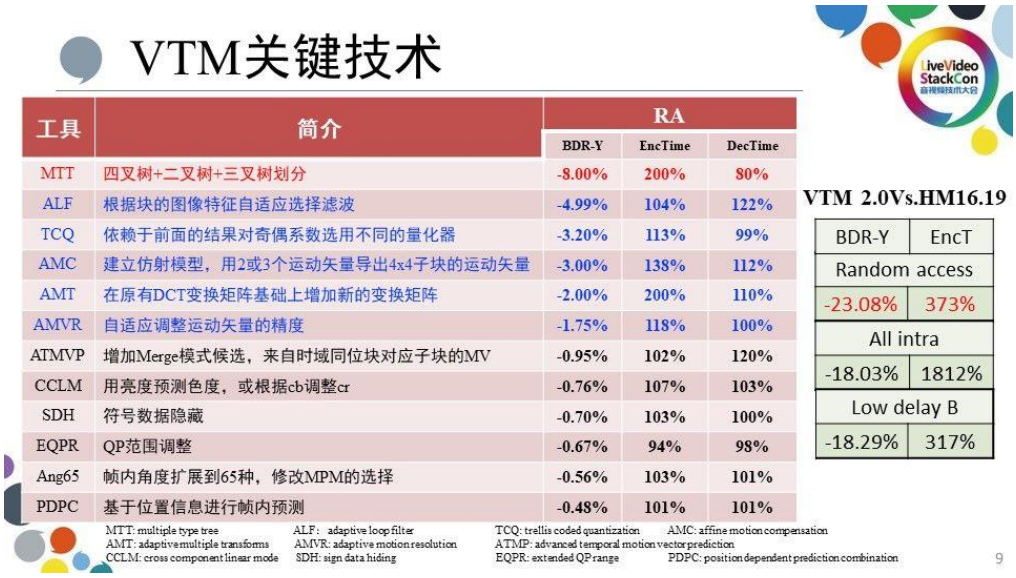
2. VVC 国际标准进展情况

会议次序	地点	开始日期	编号
12	Macau	2018-10	L
11	Ljubljana	2018-07	K
10	San Diego	2018-04	J
9	Gwangju	2018-01	I
8	Macau	2017-10	H
7	Torino	2017-07	G
6	Hobart	2017-03	F
5	Geneva	2017-01	E
4	Chengdu	2016-10	D
3	Geneva	2016-05	C
2	San Diego	2016-02	B
1	Geneva	2015-10	A

大家现在听到 VVC、266 新一代国际标准这些名词等都已不再新鲜，实际上对于新一代标准的讨论已多达 12 次会议，其中有包括快手在内的许多中国公司参与 VVC 标准的制定。

追溯中国参与确立国际视频编码标准，最早可以追溯至 1996 年 MPEG 中国代表团参与 MPEG 标准的制定工作。虽然在 1996 年之前中国对国际标准制定的影响力有限，但在 1996 年之后有许多中国高校参与其中，有力促进了我国相关行业的发展。看到这么多中国企业参与其中我倍感荣幸，虽然我们在探索 AVS，但同时我们也鼓励广大同仁参与国际标准的制订工作。

2.1 VTM 关键技术



工具	简介	RA		
		BDR-Y	EncTime	DecTime
MTT	四叉树+二叉树+三叉树划分	-8.00%	200%	80%
ALF	根据块的图像特征自适应选择滤波	-4.99%	104%	122%
TCQ	依赖于前面的结果对奇偶系数选用不同的量化器	-3.20%	113%	99%
AMC	建立仿射模型，用2或3个运动矢量导出4x4子块的运动矢量	-3.00%	138%	112%
AMT	在原有DCT变换矩阵基础上增加新的变换矩阵	-2.00%	200%	110%
AMVR	自适应调整运动矢量的精度	-1.75%	118%	100%
ATMVP	增加Merge模式候选，来自时域同位块对应子块的MV	-0.95%	102%	120%
CCLM	用亮度预测色度，或根据cb调整cr	-0.76%	107%	103%
SDH	符号数据隐藏	-0.70%	103%	100%
EQPR	QP范围调整	-0.67%	94%	98%
Ang65	帧内角度扩展到65种，修改MPM的选择	-0.56%	103%	101%
PDPC	基于位置信息进行帧内预测	-0.48%	101%	101%

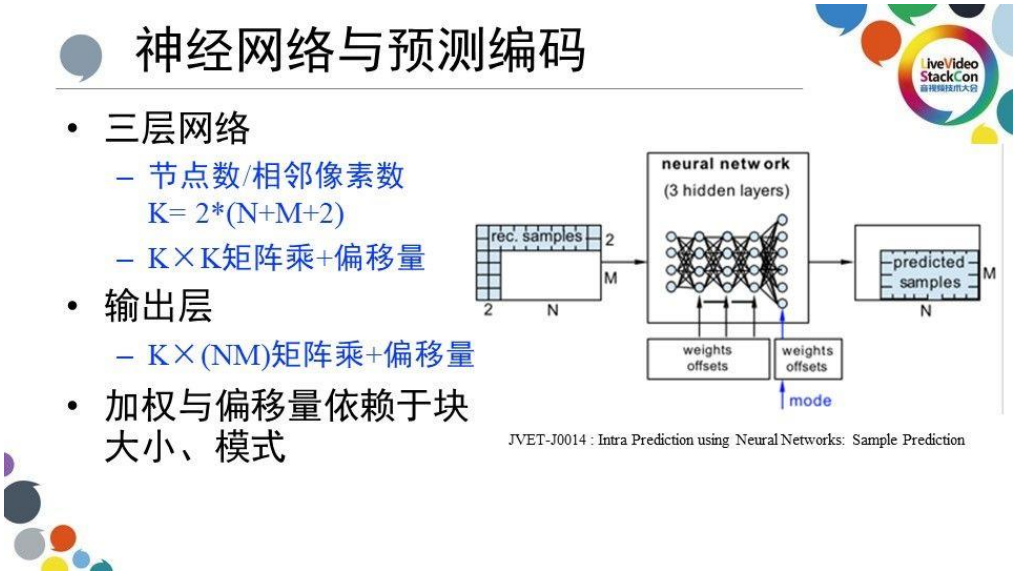
BDR-Y	EncT
Random access	
-23.08%	373%
All intra	
-18.03%	1812%
Low delay B	
-18.29%	317%

上图展示了 VTM 的诸多关键技术，其中蓝色与红色高亮的技术所带来的性能提升非常显著。当然我们对关键技术的考量离不开后端配套设施的跟进，我们在科学制定标准是一定需要在复杂度与性能间实现良好平衡，依靠技术的力量控制二者使其达到理想情况。我们使用的 VTM2.0 软件并将其性能提升用 Random access 量化，可以看到 BDR-Y 的性能提升为 23%。将其结合图中左侧表格中的工具所带来的性能提升十分可观。这里需要注意的是，虽然表格中未高亮部分的工具所带来的性能提升在 1%以下，但由于这些技术的编解码复杂度波动比较小，可显著降低复杂度，后期选择合适技术时我们也能考虑这些在复杂度上表现较好的工具。

2.2 新特色：从信号处理到深度学习

大家知道最近深度学习成为行业热门，人工智能开始为各行各业带来显著优化，深度学习同样也可以被用于优化视频编码。传统的视频编码主要基于信号处理实现预测、变换等，而现在我们更多采用统计分析、深度学习、神经网络等技术使视频编码登上一个全新的阶段。神经网络可以在编码领域大秀拳脚，同样标准制定也受其影响。大家可以从右侧的表单中看到现在许多企业与高校在参与标准制定过程中将神经网络纳入考量范围。

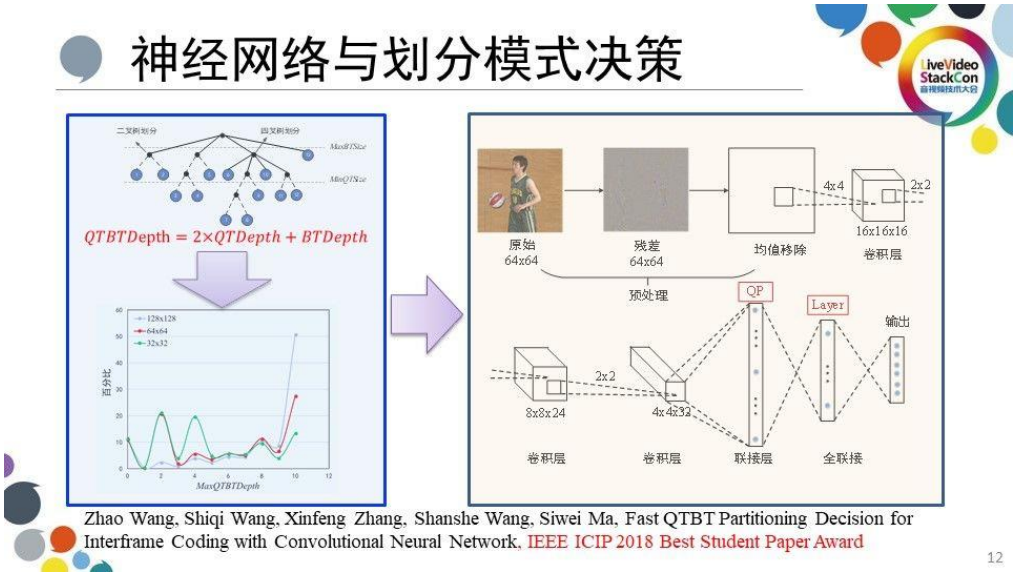
1）神经网络与预测编码

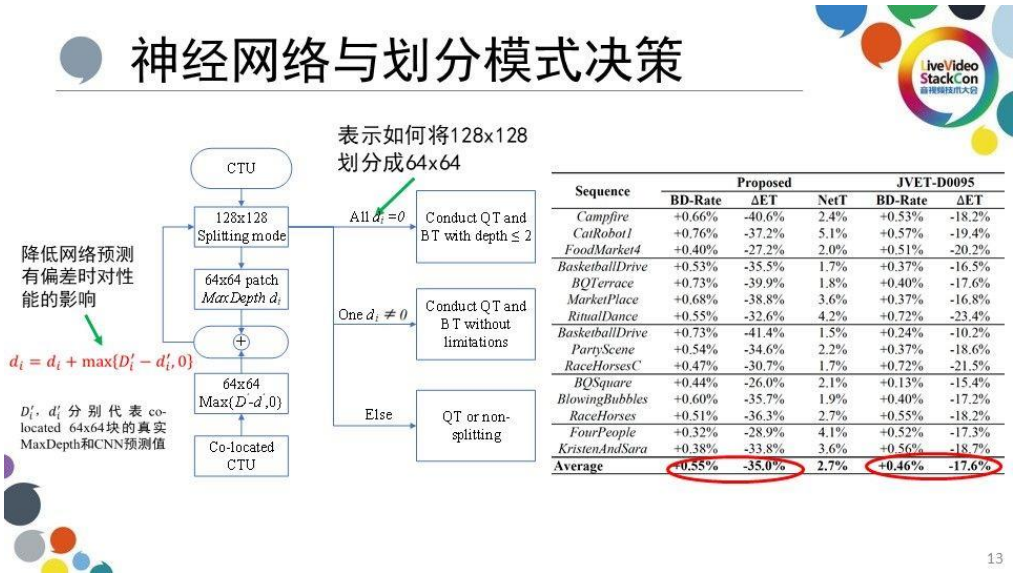


我们可以借助神经网络实现帧率预测、划分等重要工作，神经网络用与预测编码主要需要包括输出层在内的四层网络传统方法主要借助滤波器基于信号处理实现优化；而神经网络则是将节点数、相邻像素数等指标输入神经网络从而重构原始信号使其称为符合编码要求的信号，通过反复的训练寻求最优指标并确定为参数，在此之后这个训练完成的网络就可被用于帧率预测。

除了帧率预测，神经网络也可以实现滤波等功能，但由于这一部分的研究仍需要进一步完善，后续还需配套硬件的跟进才能实现理想效果。从标准制定的角度来看，我们需要跟多的时间优化完善神经网络才能使其符合写入标准的严格要求。

2）神经网络与划分模式决策



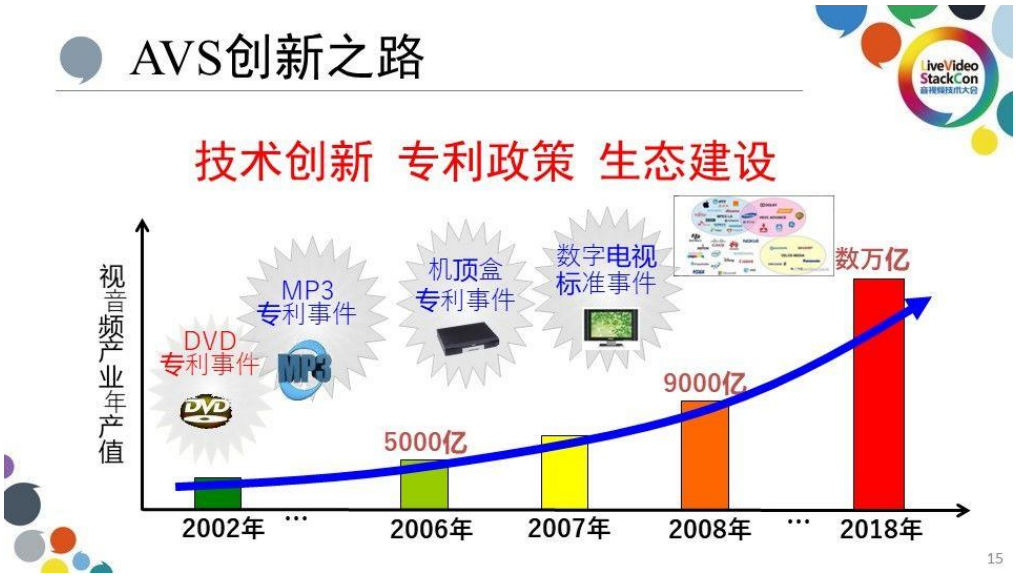


Sequence	Proposed			JVET-D0095	
	BD-Rate	ΔET	NetT	BD-Rate	ΔET
Campfire	+0.66%	-40.6%	2.4%	+0.53%	-18.2%
CatRobot1	+0.76%	-37.2%	5.1%	+0.57%	-19.4%
FoodMarket4	+0.40%	-27.2%	2.0%	+0.51%	-20.2%
BasketballDrive	+0.53%	-35.5%	1.7%	+0.37%	-16.5%
BQTerrace	+0.73%	-39.9%	1.8%	+0.40%	-17.6%
MarketPlace	+0.68%	-38.8%	3.6%	+0.37%	-16.8%
RitualDance	+0.55%	-32.6%	4.2%	+0.72%	-23.4%
BasketballDrive	+0.73%	-41.4%	1.5%	+0.24%	-10.2%
PartyScene	+0.54%	-34.6%	2.2%	+0.37%	-18.6%
RaceHorsesC	+0.47%	-30.7%	1.7%	+0.72%	-21.5%
BQSquare	+0.44%	-26.0%	2.1%	+0.13%	-15.4%
BlowingBubbles	+0.60%	-35.7%	1.9%	+0.40%	-17.2%
RaceHorses	+0.51%	-36.3%	2.7%	+0.55%	-18.2%
FourPeople	+0.32%	-28.9%	4.1%	+0.52%	-17.3%
KristenAndSara	+0.38%	-33.8%	3.6%	+0.56%	-18.7%
Average	**+0.55%**	**-35.0%**	**2.7%**	**+0.46%**	**-17.6%**

谈到使用神经网络实现划分，我们需要明确不同划分模式需要哪些技术调整。这里我们的目标是基于四叉树与六叉树实现 64x64 的划分。具体步骤为首先提取整体划分深度特征来预测划分程度，也就是判断采用 128×128 进行预测还是 64×64 进行预测。

如果选择 128x128 作为预测那么其深层次的预测准确度会大打折扣，而 32x32 的预测与 128x128 之间相差太远同样造成预测不准确，因此我们选择基于 64×64 进行深度划分预测，我们希望这种选择可以有效降低网络预测有偏差时对性能的影响。第二张图右侧列表展示的相关性能指标提升。

3. AVS 标准回顾及应用情况



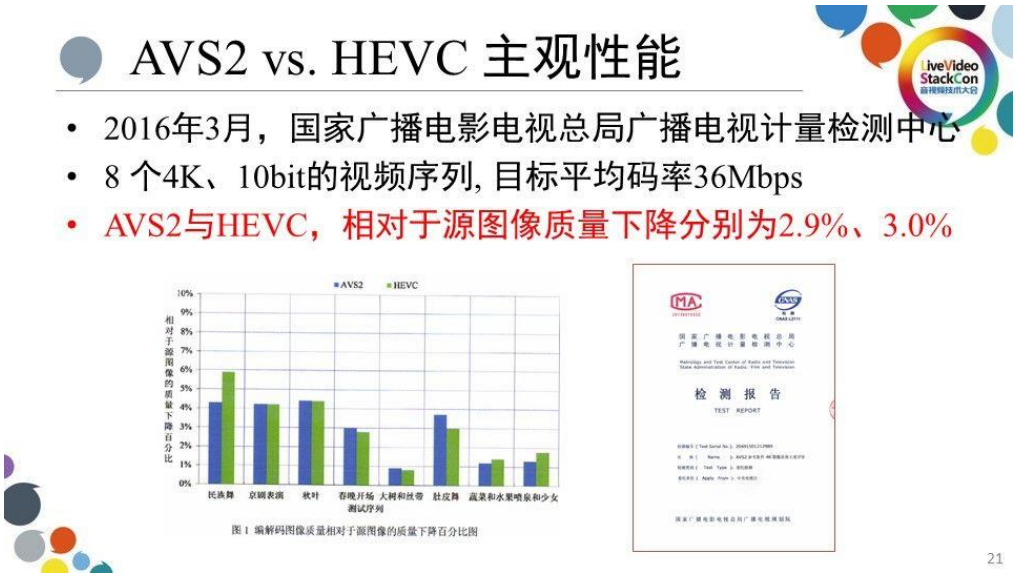
AVS 标准的确立最早可以追溯至 2002 年的 DVD 专利事件，正是因为此事件推动了 AVS 标准的产生。经过几十年的发展 AVS 对推动技术创新、规范专利政策、推动行业生态建设做出了卓越贡献。从开始的盗版横行到现在的各种专利联盟日趋规范音视频产业年产值达到数万亿，人们对技术与产业价值的追求是全行业永恒的话题。无论是 AVS 标准还是国际标准，其目的都是推动技术创新的积聚与经济价值的最大化。

AVS 的管理模式在其被提出时就有相关规范，主要由技术研发、包括专利池与专利授权在内的知识产权管理、产业联盟三部分组成。这种模式在十几年前具有非凡的创新意义，即使在今天看来也是十分先进的理念。现在 AVS 上百位工作组会员与产业联盟成员为整个标准提供有力支撑，帮助其持续优化发展。

上图展示的 AVS 在过去十六年经历的发展过程，从 2002 年 AVS 确立到 2003 年第一代标准 AVS1 基本完成，再到随后中国高清电视广播使用 AVS 标准，2012 年启动 AVS2 标准的制定，2015 年完成 AVS2 标准的制订，2016 年 AVS 被批准为国家标准并成为广电行业标准，最近的 2018 年 3 月份 AVS3 正式启动。

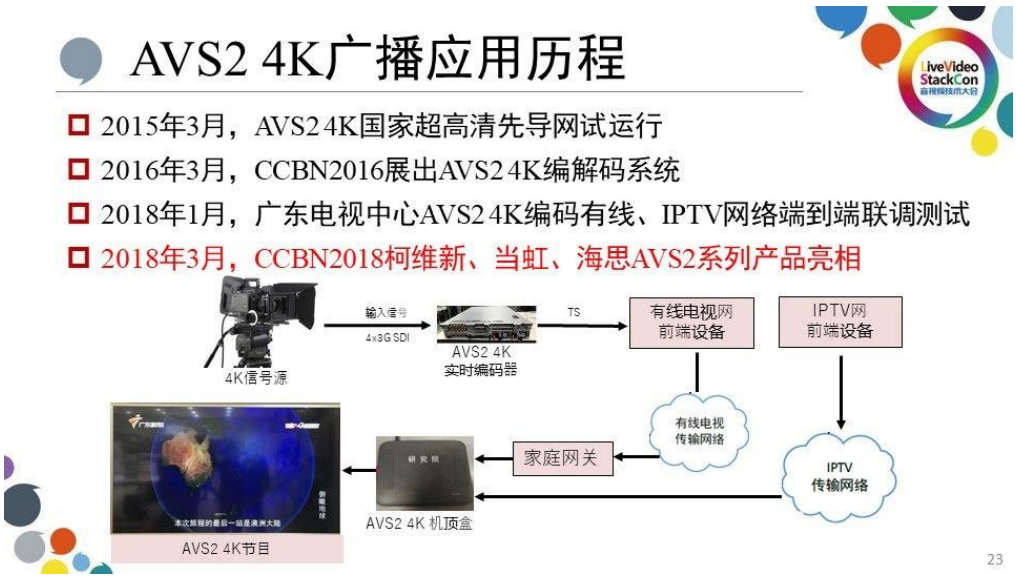
从 AVS 的标准历程中我们可以看到，可以说在过去十六年 AVS 的发展取得了很好的成绩。

落实在应用上，中央电视台已在 2018 年 10 月启动了基于 AVS2 标准的 4K 超高清频道的播出。并且全国各地相继有地方电视台开通 4K 电视频道，进一步加快了技术转化为市场效益。



如果我们对比 AVS2 与 HEVC 便不难发现，无论是在主观质量还是用户体验上 AVS2 都略胜一些。

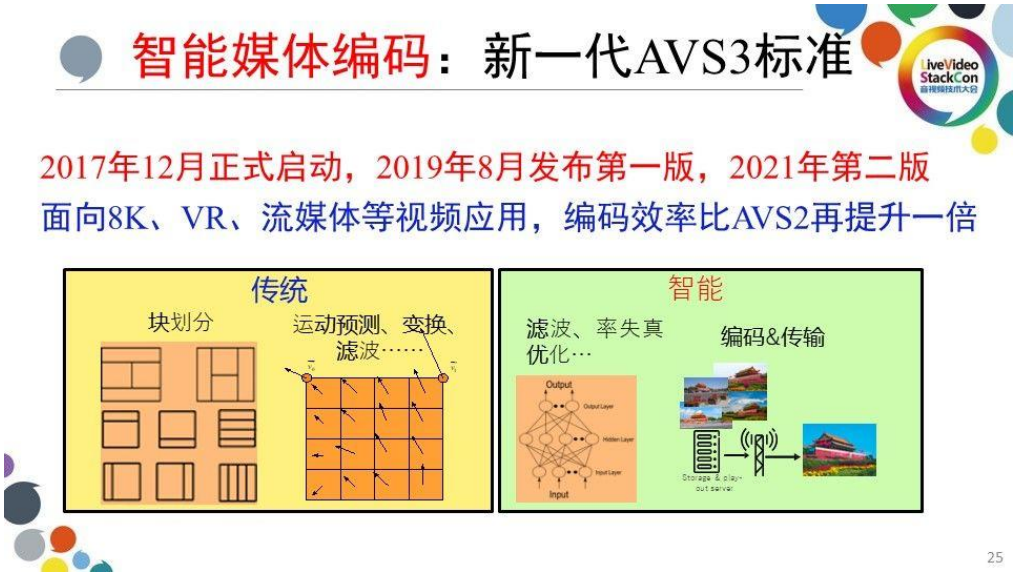
在编码其优化方面，上图展示的实验结果来自于我的学生优化的 AVS2 编码器，商用 AVS 编码器的性能还要高于上述结果。右上图我们可以看到 AVS2 的各项性能指标与其他标准下的编码性能对比可以说是不相上下， AVS2 与 x265 相同性能条件下 AVS2 编码速度快一倍。



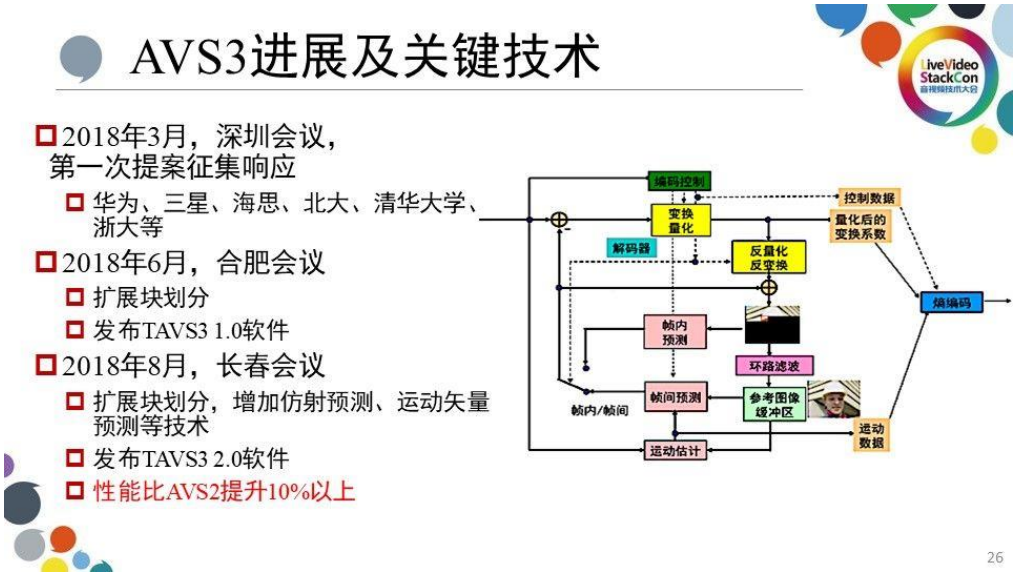
AVS 标准在软硬件复杂度上的性能指标较为优秀，这也使其受到全行业的大力推广。上图展示的 AVS2 4K 广播应用的发展历程，可以看到全行业都对 AVS2 抱有强烈的兴趣并积极参与推广建设，这极大提高了 AVS2 的普及速度与应用范围。

4. AVS 新一代标准关键技术

4.1 概述

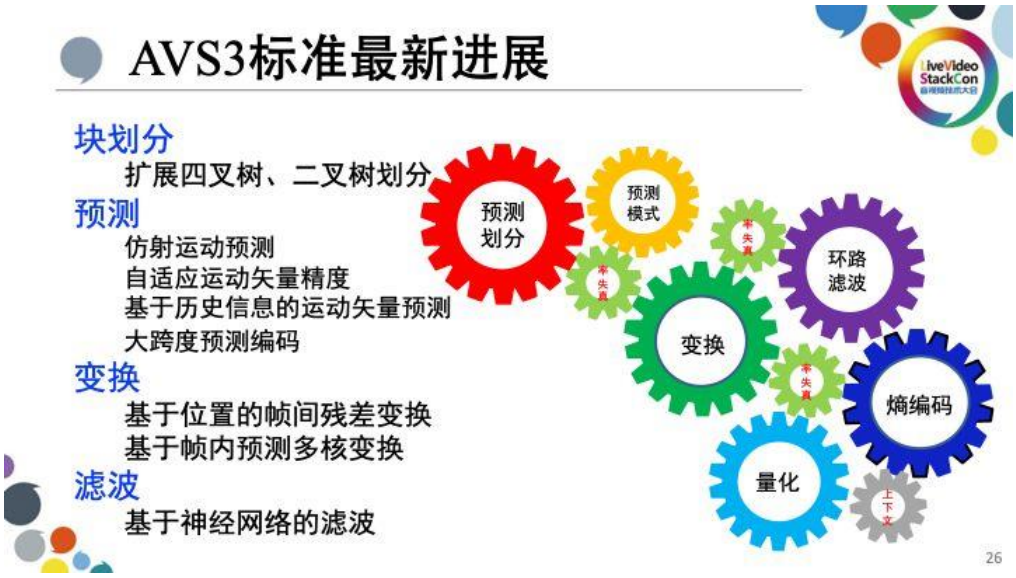


2017 年 12 月份我们启动了新一代标准 AVS3 的确立工作，2018 年 3 月份开始第一次提案征集。预计在未来的 2019 年 3 月份发布第一版 AVS3，其主要在编解码复杂度控制上做出显著优化并预计带来 30% 的性能提升；2021 年发布第二版，达到相对于第一版 50%的效率提升并实现其在 8K、VR、流媒体视频等领域的应用；最终我们希望实现的是在第二版基础上高达一倍的性能提升。从技术角度来看，我们的思路主要分为传统与智能两部分，传统部分仍然为块划分、运动预测、变换、滤波等传统策略，而在智能部分我们会划时代地使用基于神经网络的工具实现滤波、率失真优化等功能。



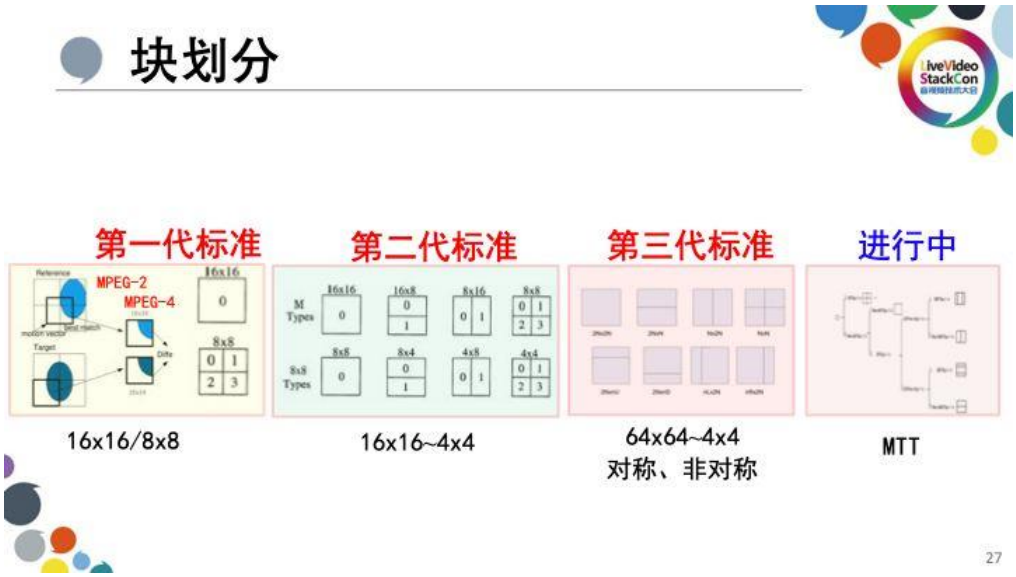
上图展示的是过去一段时间 AVS3 的会议进展与关键技术情况，到目前为止我们实现了相对于 AVS2 10%的性能提升。

4.2 最新进展



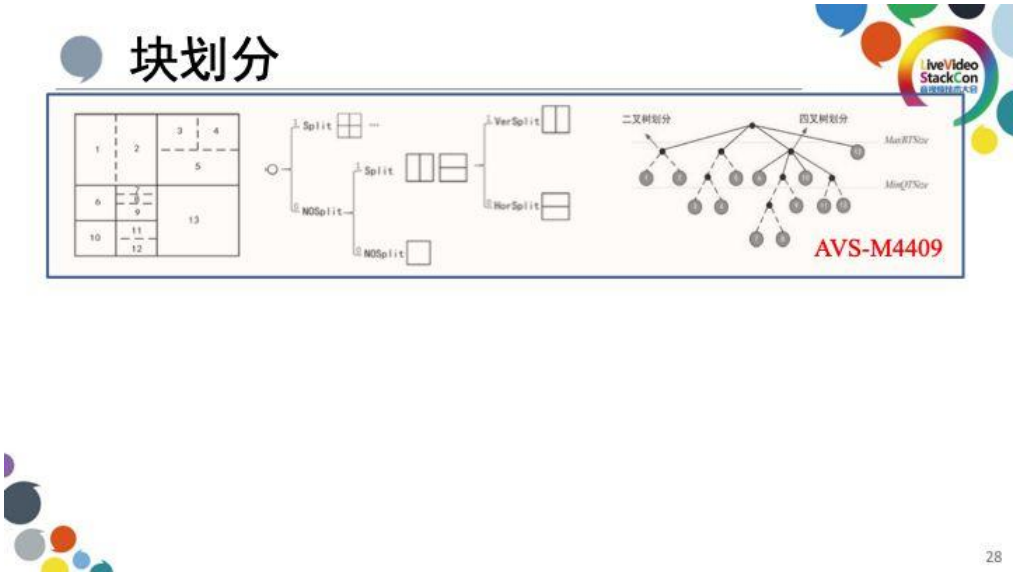
现在 AVS3 标准的框架主要基于传统部分建立，其中有包含扩展四叉树、二叉树划分在内的块划分，包含仿射运动预测、自适应运动矢量精度、基于历史信息的运动矢量预测与大跨度预测编码在内的预测，包括基于位置的帧间残差与帧内预测多核在内的变换还有基于神经网络的滤波。

1）预测划分

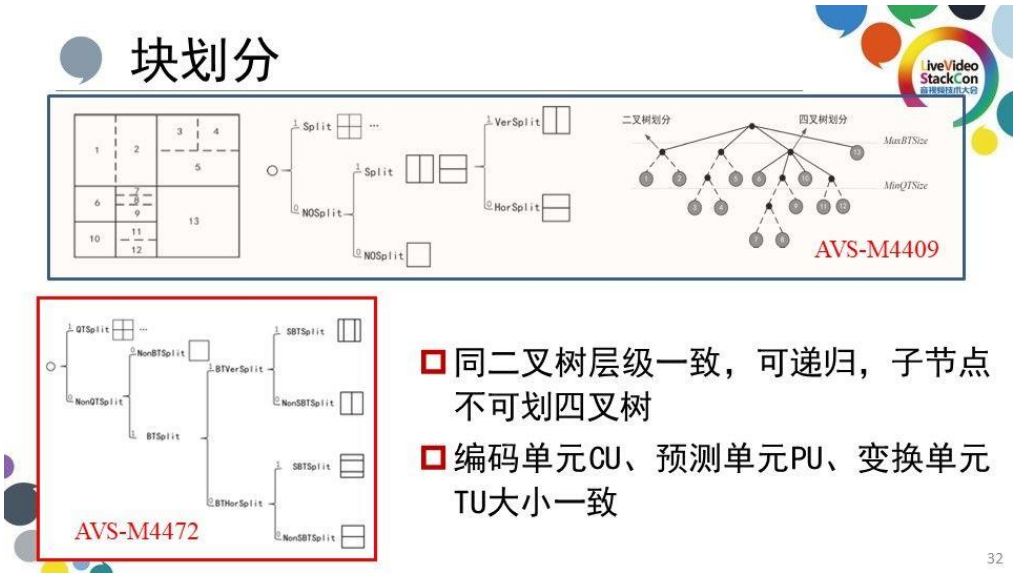


我们现在所有的标准都是基于块划分实现的，随着标准的不断迭代，划分的效率也不断提升。其本质在于通过更高效的划分降低预测误差。这里我们努力实现的为自适应运动预测与复杂视频的内容高效处理，经过预测之后的图像内容有些平滑过渡有些带有边缘或纹理；

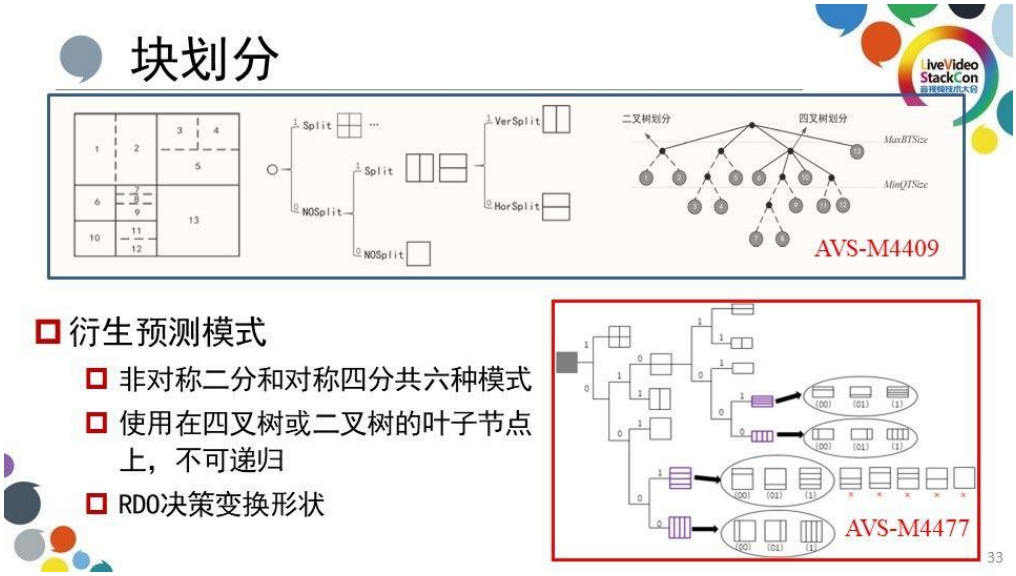
若对其统一进行某一种变换效率势必会降低，我们期待通过自适应算法将进一步提升整个过程的效率，自适应的前提是我们需要将变换量化并考虑自身滤波对视频图像内容特性的适应。随着技术的演变，块划分方案的不断迭代意味着划分方案的不断改进，如正在进行的采用 MTT 实现的多种树型划分，其本质为使用更精细的划分方案解决更复杂的图像内容从而实现理想的效果。



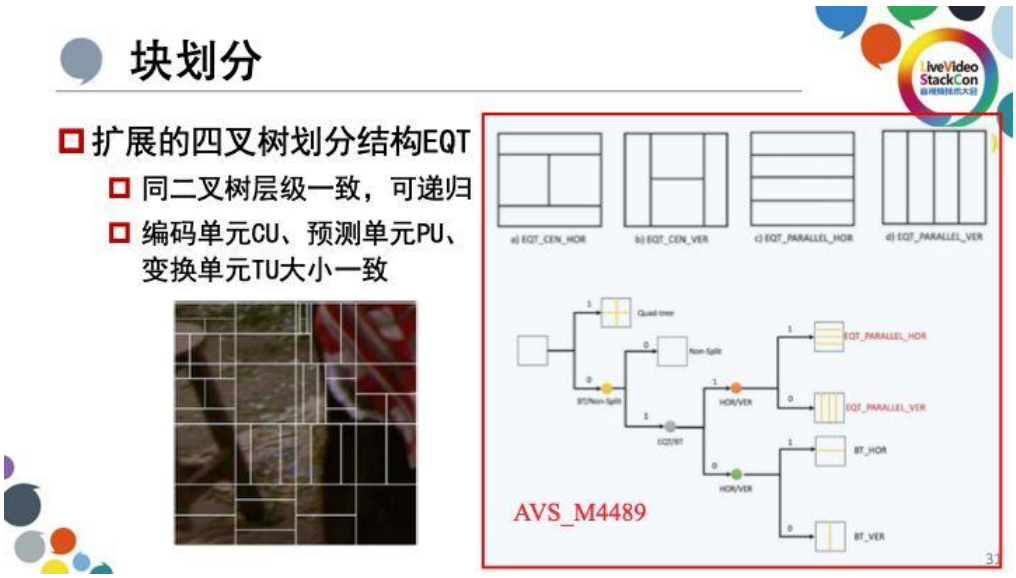
AVS-M4409 的划分过程基本上基于 VCC 的三叉树划分，同时包括了二叉树与四叉树。



而 AVS-M4472 则是二叉树层递归划分。除此之外 AVS-M4472 的编码单元 CU、预测单元 PU、变换单元 TU 大小一致。

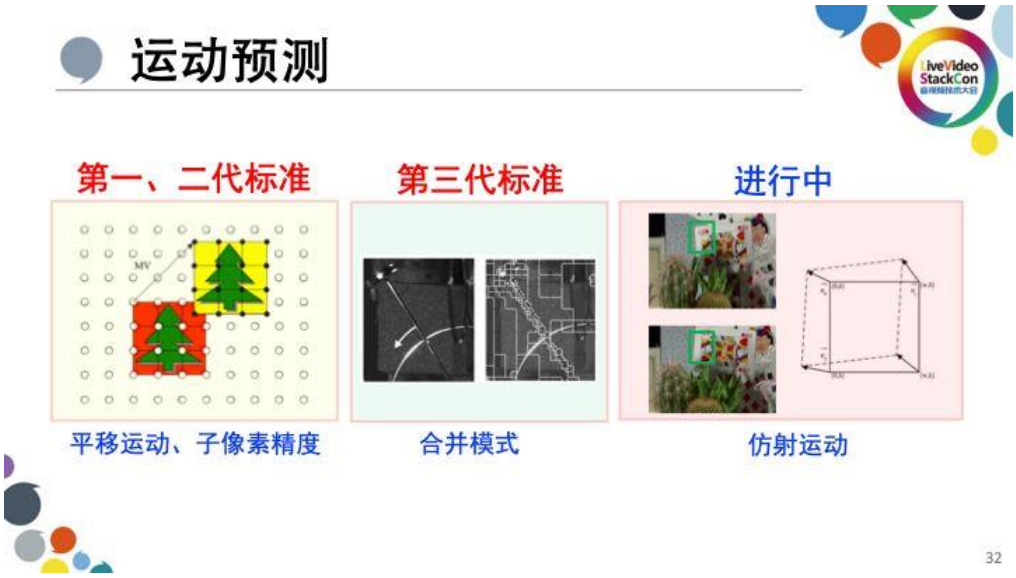


接下来发展到了 AVS-M4477 则采用了衍生预测模式，使用非对称二分和对称四分模式，并且在四叉树或二叉树的叶子结点上不可递归，使用 RDO 决策变换形状，实现良好性能更高效率的划分。

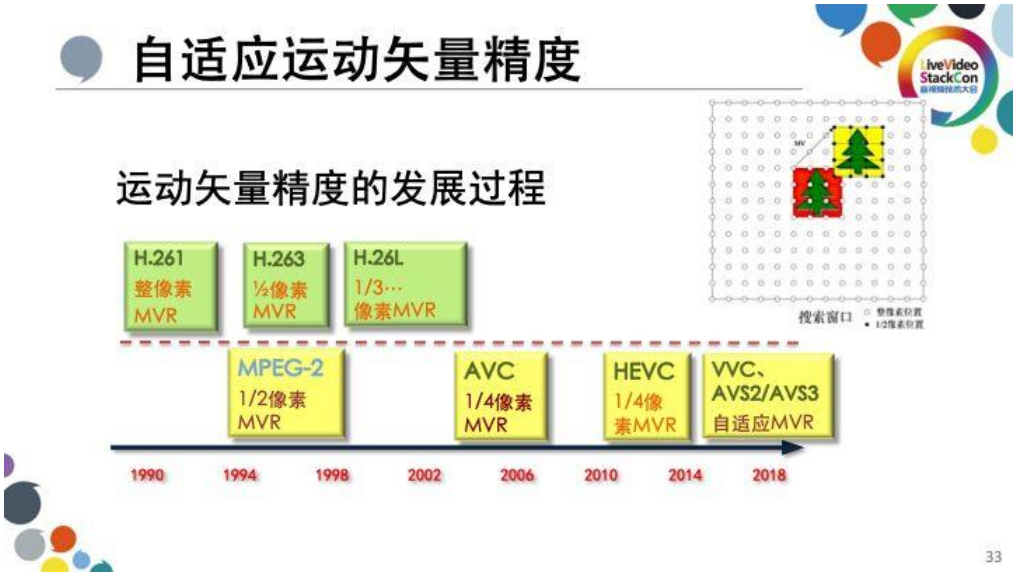


最后的 AVS-M4489 所运用的扩展的四叉树划分结构 EQT 是我们最新采用的一种划分模式，此划分同二叉树层级一直并可递归，其 CU、PU、TU 大小一致，我相信还有更加高效的划分模式等待我们探索。

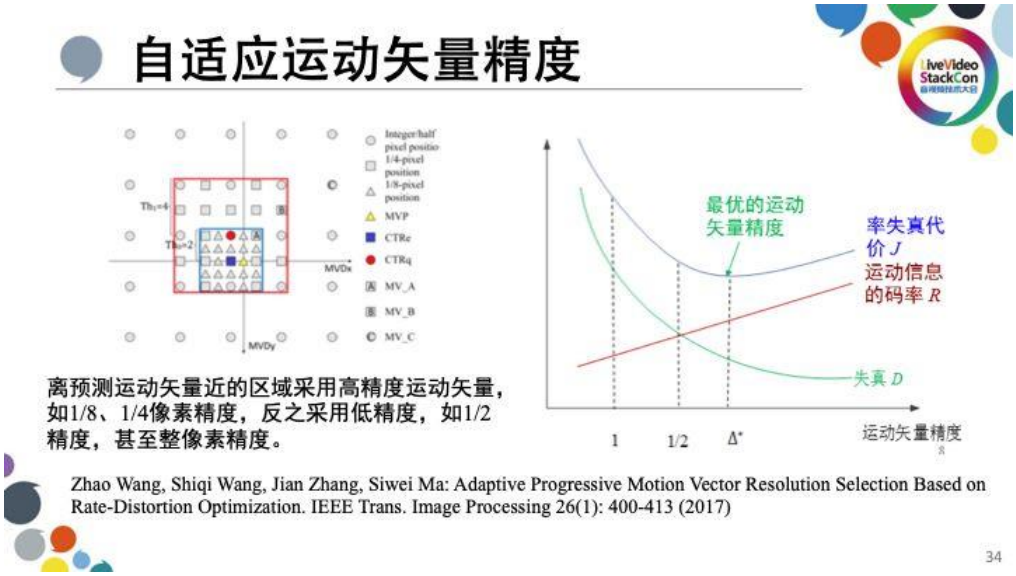
2）运动预测



在运动预测方面，第一代、第二代标准所使用的预测方案对平移运动效果较好，一旦运动状态变得复杂则容易失效；于是在第三代我们采用了合并模式，基于平移运动与向量块信息的组合更高效准确预测物体运动；而现在正在探索的仿射运动预测又把对复杂运动预测的效率提升至新的水平，我们也将其视为研究的目标。



我们现在所看到的运动矢量精度都是基于适量像素实现，出于对复杂度的控制我们需要一种较为理想的方案确定运动矢量精度。

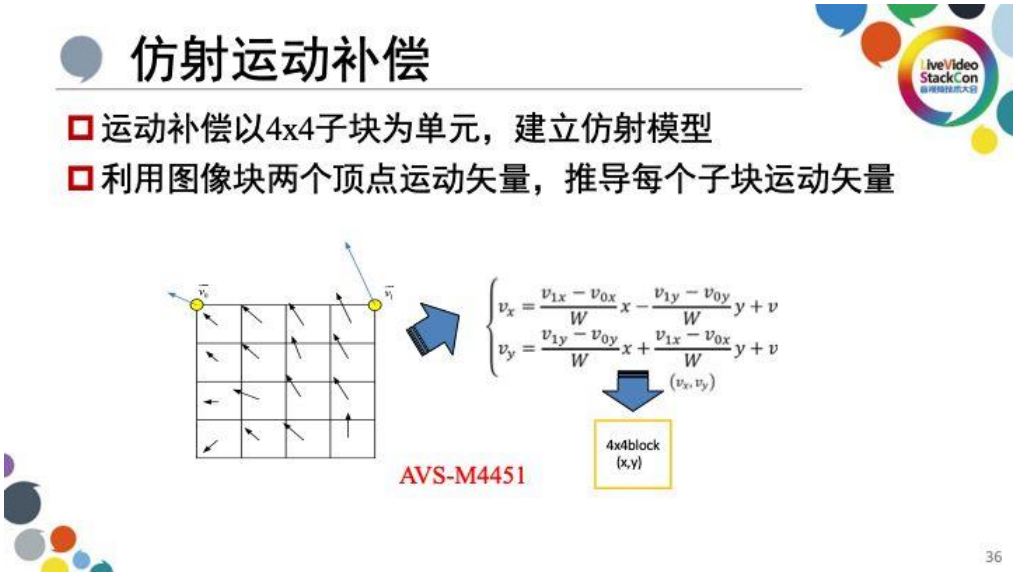


其解决方案是在离预测运动矢量近的区域采用高精度运动矢量如 1/8、1/4 像素精度，而在离预测运动矢量远的区域采用低精度如 1/2 甚至整像素精度，从而有效提高运动矢量的编码效率。比如这里在这个小的窗口里边，运动比较预测比较接近，就用高精度的，到了窗口外边就用粗精度了，在 AVS 里边是做 1/2 和 1/4 这么一个切分这样的，可以有效降低编码复杂度的同时进一步提升编码效率。

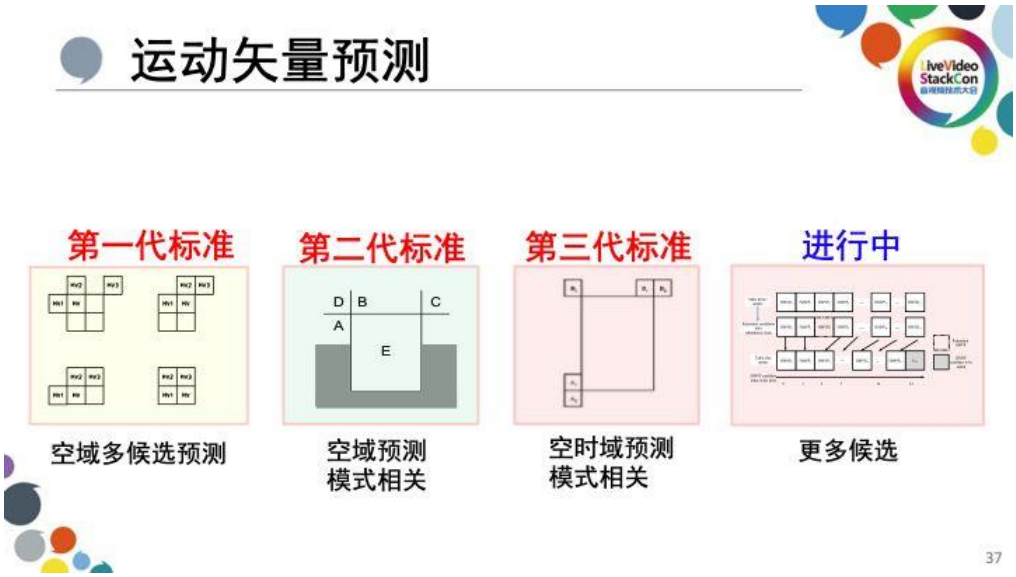
MVR Index	0	1	2	3	4
Resolution (R) in pel	1/4	1/2	1	2	4
N^{th} MVP	1st MVP	2nd MVP	3rd MVP	4th MVP	5th MVP

AMVR Bi-Index	(MVD_x, MVD_y) of list 0	(MVD_x, MVD_y) of list 1
0	Signaled	Signaled
1	Not signaled	Signaled
2	Signaled	Not signaled

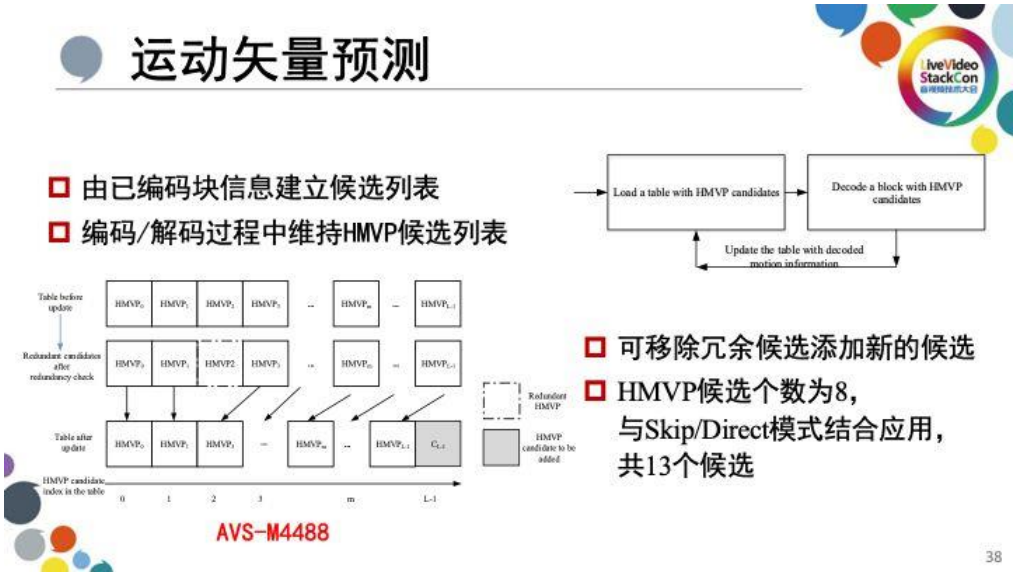
在 AVS3 中有 1/2、1/4、1、2、4 五种可选运动矢量精度，可以进一步提升编码效率。



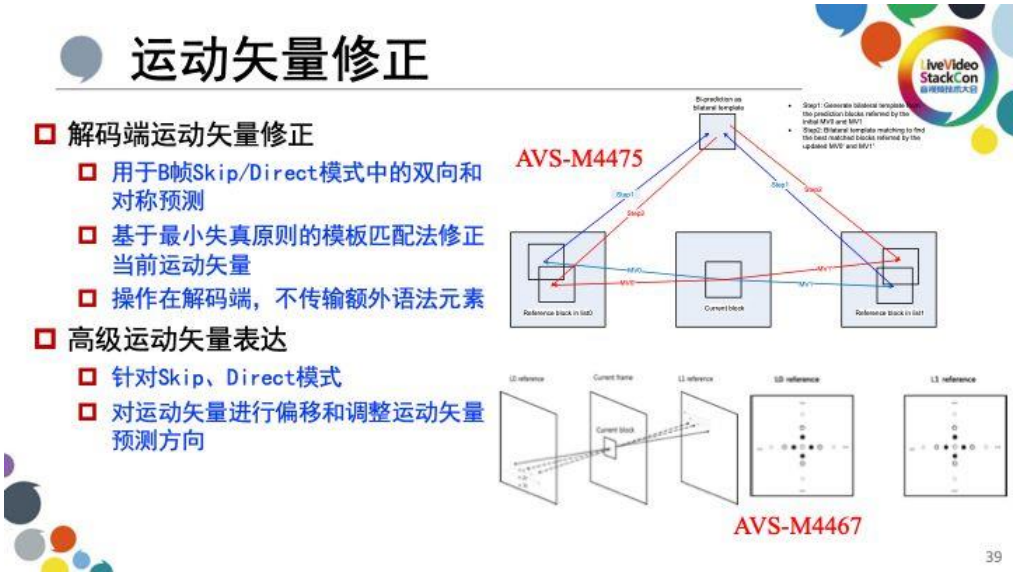
相对于传统的平移运动，仿射运动补偿可以根据上图左侧方形顶点上的运动矢量推导其中每一个小方块的运动矢量并求解运动仿射参数，从而实现相对于传统仿射运动更加高效的预测。



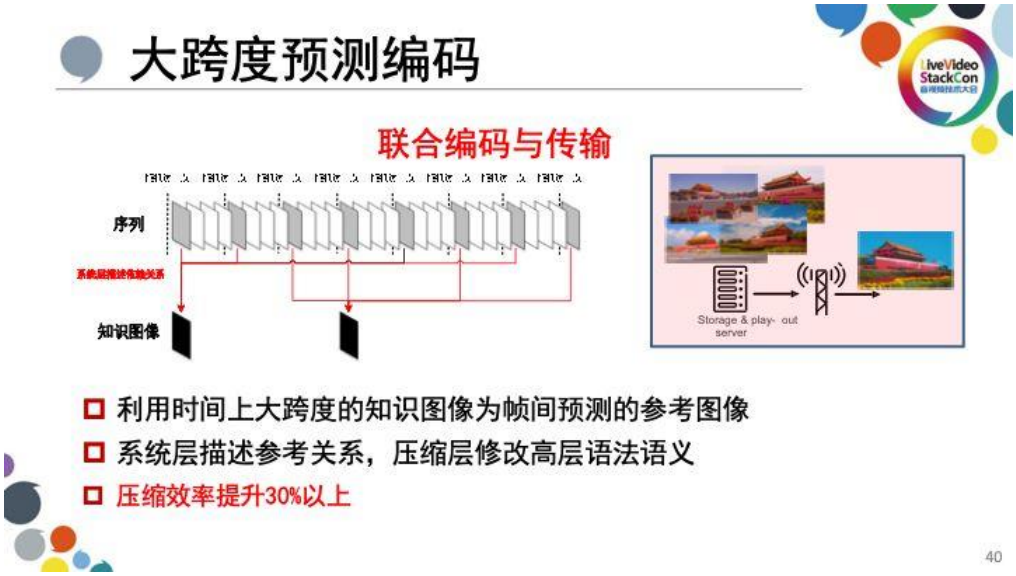
第一代标准采用了空域多候选预测的方式实现运动矢量预测，在后续几代中加入了空时域预测模式。现在我们正在探索加入更多候选预测运动矢量以带来编码增益。



AVS-M4488 采用了由已编码块信息建立候选运动矢量预测列表 HMVP，通过多个块的运动矢量构建多个列表并移除冗余候选添加新的候选，并在编码/解码过程中维持 HMVP 的候选列表，可以有效提高运动矢量预测的编码效率。



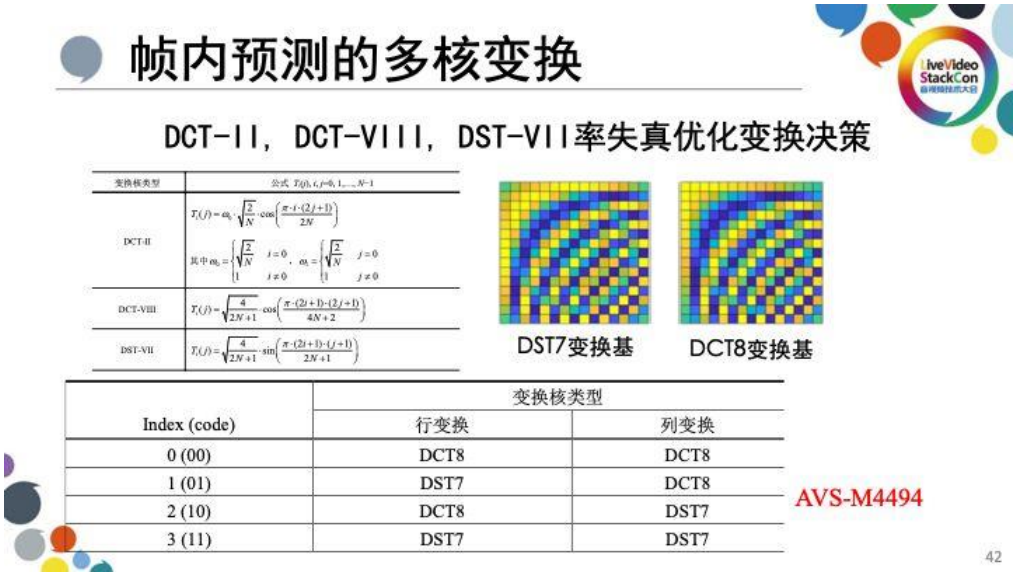
还有对 Skip、direct 模式的运动矢量进行更精细更准确的调整，也能带来明显的编码效率提升。



将编码与传输结合实现大跨度预测编码是我们正在探索的一件较为超前的方案。我们将时间上大跨度的知识图像作为帧间预测的参考图像，在系统层描述参考关系，在压缩层修改高层语法语义，并基于类似 P2P 传输原理实现视频内容之间的跨序列参考。此工作需要系统层的传输支持，大跨度预测编码可以带来 30%以上的压缩效率提升。

3）变换

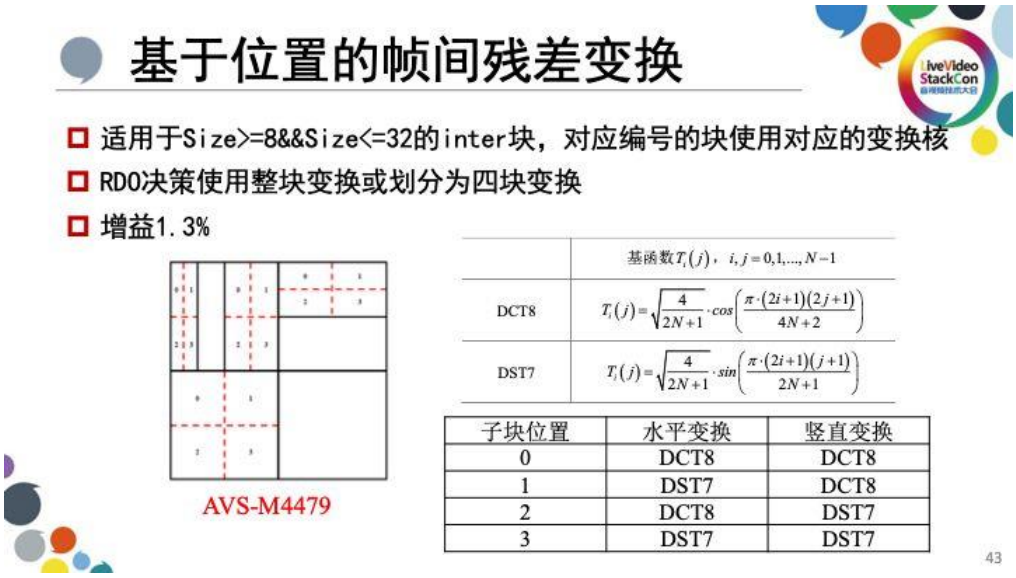
第一代标准使用的变换模式为 DCT，随后的第二代标准则采用了 ABT。DCT 在相关性较强时较为适用，而第三代标准所使用的 TU 与 DST 则更适应信号的复杂特征变化，现在我们正在探索的变换方案为多核变换。



变换核类型	公式 $T_i(j)$, $i, j=0, 1, \ldots, N-1$
DCT-II	$T_i(j)=\omega_0\cdot\sqrt{\frac{2}{N}}\cdot\cos\left(\frac{\pi\cdot i\cdot(2j+1)}{2N}\right)$ 其中 $\omega_0=\begin{cases}\sqrt{\frac{2}{N}} & i=0\\ 1 & i\neq 0\end{cases}$，$\omega_0=\begin{cases}\sqrt{\frac{2}{N}} & j=0\\ 1 & j\neq 0\end{cases}$
DCT-VIII	$T_i(j)=\sqrt{\frac{4}{2N+1}}\cdot\cos\left(\frac{\pi\cdot(2i+1)\cdot(2j+1)}{4N+2}\right)$
DST-VII	$T_i(j)=\sqrt{\frac{4}{2N+1}}\cdot\sin\left(\frac{\pi\cdot(2i+1)\cdot(j+1)}{2N+1}\right)$

Index (code)	变换核类型	
	行变换	列变换
0 (00)	DCT8	DCT8
1 (01)	DST7	DCT8
2 (10)	DCT8	DST7
3 (11)	DST7	DST7

帧内预测变换这块，行列变换组合的变换形式，早期 AVS 也做过很多的相关的研究，通过率失真优化来选择变换。

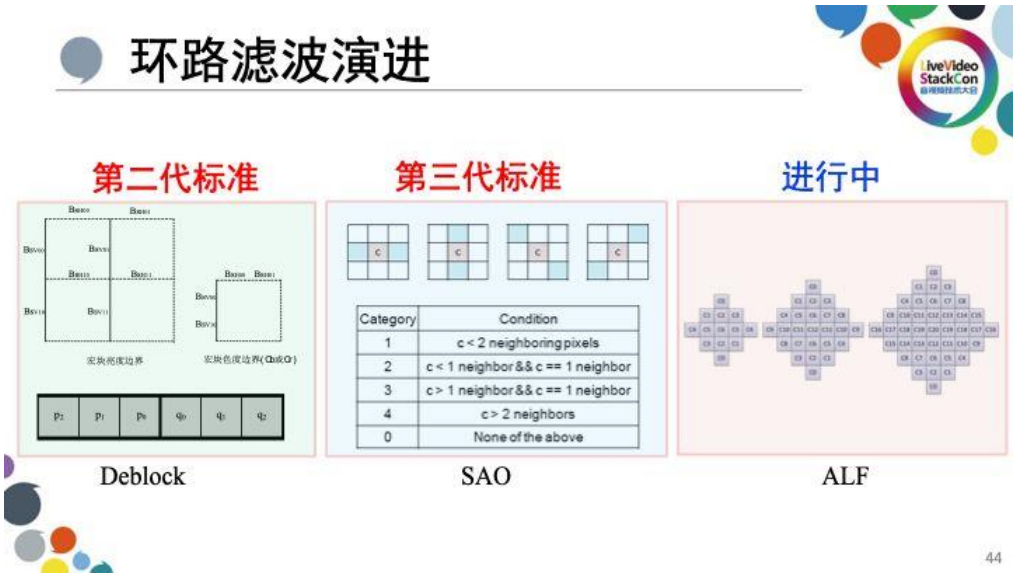


	基函数 $T_i(j)$，$i, j=0,1,\ldots,N-1$
DCT8	$T_i(j)=\sqrt{\frac{4}{2N+1}}\cdot\cos\left(\frac{\pi\cdot(2i+1)(2j+1)}{4N+2}\right)$
DST7	$T_i(j)=\sqrt{\frac{4}{2N+1}}\cdot\sin\left(\frac{\pi\cdot(2i+1)(j+1)}{2N+1}\right)$

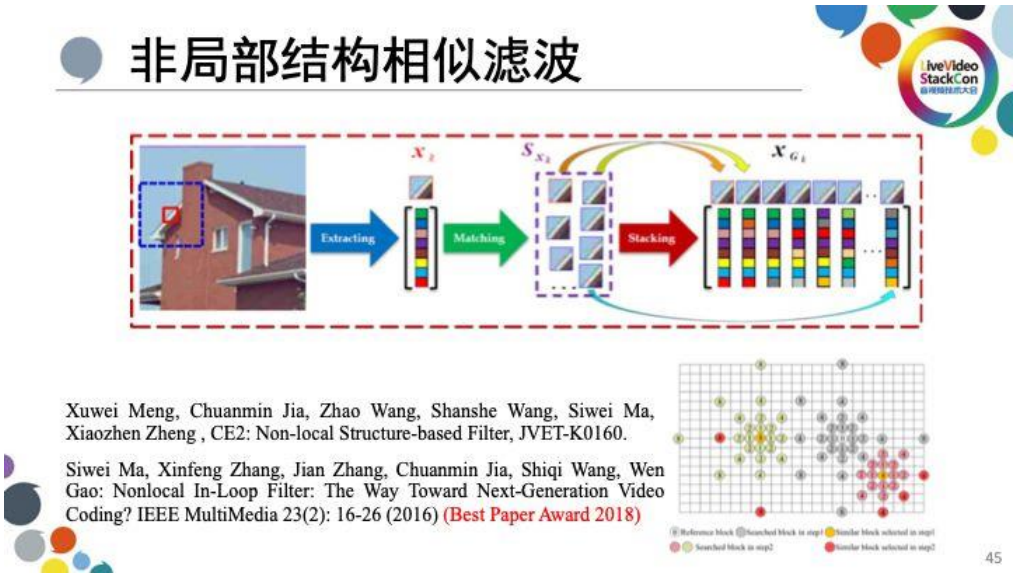
子块位置	水平变换	竖直变换
0	DCT8	DCT8
1	DST7	DCT8
2	DCT8	DST7
3	DST7	DST7

帧间残差变换是基于位置变化的，在不同块的位置其参数特性也有所区别。我们可以选择性变换从而进一步提高变换的编码效率实现性能提升。

4）环路滤波



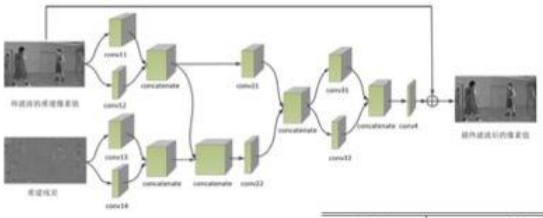
在第一代标准中没有环路滤波而只是将其作为后处理的一部分。在第二代标准中我们使用了对块边界的 Deblock 环路滤波，从 H265 这块，引入了 SAO，原理上就是从几种边缘看一看是不是有了较大的量化失真，给它加一个补偿，降低量化失真；这时候也还有有另外一中环路滤波就是 ALF，对于这一块儿关于滤波器的形状研究更多，加了好多滤波器形状，去进一步提高这个滤波器效果去，这是滤波部分的一个演变。



关于滤波，我们还探索了非局部结构相似滤波，其主要是依据统计局部多个像素发生失真得出的规律特性推断更优的全局优化策略。虽然此方案复杂度较高但效果明显，全局优化处理能够带来明显的效率提升。可能目前全局滤波方案还难以被标准采用，我们仍在探索更好的全局滤波方案。

智能编码

- 将重建残差作为网络输入，Y分量2.6%的性能提升
- 发布TAVS-AI branch软件



AVS-M4471

Layer	Layer1				Layer2		Layer3		Layer4
Conv.module	conv11	conv12	conv13	conv14	conv21	conv22	conv31	conv32	conv4
Filter size	3x3	5x5	3x3	5x5	3x3	3x3	1x1	3x3	3x3
Filter number	32	16	32	16	32	16	32	16	1
Parameters	288	400	288	400	13,824	13,824	1,536	6,912	432
Total parameters	37,904								

46

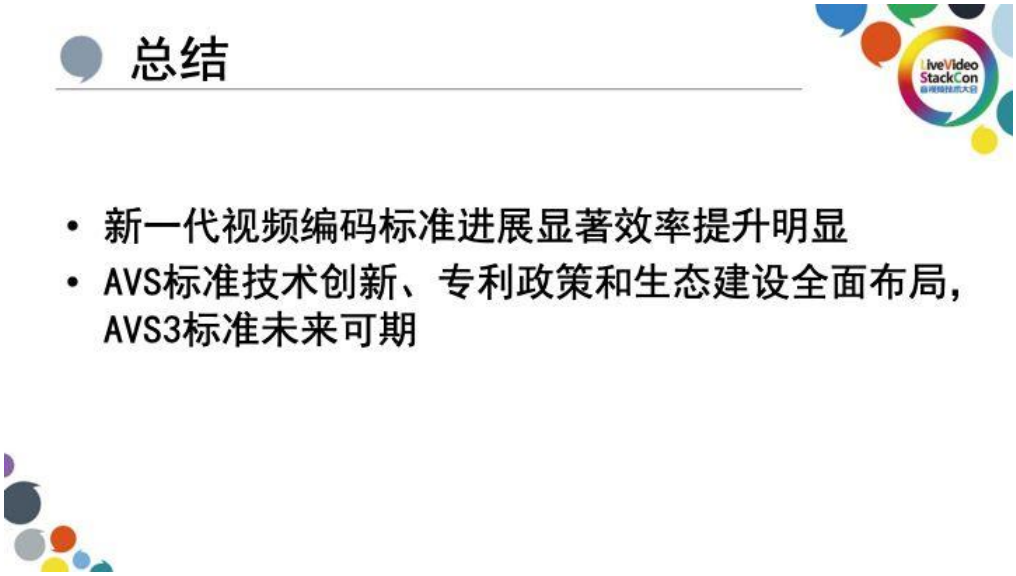
关于智能编码与智能滤波部分。由于滤波是比较独立的模块，传统滤波主要输入重构的图像即可，在这里我们同时输入了相关编码信息之后发现还能进一步提高滤波性能，这说明滤波性能提升空间还是比较可观的。

AVS3 TAVS2.1性能测试

Anchor：AVS2 RD19.4
Test： AVS3 TAVS2.1
测试条件：QP(27/32/38/45)
定参考结构，7bframes；
测试序列: AVS3 CTC

Seq			BDR-Y	BDR-U	BDR-V
720P	8bit	City	-10.18%	-16.44%	-16.80%
		Crew	-10.72%	-27.16%	-23.25%
		Vidyo1	-9.98%	-12.94%	-13.87%
		Vidyo3	-13.40%	-23.71%	-17.31%
1080P	8bit	Cactus	-10.97%	-15.97%	-15.84%
		BasketballDrive	-10.05%	-24.12%	-21.28%
	10bit	MarketPlace	-14.21%	-13.04%	-15.89%
		RitualDance	-9.28%	-13.87%	-15.03%
4K	8bit	pku_girls	-8.52%	-13.14%	-13.11%
	10bit	ParkRunning3	-7.86%	-5.70%	-8.14%
		Campfire	-8.32%	-5.96%	-15.20%
		MountainBay2	-7.59%	-7.85%	-22.06%
720p			-11.07%	-20.06%	-17.81%
1080p			-11.13%	-16.75%	-17.01%
4K			-8.07%	-8.16%	-14.63%
All			-10.09%	-14.99%	-16.48%

上图展示的是 AVS3 TAVS2.1 的性能测试结果，可以看到 AVS3 相对于 H.265 已有 10%以上的性能提升，预期第一阶段将达到 30%，我们对 AVS3 未来的发展充满信心。



我们可以看到新一代视频编解码标准的进步十分显著，效率提升非常明显，我们相信新一代视频编码标准具有蓬勃生命力。AVS 标准在技术创新、专利政策与生态建设方面都有较为全面的布局，新一代 AVS 标准未来可期。

国内之痛：传输协议与网络优

新一代直播传输协议 SRT

SRT 协议是基于 UDT 的传输协议，保留了 UDT 的核心思想和机制，抗丢包能力强，适用于复杂的网络。在 LiveVideoStack 线上分享中，新浪音视频架构师 施维对 SRT 协议的原理、优缺点特性以及在流媒体中的应用进行了详细解析。

施维　LiveVideoStack 线上分享　2020-03-06

1. 为什么选择 SRT?

毋庸置疑，现今存量最大的直播协议是 RTMP，但随着新技术的不断发展与使用场景的不断拓展，继续使用 RTMP 会令人感到有些力不从心。RTMP 协议的缺陷主要有以下四个方面：RTMP 协议缺陷。

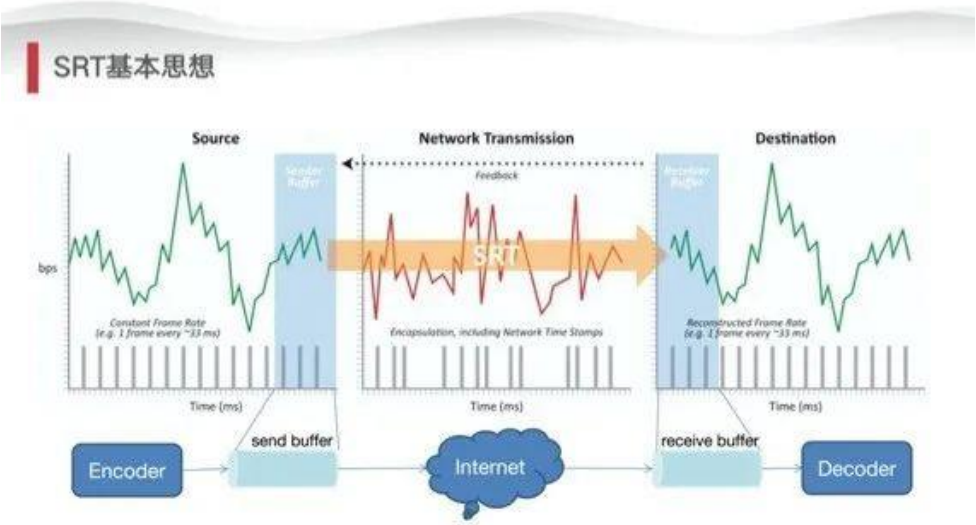
首先，RTMP 协议太老，且最后一次更新是在 2012 年；同时 HEVC/H.265/AV1 等视频格式都没有官方定义，以至于需要国内 CDN 厂商自行定义。RTMP 连接过程较长，由于 RTMP 基于 TCP（TCP 存在三次握手），除此之外，其本身又存在 c0/s0 到 c2/s2 的三次握手，再加上 connection，createstream，play/publish，总地来说 RTMP 完成一次建连需要进行 9 次会话，用于 PC 端勉强能够接受，对于移动端网络质量的要求则很高。RTMP 的拥塞控制完全依赖传输层，即完全依赖于 TCP 传输层的拥塞控制算法来进行拥塞管理，几乎没有什么优化；RTMP 本身基于 TCP 传输，无法提供带宽自适应的算法。在此背景下，众多厂商开始着手提供一些新的直播协议供行业参考。如 QUIC、SRT 等。本次我们将重点讲述 SRT 的特点与应用。

SRT 协议的特点

Haivision 联手 Wowza 在 UDT 的基础上针对音视频实时性提出了 SRT 协议。SRT 是基于 UDT 的协议（UDT 协议是基于 UDP 的传输协议，在 IETF 已经提交了 4 个版本），具有非常良好的丢包重传机制，丢包重传的控制消息非常丰富，同时支持 ACK、ACKACK、NACK。我们都知道音视频对于时间这一点非常在意，而 SRT 基于时间的报文发送，使其具有良好的防止流量突发的能力。SRT 对上层提供了丰富的拥塞控制统计信息，包括 RTT、丢包率、inflight、send/receive bitrate 等。利用这些丰富的信息，我们可以实现带宽预测，并根据带宽的变化在编码层去做自适应动态编码与拥塞控制。

2. SRT 协议原理分析

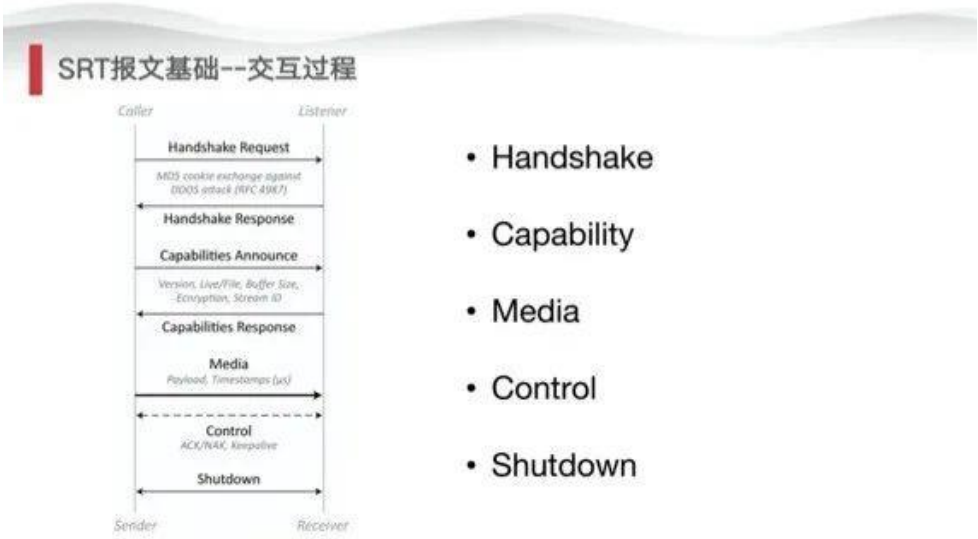
2.1 SRT 基本思想



上图可以涵盖 SRT 的基本思想：对比编码后的视音频码流（左侧绿色折线“Source”）与经过公网传输后的码流（红色折线“NetworkTransmission”），可以看到编码后的源视音频码流具有固定的帧间隔与一定特性的可变比特率，但经过公网传输后的码流，其帧间隔变得不固定且码率特性也被完全改变，解码这样的信号是一项十分艰巨的挑战，甚至根本无法解码。而如果使用加入纠错的 SRT 协议进行公共互联网传输，尽管编码后的视音频码流经过公网传输帧间隔变得不固定，但由于 SRT 协议封装中包含精准的时间戳，解码接收端可以通过该时间戳重现固定的帧间隔。更重要的是，通过规定延时量，同时划定了 send buffer（发送端缓冲区）与 receive buffer（接收端缓冲区），来自接收端的反馈信号（可以理解为后向纠错）通过一系列的设置、纠错、流量控制，使编码后的视音频码流拥有与原码流几乎一样的码率特性。Encoder 编码器处理完成的数据会被输入 send buffer，send buffer 会对数据进行时间校验，也就是按照一定的时间顺序编码并通过 internet 发送至 receivebuffer，receive buffer 按照报文上的时间戳一点点处理，确保生成的数据与源码流基本一致。整个 SRT 协议的思想基于编码，具有抗丢包、抗拥塞、抗抖动等特性。

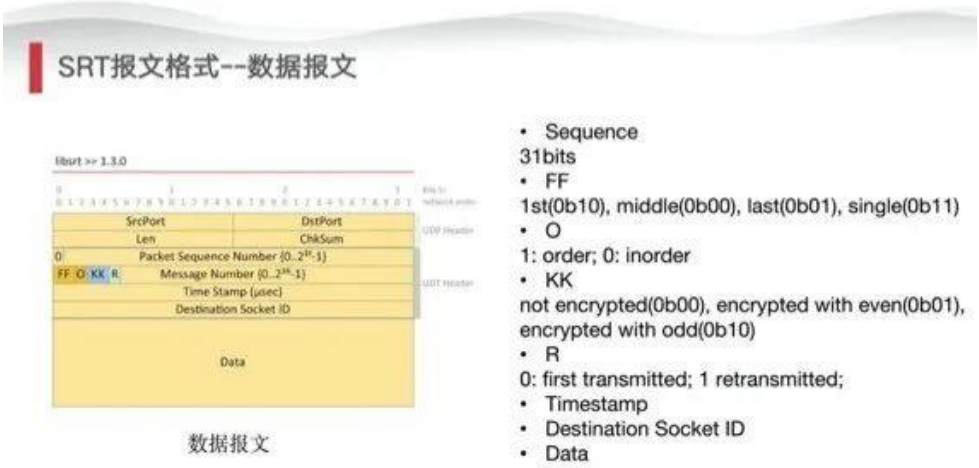
2.2 SRT 报文基础

交互过程



SRT 的报文基础也就是其交互过程依次为以上五个步骤：握手（Handshake）、重要信息（Capability）、媒体（Media）、控制信息（Control）与关闭（Shutdown）。与 RTMP 的九次握手不同，SRT 的建连过程仅需两个 RTT。也就是 Handshake Request 与 Capability Announce。Caller 与 Listener 之间的进行的 Handshake Request 是一个简化版的握手，用以提高效率；握手之后 Caller 与 Listener 间进行重要信息的交换，整个 SRT 的第一项关键步骤就是 Caller 与 Listener 在这里交换了延时量与缓冲区的大小；随后 Caller 与 Listener 之间开始媒体数据的传输，同时也传输了为恢复帧间隔而封装的精准时间戳；SRT 的第二项关键步骤是 Listener 向 Caller 同步控制信息，用以对抗公网中的抖动、丢包等一系列突发状况；最后，待至媒体数据传输完毕之后关闭传输。

数据报文



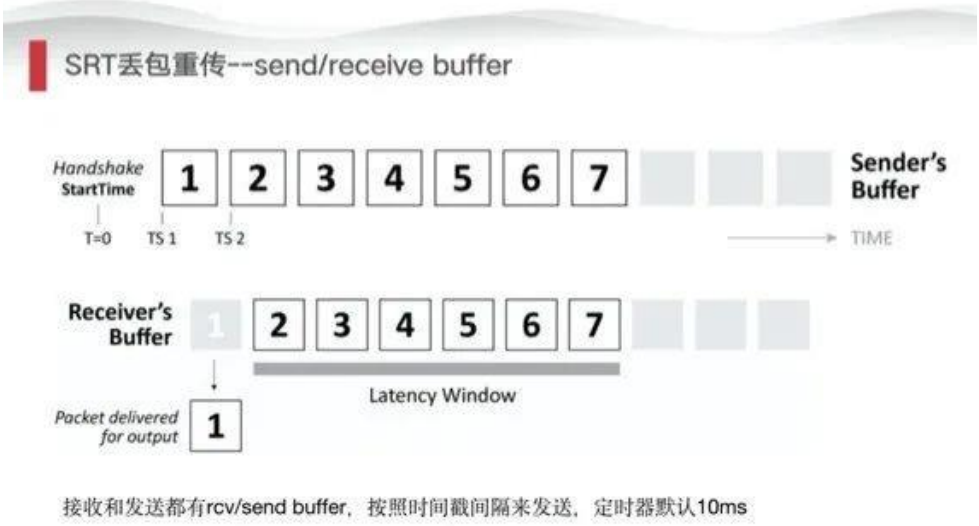
SRT 的报文格式较为简单，分为数据报文与控制报文。上图展示了数据报文的数据结构，观察结构我们可以发现上方两层为 UDP 部分，而在这之下是 UDT 部分。如果初始化为 0，则认为其是数据报文。FF 表示报文的序列， 0b10 是分片的第一个报文，0b00 是分片的中间报文，0b01 是分片的最后一个报文，0b11 表示单个报文并没有分片。KK 表示是否加密，R 表示是否属于重传报文，Timestamp 表示时间戳，Destination Socket ID 是 SRT 自定义的一个 socket id。由该数据结构我们可以看出：SRT 的数据报文拥有精准的 32 位时间戳，package sequence number 绝对够用，且结构非常简单。下图展示的则是控制报文的数据结构，右侧表格展示了控制报文的常规类型，其中值得重点关注的有 ACK、NACK、ACKACK 等。

控制报文

Type	Extended Type	Description
0	0	HANDSHAKE
1	0	KEEPALIVE
2	0	ACK
3	0	NAK (Loss Report)
4	0	Congestion Warning
5	0	Shutdown
6	0	ACKACK
7	0	Drop Request
8	0	Peer Error

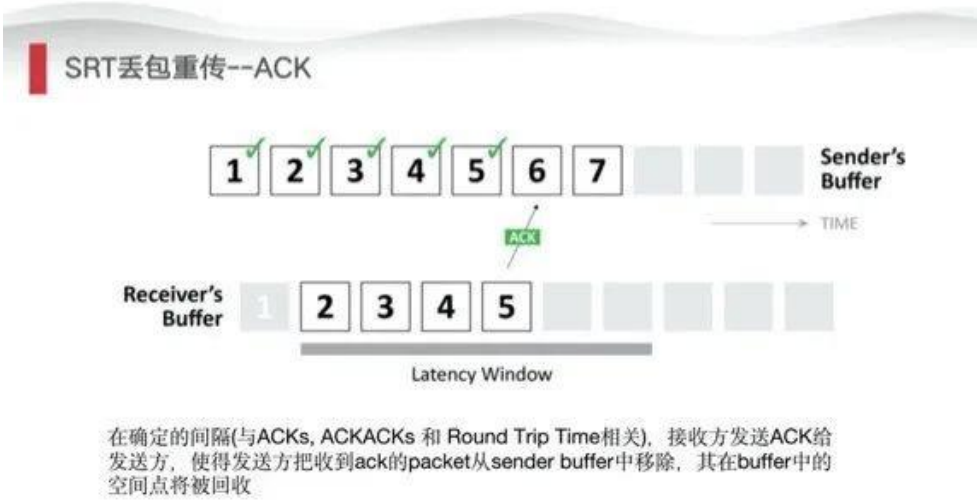
2.3 SRT 丢包重传

2.3.1 Send/Receive Buffer



SRT 在接收和发送端都有 receive buffer 与 send buffer，send Buffer 严格按照时间戳间隔来发送，定时器默认 10 毫秒。

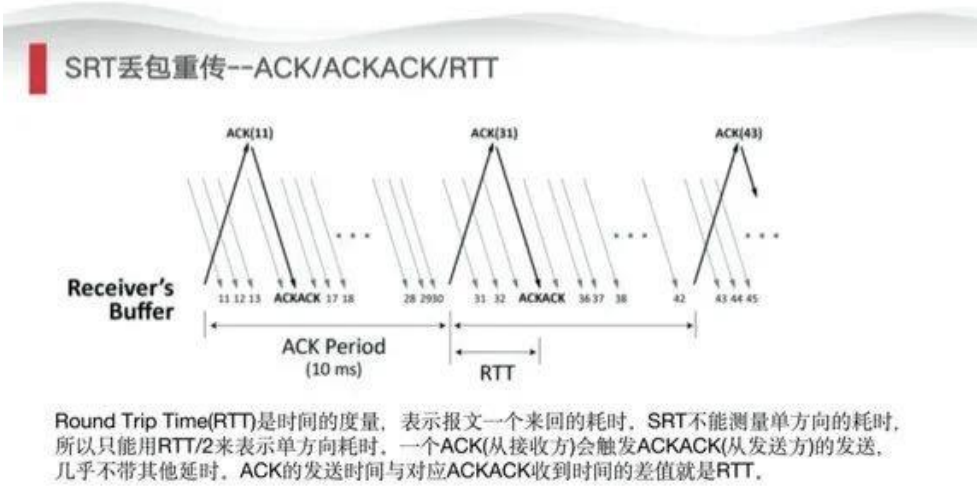
2.3.2 ACK



丢包重传最常见的便是 ACK 机制。以上图为例，假设发送端缓冲区发送五个数据包：1、2、3、4、5 给接收端缓冲区，接收端缓冲区成功接收到这些数据包之后会向发送端缓冲区发送 ACK——肯定应答已表示数据包被成功接收，发送端接收到 ACK——肯定应答之后回收空间，删除 1、2、3、4、5 这五个数据包并准备发送数据包 6。

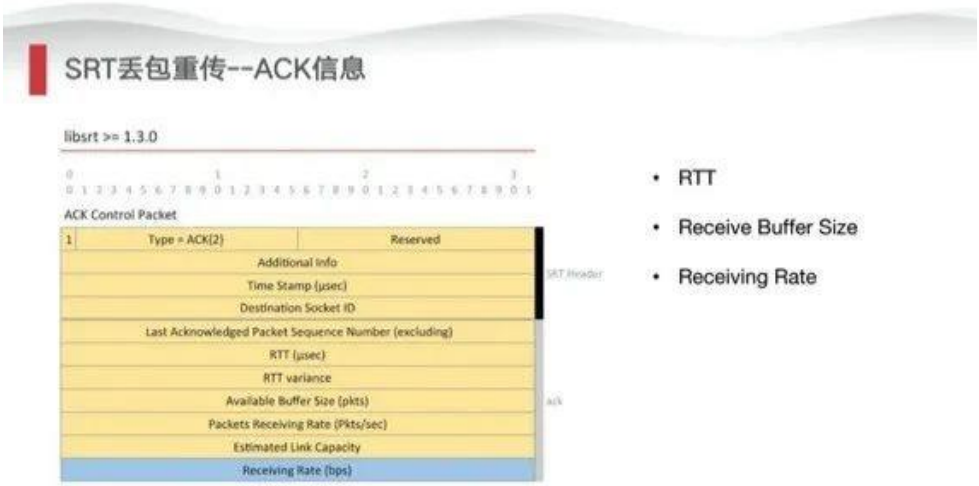
send buffer 完全按照时间戳间隔处理数据，在确定的间隔(与 ACKs,ACKACKs 和 Round Trip Time 相关)，接收方发送 ACK 给发送方，使得发送方把收到 ack 的 packet 从 sender buffer 中移除，其在 buffer 中的空间点将被回收。接收端的 receivebuffer 中存在 Latency Window 也就是延迟窗口，其作用为按照时间戳一点点上送数据。并严格按照时间戳检测，检测周期默认为 10 毫秒。

2.3.3 ACK/ACKACK/RTT



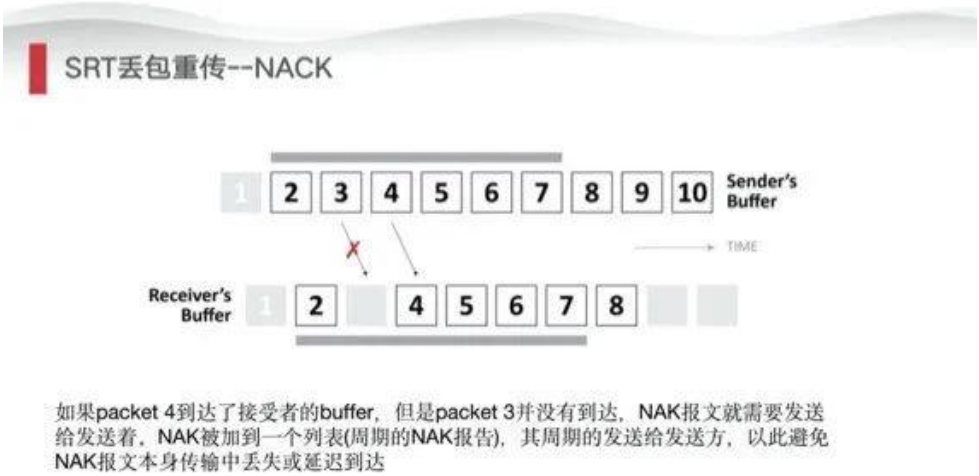
接收端在收到数据包后会向发送端反馈表示成功接收的 ACK，例如上图左侧，接收端在收到第十一个数据包后向发送端反馈 ACK(11)，而发送端在收到来自接收端的 ACK(11)之后会向对端再发送一个 ACKACK 用以表示收到 ACK(11)。这一过程最大的意义在于可让接收端计算出 RTT，ACK 发送的时间与对应 ACKACK 收到的时间之差就是 RTT——Round Trip Time(RTT)是时间的度量，表示报文一个来回的耗时。SRT 不能测量单方向的耗时，所以只能用 RTT/2 来表示单方向耗时。一个 ACK(从接收方)会触发 ACKACK(从发送方)的发送，几乎不带其他延时。RTT 可评估当前网络质量，RTT 高表示整个网络的延迟很大，RTT 低则说明网络延迟较低。RTT 由接收端计算而出，计算得出的 RTT 会通过 ACK 发送至发送端，发送端就可获知当前网络的质量如何。带宽情况是不断变化的，因此 RTT 也是一个随网络环境不断变化的实时动态值。

2.3.4 ACK 信息



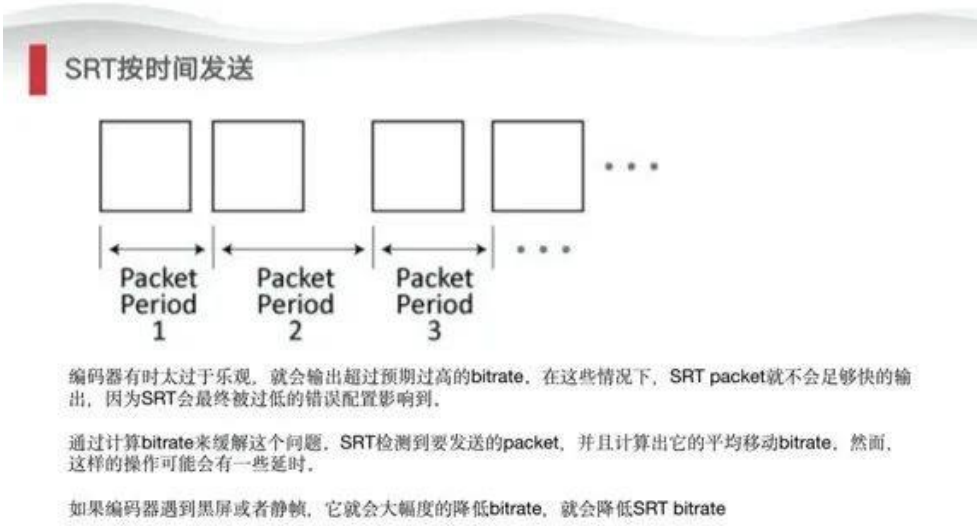
ACK 报文包含了丰富的关键信息。其中 Last Acknowledge Packet Sequence Number 表示当前收到第几个包，RTT 信息便于发送端评估网络的质量，RTT 的变化表示网络变化情况。Available Buffer Size 表示接收端缓存可用率，SRT 是可以用作文件传输的，Available Buffer Size 对文件传输的拥塞控制非常有用。当文件传输过快时接收端的内存缓存无法成功接收，此时发送端除了考虑网络还需要考虑接收端的性能与缓存是否有足够的空间与能力在短时间内接收庞大的文件。Packets Receiving Rate 表示每秒钟的收包率，这里统计的是包的个数；而 ReceivingRate 表示接收包的比特率。总而言之，接收端以 10 毫秒为周期向发送端发送 ACK，其中包括网络 RTT 信息、接收端缓存信息、接收端比特率等关键数据，其中以上三个数据至关重要，直接反映出发送端到接收端之间的网络环境状态。

2.3.5 NACK



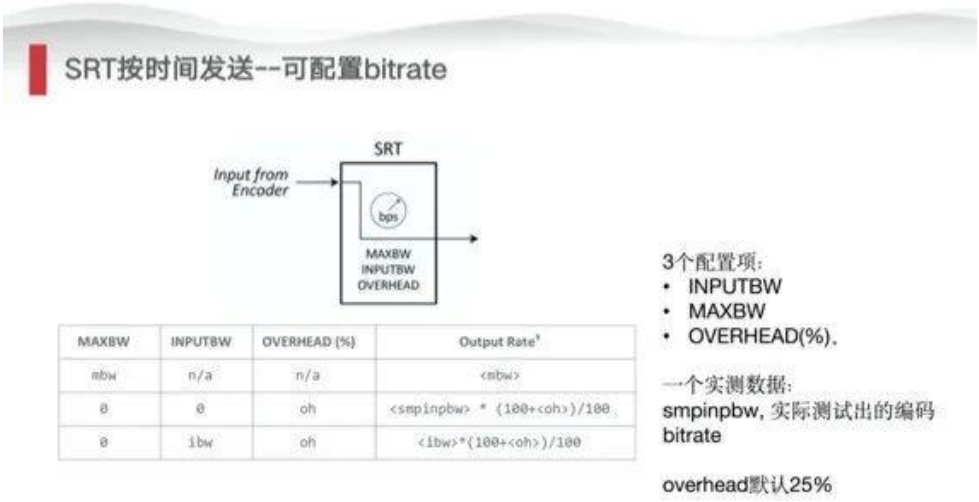
常规情况下 QUIC 或 TCP 仅提供 ACK 方式，也就是接收端向发送端反馈哪些包成功接收；而 WebRTC 的 RTP 或 RTCP 常用选择 NACK，也就是接收端向发送端反馈哪些包没有成功接收。NACK 回传的是接收端没收到的数据包的列表。SRT 同时支持 ACK 与 NACK，这样设定的原因在我看来是带宽抢占的强势。例如，接收端向发送端发送 ACK 表示该数据包已经成功接收，不排除该 ACK 报文本身丢失，导致发送端以为数据报文丢失而要超时重发，实际接收端对该数据包已经成功接收了。还有一种情况是：NACK 的周期性发送可能会造成发送端多发报文，例如发送端成功发送 5 号与 6 号数据包，却一直未收到来自接收端的 ACK 消息，当达到重发的时间时发送端再次发送 5 号与 6 号数据包，在此之后发送端收到了来自接收端的 NACK 消息，表示接收端未成功接收 5 号数据包与 6 号数据包；于是发送端第三次发送 5 号数据包与 6 号数据包。一个周期下来，发送端发了两次数据包。从协议原理角度分析，我们发现 SRT 在丢包过程中对带宽的消耗比 QUIC 与其他协议要高。 一旦出现丢包，SRT 的重传要多于其他协议，SRT 以此来抢占带宽从而达到音视频的同步效果。

2.4 SRT 基于时间发送



按时间发送是一个标准的音视频传输特性。我们知道编码器发送是以编码器的速度向外发送数据，但有时编码器会太乐观，完全按照编码输出向外发送会导致输出超过预期过高的比特率。在这种情况下 SRT Packet 就不会足够快地输出，因为 SRT 会最终被过低的错误配置影响到。

2.5 可配置比特率



SRT 中包含三个配置选项：INPUTBW 表示编码器输入带宽，MAXBW 表示最大带宽，以及 OVERHEAD(%)表示过载率。配置原则如上图所示：如果不配置 INPUTBW 与 OVERHEAD(%)而仅配置 MAXBW，则当前编码器输出的比特率是也就是 MAXBW；如果不配置 MAXBW 与 INPUTBW 而仅配置 OVERHEAD(%)，则当前编码器输出的比特率是(1080+)/100，默认为 25%；第三种情况是不配置 MAXBW 而配置 OVERHEAD(%)与 INPUTBW，最大输出比特率便为(100+)/100。SRT 最大的特点便是可配置。在有配置的情况下按照配置来做，在无配置的情况下按实际测量的比特率来做。常规情况下我们不选择进行配置，尤其是针对互联网而言，因为单一配置无法保证能够在不同时段应对不同境况的网络。通常我们选择使用实际测量出的编码比特率计算：*(1080+)/100。

2.6 SRT 简单拥塞控制——简单，太简单

接下来我们讨论一下 SRT 的流控。我们知道在丢包重传机制下，如果报文在网络传输中丢失，则发送端会重新发送。例如在某网络环境下发送端与接收端之间的带宽仅有 1M，现在由于背景流量与噪声等影响，原来 1M 的带宽减少 100k 变成了 900k，此时就会出现 10%的丢包；发送端未收到来自接收端的 ACK 或者收到 NACK 认为出现丢包，于是尝试重传这 10%的数据；但实际上发送端与接收端之间的带宽就剩下 900k 了，而加上丢包重传的发送量是 1.1M，带宽变窄发送的流量不降反增，就会导致网络情况越来越糟糕，最后造成网络拥塞卡顿频繁，最后网络崩溃。SRT 单纯地丢包重传不能解决网络拥塞的问题，出现网络拥塞的最好解决方案是限流，降低带宽流量压力。

SRT 的拥塞控制过于简单，仅在 congctrl.cpp 中实现。其包括以下几个变量：M_iFlowWindowSize 表示接收方的缓存大小，m_dCWndSize 等于 1 秒内发送的 bytes 数据 / (RTT + 10)，sendbufer 表示发送方未发送 buffer 大小。其中这里的 1 秒内发送的 bytes 数据，具体是指一个 RTT 内发送的数据。总结算法如下：m_iFlowWindowSize 用以衡量接收方缓存是否耗尽，同时监测发送方每 RTT 发送数据 size 是否大于发送 buffer 大小。接收方缓存一般在传输文件时起作用。在我看来，SRT 在拥塞控制方面做得还远远不够。

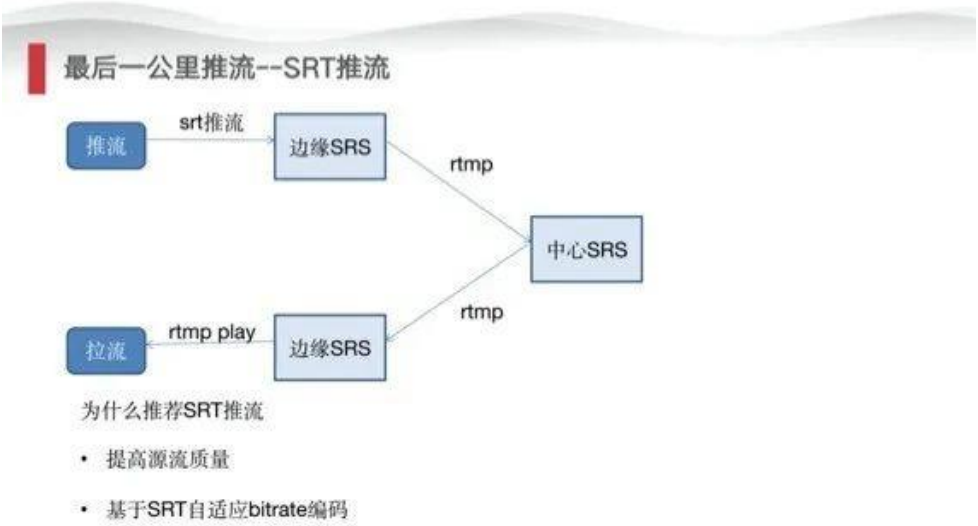
2.7 SRT 协议总结

SRT 在快速连接方面有明显优势，两次握手成功即可建连；SRT 的丢包重传策略出色，ACK、ACKACK、NACK 等提供了丰富的控制消息，还有 RTT、Receive 比特率等。但是同时我们经过测试也发现 SRT 在丢包时，发送数据的带宽占用率还是有些大，丢包率越高发送带宽占用越大。SRT 按照时间发送音视频数据，根据实际的编码比特率来发送音视频数据；但 SRT 的拥塞控制过于简单，只针对接收方缓存是否有能力接收与编码比特率是否发送过快。

3. SRT 在 SRS4.0 中的方案用

3.1 SRT 在 SRS4.0 中的方案应用

3.1.1 最后一公里——SRT 推流



我们推荐从编码器到推流至边缘节点的部分使用 SRT。其目的主要有两个：提高源流质量与基于 SRT 自适应比特率编码，发送端和接收端配合从而解决最后一公里的推流问题。是自己对 srt 的理解。SRS 支持 SRT 的接收端推流，边缘 SRS 在接收到来自推流端的 SRT 推流之后会将其转为 RTMP 并分发给中央节点，而后所有的边缘节点都可以进行 RTMP 拉流。

3.1.2 SRT 地址格式

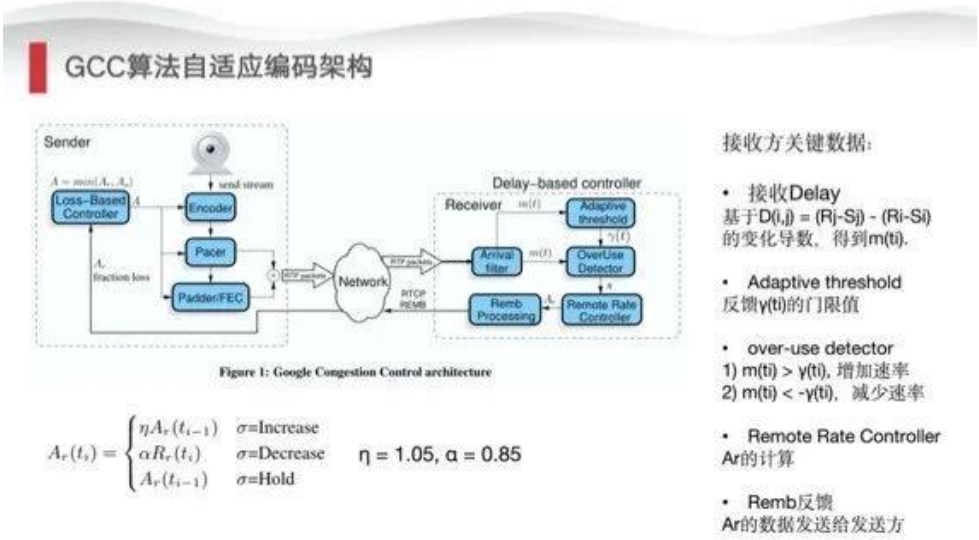
作为一个传输协议，SRT 的一个弊端在于给出一个未定义的地址，我们不清楚这究竟是推流地址还是拉流地址，那么如何进行匹配？

为方便SRT编码器推流，SRS4.0支持编码器的简单配置。如服务器IP、服务器Port以及streamid。根据 SRT 的官方文档，SRT 通过 StreamID 进行标识，简配地址格式如上图所示，带有 vhost 虚拟主机配置的地址格式如下图所示，其中 m=publish/request 表示推/拉流地址。显而易见的是，对比 RTMP，SRT 地址的可读性并不好，地址长而冗杂。

3.1.3 SRT 各种网络情况下的测试

测试各种网络情况下的 SRT 我们不难发现，丢包率增加导致带宽消耗增加，网络状况不良或发生拥塞时，发送端会发送更多的数据，这便会导致网络状况愈发恶化，丢包率变得更高，并以此恶性循环；除此之外，RTT 增加也会导致延时增加，一样会导致丢包率增加，带宽消耗更大。这里我们提出的解决方案是预测网络带宽——通过当前的 send bitrate、RTT、inflight 等数据预测网络带宽；同时动态调整编码比特率，根据预测的带宽动态调整编码比特率 1 来适应实时的带宽，避免发生拥塞并提高视频流畅度。

3.1.4 GCC 算法自适应编码架构



上图展示的就是谷歌拥塞控制算法 GCC 的架构图。如图所示：发送端将报文发给接收端，接收端由几个部分组成，其中计算接收报文 Delay，也就是计算基于 D(i,j) = (Rj-Sj) - (Ri-Si) 的变化的导数，得到 m(ti)。除此之外，接收端设置的卡曼滤波算法会计算出一个门限值，通过该卡曼滤波的计算与比对，决定是增加码率还是降低码率，最终得出下一次网络带宽的预估值（Ar），并将数据返回给发送方。

3.2 自适应码率的 SRT 推流

3.2.1 基于 SRT 自适应码率编码

基于 SRT 自适应码率编码的关键参数如上图左侧所示：rtt_min 表示 1s 内的 RTT 最小值，send_bitrate_max 表示 1s 内最大的发送 bitrate，常规情况我们大概 200 ~ 300 毫秒统计一次；

inflight 表示已经发送出去但还未接收到 ACK 的报文的个数，这种情况可能意味着报文还在传输链路的途中；BDP(bandwidth-delay product) BDP = send_bitrate_max*rtt_min 表示一个 RTT 内发送的最大字节数。如果 BDP 大于（1.2 x inflight）表示网络状况良好，可以增加编码码率；如果 BDP 小于（0.8 x inflight）则意味着网络状况不佳需要减少编码码率，其他情况则维持现有编码码率不变。这样便能较为有效地避免网络拥塞的情况发生。

3.2.2 基于 SRT 自适应码率编码——实例

我们以测试实例来验证效果：通过 Internet 推流，来自于美国的 SRT 编码器的数据推至杭州的节点，建立 SRS4.0。该测试开源地址如图中所示，使用 FFmpeg 配置编码码率为 1000kbps，传输过程中根据实际出口带宽动态调整编码码率。由上图右侧图线 1 我们可以看到自适应码率最高可达到 1400kbps，有良好的自适应效果；实际测试感受来看，视频播放平滑无明显卡顿，即便偶尔出现带宽抖动也能够迅速恢复。

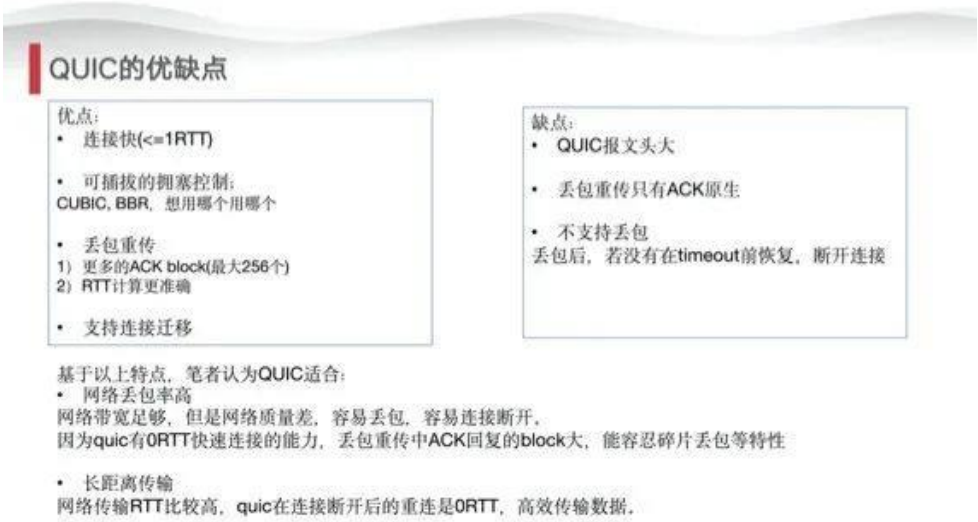
4. SRT 与 QUIC

接下来我们对比 SRT 与 QUIC，总结二者特点。

4.1 SRT1.4 的优缺点

SRT1.4 的优缺点可以简单概括为以下内容：SRT 的优点在于基于音视频按照时间戳进行收发，可有效保证音视频，同时 ACK/ACKACK/NACK 多种丢包纠正机制可有效降低延时与丢包率。SRT 对上层提供丰富的传输层数据信息如 RTT、lost packet rate、receive rate 等。当然，SRT 的缺点也不容忽视，如 SRT 的拥塞控制过于简单，需要在传输层合入 BBR 算法，原生 SRT 不支持连接迁移等。基于以上特点，我认为 SRT 更加适合编码器到最近节点的传输，也就是通过 SRT 探测出的 RTT 等相关信息实现自适应码率编码；除此之外，SRT 也适合网络节点固定、网络情况固定的环境，合理配置 lantency、send/recv buffer、ohead rate 等 SRT 参数可达到事半功倍的效果。

4.2 QUIC 的优缺点



QUIC 的优点是连接快，同时具有可插拔的拥塞控制，包括 CUBIC、BBR；另外在丢包重传方面，QUIC 拥有更多的 ACK block，最大可达 256 个，并且对于 RTT 的计算更加准确（QUIC 中存在独立的 Packet number，包括重传包在内的 Packet number 也都不相同，非常有利于 RTT 的精确计算）；最后还有 QUIC 支持连接迁移。当然，QUIC 同样存在缺点：第一，报文头大在发送报文中所占比例较高。第二，丢包重传只有 ACK 原生；第三，QUIC 不支持丢包，一旦出现丢包，若没有在 timeout 前恢复就会断开连接。基于以上优缺点，我认为 QUIC 更适合运用在网络丢包率较高的环境，因为 QUIC 具有 0RTT 快速连接能力、丢包重传中 ACK 回复的 block 较大并且在拥塞控制方面表现非常优秀。除此之外 QUIC 也适合用于长距离传输当中，因为网络传输 RTT 较高，QUIC 在连接断开后重连是 0RTT，传输数据更加高效。

4.3mediago 服务：rtmp over quic

Mediago 具有支持 QUIC 协议来传输 RTMP 直播流的特性，如 RTMP over TCP 推拉流、RTMP over QUIC 推拉流以及对 FLV 的支持。服务器之间则支持基于 TCP 服务器间 RTMP 回源与基于 QUIC 服务器间 RTMP 回源，上图中有测试链接，感兴趣的朋友可以自己尝试一下。

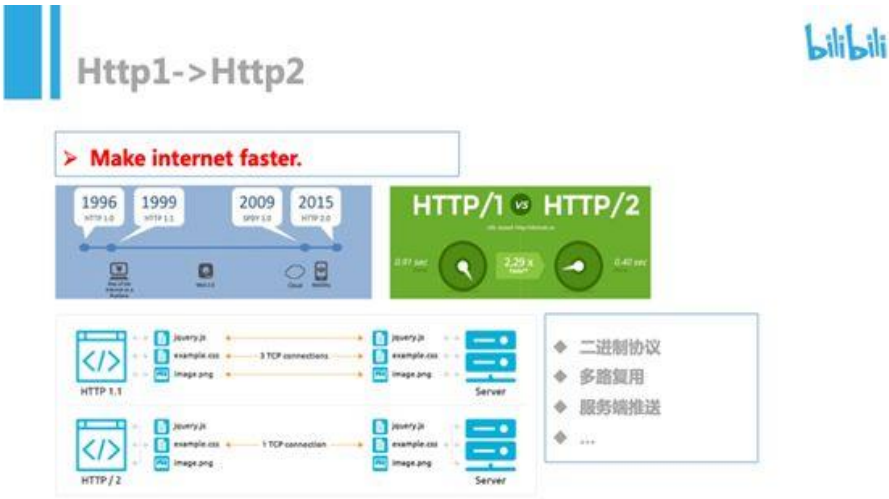
B 站 QUIC 实践之路

QUIC 协议快速建立连接、拥塞控制灵活、多路复用等特性为网络传输带来了不少收益，也更好的提升了用户的体验。哔哩哔哩高级工程师 王盛在 LiveVideoStack 线上交流分享中介绍了 B 站在 QIUC 研发与部署中的实践经验，LiveVideoStack 对分享内容进行了整理。

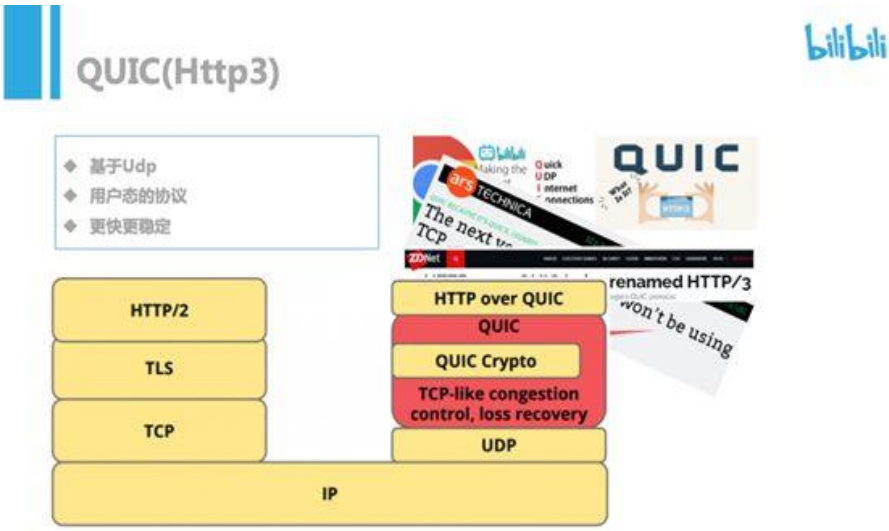
王盛　LiveVideoStack 线上分享　2019-01-28

大家好，我是来自 Bilibili 的王盛，五年多来我一直从事网络协议和点直播系统的研发工作，加入 B 站后主要负责 Bilibili 的点直播系统开发，可以说见证了 B 站的点直播系统从零发展到目前 TB 级别的整个过程。接下来我将从以下四个方面为大家分享 B 站在 QUIC 协议上的技术探索，希望大家能通过此次分享了解更多 QUIC 特性与服务端架构技术细节。

1. 从 Http1.1、Http2 到 QUIC



我们讨论 QUIC 协议一定先从 HTTP 协议出发。作为目前互联网上被使用最多的网络协议，HTTP 经历了从 1996 年诞生到现在二十多年的发展，相对于早期的 1.0、1.1 版本，现在的 2.0 版本可以说发生了非常大的变化。据第三方机构统计，2015 年 HTTP/2 版本的更新不仅加入了如二进制协议、多路复用、服务端推送等诸多新特性，更是实现了相对于 HTTP/1 近两倍的性能提升。



而我们今天讨论的 QUIC 协议其实就是所谓的 HTTP/3，Google 从提出 QUIC 协议以来一直想将其作为优化与替代现行 HTTP 协议的解决方案。从上图我们可以看出 QUIC 是完全基于 UDP 协议实现的，我们知道现在主流的两大互联网传输协议是 TCP 与 UDP，前者连接可靠的数据传输而后者连接不可靠的数据传输；

如果 QUIC 协议建立在 UDP 协议之上就需要实现一系列与 TCP 相似的数据传输操作如拥塞控制、数据重传、数据排序等，使得基于 UDP 的 QUIC 成为一种用户态的协议。目前 TCP 协议实现大部分功能都离不开内核，这就导致如果开发者希望修改调整 TCP 协议中的某一些特性与功能或根据需求进行定制化开发部署的难度非常大，尤其是涉及到内核改动的工作会带来不必要的开发成本；除了具备用户态协议的优势，为了实现更快的数据传输，QUIC 也囊括了一些类似于 TLS 安全传输层的功能，为了提高兼容性 QUIC 也提供与之前 HTTP2 一样 HTTP 语义，从而实现在应用层程序方面良好的兼容性。

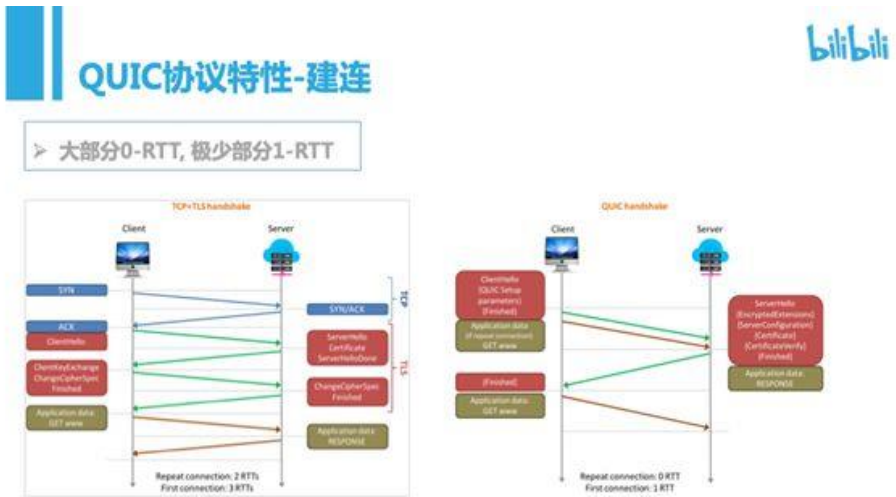
QUIC 协议从诞生至今获得了不少厂家与企业的支持，也为这些企业带来了收益与回报。纵览整个行业，国外如 Google 搜索、YouTube 视频、Akamai CDN 等等都实现了对 QUIC 协议的支持，国内如腾讯云的负载网关、QQ 空间、七牛直播云的直播推流等都在 QUIC 协议的帮助下实现了如 0 卡顿的直播推流等等技术提升，而 B 站的点直播 CDN 也实现了对 QUIC 的全面支持。QUIC 为这些企业带来的收益同样不容小觑，如为 Google 搜索页面带来了 5%的加载速度提升，使 YouTube 的视频卡顿率降低了 30%；为 QQ 空间带来了 10%的加载速度提升，帮助 B 站提升 10%视频首帧的同时减小 5%的卡顿，可以说极大提升了互联网产品的用户体验。

接下来我们将 QUIC 和 TCP 之间的性能与参数差异落实在用户体验层面，首先模拟用户同时使用分别基于 QUIC 与 TCP 的两个播放器播放同一个视频文件，理想环境下用户的直观感受是二者画面基本同步流畅；但如果使用网络模拟工具为用户播放器端到视频服务器端之间的网络传输增加 20%的丢包，此时基于 TCP 传输协议的播放器会出现明显卡顿而基于 QUIC 传输协议的播放器则仍可保持较为流畅的播放状态，二者会出现 5～10s 的播放时间差，不难发现 QUIC 在弱网环境下的传输性能优势非常明显。其次我们模拟增加每个数据包的延迟如增加到 200ms，随后使两个播放器从同一帧开始播放，不难观察到基于 TCP 协议播放器的画面明显滞后，QUIC 的首帧展示也更快。

2. QUIC 协议特性

无论是弱网环境的卡顿还是首帧展示，QUIC 相对于 TCP 的实际传输性能提升都十分明显。这与 QUIC 协议本身的一些特性紧密相关，主要包含以下四部分：

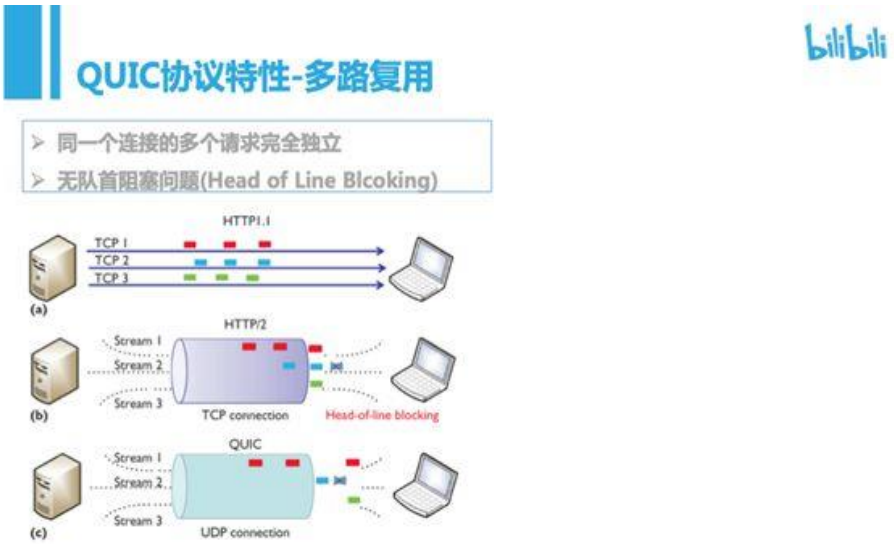
2.1 极短的建连时间



在 QUIC 的建连时间中大部分为 0-ORTT，极少部分是 1-RTT，如何实现这样的分配？首先大家知道，TCP 的一个建连包含三次握手，而如果基于 HTTPS 加密则还需包含 TLS 一层的一个握手，同时增加 1 RTT 的时间；综合来看，已完成建连的 TCP 连接进行握手大概需要 2 RTT，而首次建连的 TCP 则需 3 RTT，其中便包括 TLS 的证书交换。但对于 QUIC 协议，若客户端之前未建连，其第一次建连时需在客户端直接生成证书与和协议栈相关的配置且附带一个 ID，这些数据会与请求数据一起直接发送并保存在服务端，假若服务端已存在以上数据那么系统会直接进行校验操作并直接回复数据；

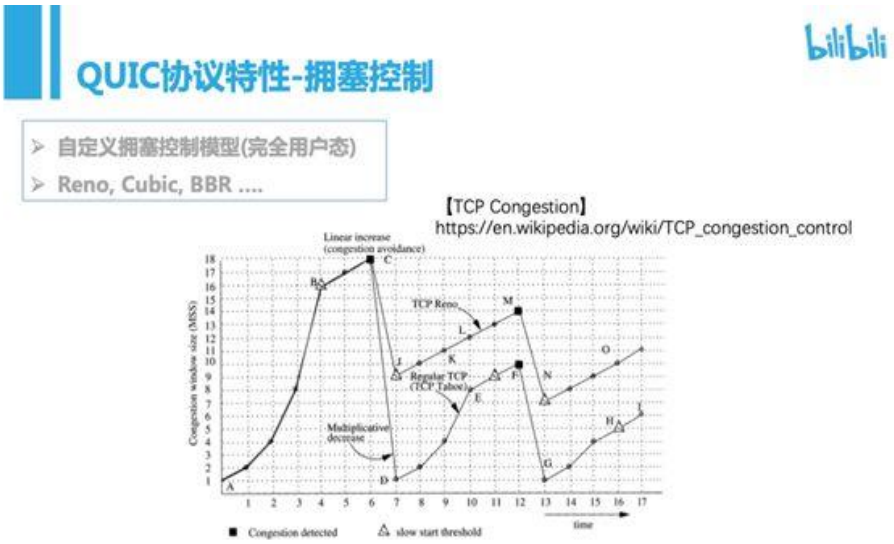
而已建连的客户端则仅需 0-RTTs 时间，初次建连的 QUIC 服务端为便于进行建连与校验会把 ID 保存在服务端本地，待服务端完成校验即可发送证书至客户端并进行客户端校验，校验成功即可直接启动数字交换；且对于第一次建连的 QUIC 而言只需 1 RTT 就可完成建连并允许后续的数据交互，以上就是 QUIC 的建连时间特性——大部分 0-RTT、极少部分 1-RTT。

2.2 多路复用



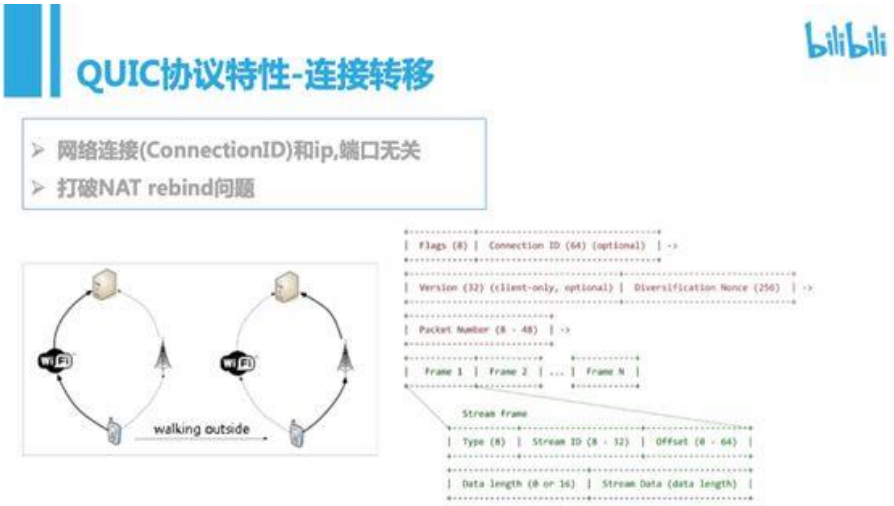
QUIC 协议的第二条特性为完全多路复用，以 HTTP1.1 为例，每条数据流基于一个 TCP 连接，每个 TCP 连接都单独传输数据，但此 TCP 连接方案会明显增加服务端与客户端的并发负载；HTTP/2 对此问题进行了有效优化也就是采用多路复用的传输策略，通过一条 TCP 连接传输多路数据，但此方案容易造成队首阻塞问题：以上图展示的 HTTP/2 数据流为例，若 Stream 2 丢失那么 Stream 1 与 Stream 3 都会被阻塞，直到丢失的 Stream 2 数据重传完成之后 Stream 1 与 Stream 3 才能被继续传输，这种因为一条数据流的丢失重传等问题造成其他多条数据流阻塞的现象被称为队首阻塞问题；QUIC 对此问题进行了修正，也就是把每个重传过程安排在每条 Stream 中单独完成，由于 Stream 本质上是一个基于 UDP 的小数据包，所以这种方案并不会造成队首阻塞问题。

2.3 灵活的拥塞控制



QUIC 的第三个特性被称为灵活的拥塞控制，QUIC 能够在操作系统层面实现完全用户态得益于协议栈非常灵活的拥塞控制。如我们之前熟知的被用于 TCP 的 BBR 拥塞控制算法也是一开始在 QUIC 上实验成功之后才被移植到 TCP 内核中。当然这种将模块移植进内核的尝试是一个非常漫长的过程，但为了实现较为良好的拥塞控制并达到用户态要求，此方案不失为一种良策。

2.4 连接转移



QUIC 的最后一个特性被称为连接转移。什么是连接转移？假设当你正在家中使用 WLAN 观看视频时需要外出办事，当你走出家门时手机也会从通过 WLAN 下载数据改为通过蜂窝网络下载，在 WLAN 与蜂窝网络切换的瞬间 TCP 连接会被切断再重连，这是因为 TCP 采用包括原

IP、目标 IP、原端口、目标端口在内的四元组方式表征一个连接；QUIC 并未采用四元组的表征方式，而是将数据包中一个 64 位的数值 ID 作为表征，无论 WLAN 与蜂窝网络如何切换，只要发送至的服务端数据当中的 ConnectionID 一致，服务端就会默认其为同一个连接，从而避免出现切断重连的过程。

3. 视频云的 QUIC 实践

3.1 Roadmap

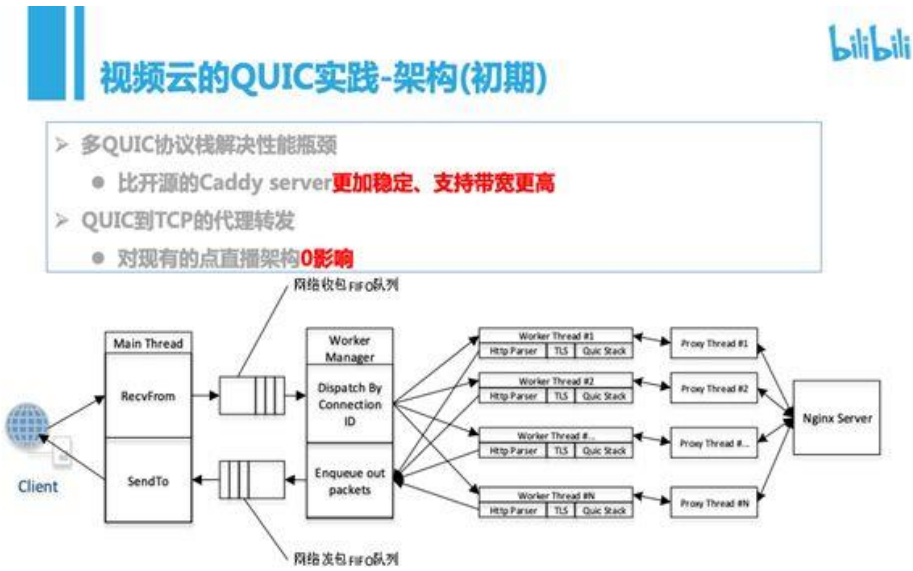
接下来我将重点介绍 B 站开发 QUIC 协议的实践经验。我们在开发 QUIC 协议之前，遇到了以下两个难点：首先，作为一个正在开发中的协议草案，QUIC 协议标准本身需要频繁进行迭代更新，大约 1.5 个月更新一个版本。如现在被广泛使用的 QUIC43 就是不到一年前从 QUIC37、35 或 39 等版本迭代而来，而不断跟进此协议标准的迭代升级确实是一件很难的事情；其次是开发 QUIC 网络协议本身是一个比较底层的实践，需要开发者具备较强的技术素养才能攻克其中诸多的技术难点。B 站已在服务端全面上线 QUIC，客户端的 QUIC 也正在灰度上线中。

3.2 技术选型

我们尝试探索多元化技术选型，通过一个季度的调研确定了以下几个可实现 QUIC 的常见开源项目：Libquic、Caddy&Quic-Go、Stellite&Isquic 与 Chromium。我们尝试借助这几个开源项目快速搭建并上线 QUIC 服务端，并仔细对比这几个方案的利弊：Libquic 虽然具有一些可在 Chromium 中实现的优势，但其早在两年前就已经停止更新；Caddy&Quic-Go 作为使用（……long）实现的单协议栈，其测试性能无法满足我们的上线标准；Stellite&Isquic 作为商业解决方案不但需要我们花大量成本进行部署，而且其协议栈的功能也不够全面；Chromium 作为 Google 官方的开源项目，虽然与 QUIC 协议标准始终保持一致的版本更新，但其接近两千多万行的庞大代码量明显增加了我们的运维压力。基于以上调研结果，最终我们决定使用 Chromium 快速搭建 QUIC 服务端，主要分为以下两步：首先使用 Chromium 的 QUIC 协议栈模块打包一些 API，其次是自主实现能够满足生产环境的 UDP Server，结合以上两者完成 QUIC 服务端的上线开发。

上图展示的就是我们在调研与探索过程中参考的资料来源。

3.3 初期架构



上图展示了 B 站在 QUIC 服务端实现的一个初期架构，为了与 B 站目前点直播系统的（NGX）体系兼容，我们仅将 QUIC 服务端作为一个代理接收来自公网的数据并转发至本机的（NGX ）服务器。由上图架构不难发现我们的工作重点是根据在主线程上进行收发的 UDP 数据包中的一些连接，将 ConnectID 分发至不同的工作线程。每个工作线程上都会运行一个 QUIC 协议栈，从而进一步发掘整个 QUIC 服务器的性能。需要注意的是，非单一的 QUIC 协议栈在处理网络 UDP 数据包或 QUIC 数据包时存在限制条件，也就是必须按照其 ConnectID 进行分发，否则就会出现一个连接进入两个协议栈的错误，继而导致数据丢失甚至服务宕机的状况。在此限制条件下此架构可实现如负载均衡等功能。

与此同时我们通过 QUIC 到 TCP 的代理转发，把 QUIC 的 HTTP2 的协议转发为（NGX）Server 使其能够为本地的（NGX） Server 服务并接收（NGX） Server 的回复，最终将（回复）转化为 UDP 包返回至 QUIC 客户端，这便是我们 QUIC 服务端的初期架构与细节。

3.4 使用方式

用户可直接使用内核支持 QUIC 的 Web 端如 Chrome、QQ 浏览器、Opera（内核）观看 B 站的视频；与此同时我们也会尽快灰度上线移动端的 QUIC 灰，为大家带来酣畅淋漓的 QUIC 高效传输体验。

4. QUIC 展望

展望未来的 QUIC，机遇和挑战并存。作为较早入局 QUIC 探索的开发团队，我们除了期待更优秀的适合视频传输的拥塞控制模型算法早日出现，也期待有更多建立在 QUIC 之上的协议如 RTMP 等的出现；当然，我们也面临着诸多挑战，如一旦 gQUIC、iQUIC 或 HTTP/3 成为标准，那么我们的协议栈就会面临大范围的改动，同时迭代更新也会增加我们的开发成本；其次，由于不存在流量控制，容易被用于流量攻击，UDP 包存在被运营商特殊对待的可能；最后是移动端的不同平台（如 Android、iOS）对 QUIC 的态度与支持力度，我们的愿景是与全行业志同道合者一起致力于 QUIC 的标准化，彼此分享研究技术亮点，共同努力使用户的音视频流媒体观看体验登上一个新的高度。

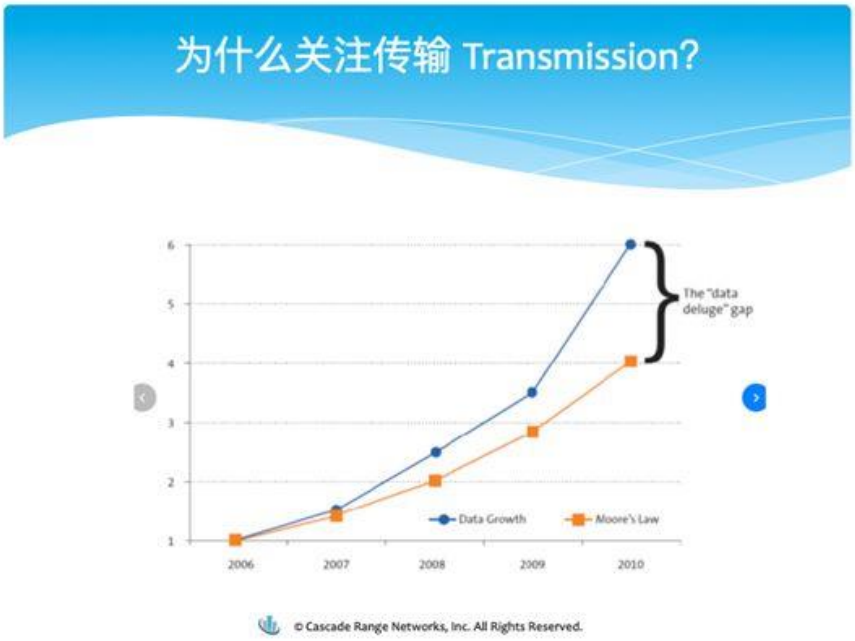
TCP 的困境与解决方案

TCP 协议是互联网应用最广泛的数据传输协议之一，在过去的 40 年中改变了世界，但也成为了新的技术瓶颈。Cascade Range Networks, Inc CTO/联合创始人 范醒哲在 LiveVideoStack 线上交流分享中详细解析了 TCP 面临的困境与可行的解决方案。本文由 LiveVideoStack 整理而成。

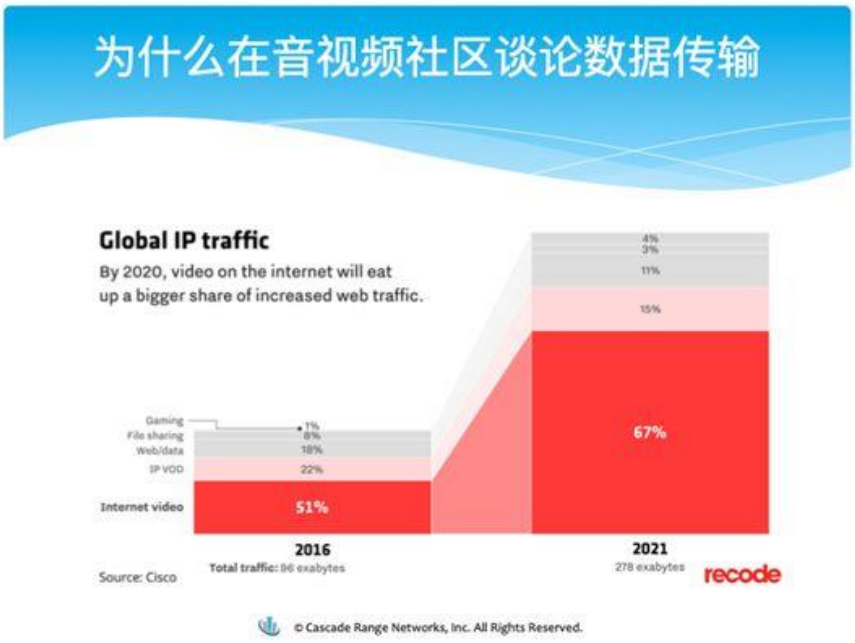
范醒哲　LiveVideoStack 线上分享　2019-01-21

大家好，我是来自 Cascade Range Networks 的范醒哲，本次我为大家准备的分享主题为“TCP 的困境与解决方案”。作为互联网使用最广泛的传输协议，TCP 带来巨大改变的同时，也面临一些亟待解决的问题，接下来我将围绕音视频行业与大家讨论以下相关内容：

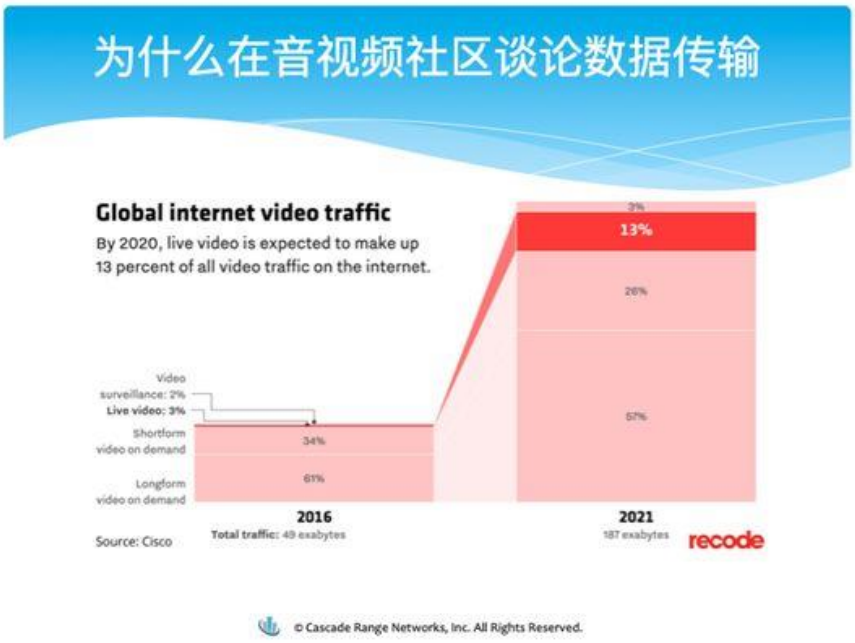
1. 为什么关注 TCP?



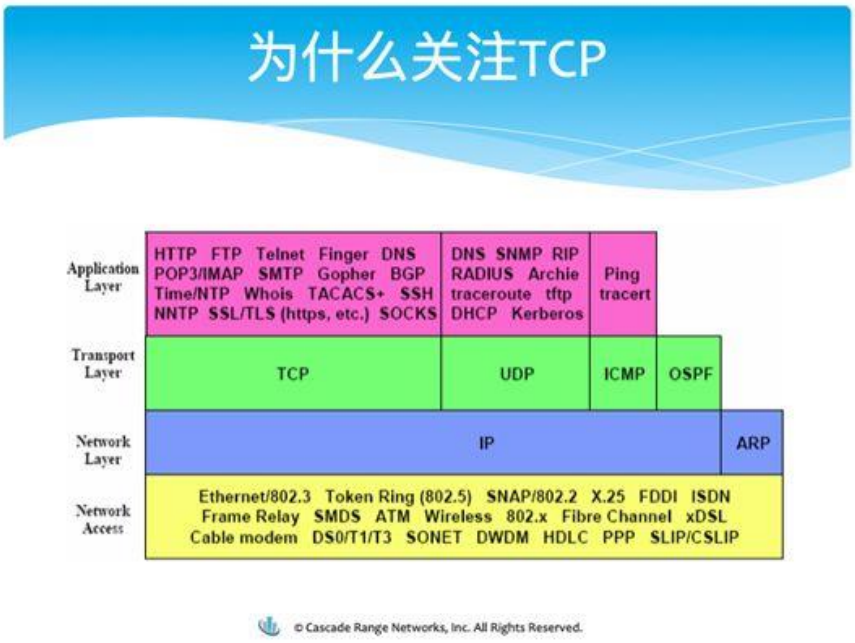
为什么我们需要持续关注传输问题？最根本的原因是数据量增长的速度远远超出带宽增长的速度。即使 5G 时代即将到来，传输问题依旧是技术实践当中的关键性命题。看似 5G 时代下强大的带宽会让传输问题迎刃而解，但在实践中不难发现数据的增长速度远远更快。随着 5G 时代到来的是超高清视频、3D、VR、AR 等数据量极大需要带宽更多的音视频应用，这就使得带宽成为一项技术瓶颈始终制约音视频行业的未来发展，我们需要一个能够妥善处理带宽问题的解决方案。



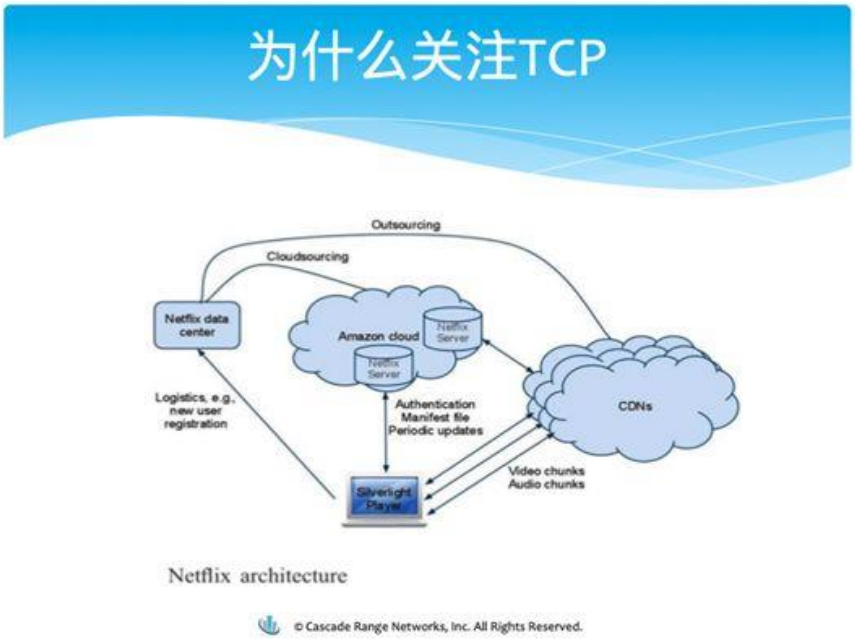
我们在音视频社区讨论数据传输，主要是因为数据传输虽然是一个网络概念，却与音视频技术中的各种技术存在一定相关性；而音视频数据现已经成为互联网中数据传输的主要对象，其占比预计会从 2016 年的 51%增长到 2021 年的 67%。



而 Live Video 方面则会出现 4 倍的增长，2021 年达到 13%，这使得我们不能不关心其未来发展。



即使在现在的音视频行业中有很多解决方案都是基于 UDP 协议，但作为一个承载互联网中大部分应用的传输协议，TCP 在可预见的未来依旧是最主要的协议。



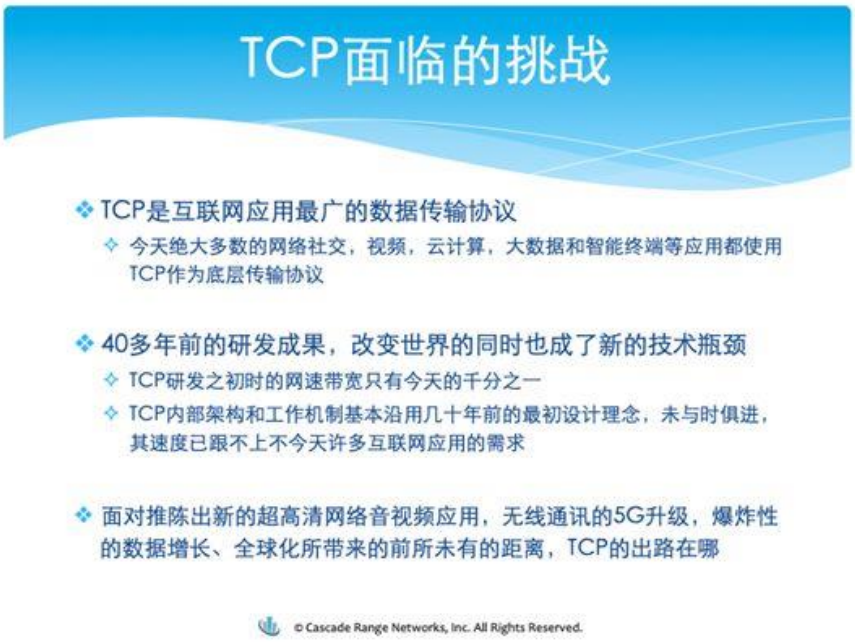
例如对于全球最大的流媒体平台 Netflix 而言，从 Netflix 数据中心到 CDNs 的 Outsourcing，从数据中心到 Amazon cloud 的 Cloudsourcing 以及用户从 Amazon Cloud 获取视频数据等数据流的后端都在使用着各种基于 TCP 的方案。

虽然有些后端已经陆续使用基于私有 UDP 协议方案的大规模数据传输，但前端的大部分数据传输尤其是用户从 Amazon Cloud 获取视频数据还是基于 TCP 方案来实现视频分发。

作为未来智慧城市中不可或缺的一部分，安防系统中的智能摄像头到 NVR 再到云的大部分数据流都是基于 TCP 实现。一些拥有私有云的企业可能使用基于 UDP 的解决方案，但如果接入数据至公有云则仍需要 TCP 进行承载。

除了上述案例，家庭娱乐与未来的远程医疗都需要大量的音视频数据作为支撑，其传输也主要由 TCP 承载。

2. TCP 面临的挑战与问题

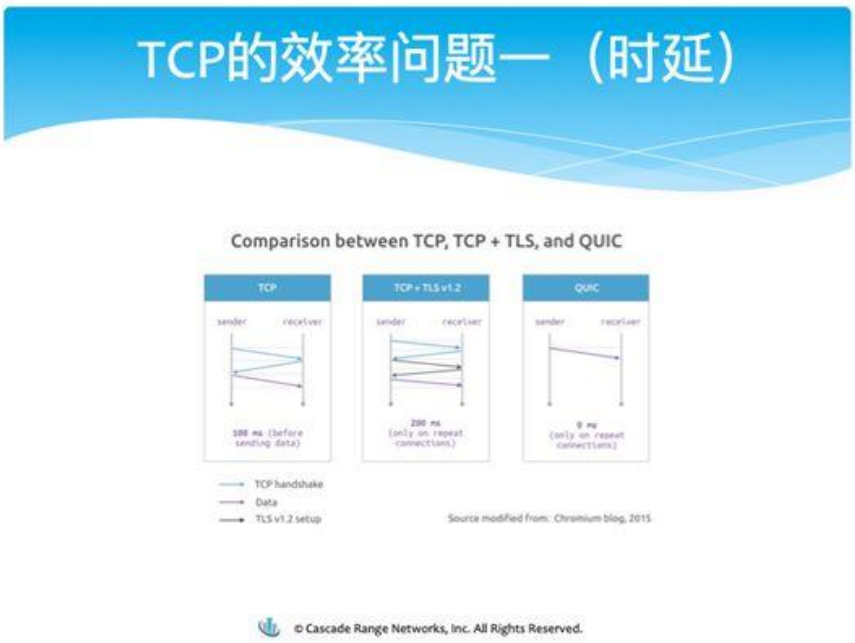


TCP 作为互联网应用最广的数据传输协议，所面临的挑战值得我们一探究竟。首先，由于今天绝大多数的网络社交、视频、云计算、大数据与智能终端等使用 TCP 作为底层传输协议的应用都

是基于 HTTPS 实现的，每项应用对性能的要求侧重点不同——交互视频、游戏等侧重低时延，高清视频、在线 4K 播放等侧重数据传输速度，这些不同应用的不同侧重需求使 TCP 的设计充满挑战；其次，由于在 TCP 的研发初期带宽资源较为匮乏，而随着通信技术的发展带宽资源与质量明显提升，互联网带宽已有几个数量级的提升，此时的 TCP 必须对几十年前的内部架构与工作机制进行一定技术改进才能保证在高带宽中高速的传输数据——如优化 TCP 的主体架构等，以避免 TCP 面临一些具有海量数据应用时在速度等性能参数方面力不从心；最后，随着全球数字化及超高清网络音视频应用的层出不穷，远距离与大数据也给 TCP 带来了很多前所未有的挑战。

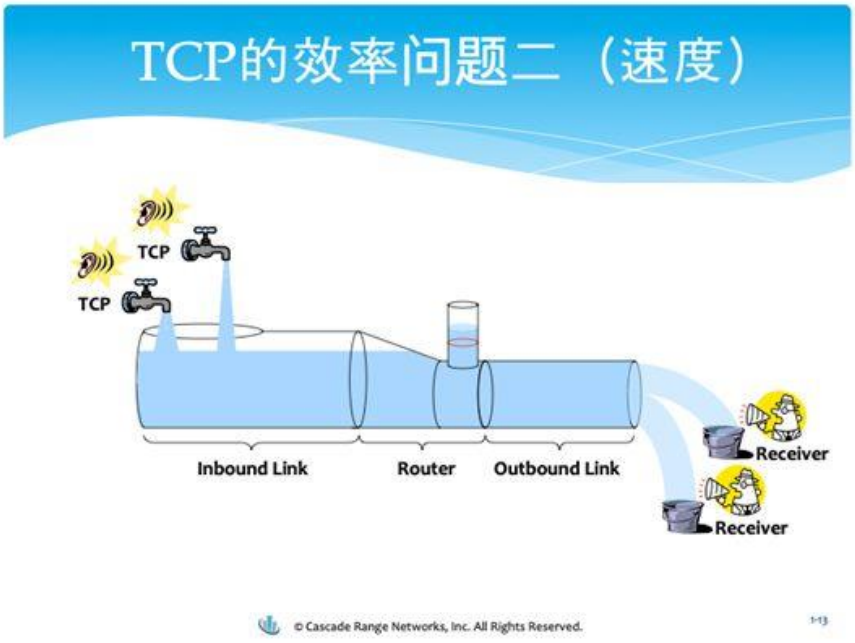
具体而言，TCP 面临的主要问题是带宽资源的浪费。在互联网初期带宽资源不多，就好比路不宽不好，后来路在逐渐拓宽升级等，但是 TCP 这辆车却一直没有升级很多，就像一辆老旧汽车在高速路上无法快速飞奔。另一方面，有如下班高峰期的道路拥堵，网络拥塞也是 TCP 亟待解决的问题，这方面的研究可以堪比自动驾驶技术极大提升高速公路的车流量，使每一位司机都变成一位守规的开车人。

2.1 TCP 的效率问题——时延

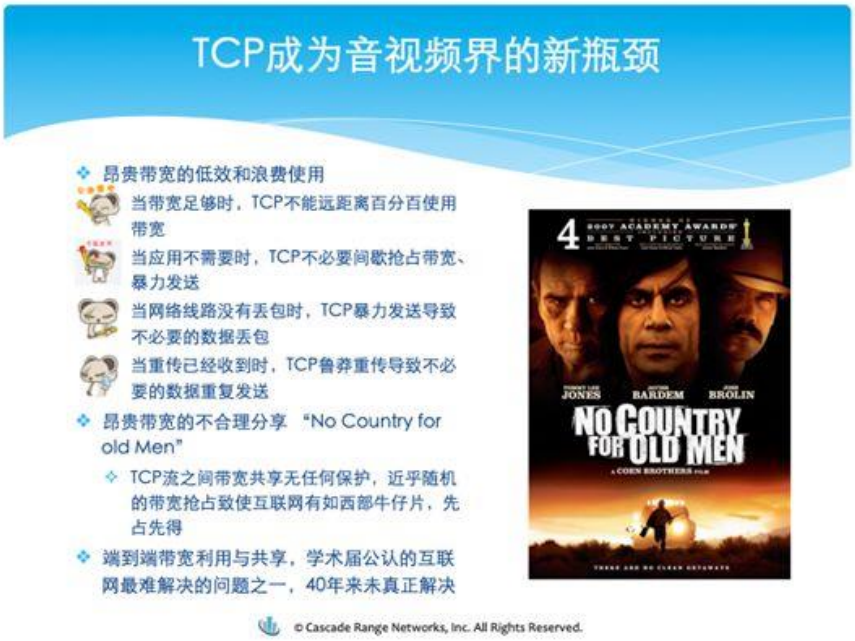


时延程度对一些特殊场景下的音视频而言至关重要，如在线课堂应用场景中一旦讲师的音画出现明显时延就会导致讲师与听众之间无法正常交互；而像游戏直播等出现明显时延则会极大影响游戏直播效果与观众观看体验。

2.2 TCP 效率问题——速度



TCP 的速度问题同样至关重要，如在未来 4K 在线播放等对数据传输速度要求极高的应用场景，一旦速度过低无法达到流畅播放的要求，观看就会经历大量卡顿等待或者变码率造成的清晰度下降，体验便会大打折扣；而如果速度过高则会造成网络的拥塞与丢包、重传，进而造成带宽利用低下等问题。



互联网技术的发展，带宽自始至终都是非常昂贵的资源。此环境下的 TCP 正逐渐成为一个制约音视频发展的新瓶颈，其限制首先在于当带宽足够时 TCP 无法实现 100%的带宽使用率，这时会出现物理带宽足够播放 1080P 等高清视频时却出现严重卡顿，即带宽未得到 100%利用的现象。我们需要在考虑带宽投入的同时重视带宽的利用效率，避免 TCP 成为效率的制约因素。

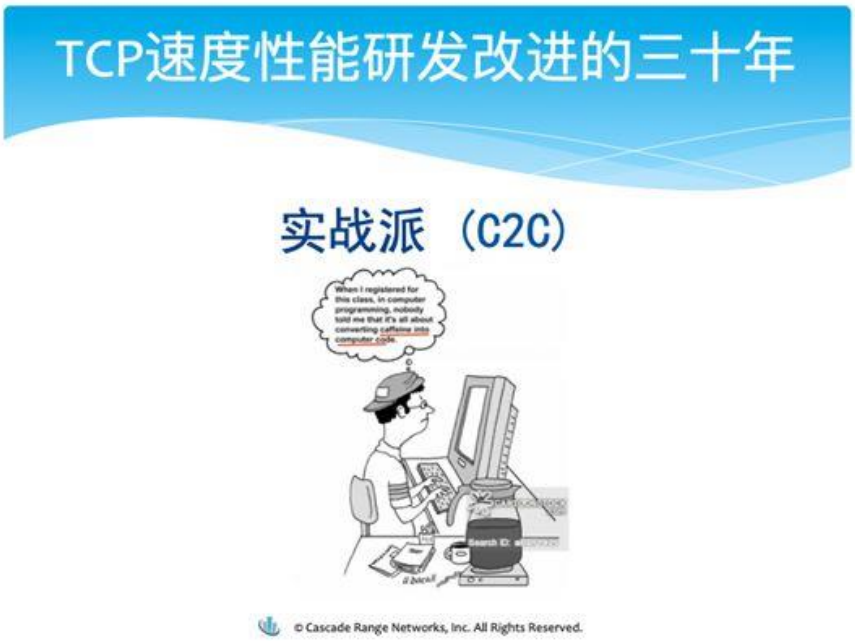
TCP 瓶颈的其他方面包括：

1）当应用不再需要时，TCP 不必要的间歇抢占带宽与暴力发送会对正常的视频流传输带来极大干扰从而造成视频卡顿、非必要丢包等状况。

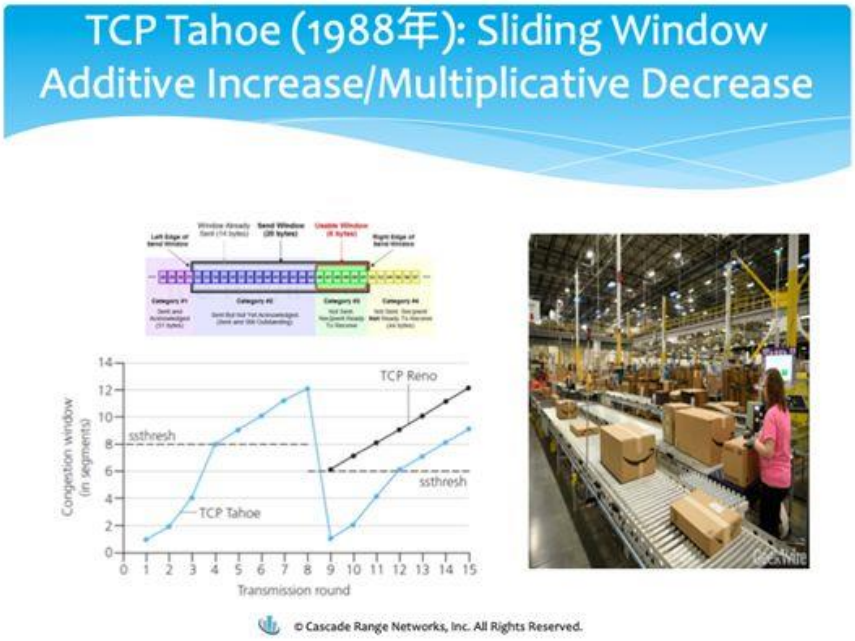
2）TCP 鲁莽的重传也会导致不必要的数据重复发送，进而浪费宝贵的有限带宽资源。总而言之就是 TCP 流之间带宽共享无任何保护，带宽抢占近乎随机，使得带宽资源分配不合理，这种端到端的宽带资源利用与共享命题是 40 年来学术界公认的互联网最难解决的问题之一。

3. TCP 速度性能研发改进

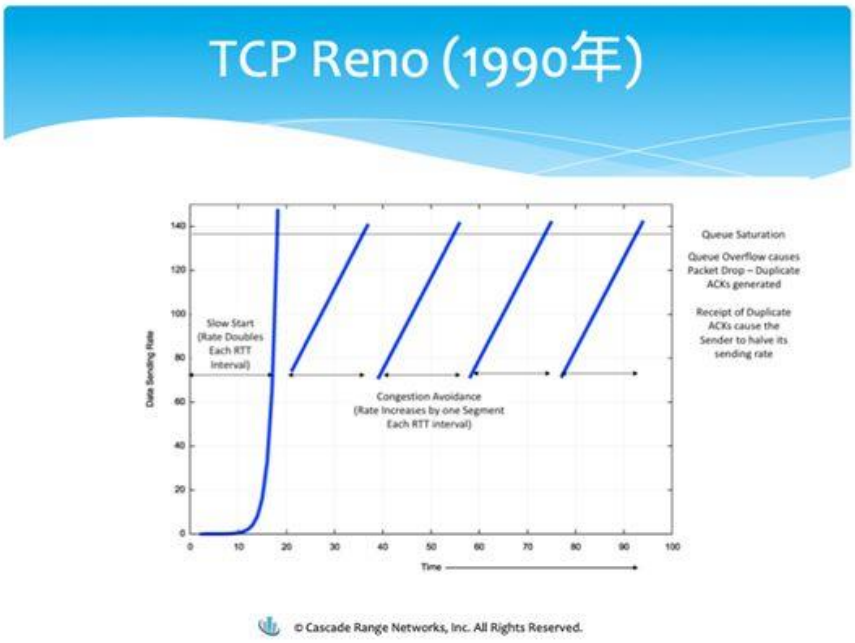
3.1 实战派



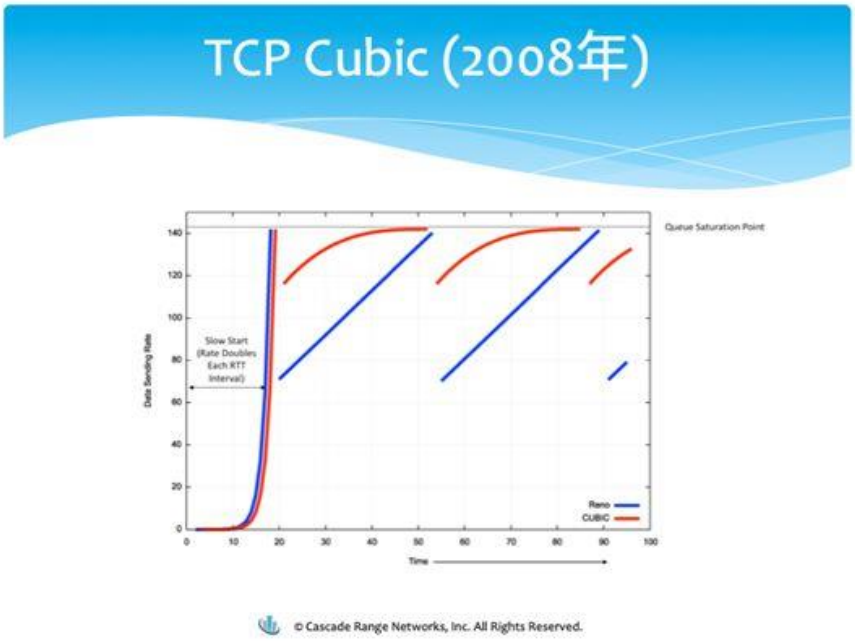
实战派主要是通过对 TCP 代码的修补，小幅提升 TCP 的性能。



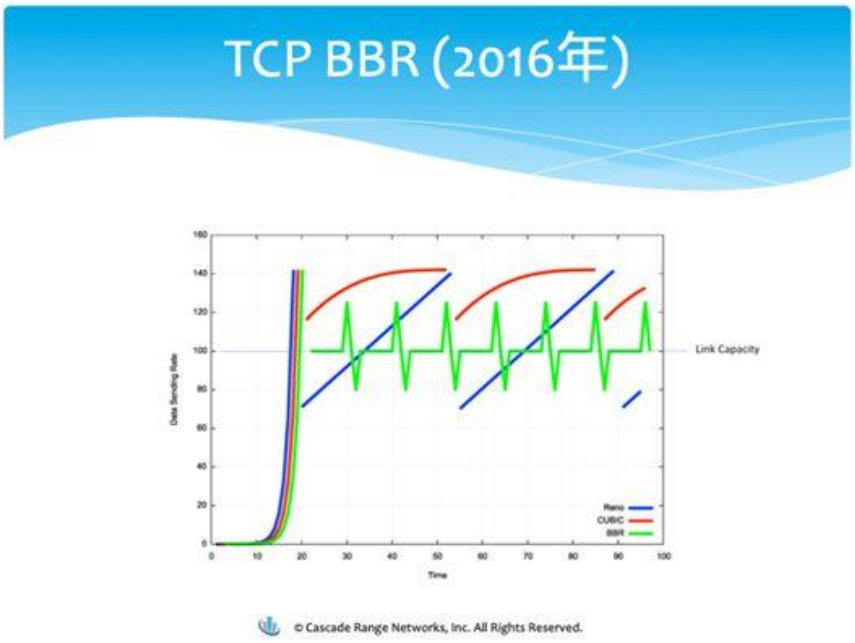
TCP 从初期便使用一种被称为“滑动窗口”的方式进行发包，也就是类似于批量流水线的方式，如上图右侧展示的那样，工人对应 TCP 的一段代码，通过调整此滑动窗口的大小来确定发什么包及发包的速度。



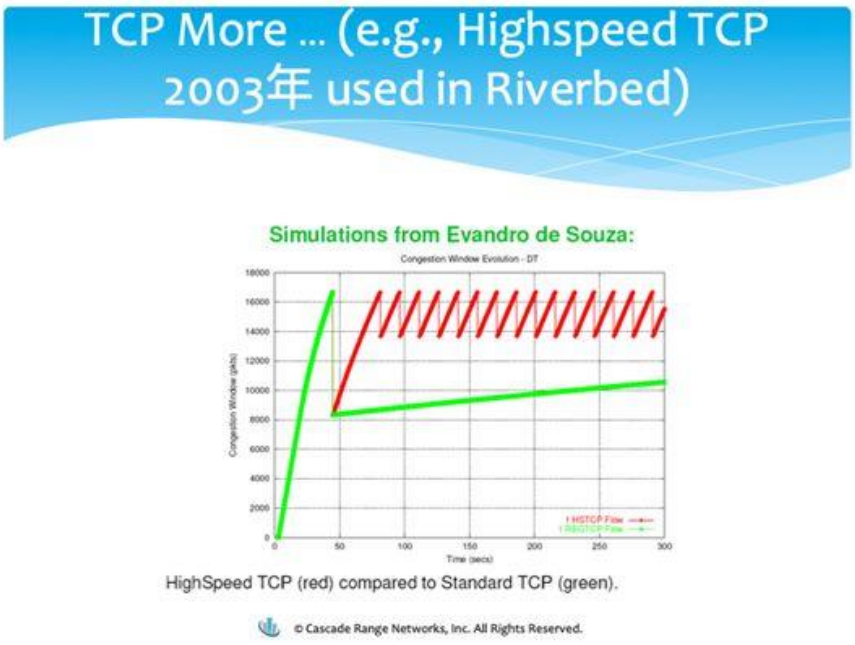
早在 1990 年 TCP 便采用被称为 AI/MD 的滑动窗口发包模式，AI 即逐步线性探测带宽，MD 即在拥塞时指数递减滑动窗口，如每探测到拥塞或者丢包时将滑动窗口减小到原来的一半， 从而减慢发送速度。



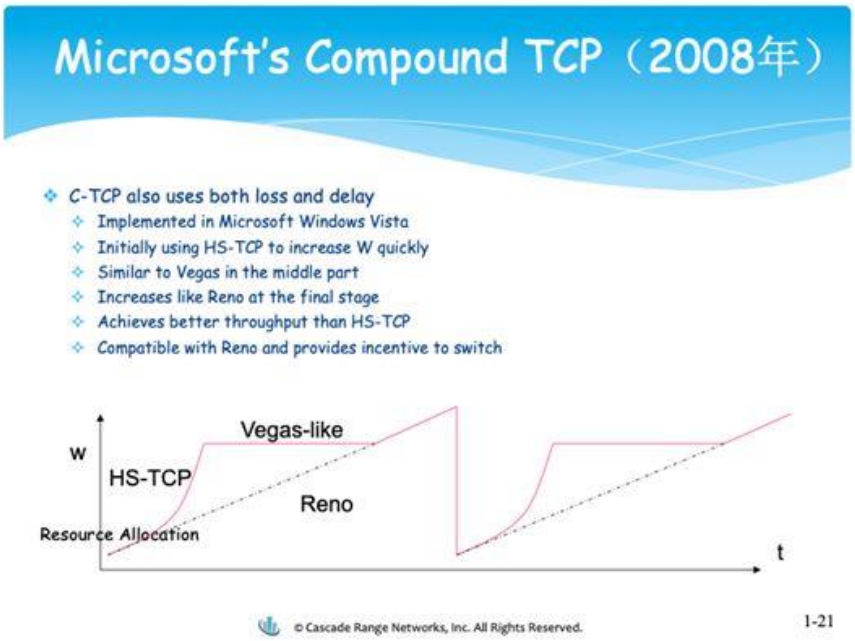
但实际上 TCP 的 AI 部分反应速度十分缓慢，在面对高带宽环境时明显力不从心。因此在改进的 TCP Cubic 上引入了一些非线性探测方法，可以简单的理解为将传统的线性搜索改进为二分搜索从而提升搜索速度。



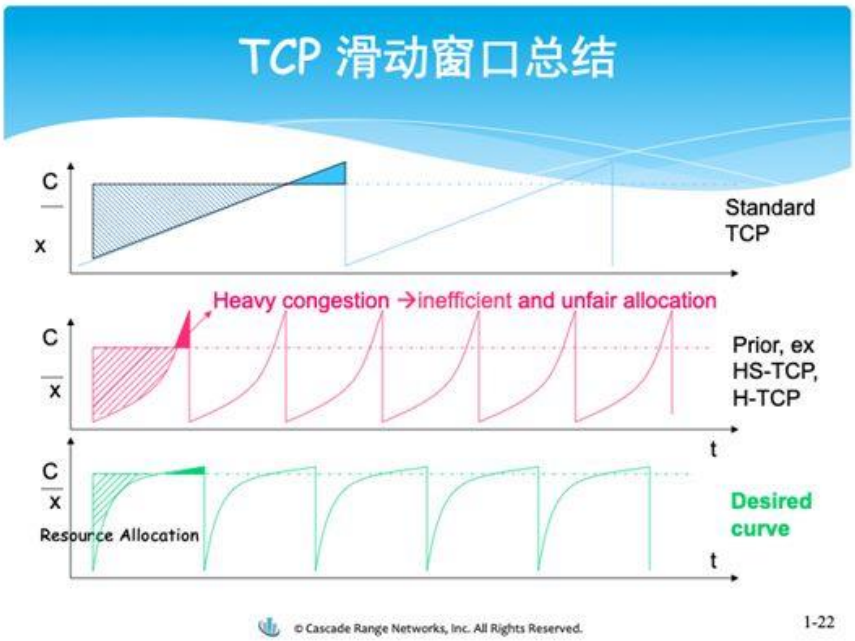
除此之外，BBR 也是是近年新起的一个拥塞算法，其原理是通过短暂的暴力发送来探测带宽并根据估算的带宽决定发送速率。



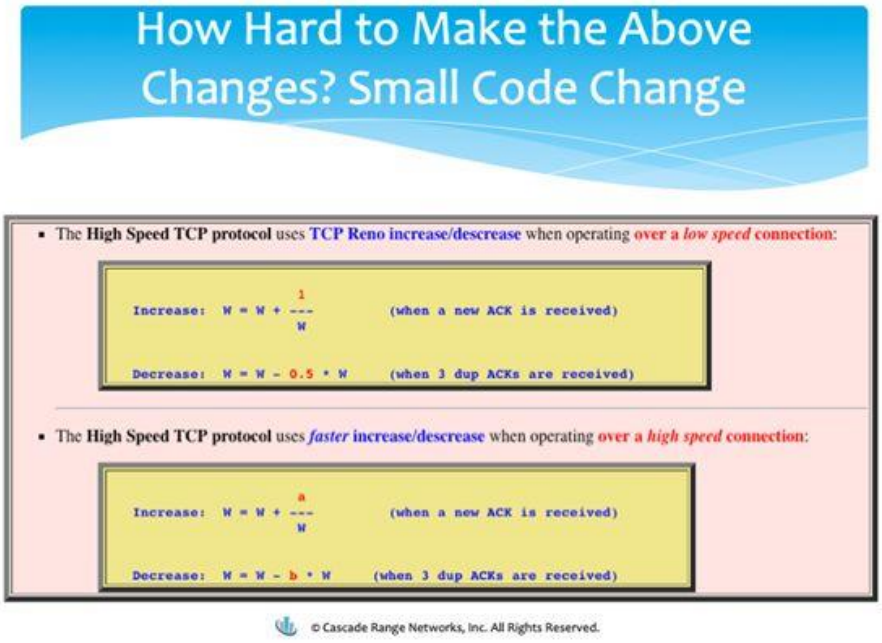
当然，近些年还有一些商业算法面世，例如由 Riverbed 公司开发的 Highspeed TCP，其是在原来 AI/MD 的基础上，把 MD 的窗口减半改为减小 1/8。



而微软则尝试通过加快网络带宽探测的速度，细节如上图所示。



从最初的滑动窗口改变策略，到后续的各种改进，业界已经将滑动窗口策略改进作为优化 TCP 性能表现的常见思路。

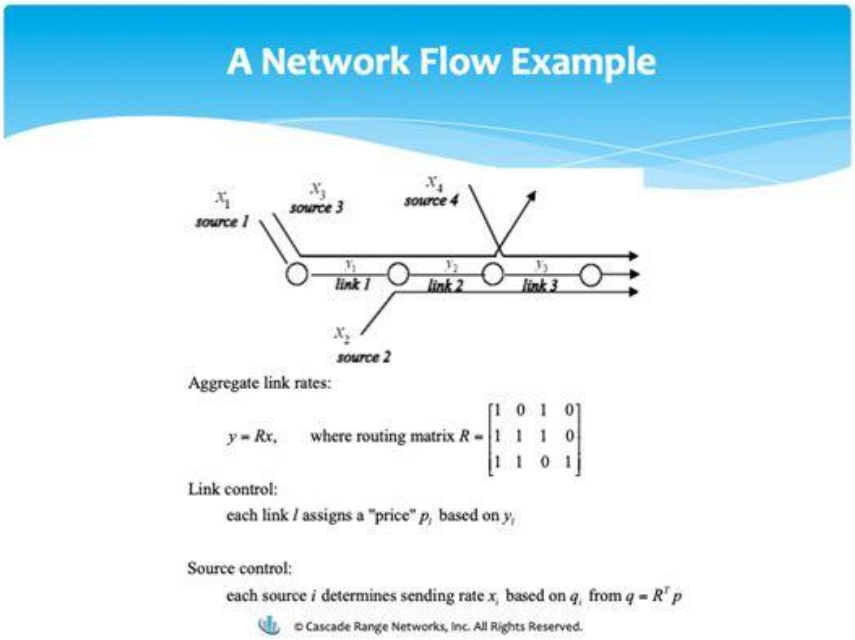


落实到内核代码层面的改动，改进一个滑动窗口协议算法的工程量一般小于几百行代码。

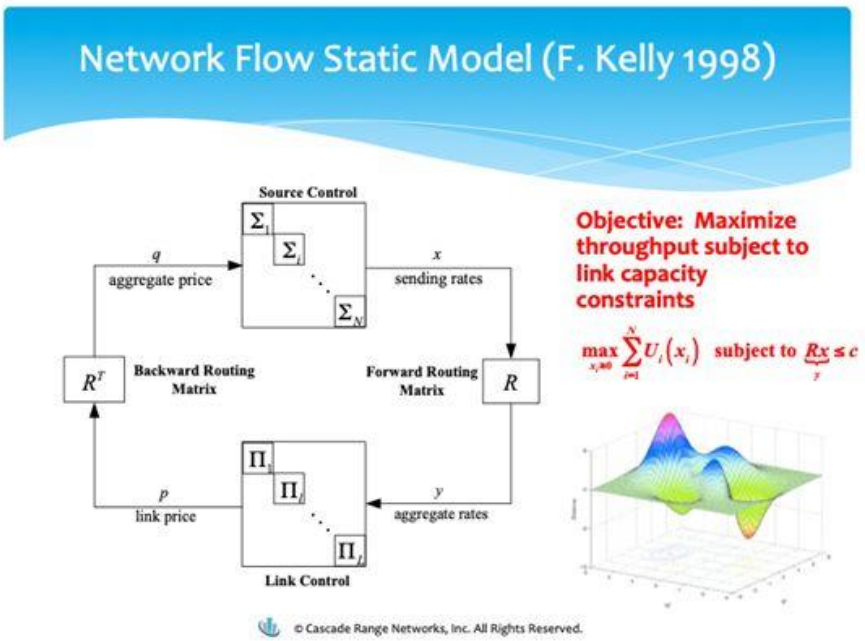
实战派的基于反复试错的实践对音视频行业而言的确始终是姗姗来迟，以至于近些年来音视频行业对各种网络状况下的速度诉求远远超出了 TCP 自身在弱网情形下的速度提升，所以慢慢有群体提出使用 UDP 取代 TCP 的各种解决方案。相对于较为功能全面的 TCP，UDP 更像是一张白纸，基于 UDP 的开发设计具有一定后发优势。基于 UDP 的方案也主要在应用层开发， 而 TCP 的开发几乎都在操作系统内核中，难度极大。

3.2 学术派

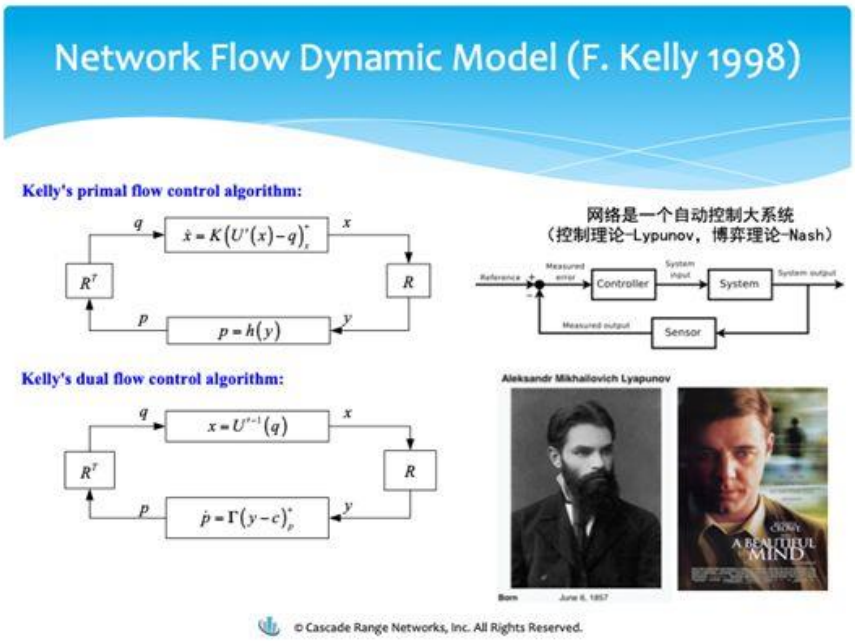
学术界在过去三十多年中也在不断探索针对 TCP 的速度性能改进与研发。虽然学术派的想法距离我们日常运维实践来说较为遥远，但一些开创性方案同样具备一定指导意义。



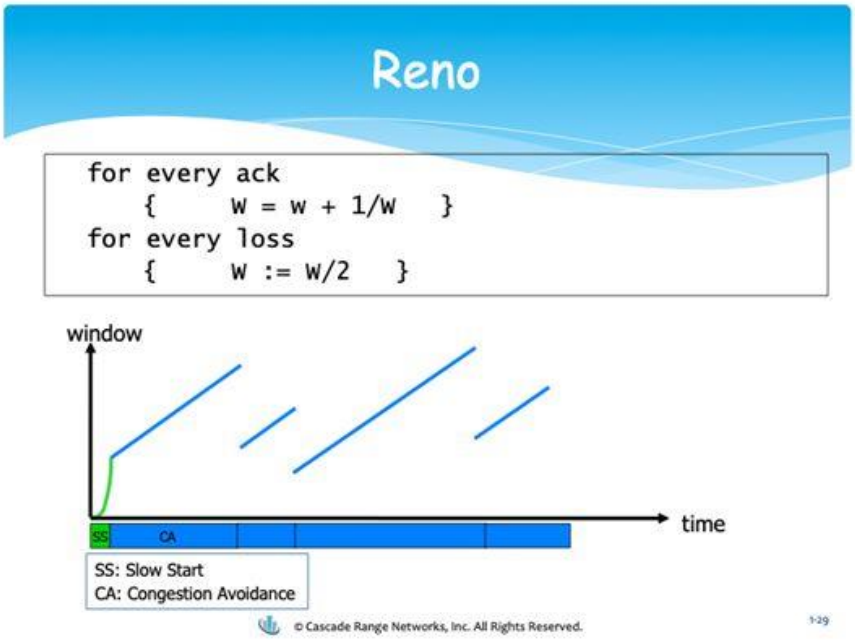
学术界的重要思路是通过建模与仿真实现对 TCP 的性能优化，上图展示的是一个网络拓扑，其中包括 4 个 source 与 3 个 link。link1 的速率对应流 x1、x3 的速率之和，link2 的速率对应流 x1、x2、x3 的速率之和，link3 的速率对应流 x1、x2、x4 的速率之和，总体来看就是每个 link 的速率等于各单流速率之和。而我们通过测量往返时延来量化拥塞程度，x1 的往返时延等于流 y1、y2、y3 上的往返时延之和；x2 的往返时延等于流 y2、y3 上的往返时延之和。从建模的角度我们可以看到这是一个非常规整的对偶问题：使用线性代数表述，链接上的速率等于矩阵 x 链接速度乘以矩阵 R，而拥塞正好是将此矩阵顺/逆时针旋转 90 度后再乘以每个链接上的时延即可得到对应链路的往返时延。



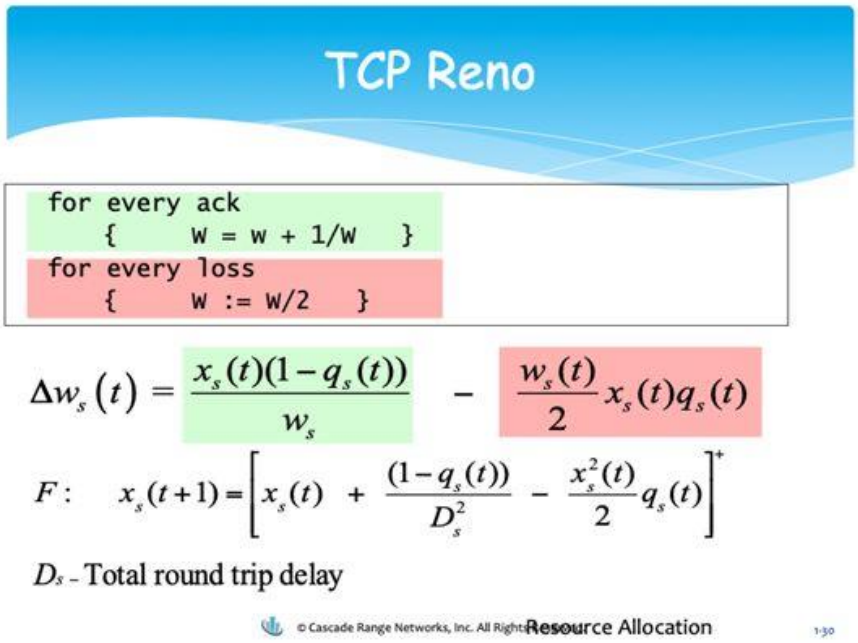
1998 年，英国一位院士在研究网络建模时发现 TCP 协议与 AMQ 协议实际上是最大效用网络全局优化命题。上图右侧公式中的 c 代表带宽，我们需要在保证速率不超过带宽的情况下尽可能提高网络效用性能，所谓的各种网络效用即对应了各种 TCP 协议。



Kelly 教授不仅将此问题建模，还得出此问题的两个分布式解，这两个分布式解正好分别对应了 TCP 协议与 AMQ 协议。这种将网络看作为一个分布式自动控制系统的想法是开创性的，使得我们可以将具有近百年历史的控制理论与博弈理论应用于网络的研究中。



以 Reno 为例，上图展示的两段代码中，第一段用于进行线性探测带宽，第二段用于探测到网络拥塞时将窗口数量减半。



两段代码对应两段数学公式，$\frac{x_s(t)(1-q_s(t))}{w_s}$ 对应线性探测带宽，$\frac{w_s(t)}{2} x_s(t) q_s(t)$ 对应探测到网络拥塞时将窗口数量减半。

Reno Utility function

- Reno

$$F_s : x_s(t+1) = x_s(t) + \frac{(1-q_s(t))}{D_s^2} - \frac{x_s^2(t)}{2} q_s(t)$$

- At equilibrium

$$x_s = x_s + \frac{1-q_s}{D_s^2} - \frac{x_s^2}{2} q_s$$

$$\Rightarrow q_s = \frac{2}{2 + x_s^2 D_s^2}$$

- Utility function

$$U(x_s) = \int f_s(x_s) dx_s = \int \frac{2}{2 + x_s^2 D_s^2} dx_s = \frac{\sqrt{2}}{D_s} \tan^{-1}\left(\frac{1}{\sqrt{2}} x_s D_s\right)$$

© Cascade Range Networks, Inc. All Rights Reserved. 1-31

于是我们就可以推导出 TCP 协议的效用公式，这种建模可将不同的 TCP 协议对应至不同的数学方程当中， 不同的 TCP 对应不同的效用也就顺理成章地解决了大家关心的带宽共享和使用问题。除此之外，实际传输速度的快慢也即等同于趋于平衡点的速度快慢，也就是将原命题转化为一个自动控制问题，这样我们就可以将很多自动控制理论用于此网络命题的研究之中。

TCP理论研究在音视频行业的局限性

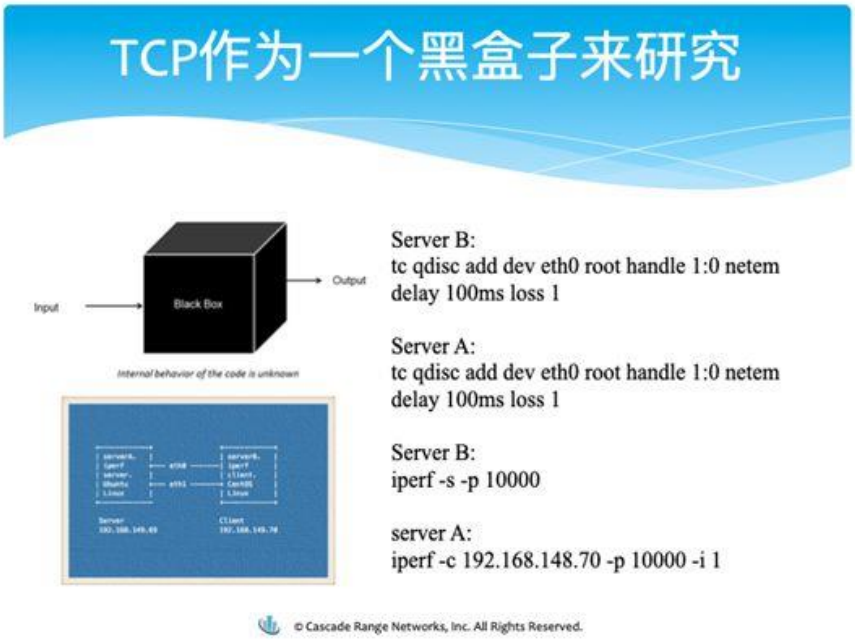
- 很多研究停留在院校，并未真正被音视频行业所知晓和实践，学术界有上千篇关于TCP研究的文章
- 院校也没有机会接触音视频界实实在在待解决的问题，一些实际的问题也未能在建模中充分考虑
- 学校的方案直接应用的音视频行业往往效果不好

© Cascade Range Networks, Inc. All Rights Reserved.

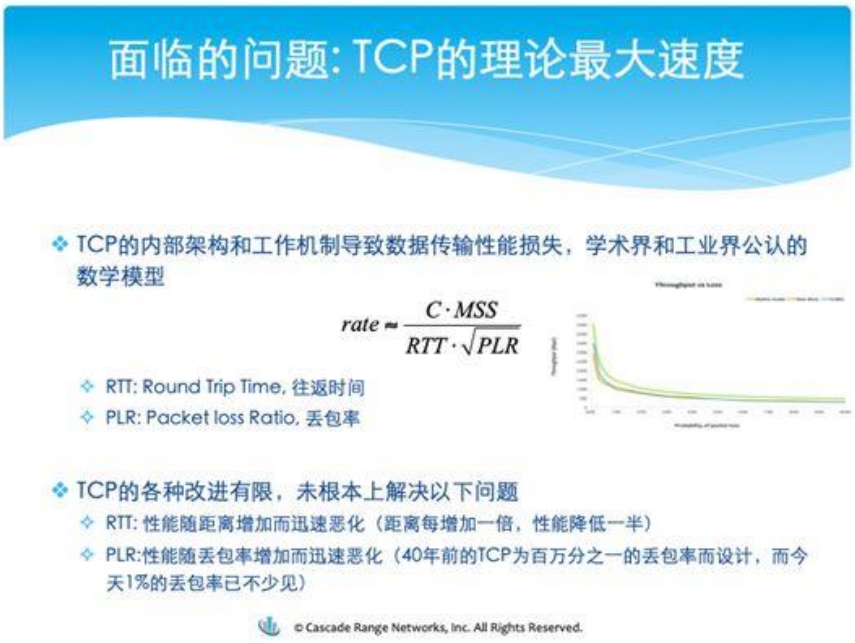
虽然在过去三四十年学术界通过不断的建模与仿真从未停止过对 TCP 的理论研究，但这种建模与仿真对音视频行业来说具有一定的局限性。

首先，尽管学术界有上千篇关于 TCP 研究的文章，但这些研究很多都仅停留在高校院所或学术期刊上，并未真正被音视频行业所知晓与实践，许多理论仍停留在数学公式阶段，无法在短时间内被转为工业应用；加上院校也没有机会接触音视频行业实实在在亟需解决的问题，缺乏对音视频行业的深入理解与思考，一些实际的问题容易被研究者忽视；以上种种原因导致学术界有关 TCP 的研究无法在实际的音视频行业得到较为成功的应用。

3.3 研究思路



无论是尝试通过底层代码进行优化的实战派还是借助仿真与建模进行优化的学术派，在评估各种 TCP 的好坏时，都可以将 TCP 作为一个黑盒子来研究。我们作为终端用户也可以用同样的办法来评估，普通用户进可以忽略 TCP 内部的代码与建模方式，将测试重点放在量测各种网络情况下的 TCP 性能，这种性能分析的实现门槛较低，只需一台电脑即可完成。如图所示，我们可以通过两台 Linux 虚拟机上的 tc 命令行创建两个网损：配置分别从 B 到 A 和 A 到 B 的两个 100ms、丢包 1%的单向网损，此情况下就得到了一个 200ms 1%丢包的双向网损；接下来我们就可以用一些业界的常用量测工具如 iperf 来量测 TCP 的性能，这便是评价 TCP 性能的最快方式。



通过类似的量测与建模研究，学术界和工业界得到一些 TCP 最大理论速度的经验公式，如上图展示的一个公式，由此公式我们可以发现，TCP 的理论最大速度与传输距离成反比，与丢包率的开平方成反比。这也就是为什么当在访问距离较远的网站时 TCP 会经常出现一些性能问题，例如一个可以在局域网内流畅观看的高清 4K 视频流被跨国传输到海外时便会出现无法正常播放等一系列问题。同样，TCP 对丢包的极其敏感也与公式中的开平方有关系，假设线路信号错误丢包为百万分之一，开平方后即为 1/1000 在公式右半边的分母中，而实际拥塞丢包为百分之一，开平方后的数值为 1/10，其与 1/1000 存在一百倍的差距，这便导致了 TCP 对丢包的敏感性。拥塞丢包就好比道路堵车在现实中是无法避免的。

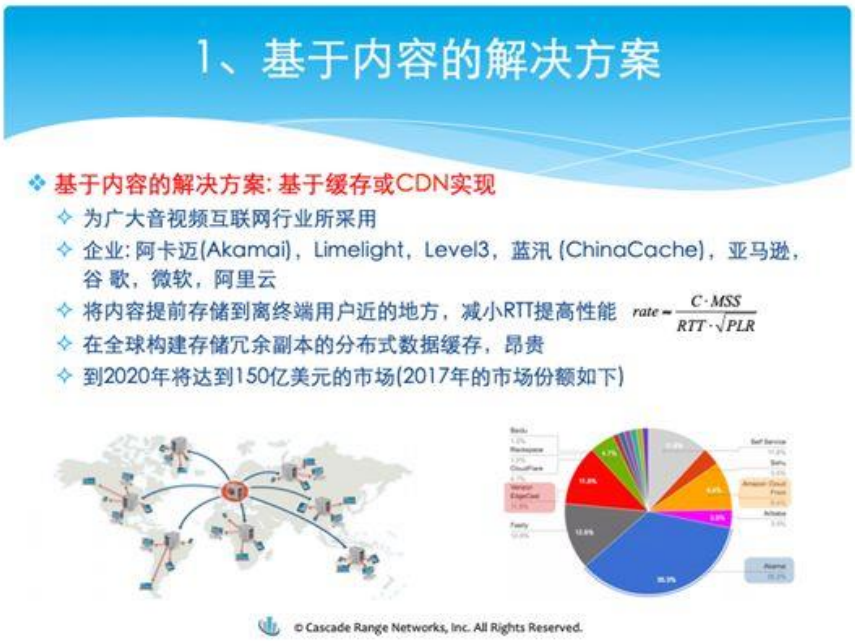
面对 TCP 的上述种种缺陷我们应该如何改进与解决?

4. 解决方案

4.1 基于内容的解决方案

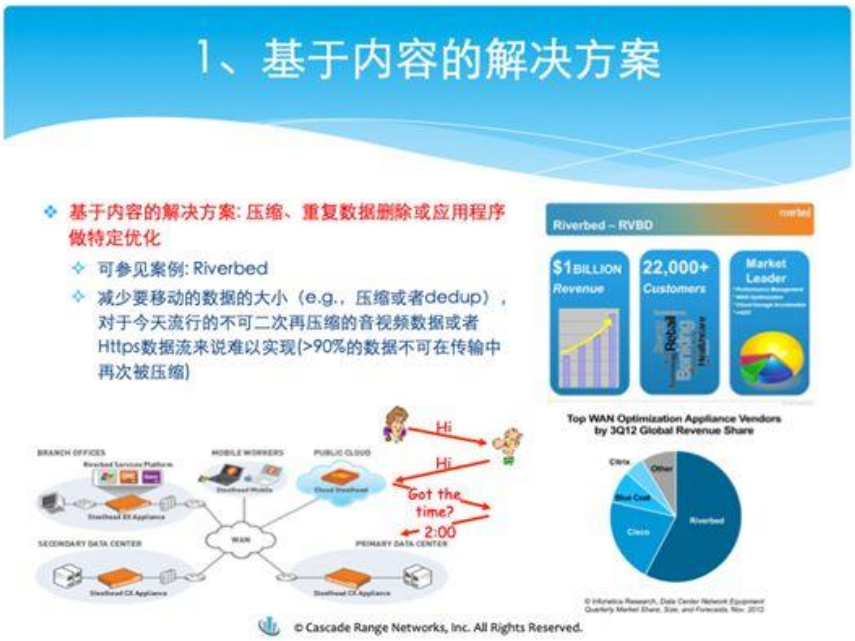
我们可以将此方案概括为基于缓存或 CDN 进行数据压缩，重复数据删除或对应用协议进行特定优化。

1）基于缓存或 CDN 实现



我们可以将第一个方案概括为基于缓存或 CDN 进行数据压缩，重复数据删除或对应用协议进行特定优化。此方案被绝大多数音视频企业所采用，也是 CDN 的主要收入来源之一。在这里 CDN 主要为企业提供两部服务：带宽与覆盖。后者主要依靠 CDN 的多城市布点以降低往返时延来解决。虽然此方案间接解决了一些 TCP 的短板，但却是非常昂贵的。

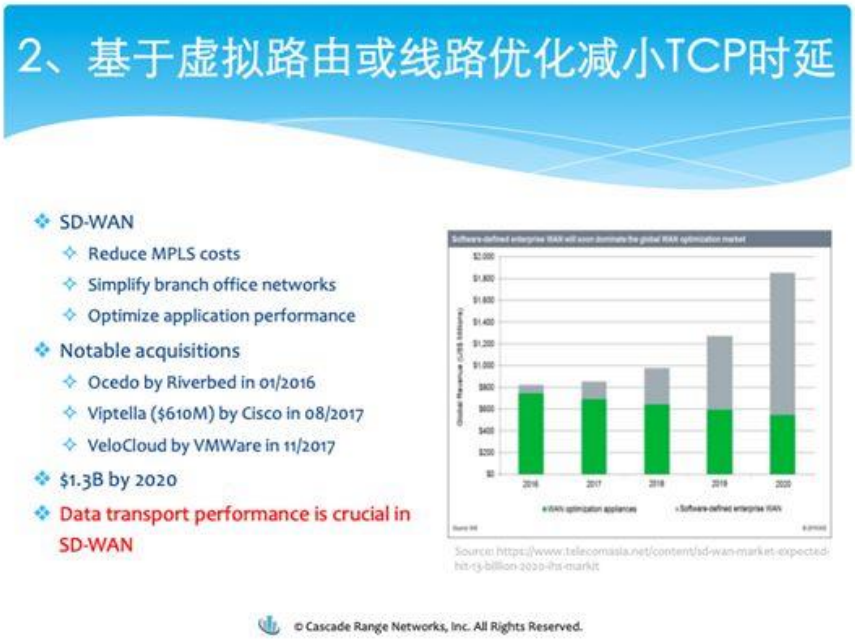
2）删除压缩、重复数据或优化特定应用程序



基于压缩、重复数据的删除与应用程序特定优化的解决方案在以前受到追捧，如 Riverbed 公司便使用软件加速核心来压缩数据与删除重复数据。但在音视频行业中，已经上传的数据几乎不能够进行二次压缩，尤其对于 HTTPS 数据流而言更是难以实现；而应用程序特定优化是指提升单次交流的效率，主要用于解决在线视频交互的时延问题，这种提升对大数据音视频应用的是非常有限的。

即便如此，基于内容的解决方案本身并未真正解决 TCP 的诸多问题，而是弥补了 TCP 的相关短板，通过覆盖、内容压缩等降低 TCP 的发包数量。

4.2 基于虚拟路由或线路优化减小 TCP 时延



此方案主要针对路由或线路的优化，多用于一些手游加速。其关键在于基于缓存或 CDN 进行扩展，通过购买或租用线路来优化小数据传输路径，从而解决单个数据包端到端的时延问题，但由于其云路位于由 ISP 控制的物理路由之上，优化受到限制，且所需花费的成本十分庞大，维系一个 SD-WAN 仅运维所需成本就高达上千万，且性能提升十分受限，同样也是非常昂贵的解决方案。

4.3 基于 UDP 的解决方案

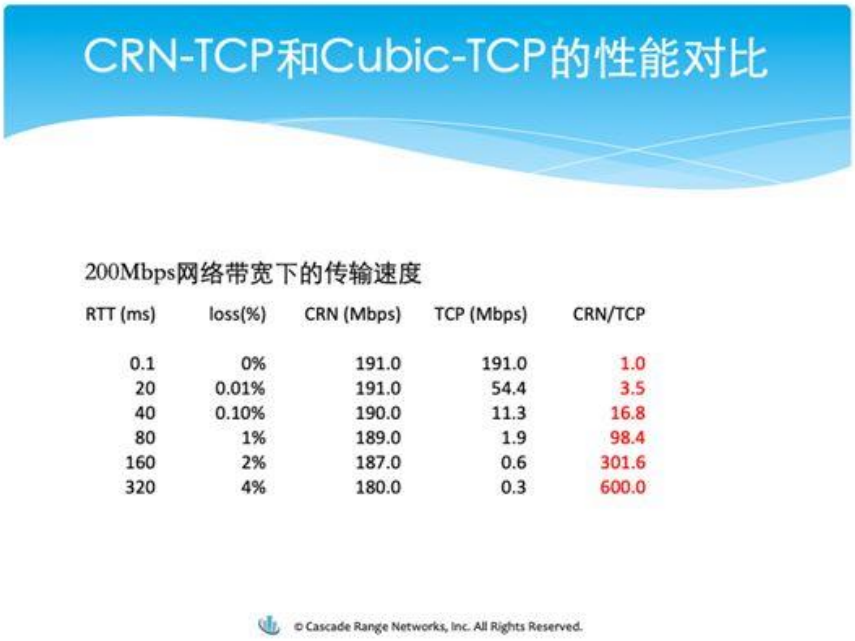
TCP 的诸多久未改善的缺陷使得业界更加倾向于使用基于 UDP 的解决方案，随着音视频行业的高速发展。TCP 的发展停滞不前，越来越多的企业不断探索新的改进方向，基于 UDP 便是其中的一种。其优势在于 UDP 是一个应用层封装，开发者在 UDP 上开发一个协议的难度远小于内核层级的开发，具有一定后发优势。当然，UDP 的缺陷在于由于其端口开放的设计易导致一系列安全隐患，不易受 IT 部门青睐；加上 UDP 本身没有可靠的性与拥塞控制，因此基于 UDP 的应用层传输协议必须在应用层得到重建，这就会造成额外的软件开发与 CPU&RAM 的资源浪费；最后，UDP 作为一个双端方案，需要数据发送端与接收端的双端部署，这对于那些不能控制收发两端的企业来说无疑是不能接受的；同时，面向广大消费者的双端方案也面临许多问题，如某在线教育公司为保证在线课堂视频交互的成功率会要求学生使用 PC 的 Chrome 浏览器或手机端的 Chromium 内核浏览器观看视频，这体现了 UDP 无法在短时间内大规模进入消费市场的现状，可以说在覆盖和穿透上并 UDP 和 TCP 相比还有很大的距离 。

4.4 改进的 TCP 拥塞控制

上图展示的就是前文提及的改进 TCP 拥塞控制算法的一些优劣与实例。

经过这么多年的发展我们可以看到，UDP 协议的成功从侧面反映了 TCP 协议的失败。TCP 协议并未妥善解决音视频行业所面临的时延与卡顿问题，尤其是在处理低时延交互、跨国传输与卡顿频率、每次卡顿等待时间等问题时均显得力不从心。现在的 TCP 无法满足安防、交互直播等诸多行业的需求。

我们看到业界呼唤新一代与 TCP 兼容的高速数据传输技术能够从容应对音视频大数据和智能终端的挑战，借助全新设计的内部架构与机制和在智能检测、控制、信号处理方面的全新理论基础，从根本上解决核心传输问题并与现有个解决方案协同工作。



RTT (ms)	loss(%)	CRN (Mbps)	TCP (Mbps)	CRN/TCP
0.1	0%	191.0	191.0	1.0
20	0.01%	191.0	54.4	3.5
40	0.10%	190.0	11.3	16.8
80	1%	189.0	1.9	98.4
160	2%	187.0	0.6	301.6
320	4%	180.0	0.3	600.0

面对音视频行业的需求，我们经过不屑努力研发了新一代 TCP 传输协议，即 CRN-TCP，主要用于解决高清视频播放时的卡顿问题，上图展示的是实验室中的实测数据，其中第一列是不同的网络时延，第二列是不同的网络丢包，最后一行是 CRN-TCP 相对于传统 Cubic-TCP 所带来的性能提升倍数。CRN-TCP 在带宽允许的高丢包与高时延长距离网络上实现了 4K 高清视频的无卡顿播放。

我们希望与音视频行业相关人士一起努力，实现在不做任何应用与网络架构的变更下降低音视频流的时延与卡顿，提升音视频用户在弱网下的体验；同时实现最大的带宽利用率与最高的带宽使用效率，从而节省昂贵的带宽开销并实现与音视频数据流的公平带宽共享；新一代协议全面兼容现有 TCP 协议并支持标准 TCP 应用端口与各种互联网应用以及各种网络优化解决方案，达到对全平台的支持与兼容。

服务端架构演进与实践

视频云下半场 向前走还是向“厚”走?

从 2016 年至今，流量的增长基本进入到了平稳期，此时，面向产业界和开发者，我们如何提供更多、更新的能力给到他们，提升平台的用户体验？本文来自腾讯云视频业务产品总监黄斌在 LiveVideoStackCon 2019 深圳站上的精彩分享，希望和业界一起探讨视频云下半场的方向与定位，也希望与产业界同仁一道，共建更好的大视频生态。

黄斌　LiveVideoStackCon 2019 深圳　2019-12-26

1．直播行业中的难点

腾讯云内部最早在 2015 年下半年开始进入视频云领域，将腾讯多年在音视频编解码、音视频通信以及海量并发业务的经验逐渐开放，当时我们也是新进者，定位是在 OVP（在线视频平台），类似国外的 brightcove 及国内的 CC 视频，我们在教育、在线视频等领域进行了尝试。不过真正确定业务重点方向是在 2016 年，2016 年也是国内的直播元年，行业的爆发让团队意识到直播的流量是非常大的，在高并发情况下如何能做到视频流畅无卡顿、并能提供丰富的 IM 通信、保证互动连麦等环节的正常进行，这正是我们的技术优势所在，我们抓住了直播的这个风口。

我自己将市面上的产品按照碎片消费、按需点播、实时互动和定时播放四个维度分为四个部分，右上象限这类实时性高，低制作成本的这些应用，正是这几年集中增长爆发的应用。

这些新兴应用，往往都有复杂的功能特性，对于头部企业都有强大的研发团队，有很多能力去构建上图中的要求，但对中小型企业来说，能把上述功能都做出来，并且要做好的话，面临的挑战是非常大的，其中每一个功能都需要投入一个小团队深入去做，由此带来的成本也会非常高。

上图中列举了视频云需要的基础服务、音视频计算集群、CDN 加速网络，以及终端能力和社交互动系统，每个部分所需要投入的人力和时间成本，可以更直观的看出直播应用的开发所需要的大资源投入以及高技术挑战。

2. 腾讯云直播方案

2.1 腾讯云互动直播方案

我们基于在 QQ 平台里音视频通信技术能力的积累，我们设计出了第一套腾讯云互动直播方案。

- 第一步，基于类 RTP 私有协议实现了一套多路低时延音视频系统，保证主播和连麦互动观众之间可以实现低至 400ms 的低延时音视频通信，放到现在也能够接近多人实时通话场景的应用；
- 第二步，团队借鉴了 CDN 的架构实现多级分发，满足单房间内百万观众都能实现低时延观看，并且可以通过重定向逻辑随时上麦，实现连麦互动场景；
- 第三步，为了满足 H5 观看，旁路转码推出一路标准流媒体协议（RTMP/FLV/HLS）的流，甚至能够录制和回放，实现既有私有 RTP 协议链路，也有标准流媒体协议旁路系统。

此套方案在直播方案基础上实现了很好的兼容性，因此 2016 年腾讯很快就拿下了泛娱乐和社交这类平台的头部客户，而此后不少直播平台也是基于类似的思路实现，而腾讯在其中提供了标准云服务。

2.2 腾讯云标准直播方案

针对中小型的社交直播需求，我们也提供一套基于标准协议的，更轻量化的直播服务流程，从推流到云端的转码、切流和混流，这些从现在的眼光看依然是非常经典的直播方案，以上就是腾讯在 2016 年面向直播的两套主要架构，基于这两套架构腾讯也将主流的泛娱乐直播场景都囊括在内。

3. 向前走-制作域

从 2018 年到 2019 年，视频流量的增长开始进入平稳增长期，经过几年的持续增长，平台都进入到精细化运营阶段，包括用户、服务体系、成本和质量各个方面都需要做优化。与此同时，我们回到标题所提出的问题“视频云下半场，向前走还是向“厚”走？”

刚刚提到我们的产品方案在云端处理、CDN 的分发和终端 SDK 这个链条上的方案覆盖度都已经比较完整了，而在内容供给侧或者说制作域方面，会发现随着近两年短视频的增长，内容制作工具化的诉求开始陆续体现出来，甚至逐渐演变为刚需。在做过不少分析后，最后团队还是决定落地，制作相关制作域或制作云的方案，下面我将为大家介绍团队在制作域方面的一些产品。

3.1 腾讯云剪

腾讯云剪作为云化的内容协同编辑工具，首先具备和传统编辑软件类似的基本编辑甚至复杂的编辑功能，另外在内容方面，我们整合了腾讯生态伙伴的大量的视频内容，包括电竞、体育、综艺、娱乐等，这里除了原始素材，我们也提供经 AI 结构化分析后的，带着标签结构化信息的素材供创作者二次加工使用；另外我们还提供特色的精彩集锦、节目拆条等特色能力。

3.1.1 腾讯云剪开放平台

腾讯云剪作为 SaaS 的产品形态提供给到创作者，同时我们也将其基础能力提炼成一个 PaaS 能力平台，除了最基本的视频处理、上传和导出，包括媒资管理以及复杂的转码检索能力，这两层都能够以 PaaS 的方式提供，有这样需求的用户或合作伙伴需要腾讯云剪的这些能力去做成自己的编辑工具，那我们就提供给伙伴 PaaS 平台能力。

3.2 腾讯微剪插件

除了 Web 端的编辑工具之外，团队还和微信一起在做小程序端的音视频编辑插件能力，需要微信小程序及客户端团队提供一组最底层的音视频合成、打包和渲染的接口，在此基础上我们利用自身的技术能力再封装出一套编辑工具插件。目前这款腾讯云联合微信小程序、腾讯音乐集团爱听卓乐公司三方共同合作研发的腾讯微剪——小程序音视频剪辑插件还在不断地打磨和完善，相信在不久之后就可以与大家见面。该插件集成了由腾讯音乐集团提供版权音乐在线曲库，基于腾讯视频云强大的音视频处理技术开发并封装完成。开发者们之后仅需在小程序插件市场申请后即可快速调用，为小程序增加强大的音视频剪辑功能。

3.3 视频云终端能力 Update

在短视频制作方面，近期我们还对我们的美颜动效能力进行了全面升级，除了基础的美颜、特效贴纸之外，还增加了高级的美妆美颜、肢体识别特效、绿幕抠图和人脸融合等能力，特性及性能方面有了极大升级。

3.4 腾讯云导播台 LVC

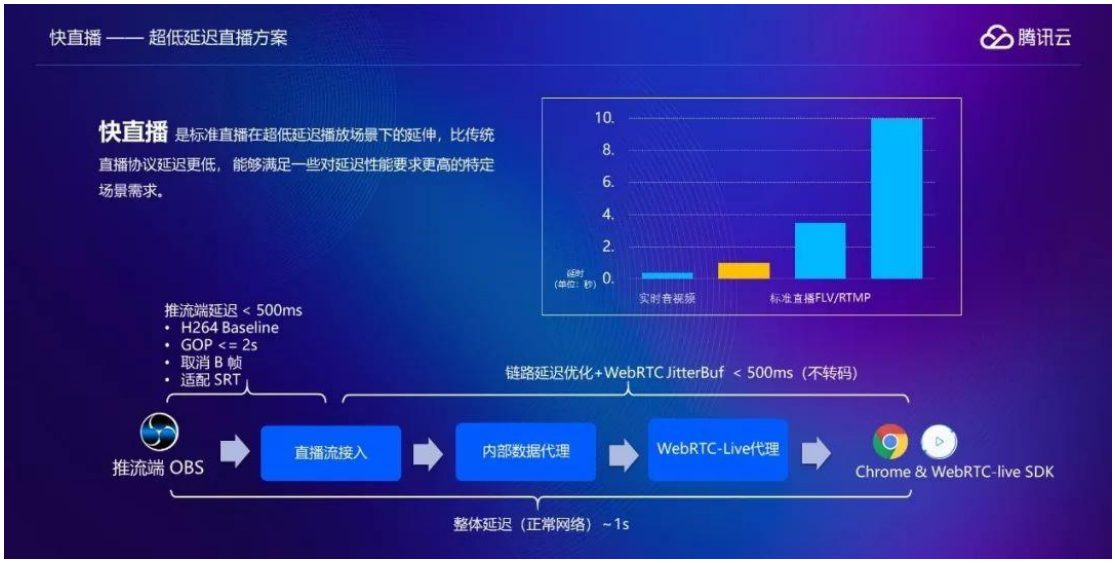
由于腾讯云本身在云端后台有不少混流、切换的能力，团队也将这些能力包装成可视化页面，变成云导播的产品形式开放出来，这个能力在企业培训、教育、节目制作等场景都有广泛的应用。腾讯云导播台也提供 API 的方式去调用预监、转场等功能，面向普通用户提供完善的页面，可在页面直接进行导播操作。

3.5 传输及全链路质量监控

传输方面在音视频中也是一个比较强的场景，今年英雄联盟决赛在法国举行，编码器在现场切出的 TS 切片，通过我们提供的传输能力，实现了 ts over SRT，将清流信号从欧洲现场从公网传回了上海及北美，对比我们发现公网传输的两条链路做到了无丢包和低延迟，传输过程中我们也提供视频传输通信质量的监控，包括码率、卡顿、帧率等指标。

4. 基础编解码及平台能力

4.1 快直播——超低延迟直播方案



关于我们的 WebRTC CDN 产品，我们对外叫“快直播”。我们希望它会作为我们直播 CDN 升级的一个重要能力。WebRTC 本身可以做到低于 1 秒的超低延迟，并且延迟可控，不会持续积累。作为实时通话标准，它有一整套应对网络波动和弱网的策略，在超低延迟的基础上，卡顿不会落后于其他高延迟流媒体方案。另外，WebRTC 已经被绝大多数浏览器广泛支持，不需要安装插件就能播放；而且，WebRTC 技术依然在快速演进，未来会支持 AV1，SVC 等；我们也在网络层与 QUIC 进一步结合，形成更好的传输方案；如果在带宽成本、节点资源覆盖上面持续优化，相信未来基于 WebRTC 的 CDN 将会成为直播内容分发的一个趋势。

4.2 全球部署

除了超低延迟直播方案以外，出海也是明年腾讯云发展的重点，从系统平台来看，在东南亚、中东、印度这些地区的节点流量增长还是非常快的。与国外云厂商相比，技术及产品层面来讲，国内的视频云与 Akamai、AWS 的产品平台还是有些差异的，国内的视频云从视频的采集推流到云端转码处理再到 CDN 分发，包括最后的终端播放，基本是一套相对耦合的闭环方案；而像 AWS 这样的云厂商，平台产品会相对解耦一些，独立的源站及转码模块、视频传输模块、CDN 模块，每一个都可以独立接入服务，彼此可以不用相互依赖；从格式及协议来说，海外 HLS、DASH、CMAF 占主流，而国内视频云一般是 RTMP 上行，FLV、HLS 下行为主，在这个层面来说，国内客户出海基于国内视频云团队的方案，业务接入上会相对平滑一些；

4.3 开放媒体联盟

在下一代视频边面标准方面，腾讯视频云直播、点播、媒体处理全线产品均已支持 AV1 标准，腾讯云也是国内首家直播+点播同时支持 AV1 视频处理业务的公有云厂商。在今年 10 月，腾讯已正式加入开放媒体联盟(Alliance for Open Media，AOMedia)，推进视频 AV1 标准商业化，并成为董事会成员之一，也是董事会成员中迄今为止唯一的一家中国企业。腾讯多媒体实验室和腾讯云正积极推进视频 AV1 标准的商业应用。AV1 中的极速高清转码服务，可以让用户在低码率下享受与高码率相同的画质体验。通过极速高清转码技术，直播、点播业务相比开源软件在相同画质下平均分别节省超 30%+和 40%+的带宽成本。或在同等带宽条件下，提供更加高清优质的视频画质体验。腾讯云一直以来都潜心深耕音视频领域，不断完善自身产品体系，并结合用户业务场景不断创新，为用户带来极速高清、画质增强、老片修复等技术产品。

5. 共建开发者生态

近两年，腾讯云正积极的参与各类技术开源社区，共建开发者生态，举个例子，腾讯视频云提供了近 20 套各类音视频处理 SDK，其中最主要的 SDK 对应 demo 的源码我们都分享在 GitHub 上供开发者参考，极大降低开发者的工作量及集成门槛。

综上，腾讯视频云整个产品矩阵，从最初的底层基础设施发展到音视频 PaaS 平台，再到目前在多个场景积极的尝试 SaaS 应用，从内容源端到视频云端再到播放终端，也提供了层次丰富的产品方案方案，面对不同垂直场景和行业的能力，我们又推出了多达 17 套的行业场景方案。我们通过整合这些技术能力、基线产品及解决方案，将其对外开放给到开发者和合作伙伴，构筑更好的大视频生态。

新浪微博：大规模离线视频处理系统的架构设计

微博视频平台在 4 亿月活用户吃瓜嗨聊的高并发、大流量背景下，既要保证用户微博生产和消费体验，又要支持业务快速迭代，确保正确性、稳定性和高可用性。本次演将以微博视频大规模视频离线处理系统的架构设计为主题为大家带来大规模分布式系统的架构设计，性能优化和高可用保障等一线实战经验。

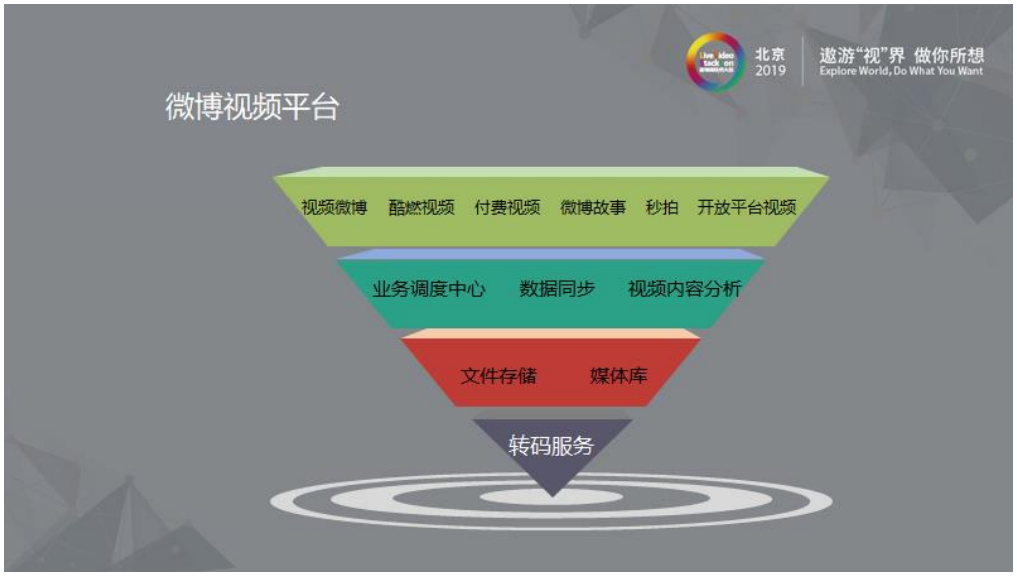
霍东海　LiveVideoStackCon 2019 北京　2019-11-21

大家好，我是来自新浪微博视频平台和微博平台研发部的架构师霍东海，从 2017 年加入微博，目前在微博视频平台负责微博视频离线处理系统架构等相关工作，包括大规模离线微服务系统的架构设计和服务保障体系的建设等。近期专注于视频平台技术体系的提升对用户体验提升的帮助，主导构建了微博 SVE(Streaming Video Engine)系统，支持大并发场景下对视频进行并行转码，大幅度提升转码效率。

1. 背景介绍

微博本身有大并发、大流量的特性，有 4 亿+的月活，同时微博也是一个开放平台，支持多种第三方分享，每天都会有百万视频分享需进行处理。

微博视频业务大概分两种业余形态，一个如左图所示，是竖版的短视频分享，另一个是如右图所示的稍微长一点的横向播放的短视频。微博视频还有一些特殊的场景，例如在微博 PC 端点视频按钮会跳转到酷燃网，它是一个 5 到 15 分钟的短小综艺类视频分享的网站，如图中，下面都是一些优酷，爱奇艺，腾讯等视频网站分享到微博的视频。

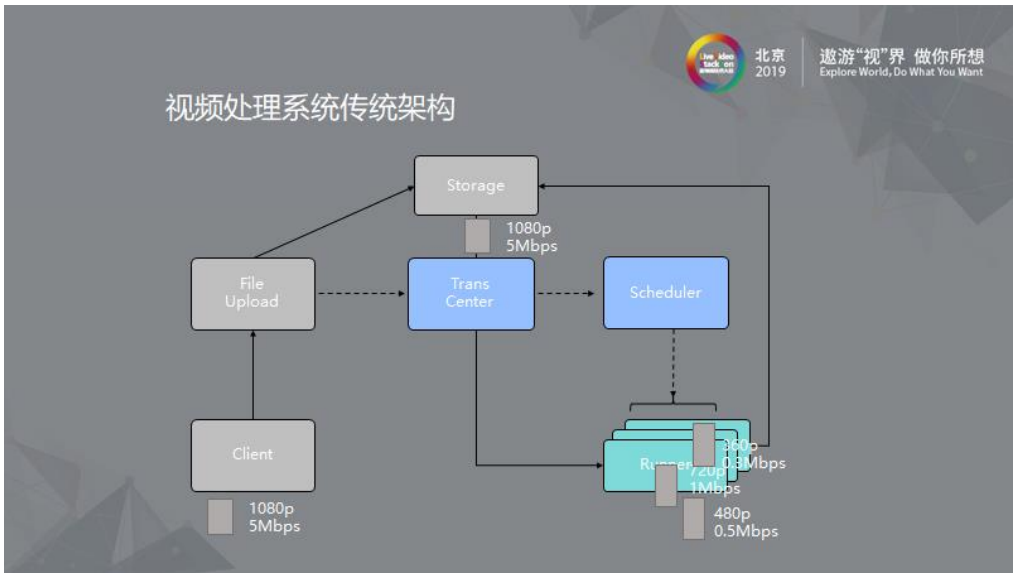


我们微博视频团队面临的业务场景是及其复杂的，我们要在复杂的场景下解决视频处理的问题。如图中，我们有微博视频，酷燃视频，付费视频，微博故事，秒拍，以及通过开放平台接入的视频分享网站，微博在最上层会接入极其多的业务方。中间会引入业务调度中心，即业务调度层，对上层业务进行调度。

另外是数据同步，所有的视频呈现在微博都是博文的形式，始终是需要和我们自己的系统进行交互的。业务调度层的另外一个作用就是对视频内容进行分析。往下一层是文件存储，媒体库层。文件存储包括文件上传，文件存储方面等的问题。媒体库是视频对象的源信息，如视频分辨率 URL，视频长宽，用户 ID，博文内容等信息存储。最下层是转码服务。我们重点介绍的就是转码服务在微博场景下的思考。

2. 微博视频转码服务架构与挑战

2.1 视频处理系统传统架构



在讲微博面临的问题之前，先来了解一下视频处理系统的传统架构。例如，某一用户在 PC 端或手机端有一个 1080p，5Mbps 的视频需要上传。

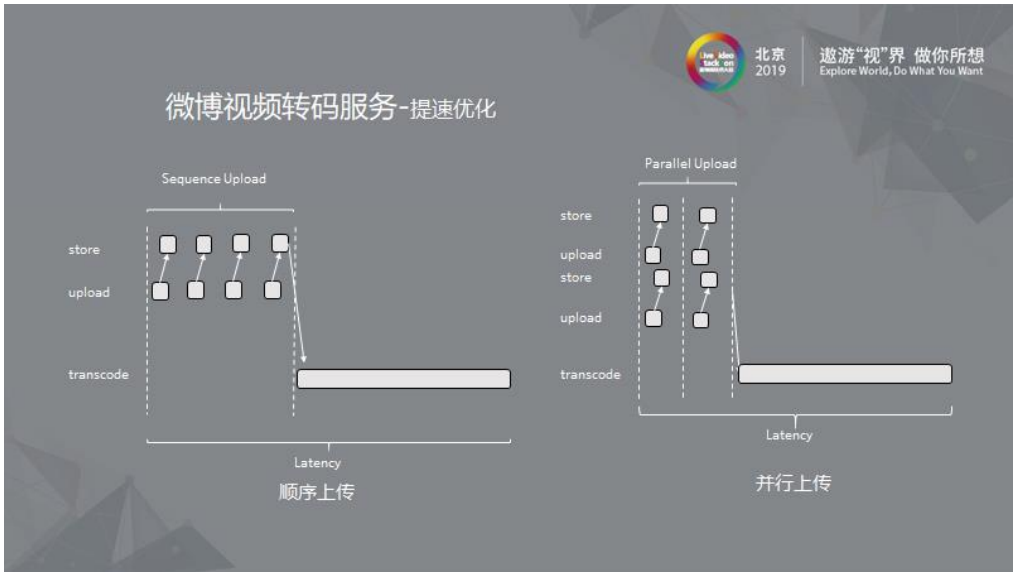
在传统的架构中，会先将文件传到文件上传服务，文件上传服务将其传到底层存储。传到存储后，文件上传服务会告知转码服务文件需进行转码。转码时转码服务通过调度器将转码任务传到对应的转码集群中的转码服务器。真正转码的机器，从存储中下载用户上传的源文件，转换成特定格式后回存到存储中。

2.2 微博视频转码服务 – 业务繁杂

对微博视频而言，我们有非常繁杂的业务，例如业务方会有不同的水印，一些用户会对自己的视频有特殊要求，另外系统要能满足线上验证优化转码算法的需求，再加上转码服务本身会提供抽帧等基础服务，要使这些融合在一起快速方便的支持业务方的需求，我们面临很大的挑战。

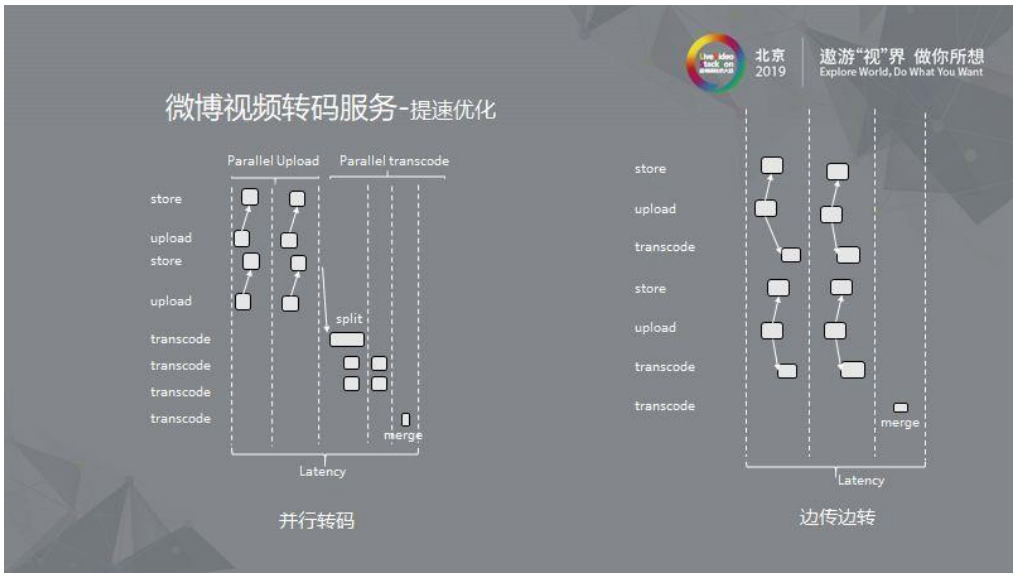
2.3 微博视频转码服务 – 提速优化

另外，在优化视频基础体验的时候，我们会提出并行上传来提高用户上传成功率，做类似断点续传的功能，我们还会做并行转码完成云厂商提出的分片转码。甚至我们做到了用户边转边存，使视频在用户手机端完成分片，一边分片一边上传，上传的同时后台进行转码，上传完成的同时，转码即可完成，最后合并视频完成发送。这极大的提高了用户上传视频到发布微博这一过程的体验。



提速优化举例

第一个是顺序上传。现在顺序上传的过程一般是二进制切片，切片后依次上传，整个系统延迟会比较长，做并行上传时，例如两个进程同时上传，会比顺序上传提速一倍。

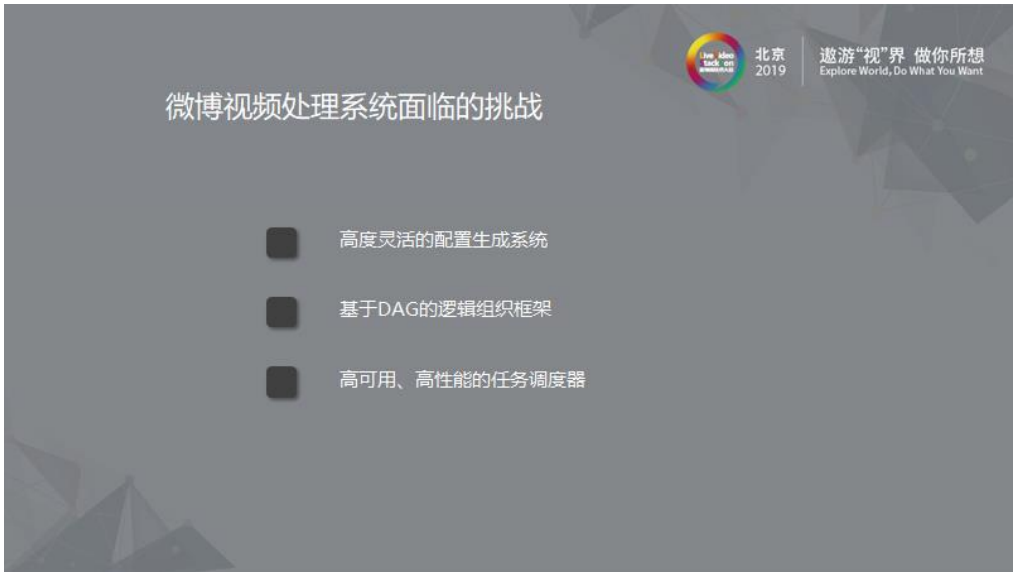


在做并行转码时，相当于把视频做成二进制分片上传后，合并起来进行转码。转码时再将视频切分成不同时长的片段进行分片转码，完成后合并视频。这种方式下通过提高并行程度降低了延时。在边传边转方式下，客户端上传存储后，马上进行转码，客户端操作与服务端服务并行，最后服务端会将源视频、目标视频分别合并。

2.4 微博视频处理系统面临的挑战

我们面对业务繁杂，需进行基础服务优化的双重挑战。另外，微博业务具有很强的实时性，这就要求我们每环节都得快速完成，包括我们实现代码的时间，接入业务方上线的时间。我们必须实现一个低延时、高并发、高可用、高性能的视频转码服务。

视频转码服务本身需要大量计算，需要大规模的集群支持这项服务。我们面临的另外一个挑战就是对大量集群的管理。由于我们使用了分片转码，边传边转的优化方式，一个视频切成十片，转码量会变成十倍，这导致转码任务量陡增，同时也会产生一个更细粒度的调度。切片给我们带来更加复杂的任务依赖关系，我们要管理切片、分片并行转码以及合并整个过程中的任务依赖。过程中步骤越多，失败率越高，越要求系统有更高的健壮性降低失败。



我们今天主要讲的就是如何实现一个低延时、高并发、高可用、高性能的系统，我将主要从以下几个方面来说明。首先是高度灵活的配置生成系统，相当于将业务相关的东西从主系统中抽离放到配置系统中，使主系统专注于基础性能优化和基础服务。第二点要讲的是基于 DAG 的逻辑组织框架即用工作流引擎去组织任务之间的依赖。最后会讲一下高可用、高性能的任务调度器对系统的重要作用。

3. 微博视频转码服务架构设计

3.1 木林森

高度灵活的配置生成系统

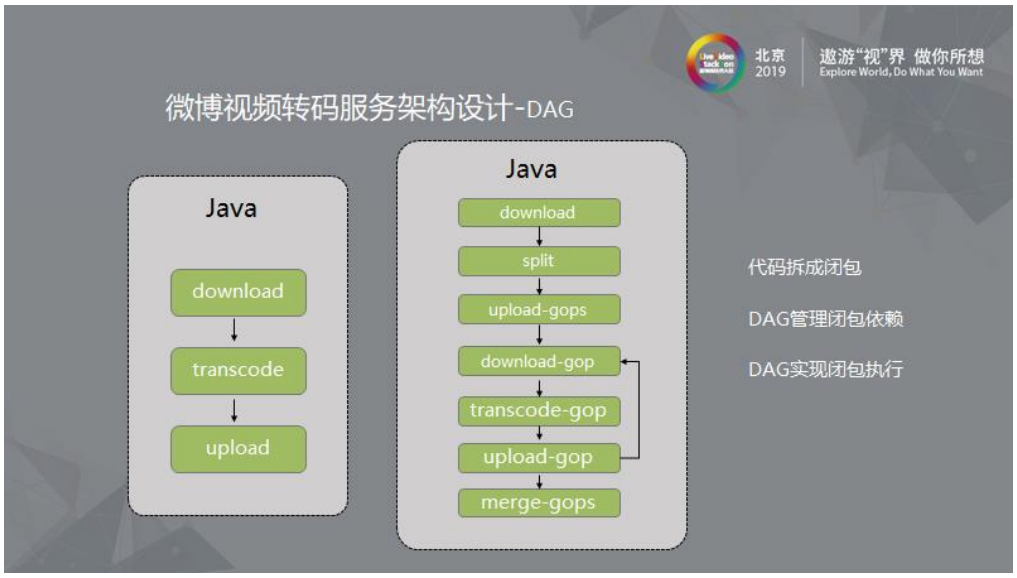
对于灵活配置，我们取名为木林森。它是一个基于树形结构的规则引擎，即我们的配置结果都是树形结构，多棵树即可组成森林，所以我们取名为木林森。

木林森支持灵活的配置生成。微博有些业务场景下产品方只要求快速而不在意视频输出属性，这时我们可以直接使现有输出业务与输入业务连接完成业务接入。用这种方式可以提升新业务接入效率。下面是一个简单的示例。

如图，例如我们有微博原生视频接入业务，现在要接入的亿幕视频希望与原生视频有相同的输出。此时我们在输出业务以下到转码输出都不需要改变，我们只需要将节点连接，输入的亿幕视频就与微博原生视频有了相同的输出。只需通过后台点击配置就可将视频接入。右图所示是微博视频的输出配置。

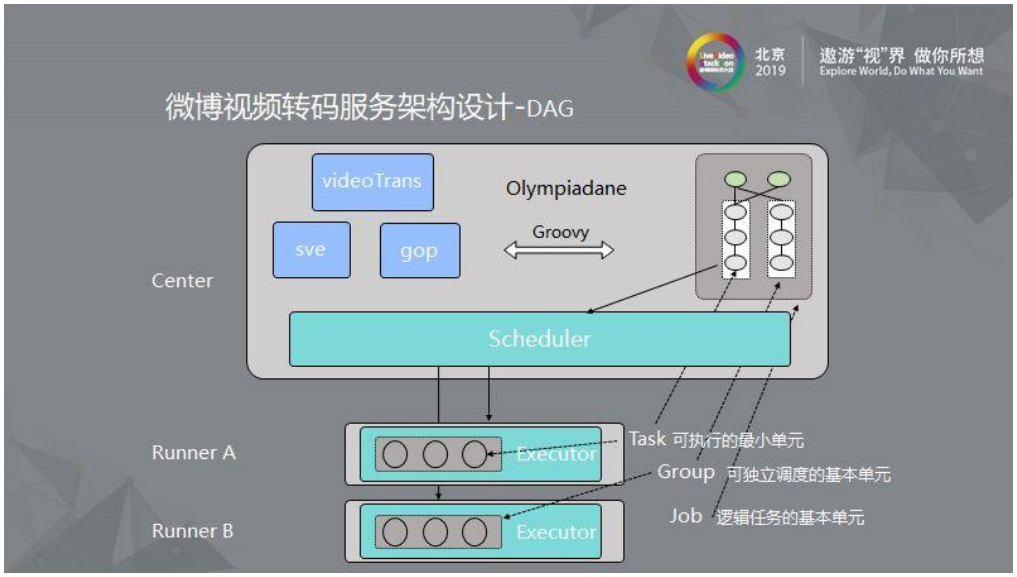
复杂场景下，原生视频，秒拍视频，VIP 视频的输出业务配置如图。不同用户端视频经过系统输出的视频是不同的。通过配置可完成复杂场景下的业务逻辑抽离。

3.2 DAG



基于 DAG 的逻辑组织框架

我们自己实现了一套工作流引擎框架来支撑我们的业务，首先介绍一下框架的思想。我们是基于 Java 开发的，这里用 Java 举例。对于一般的上传系统，代码实现只有下载、转码、上传的过程。在这一段代码的基础上，我们要实现分片转码，边传边转等复杂的逻辑流。最简单的方法就是我们将一般上传的代码复制改动，这时我们的方式如右图。上传过程变为下载，切片，将切片结果上传，下载切片，切片转码，上传切片，然后使这个过程循环往复，这时可多台机器并行工作，最后将切片合并。由于过程复杂，所有我们希望能用有向无环图连接组织，将基础服务固化，通过脚本将不同功能组织起来。这时我们无论下载文件转码上传还是分片上传，只需简单连接即可，两个下载之间的代码是不需要改动的。我们将可固化的部分固化，将代码拆成一个个可独立执行的闭包，通过 DAG 管理包与包之间的关系，在 DAG 内部实现闭包的执行。这就是我们关于 DAG 框架的想法。



这是我们转码服务的图示。如图中，Center 部分就是中央调度的服务，Runner 部分是执行转码任务的服务，videoTrans 是 DAG 组织任务间关系的脚本。我们的脚本通过 Groovy 实现。框架名字叫做 Olympiadane，是通过 Groovy 引擎执行连成的关系，其中的 Group 是可独立调度的单位，即图中白色的部分。任务先经过调度器，调度器根据情况分发到执行器，执行器内部根据前后依赖关系顺序执行 Task，在此例中就是下载，转码，上传。另一台机器也是一样的。这样就实现了执行流与业务之间的解耦，如果要接入其他的新服务的话，我们只需要再实现一个 Task，将此 Task 的依赖关系放入脚本即可完成。另外通过脚本生成的就是图中的 Job。这是我们 DAG 架构。

实现 DAG 框架后，可以通过脚本快速接入支持业务，由于脚本变动但 Feature 不是经常变动的，所以 Feature 可在脚本间共用，也可以独立测试，这样我们便可以完成可组装的独立组件。这些独立组件具备可以独立测试，易扩展，易部署，高性能的特性。

这里描述的是 DAG 的过程。前面我们提到的分片转码，过程分是下载，切片，上传分片结果几步。

例如有三个机器同时并行完成下载分片，转码，上传结果的工作。当并行过程结束后，会有一个新的依赖关系，如图中，下载所有分片，合并转码后的视频，不同清晰度的文件都是在不同机器上并行工作的。这是对我们转码服务优化中，通过 DAG 组织的一次实践。

如图中，灰色的部分变成了绿色，这表示这个过程是可以观测的，这也是通过 DAG 方式实现的一个优势。我们可以观测任务执行到了哪一步，也可以更快的定位出现问题的地方。另外，由于我们进行了拆分，可以对独立 Feature 进行 DAG 切面，例如我们可以统计它的耗时，这样我们可以知道哪种类型的业务耗时较长，也有助于观察系统的稳定性。

在 DAG 的优化方面，我们通过字节码编译技术做到了脚本快速执行。另外我们引入了一些 Protostuff 技术快速完成资源存储。

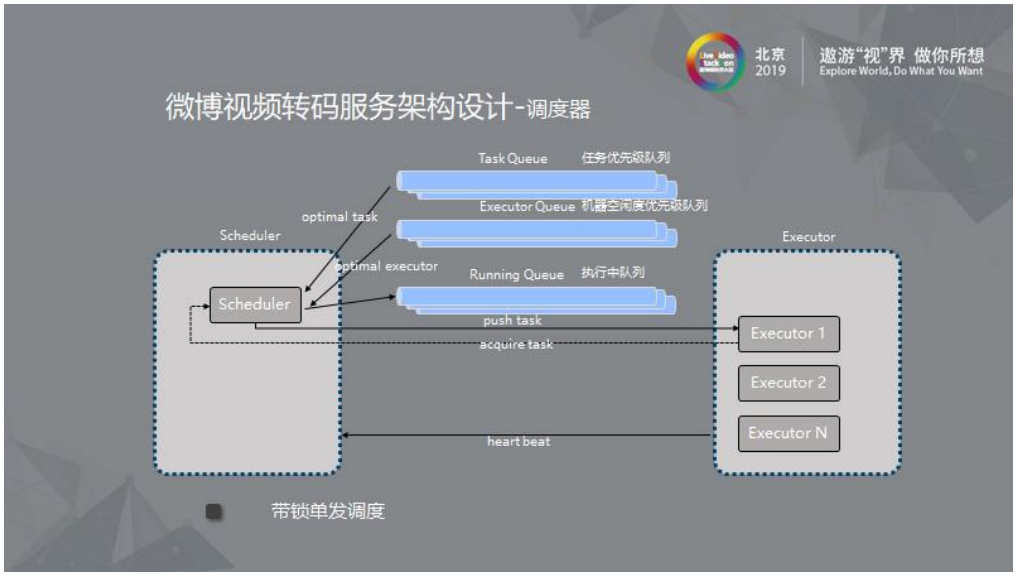
3.3 调度器

高可用、高性能的任务调度器

我们通过木林森将业务系统抽离出来，通过 DAG 系统将实现时的依赖关系抽离，因此我们需要一个好的调度系统来支撑。由于我们进行了切片，因此调度任务达到了万次每秒。我们也会需要更细粒度的调度任务，比起粗粒度的调度，对基础组件性能要求更高。

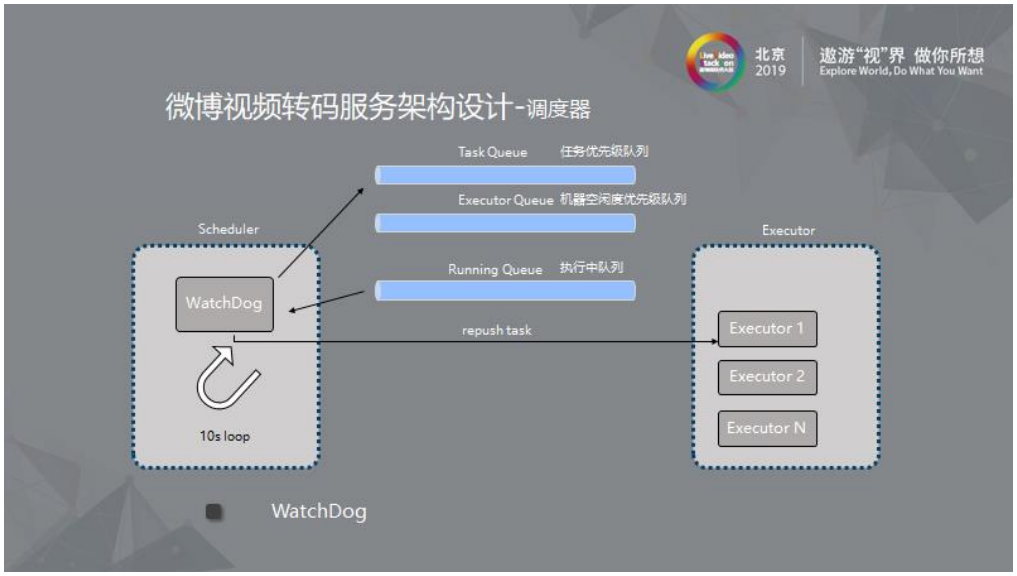
我们对调度器的另外一个设计目标就是调度占比要低于百分之五，这就意味着系统损耗更低。这对调度器有极高要求，我们要使百分之九十九的调度任务在 10ms 内分派到对应机器，并且我们希望它的调度是最优调度，即能准确把任务分派到空闲机器。

在设计调度器时我们也做了一些思考。我们对中心化调度器和非中心化调度器做了对比。中心化调度器的调度准确度高，它将资源队列信息放到中心化存储中，对监控更亲和。但是它的资源依赖较多，我们将队列放到了资源中，因此资源访问读写中会产生一定依赖，也会有一定性能损耗。对于去中心化调度器来说，它的扩展性更强，但是它存在调度不准确的问题。最终我们选择了中心化调度方式。



上图是调度器调度过程。左边是调度器，右边是执行器。调度器和执行器之间通过心跳注册，心跳时间是可配置的。注册完后会将机器信息放到机器队列，中心资源中有一个任务优先级队列，我们可以对不同任务映射不同优先级。另外一个是机器空闲优先级队列，就是我们将机器空闲度映射为优先级。在派发时，我们会取到高优先级任务，取到空闲度优先级高的执行器，然后将任务派到指定机器，即可将任务放到执行队列中。执行队列的重要作用在后面会讲到。执行结束后，会进行一次回调，从执行队列中移除任务。

我们通过三个队列完成任务调度，由于存在资源依赖，所以我们对这些资源进行了哈希计算，不同机器可以使用不同资源，只要资源满足就可分派任务。

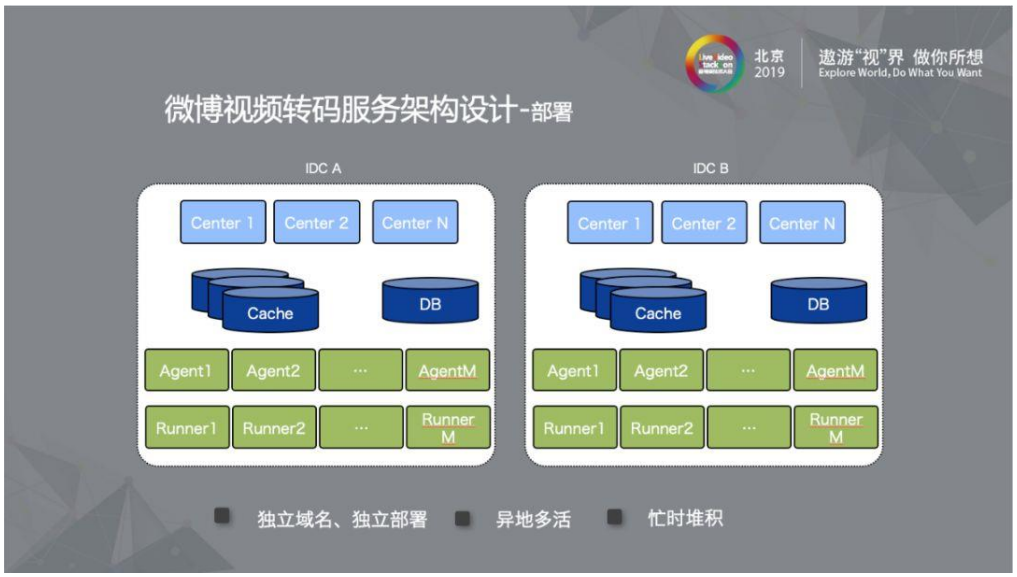


但是这里会有一个问题。心跳会汇报情况，但它会有一定延迟，如果 Executor 与资源中存储的状态产生差异，任务可能会被分派到一台无法工作的机器。

为解决此问题，我们设计了一个带锁的双发调度。与之前介绍的相同，我们依然从队列选择机器。不同的是，我们会在空闲优先级队列中取到最优的同时，取一个随机机器去完成分派。分派后，执行器会再一次调用调度器确认由谁完成任务，再去执行。当最优任务不可执行时，另外一个机器可完成任务。这样，我们就可以实现百分之九十九的任务在 10 毫秒内完成分派的目的。同样的，带锁双发调度也会有哈希计算的存在。同时，我们会使用 WatchDog 观察执行队列中的任务是否在规定时间完成，若没有完成，我们会重新触发调度器分派任务。这样我们可以有效减慢失败率提升。

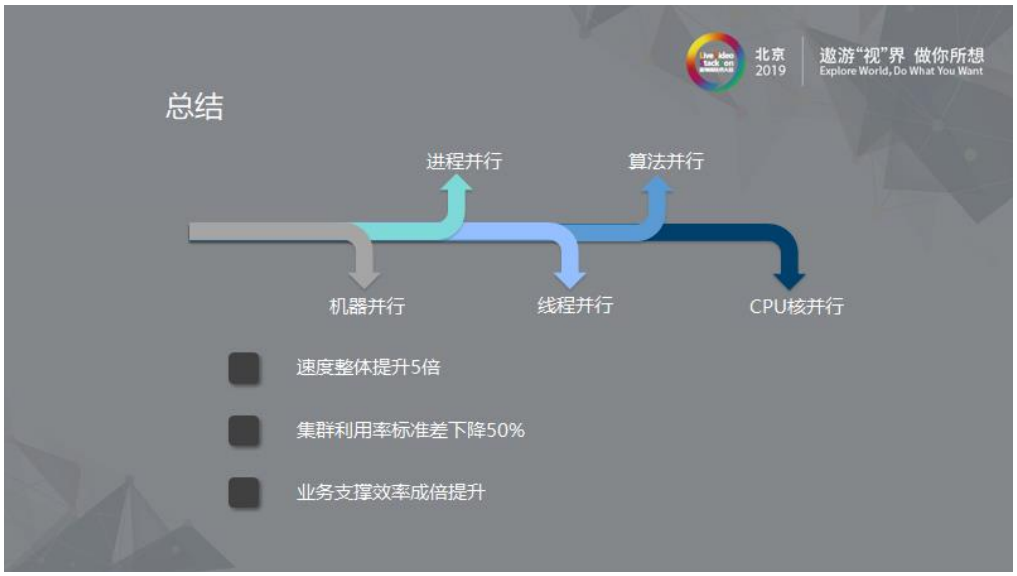
通过以上设计，我们的调度器可以实现毫秒级派发。对于微博业务来说，可能会出现紧急大流量出现，我们在设计时也考虑了水平伸缩方式，使它支持弹性扩缩容。通过 WatchDog 机制，我们可以实现宕机自动摘除。在实际应用中，我们每天都会有机器扩缩容，我们的做法是让待扩容机器不接受任务，先完成已有任务，再做机器扩缩容工作。但这样做还会有未完成的任务存在，通过 WatchDog 机制，我们可以确保这些任务重新分派完成。同时我们实现了 4 个 9 可用性。

3.4 部署



在转码服务部署方面，我们在两个 IDC 部署了完全相同的两套资源，它们有独立的域名，独立的部署。这么做的好处是我们可以在两个机房间随意的切流量，任一机房出现问题，我们都可以切换，但是两个机房的部署并不是一比一的冗余。我们常备的机房是一个大规模集群，另一个机房是一个小规模的，或许只有常备机房十分之一的量。两个机房在使用时可以分开，例如我们转一些不影响用户发博的转码输出时，可以使用小机房完成任务，这样大机房出现“灾难性”情况时，可以把流量切到小机房。当然小机房是不能满足那么大流量的，但是调度器本身的队列有堆积的特性，可以将堆积的任务慢慢执行。没有大量机器冗余可以充分利用机器。

4. 总结



接下来，我将所有内容做一个总结。通过转码服务的优化提升，我们整体转码提速了 5 倍，集群利用率标准差下降了百分之五十。这里的标准差指由于分派任务不合理，分到不同清闲度的任务导致机器 CPU 利用率忽高忽低的情况。通过好的调度方式，对集群的利用率标准差有很大降低。另外，我们业务支撑的效率也成倍提升。我们通过各种方式进行解耦，将变化大的和变化小的分开放到不同位置。

在服务架构设计开发的过程中，我们使用了很多并行手段，包括机器并行、进程并行、线程的并行以及算法、CPU 核的并行等，通过这些手段发挥机器最大的价值。今天我们设计的目的是一个低延时高可用的系统，我们用到的并行上传，并行转码，极致上传等手段都用了算法上分治、递归、贪心的思想。在高可用方面使用了高内聚低耦合的思想，用到了动静分离、自动容错、异地多活的手段。最后想和大家分享，我们在做系统架构、设计优化不知道该如何实现的时候，就可以无脑的把这些高内聚低耦合、空间换时间等常见的思想往系统上套，或许就可以得到想要的结果。优化架构并不是多么复杂深奥的东西，我们要去思考的是前人总结的这些手段在我们自己系统上的能否达到效果，如果得到了满意的效果，那么这些思想就会转化为我们自己的东西。

跨 Web 与客户端的优化之路

360 视频云 Web 前端 HEVC 播放器实践剖析

360 视频云前端团队围绕 HEVC 前端播放及解密实现了一套基于 WebAssembly、WebWorker 的通用模块化 Web 播放器，在 LiveVideoStackCon2019 深圳的演讲中 360 奇舞团 Web 前端技术经理胡尊杰对其架构设计、核心原理，具体痛点问题的解决方式进行了详细剖析。

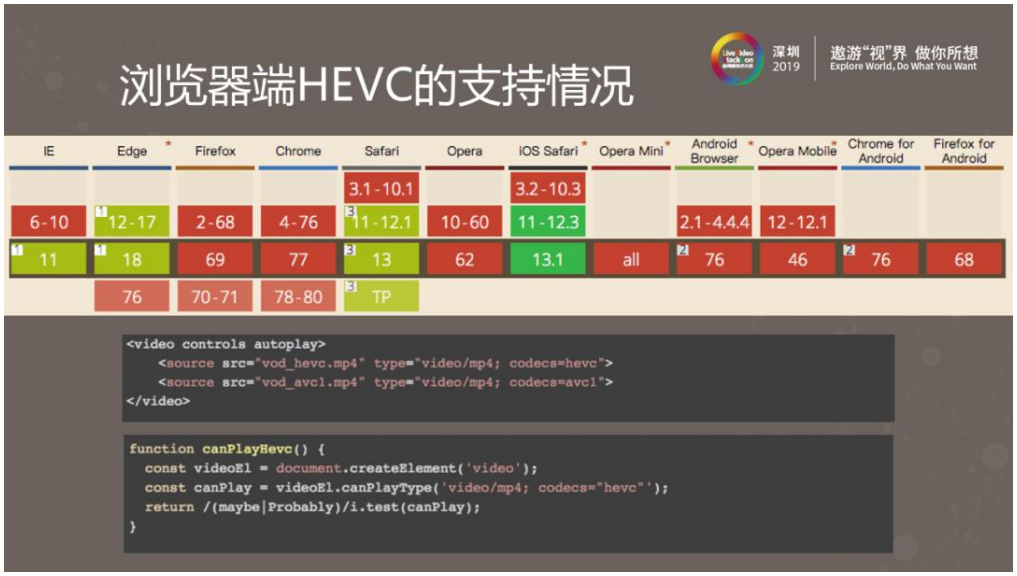
胡尊杰　LiveVideoStackCon 2019 深圳　2020-04-07

奇舞团是 360 集团最大的大前端团队，同样也是 TC39 和 W3C 会员，拥有 Web 前端、服务端、Android、iOS、设计、产品、运营等岗位人员，旗下的开源框架和技术品牌有 SpriteJS、ThinkJS、MeshJS、Chimee、QiShare、声享、即视、奇字库、众成翻译、奇舞学院、奇舞周刊、泛前端分享等。

奇舞团支持的业务基本上涵盖了 360 大部分业务线。我个人最开始的时候也曾带队负责 360 核心安全平台的 Web 前端支持，包括大家耳熟能详的安全卫士、杀毒软件等。随着公司的业务发展，后面也负责了 IoT 业务前端支持，最近两年主要配合 360 视频云的一些 Web 前端支持工作。基于 HEVC 的播放器，实际上就是来源于我们最近做的一个叫 QHWWPlayer 的播放器。HEVC 并不是一个新鲜事物，但对于我们团队来说，Web 前端的 HEVC 播放器一直是个亟待优化的领域。虽然移动终端或 PC 端 HEVC 播放器已经遍地开花，但在 Web 端仍旧有很多地方需要改进。包括现存一系列智能硬件产品，也在固件采集端已经应用了 HEVC 的编码，不过如果想让其在 Web 端呈现并达到用户需求仍需加倍的努力。本次分享将从以下几个维度展开，希望能给大家带来一定的参考价值。

1. 需求背景

1.1 浏览器端 HEVC 的支持情况



上图展示了 HEVC 在浏览器端的支持情况，其中红色代表不支持的浏览器对应版本，绿色代表对 HEVC 具有良好的支持，青色代表无法保证浏览器可以很好地支持 HEVC。总体上来说 HEVC 在浏览器端并不是一个得到广泛支持的靠谱方案。

一般情况下，PC 端浏览器都给我们提供了相应的 API，如果我们的业务场景是支持 HEVC 的浏览器，可尝试有效利用浏览器的原生能力。基于浏览器原生 video，配置 source 时指定解码器，告知浏览器当前视频采取的是哪一种编码方案。如果浏览器自身有能力进行解码那么其自然会走入"支持 HEVC"的逻辑分支当中。也可以另外通过 JS 实现检测功能，JS 也提供了相应 API——canPlayType 来判断当前浏览器环境是否支持 HEVC 解码。

但如果以上流程无法得到有效支持呢？这也是本次分享我们讨论的重点。

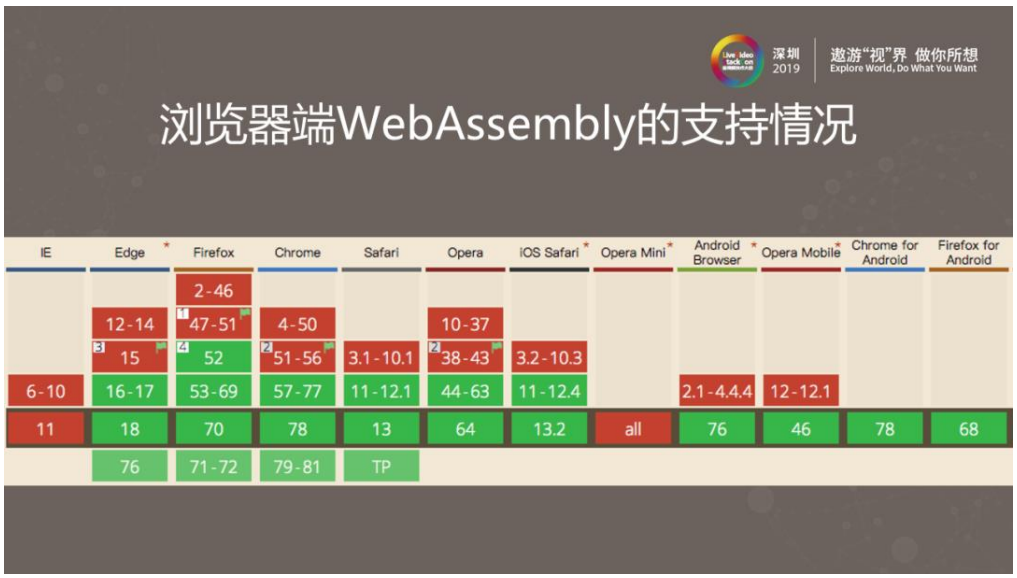
1.2 Web 端解码方案

浏览器端视频解码总共有以上三种方案，首先就是前文我们提到的基于浏览器原生能力的播放，例如基于 video 标签拉流、解码以及渲染播放，整个过程完全由浏览器实现。第二种方案是首先通过 JS 来下载视频流、对视频流进行解封装与转封装处理，最后再通过浏览器提供的相关 API，交由浏览器原生 video 进行解码与渲染播放。如开源社区当中的 HLS.JS 或 FLV.JS 等就是基于该思路。

但是 HEVC 不能仅靠解封装与转封装来实现，因为其本质上在解码层就不支持。因此第三种方案就是：JS 下载的视频流首先经由解封装（解密）处理，并在接下来进行解码，解码完成后渲染播放。如果我们这里转成浏览器普遍支持的解码格式并让 video 标签进行播放，尽管理论上可行，但成本显然是非常高的，并且中间存在一个无端的浪费。因此这里通常直接采用浏览器端 Canvas+WebAudio API 实现视频与音频的渲染，而不再使用浏览器原生 video 能力。这里如果使用纯浏览器原生的 JS，由于 JS 天生单线程执行的弱势，会导致整个处理的效率比较差。

近期，万维网标准化委员会正式推出了 WebAssembly 规范。一方面我们可以借助 WebAssembly 高于 JS 的能力，实现更加出色的大规模数据处理与解码，另一方面基于 WebAssembly，我们也能方便地将传统媒体处理中基于 C 或 C++开发的一些媒体处理能力集成在浏览器端执行，并且可通过 JS 来调用 API。对于熟悉传统 Web 前端开发的我们来说，这也是一个值得我们坚持探索与实践的全新领域。有了 WebAssembly 之后，我们就可以让部门内擅长视频处理的专家级同事来配合实现更加出色的浏览器端视频播放，相对以往的开发流程来说，无论是能力、成本控制还是效率与灵活程度都有十分显著的提升。

1.3 浏览器端 WebAssembly 的支持情况

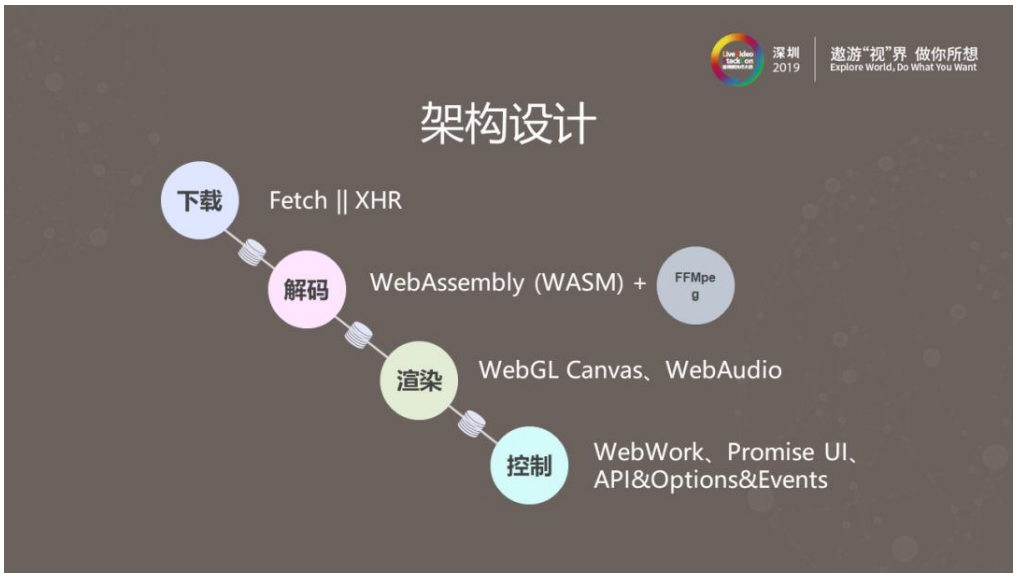


上图展现了浏览器端 WebAssembly 的支持情况，尽管个别低版本的浏览器有一些支持限制，但随着标准化委员会对该标准的不断推进，情况会变得越来越好。在包括一些混合式场景，例如 APP 内嵌（比如聊天工具或通讯工具当中打开一个链接）等情况，是否支持也取决于 WebView 本身提供的能力以及 WebAssembly 的支持情况，总体上来说趋于向好。

1.4 HEVC 播放器需求目标

HEVC 播放器的需求目标，就是基于 JavaScript 相关 API，配合 FFmpeg+WASM 达成 HEVC 在浏览器端的解码&解密、渲染播放的需求，接下来我们就开始研究如何落地这一目标。

2. 架构设计



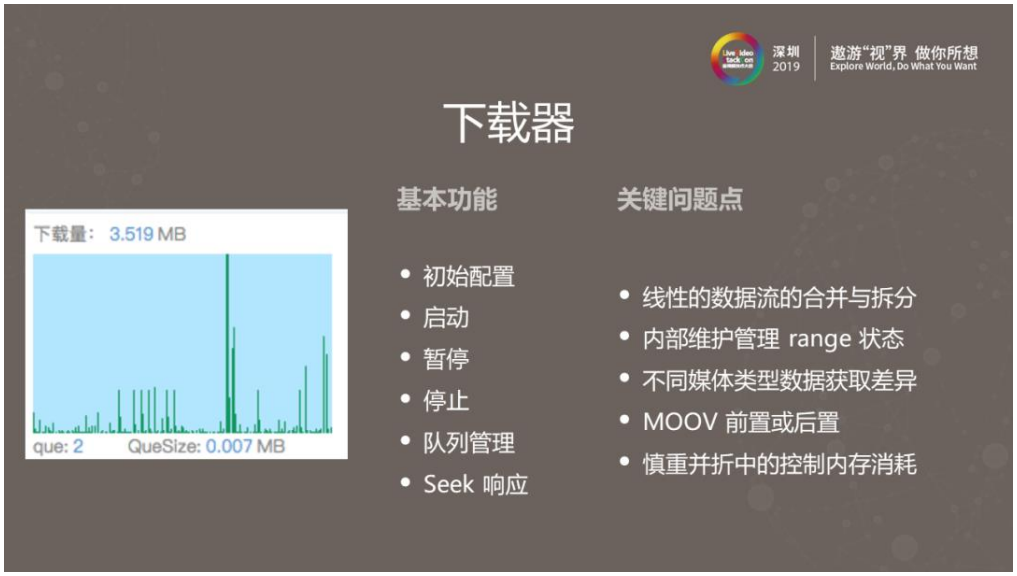
总体架构设计思路如上图所示，首先我们需要一个专门负责下载的下载器，该下载器也是基于浏览器的 JS Fetch 或 XHR API，以实现文件获取或直播拉流等操作。成功拉取的视频流会被存储在一个数据队列当中，随后基于 WebAssembly（WASM）+FFmpeg 的解码器会来消费处理队列里这些流数据，解码出音视频数据，并放置在音视频帧数据队列当中，等待随后的渲染器对其进行渲染处理。渲染器基于 WebGL+Canvas 与 WebAudio 调用硬件渲染出图像与音频。

最后则是控制层用于贯穿整体流程中下载、解码、渲染等独立模块，同时实现底层一些基本功能：如之前我们提到 JS 为单线程，而浏览器提供的 WebWork API 可拉起一个子线程。该流程中每一个模块都是独立的，队列中的生产与消费过程也是异步进行的。(我们可基于 JS 本身一些比较好的特性实现诸多便捷的功能。例如基于 Promise 可以将异步过程进行较为合理的封装，并呈现一些异步处理逻辑流程的关键环节的控制到 UI 层。)

除此之外，还有控制层的一些基础配置选项，包括播放器本身的一些事件或消息的管理，都可以基于控制层来实现。

3. 分解实现

3.1 下载器



下载器作为一个基本模块独立存在，具有初始配置、启动、暂停、停止、队列管理与 Seek 响应（用于进度条拖拽）等基本功能。上图左侧图标是在开发完成后，基于下载器的事件消息呈现的数据可视化结果。（柱状图表示单位时间下载量，这里我们可以看到的是，下载量并不均匀，其中的变化可能取决于推流端、服务端、用户端，也可能取决于整个网络环境。）

下载器方面需要留意五个关键问题点：

线性的数据流的合并与拆分

我们应当进行线性数据流的合并与拆分。理论上浏览器从服务端下载一个视频流的过程是线性的，但浏览器的表现实际上并非如此，二者的差异可能会很大。

例如当一个浏览器启动并基于 JSFetch API 抓取流，其过程也是通过 API 监听数据回调来实现，每次回调可能间隔会很短、数据量也只是一个很小的一千字节左右的数据包。但有些浏览器的表现并非如此，它们会等抓取到一个 1M 或 2M 的数据包之后才反馈给 API 回调。

而那些过于零碎的数据直接丢给队列或之后的流程来处理，这样势必导致更频繁的数据处理；数据包体积大的直接队列和后续流程势必增加单次处理成本。因此对线性数据流的合理合并与拆分十分必要，整个过程也是结合初始配置来实现阈值控制。

通过阈值调节控制，我们希望能够做好用户端浏览器硬件资源消耗，与该业务场景下媒体播放产品服务体验之间的取舍与平衡。

内部维护管理 range 状态

除此之外，下载器实际上也需要内部维护管理 range 状态。例如当用户选择点播时，我们需要明确是从哪一个字节位置到另一个字节位置下载传输中间这一片数据。而在直播过程中，则可能出现由网络环境造成卡顿或用户端主动暂停的现象，此时下载器需要明确知道播放或当前下载的位置。

不同媒体类型数据获取的差异

第三点是不同媒体类型数据获取的差异，也就是下载器针对不同的媒体类型开发不同的下载功能。例如一个 FLV 直播流可以理解为是一个连续的线性的数据获取，而点播则以包为单位获取。对于 HLS 流需要获取 m3u8 列表，完成分析之后再从中选取数据包的地址并单独下载，随后进行流的合并或拆分。总地来说，我们需要保证数据的最终产出尽量均匀存储到队列中，以便于后续的一系列处理。

MOOV 前置或后置

在媒体处理中像 MOOV 等的索引数据有前置与后置两种情况，这里需要注意的是，我们的播放器基于 Web 端。

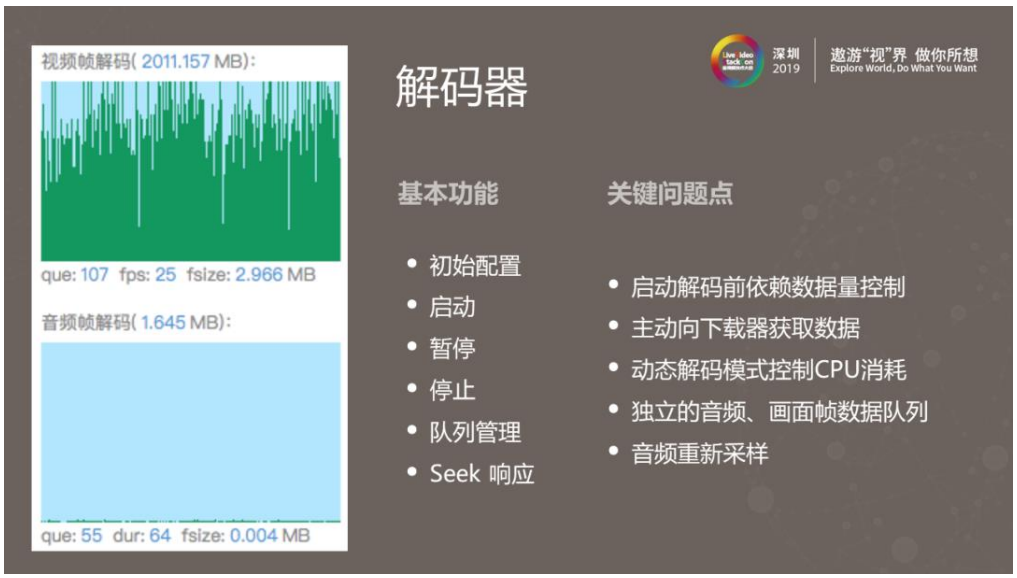
若索引文件为后置，如果播放器直接下载了一部分数据就直接丢给 FFmpeg 解码器进行解码，由于 FFmpeg 解码器无法获取索引，当然也就无法解码成功。除非解码器等待整体媒体源下载完毕，实际上这样是不现实的。

另外由于我们无法控制 MOOV 索引数据的体量，前置索引的大小无法确定，尤其对于一些特殊情况，这种逻辑会带来很多问题。（但是这里有一个取巧的办法，就是我们可以尝试首先抓取前面几个数据包，探测 MOOV 边界，并基于此得到 MOOV 的长度，从而判断取舍在什么时机启动后续的解码。）

慎重并折中的控制内存消耗

最后，慎重并折中控制内存消耗也至关重要。例如尽管较大的缓存能带来流畅的播放，但在 Seek 时就会带来很大的浪费，我们则需要根据服务所在的应用场景、帧率码率等来实现合理的折中与取舍。

3.2 解码器



下载器之后，整个流程的核心能力就是解码器。解码器的基本功能与下载器相比大同小异，需要特别关注的是解码器并不是像下载器完全是去调用一个原生的 JS Fetch API 或 XHR，而是在启动 WebWorker 之后再启动 WebAssembly（这里的 WebAssembly 依赖中是引入了定制化的 FFmpeg API，以解决解容器、解码等需求），并实现一些 API 的交互。上图左侧展现了音频与视频帧解码数据队列的可视化结果。

解码器方面，需要关注的关键问题主要有以下几点：

启动解码前依赖数据量控制

刚才讲到 MOOV 前置与后置时我们也提及这一点，也就是在启动解码前做好数据量控制，明确其数据量是否已经达到 FFmpeg 的基本需求。如果索引文件的数据还没有完全给到就直接使用命令行启动 FFmpeg，那么就会出现报错的情况。我们应当结合数据量的精准控制来对解码器的启动时机做合理的判断。

主动向下载器获取数据

解码器需要主动获取下载器生成的数据队列，这样系统便可根据数据消费效率获知当前解码器是否处于繁忙的状态。同时，主动向下载器获取数据也能在一定程度上减轻 CPU 的负担，并可根据 CPU 的负载来决定当前从下载端应该获取多少数据。例如如果 CPU 负载较大则数据队列自然会出现累积，我们可以在下载器初始化时设置一个阈值，如果数据队列积累达到该阈值则下载器暂停下载，这样就可合理控制处理的整体流程并确保播放的正常。

动态解码模式控制 CPU 消耗

整个解码过程实际上还依赖 CPU 的性能，如果单帧解码的时间较长，例如一个帧率是 25 的视频，仅单帧解码就需耗费半秒钟甚至更长时间，此时如果我们依然按照这样半秒钟或更久的频度解码，则解码数据生产效率完全跟不上渲染的自然时间进度，效果肯定不符合预期，播放也会断断续续。因此我们需要针对不同的应用场景，使用动态解码模式（主动丢帧）控制好 CPU 的消耗。例如在直播或安防场景下，我们可以舍弃一些指标以保证解码与传输的时效性。

独立的音频、画面帧数据队列

如上图左侧所示，独立的音频与画面帧数据队列分别管理；比如我们启动丢帧策略的话，会看到画面帧数据量变少，但声音没有变化。

音频重新采样

采集端编码数据的音频采样率需要结合播放端的支持情况来留意兼容问题。浏览器是一个比较特殊的应用场景，各浏览器对音频渲染中采样率的支持程度也是不同的。

例如安防场景对声音的要求并不是很高，通常 16,000 的采样率即可，但是如果想在浏览器端播放视频，则部分浏览器要求至少 22,050 的采样率，否则浏览器端播放无法成功识别并渲染音频数据。FFmpeg 本身可以进行音频重新采样，因此我们可以在解码器端加入相应的配置项，如果用户有该需求那么就可以启动音频重新采样，重新把 16,000 的音频采样率重采样成符合浏览器所要求的 22050 采样率。有了符合要求的独立的音频与视频数据帧队列，接下来也自然就能基于浏览器实现对音视频的渲染与呈现。

3.3 渲染器

渲染器的基本功能与下载器、解码器相似，不同之处在于以下几个关键点：

依赖解码、UI 提供画布

渲染器需要浏览器提供一个独立的画布用于绘制相应的视觉画面内容。在 UI 模块初始化时呈现出一个画布的容器，渲染器渲染生成的画面才能表现在网页上。

除此之外，渲染器依赖解码器解码生产出的音视频帧数据才能进行音画渲染。

主动向解码器获取帧数据

这一点与解码器向下载器主动拿数据相似。

分缓存队列、渲染队列

渲染器会消费处理等待渲染的帧数据队列，只不过帧数据会被分为缓存队列与渲染队列。

而之前我们介绍的下载器与解码器，本身只有一组数据队列。为什么要这样呢？渲染器调用 WebAudio API 将音频数据传输给浏览器进行 PCM 渲染时，无法将已经通过该 API 传输给浏览器的数据做取回控制，因此就需要记录当前已经给了多少数据到浏览器，这就是“渲染队列”。而“缓存队列”则是从进程中获取一部分数据先存储在一个临时队列当中，从而避免频繁地向处于另一个独立 WebWorker 中的解码器索取其音画帧队列数据，而带来不必要的时间消耗。

音画同步、倍速播放、Waiting

音画同步、倍速播放以及判定是否处于等待状态至关重要。比如要追求直播的低延时，网络抖动导致数据堆积发生的时候，倍速追帧是个有效的办法。

动态码率变化

一个视频在播放的过程中，可能随网络状态的波动出现码率的动态变化，例如为适应较差的网络状况，播放器可以主动将媒体流获取从一个较为清晰的高分辨率变化到一个比较模糊的低分辨率源。

而再渲染中，基于 WebGLCanavas 的渲染器，我们首先需要对 YUV 着色器进行初始化操作，而 YUV 着色器的初始化，依赖于其所绘制的数据对应的分辨率、比例与尺寸。如果最开始的分辨率、比例和尺寸与之后要渲染的数据不一样，而我们又未对此做相应的响应适配，那么就会出现画面绘制花屏的情况。而动态码率变化就是要随时响应每一画面帧所对应的分辨率变化，对 YUV 着色器作动态调整，从而保证画面的实时性与稳定性。

从下载、解码到渲染，视频播放器的基本流程就此建立，播放器便有了获取媒体数据、完成解码、呈现音画效果的基本能力。

3.4 UI

基本的 UI 如上图左侧所示，上半部分是整个播放器在实例化之前我们可以去做的一系列初始化配置。图中所示的仅是一小部分参数，例如媒体源的地址、是否启用了加密 Key、对应的解密算法，包括渲染时为满足某些特定场景下的需求，音视频是同时进行渲染还是在主动控制下仅渲染音频或视频——例如在安防监控业务场景，会有一些设备需要音频采集、另一些不需要，或者干脆播放时就不想播放源流音频等等。若在这里播放器不做判定支持，则存在由于音画同步控制依赖音频帧视频帧时间戳比对，但没有音频帧数据的原因导致无法正常播放，而播放器使用者能进行主动控制则可以避免该问题。

UI 的基本功能包括实例化、用户操作触发后续流程涉及的各模块接下来要做什么，还有状态信息响应展现，也就是根据用户交互行为和播放器工作状态作出反馈与信息传递。

另外，UI 也需要对相应的状态变化作出响应，例如用户控制当前播放器从正在播放切换到暂停，那么 UI 层面则需要针对用户操作进行相应的变化。还有快进、拖拽进度条等等。

3.5 控制层

最后的控制层至关重要，首先控制层隔离校验对外暴露的参数及方法。播放器可实现或具备的特性有很多，不可能全部暴露给用户。在播放视频时，下载与解码的数据实际上存在一个前后呼应的关系，如果我们不考虑用户行为与需求，在网页上呈现播放器的所有特性。而用户也不对其进行科学性选择与判断，而是随意调用 API，势必会带来矛盾、冲突与混乱。因此我们需要隔离配置信息、校验对外暴露的控制参数及方法，以避免可能存在的冲突。

另外根据之前的介绍我们可以看到，不同模块的基本功能大致相同。因此在控制层我们需要统一各模块的生命周期，并完成用于调度各模块工作的基础类的实现。

每个独立的模块什么时刻可以实例装载？什么时刻销毁？该模块是否支持热插拔？各模块生命周期状态的管控与事件消息的监听与调度…… 这些都由控制层进行管理。

有时我们需要做一些取舍，例如编码器并不是基于 FFmpeg，而是基于我们自己的解码解决方案，那么就可以尝试在播放器实例化时候，更换对应模块当中相对应的部分依赖为自己的解码方案；如果我们需要调整播放器 UI 层界面样式，那么就可能需要定制自己的 UI 模块……

在这个播放器实现中，为了规避单线程一些弊端，我们基于 WebWorker API 对重点模块开启子线程。

而 WebWorker 本身的设计存在各种不便：

首先，要求我们必须单独打包一个 JS 文件，基于 new Worker(“*.js”)引入到项目中。

但我们整个播放器作为 SDK 项目的构建来说，通常只产生一个 JS 文件发布出去，才是合理的。

如果同时产生多个 JS 文件，这对我们的调试、开发或后续应用等来说都不方便。

针对这个问题我们结合 Promise 实现了 PromiseWebWorker，PromiseWebWorker 相对于原生 Worker，参数不再必须是传入一个 JS 引用路径，而是可以传入一个函数。

这样一来我们就可以在项目编译时生成一个独立的 JS 文件，在播放器的执行过程中将其中 worker 依赖的那部分函数内容生成一个虚拟的文件依赖地址，作为 WebWorker 执行的资源。

其次，WebWorker 原生能力实现父子线程之间数据传递通讯，只能通过 postMessage 传送数据、通过 onMessage 获取传送过来的数据，这对于频繁的数据交互中想保证上下文关联对应关系是比较麻烦的。PromiseWebWorker 则借助了 Promise 的优势，对以上整个数据交换过程做严格的应答封装处理，从而实现播放器功能的健壮可靠。

http://lab.pyzy.net/qhww 中是播放器 DEMO 展示地址若对此感兴趣可以前往试用研究

要点回顾

调度控制层控制下载器、解码器、渲染器与 UI&交互四大模块，如果要做某功能模块的业务定制化开发、功能增强补充，对对应独立的模块内部进行优化并做出相应的功能扩展或者调整即可。

4. 难点突破

开发过程所遇到的难点总体可以用以上三点来概括：首先基于 WebAssembly 工具链（emscripten.org），借助 EMSC 编译器我们可以直接将一个 C 和 C++编译成 JS 可用。这一过程本身存在诸多不便之处，主要是因为其本身对系统一些底层库的依赖或对于开发环境的要求，导致可移植性并没有那么好，当需要跨机器协作时容易出现诸多问题。现在我们内部的解决方案是自己找一台专用机器来配置做为编译发布使用。

第二点是队列管理与状态控制，只有精确实现队列管理与状态控制，我们才能保证整个程序能合理稳定的执行。

第三点就是项目构建打包，我们要解决前端一些构建打包的习惯以及其在逻辑需求上存在的一些冲突。

5. 未来展望

展望未来，我希望未来浏览器能对 HEVC 有更加出色的支持。本次分享虽然是一个播放器，但我们知道 FFmpeg 的能力不只是解码播放，还可以做更多实用工具的发掘实现。同时我也希望未来媒体类型百花齐放，甚至私有编解码也能够形成 Web 端场景更规范灵活的解决方案。WASM 成熟、标准化完善、各业务领域对应解决能力的细分，也是很值得期待的一件事情；而回到播放器本身，字幕、AI、互动交互等都是能进一步提升音视频播放服务的可玩性与用户体验方向值得研究的方向。

微博客户端播放器的演进之路

近年来新浪微博围绕着快速上线、快速启播、成本节约、稳定性等问题对客户端播放器进行了一系列优化，以求为用户提供更好的服务体验。本文来自新浪微博高级多媒体工程师刘文在 LiveVideoStackCon 2018 音视频技术大会中的演讲，并由 LiveVideoStack 整理而成。

刘文　LiveVideoStackCon 2018 北京　2019-02-20

大家好，我是来自新浪微博的刘文。今天给大家分享的题目是微博客户端短视频播放器的演进之路。在这里主要给大家介绍一下为了更好的服务业务，我们是如何对播放器的体验进行改善、优化的。

主要内容包括以下三部分：
1、微博短视频业务介绍；
2、基于漏斗模型的体验优化；
3、画质提升优化；

一、 微博短视频业务介绍

如下图所示，短视频在这几年的大爆发我们都是有目共睹的。从 2015 年底，就 Facebook 和 Snapchat 来说，它们的日均播放量都已经达到了 65 亿以上。从那时开始，国内也有很多公司开始跟进，开始布局自己的短视频业务，微博也是从那个时候开始加入了短视频业务。

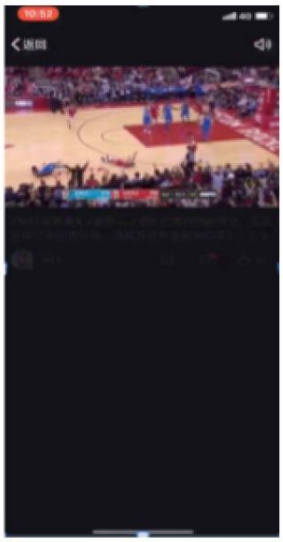
微博短视频业务介绍

国内短视频现状

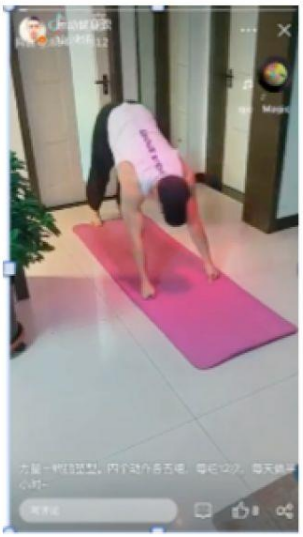
数据来源：中商产业研究院整理

如上图所示，从 2016 年至今，中国短视频的市场规模增长速度是非常之快的。那么经过这几年的发展，微博短视频业务的形态是什么样的呢？

微博短视频业务介绍



微博视频



微博故事

从目前来说，微博的短视频业务主要包括两种形式：微博视频和微博故事。微博视频存在于信息流和推荐流，当用户滑动到视频内容时，触发自动播放，主动为用户播放视频的内容。微博故事主要是竖版视频，有独立入口，主要用来分享用户最近 24 小时内发生的故事。

那么，针对上述两个业务场景，我们的播放器技术做了哪些事情呢？

二、 基于漏斗模型的体验优化

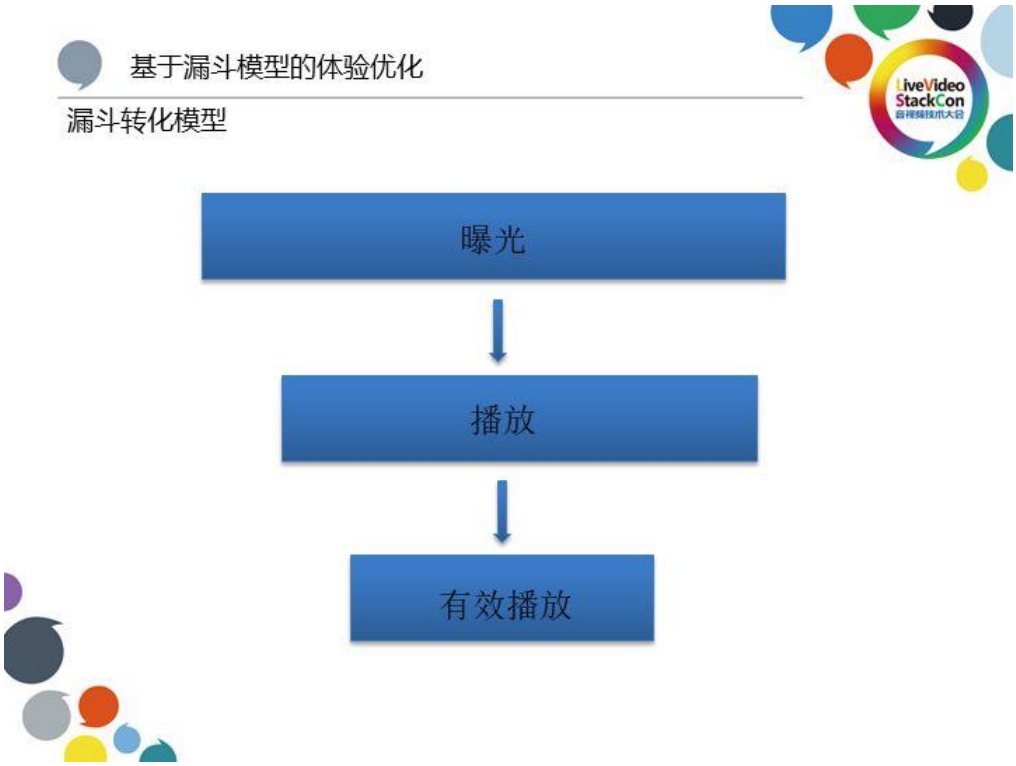
1、业务指标

为了服务好业务，我们首先要思考业务的核心指标是什么？

业务的核心指标是有效播放，有效播放指的是大于 3 秒的视频播放，这是播放器优化中的一个重要指标。

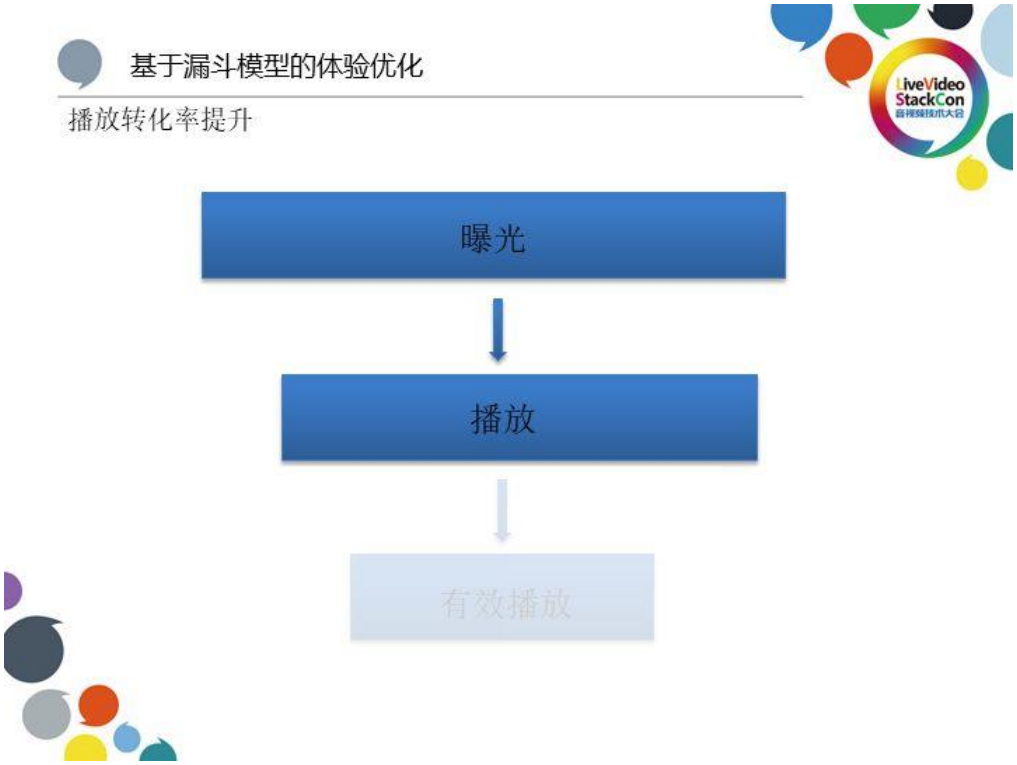
2、漏斗转化模型

业务指标有效播放存在这样一个漏斗转化模型，包括从曝光到播放再到有效播放。

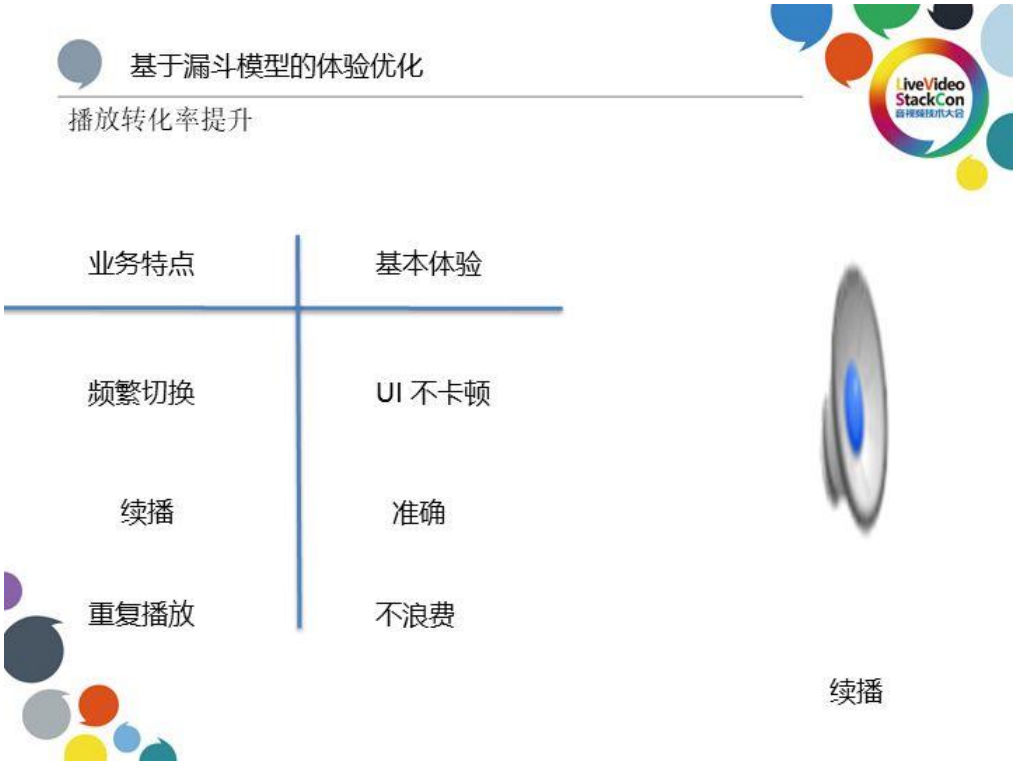


有效播放首先来自于所有的播放，而播放又来自于对视频内容的曝光，那么在有效播放的指标转换过程中，从播放器的角度看我们可以在哪些环节来开展一些优化进而实现指标的提升呢？我们一层层来看。

3、播放转化率提升



首先，我们来考虑漏斗模型的第一层，从曝光到播放阶段怎么提升播放转化率。我们上线了自动播放业务。自动播放指的是在信息流里碰到视频就自动播放，也就是只要曝光就触发播放。从理论上来说，如果我们上线了这个功能，那么曝光到播放的转化率就是百分之百。要上线这个功能，我们必须先要解决一些基本的体验问题。



自动播放在业务上面临哪些挑战呢?

第一，自动播放涉及到多个视频间的频繁切换，这样一来，它对视频基本体验的要求是 UI 不卡顿。第二，对于已经播放过的视频，当再次播放时，我们希望它能延续上一次播放结束的地方继续播放（续播），对于续播我们要求它的播放是准确的，这样才能够保证良好的体验。第三，在信息流、视频流滑动的过程中，对于单个视频自动播放有可能会引发重复播放，对于重复播放我们要求不会造成用户数据的浪费。在实际上线这个功能时，上述问题我们都需要一一解决。

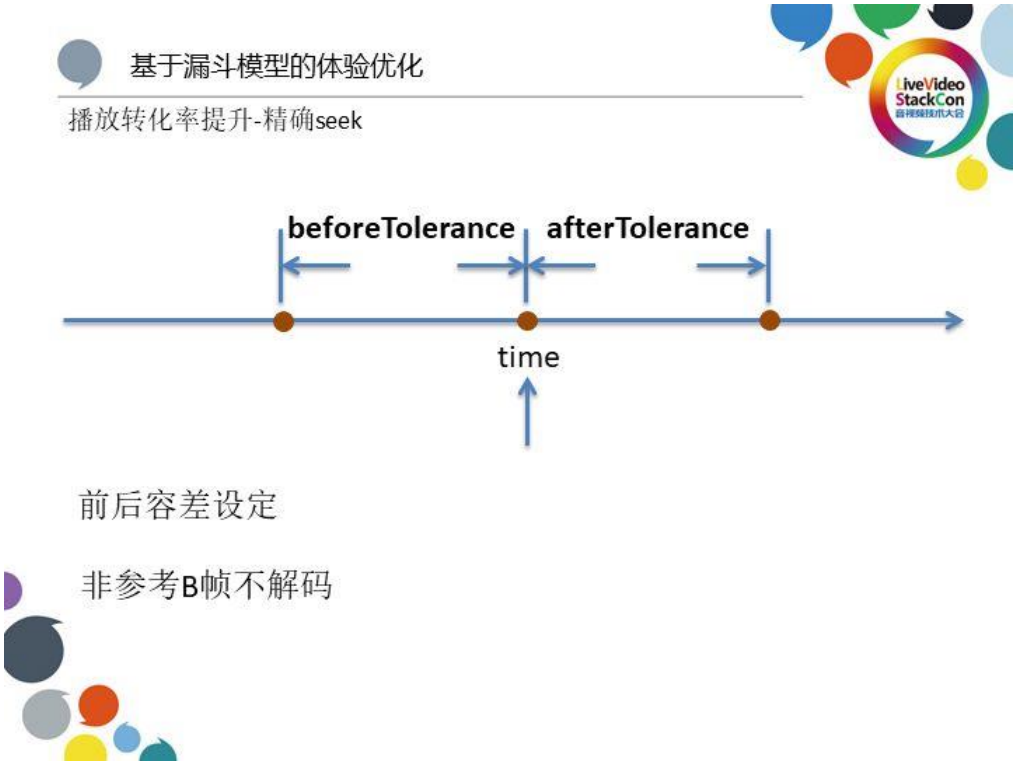
基于漏斗模型的体验优化

播放转化率提升

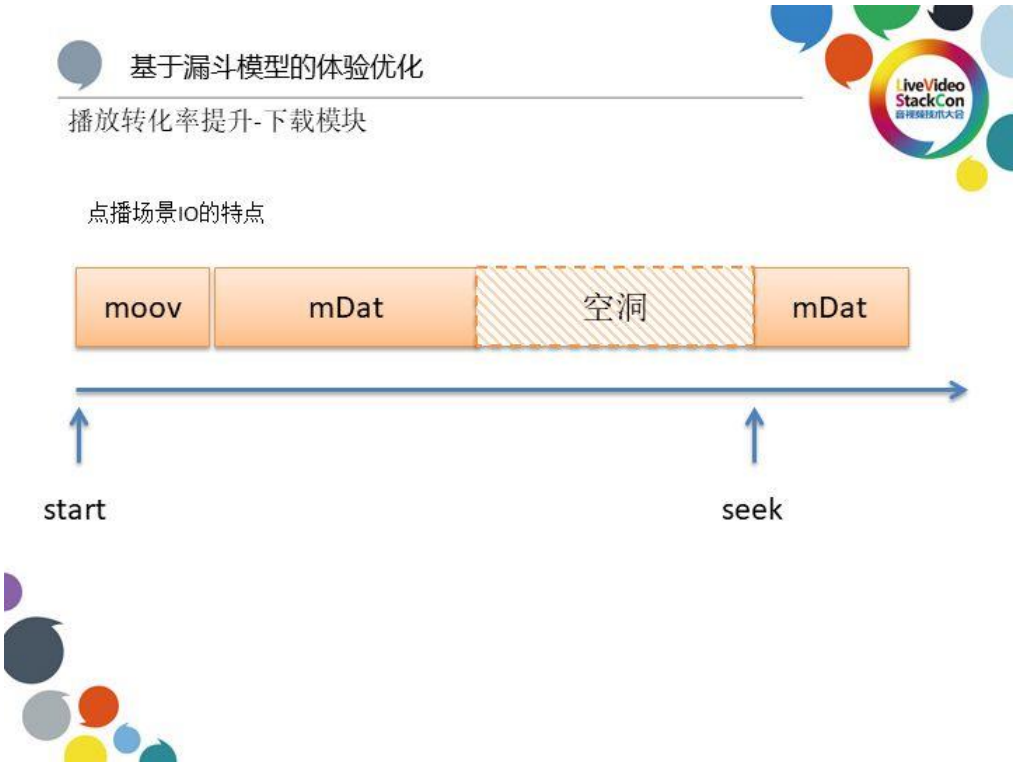
业务特点	基本体验	解决办法
频繁切换	UI 不卡顿	播放器API异步化
续播	准确	精确Seek
重复播放	不浪费	具有缓存能力的下载模块

为了解决这些问题，我们的解决办法是什么呢?

第一，为了解决 UI 卡顿的问题，我们将所有播放器 API 异步化。第二，为了保证续播的准确性，我们支持了视频的精确 seek。其中为了提升精确 seek 效率，参考了 iOS 播放器的接口设计，支持 seek 前后容差的设定，并且对非参考 B 帧不进行解码，尽可能的减少精确 seek 中的解码消耗。第三，为了不浪费用户的流量，对于重复播放，我们设计了具有缓存能力的下载模块。

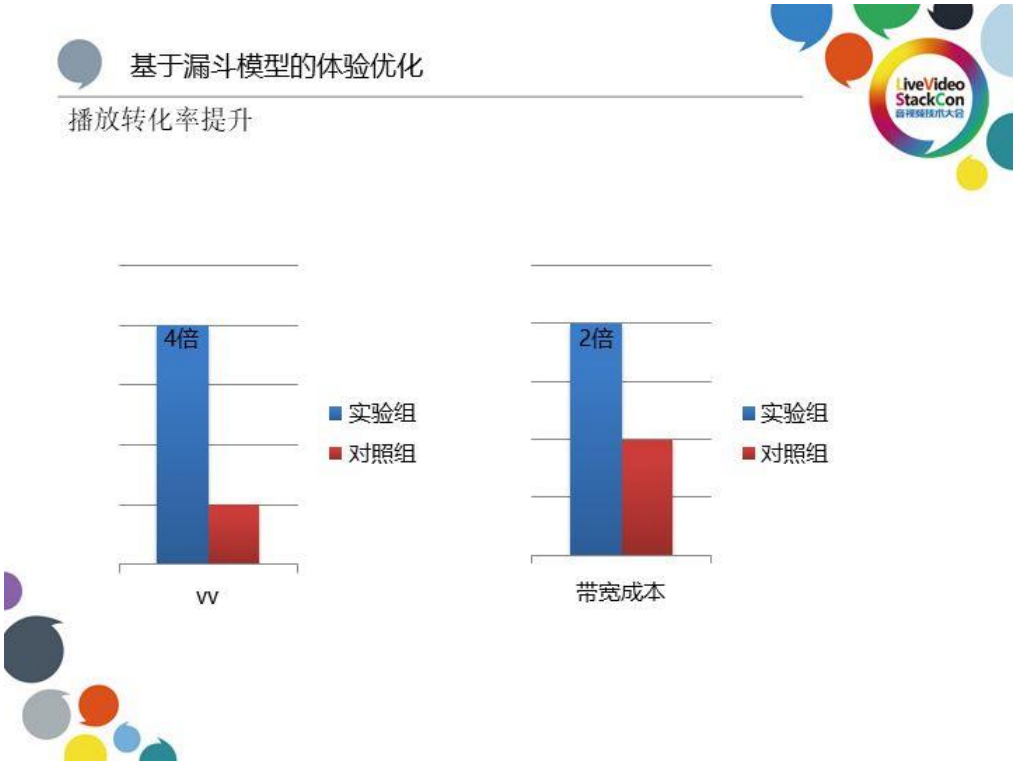


下载模块的工作原理是代理播放器的 IO。此外，我们需要考虑点播场景下的用户行为特点，用户的 seek 拖拽会造成缓存文件出现空洞，下载模块必须支持分片缓存管理， 如下图所示。



最终的下载模块建立在分片缓存文件管理的基础上，包括接口层、IO 管理、缓存文件系统。通过播放器代理接口接入到播放器，解决了自动播放过程重复下载浪费的问题。

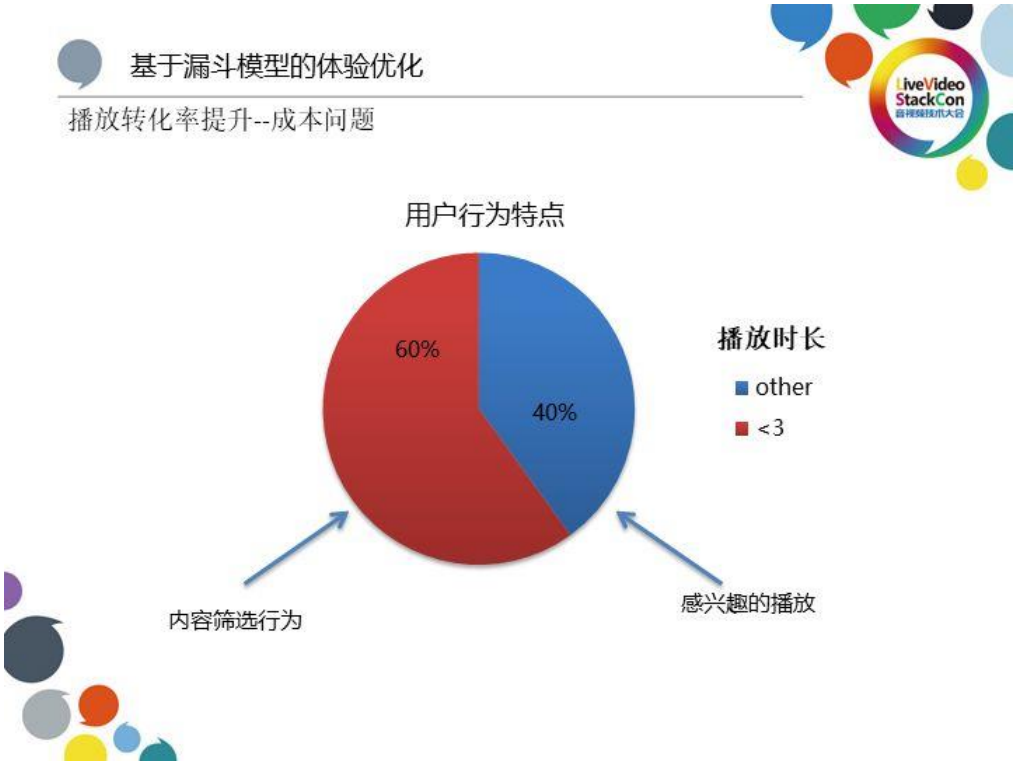
2015 年，我们实现这个功能的灰度上线，我们对实验效果进行了对比统计，播放量增加了四倍，但带宽成本也增加了两倍。



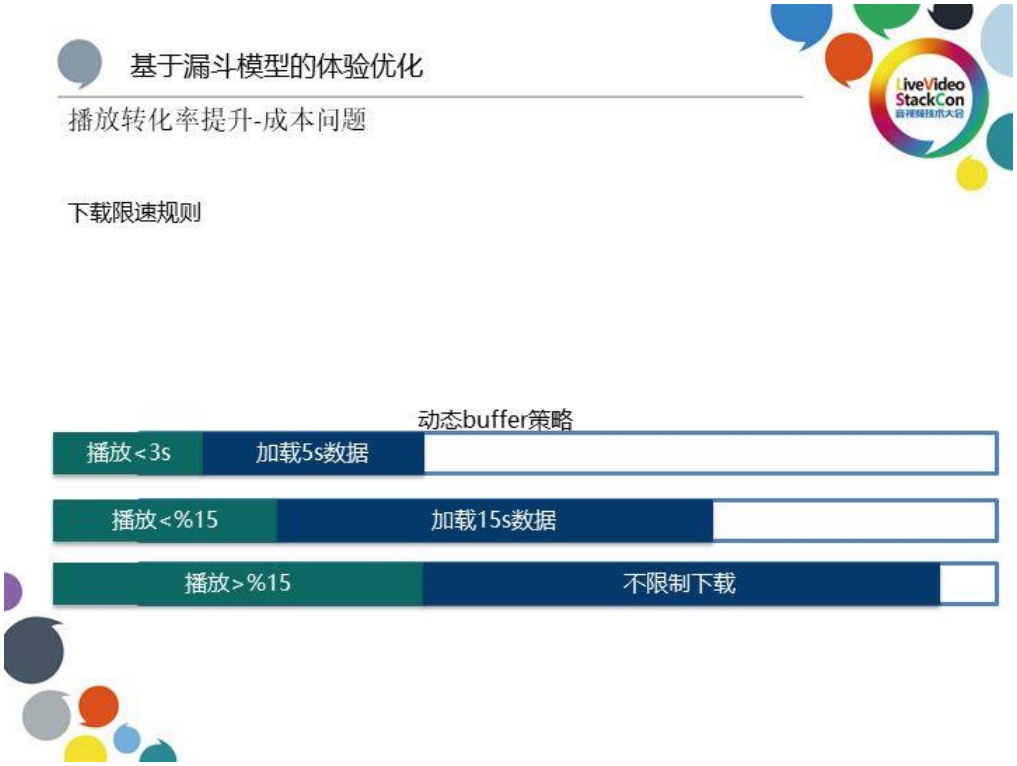
由于带宽成本太高，我们无法实现全量上线这个功能。从播放器的角度来说，我们应该尽可能的避免流量的浪费。在自动播放场景下，流量浪费的形成原因和发生的规模是什么样子的呢？

由于下载没有限速。在自动播放的过程中，数据加载过快，已经的播放进度远远小于已经下载的进度，如果用户没看完视频很早退出，就会存在很大的下载浪费。如下图所示。

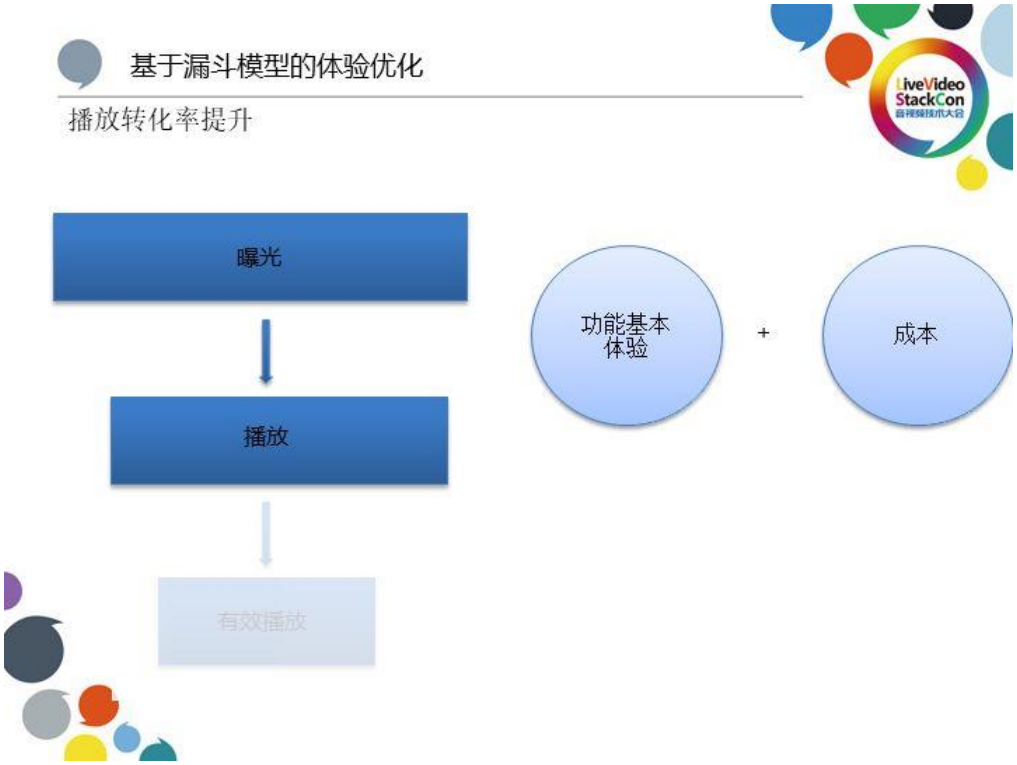
在用户观看的过程中，他们会筛选自己喜欢的内容播放。通过日志统计发现，有 60%的视频播放时长是小于 3 秒的，也就存在大量的播放早退现象。所以，在自动播放场景下，视频播放存在大量的下载浪费。



为了解决这个问题，我们必须进行一定的下载限速。下载限速的规则如下图所示：



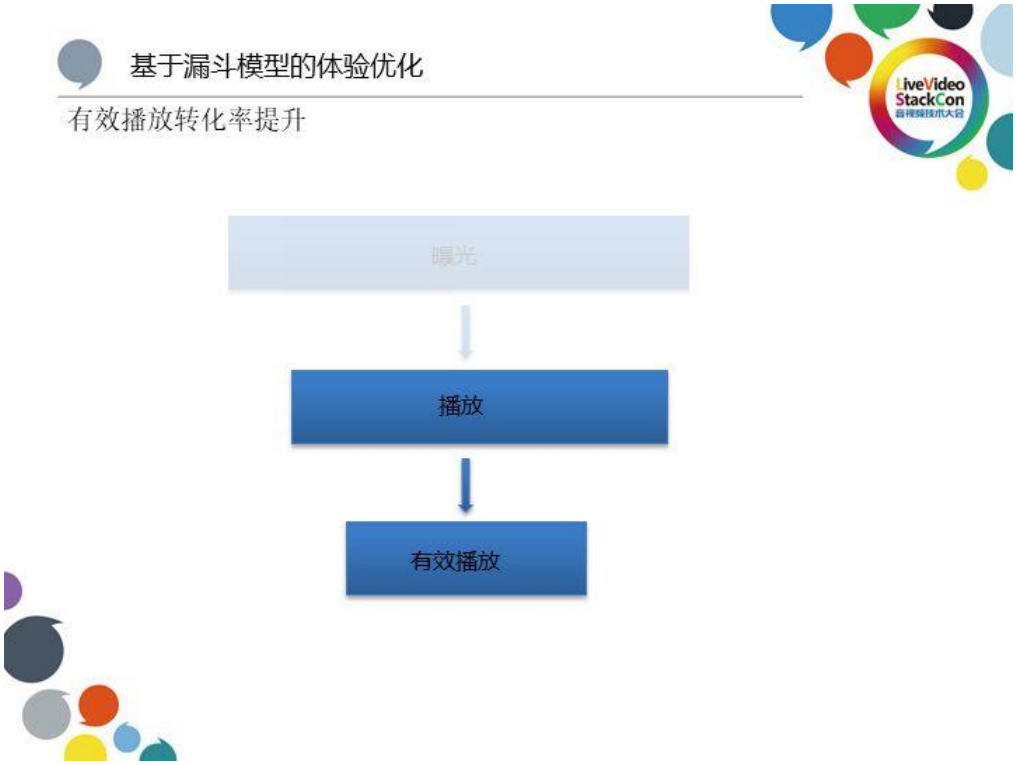
当播放器播放时长小于 3 秒时，只加载 5s 的数据；当播放时长小于视频总时长的 15%时，加载 15s 的数据；当播放时长大于视频总时长的 15%时，不限制下载。



在 2016 年上半年，我们通过解决自动播放功能带来的基本体验问题以及成本过高的问题，实现了自动播放功能的全量上线，促成了漏斗模型中从曝光到播放的转化。

4、有效播放转化率提升

再来看漏洞模型的第二层，我们需要努力实现播放到有效播放的转化。



首先，我们要回答播放最终都可能会转变成什么？然后看哪些部分是可以减少，并向有效播放转化的。播放最终会转变成三种状态：异常、取消播放和成功播放。

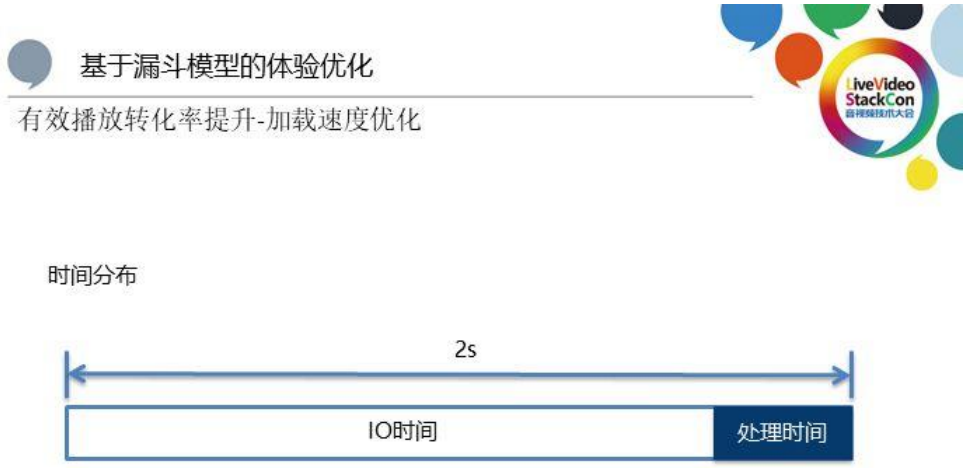
异常可以细分为播放错误和崩溃两种状态，一般是由网络出现问题或者播放器出现 bug 导致的，这是比较严重的体验中断行为是我们必须要处理的。取消播放指的是用户等了一段时间后，视频没播出来，用户取消了观看。可以细分为等待播放时间小于 1 秒和等待时间大于 1 秒两种状态。

等待时间小于 1s 的取消，我们认为是快速滑动中自动播放产生的无效播放，这部分没办法去缩减。等待大于 1s 的，是用户想看但是由于加载慢最后选择了放弃。这部分我可以通过秒开优化去转化。成功播放包括小于 3s 的播放和大于 3s 的播放。小于 3s 的播放，我们认为是对播放的内容不敢兴趣。这部分播放器优化切入的点很有限 。

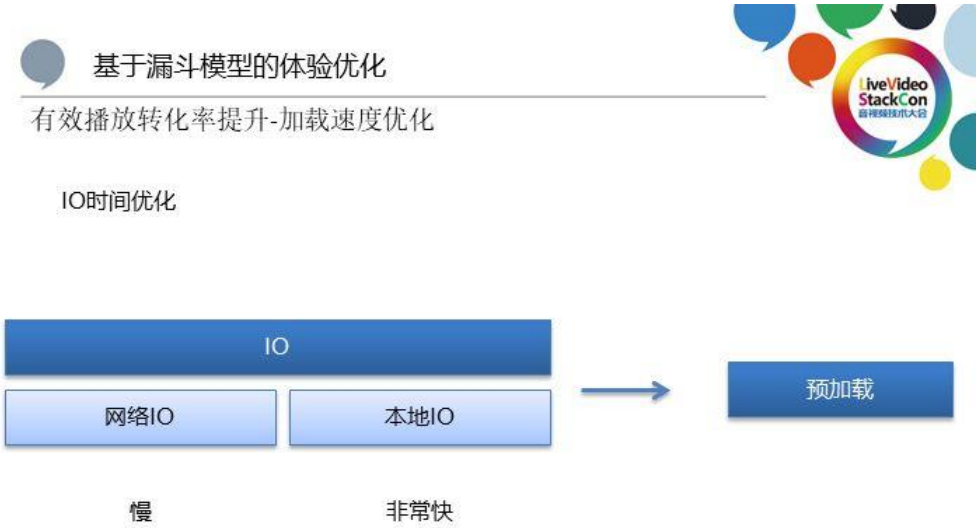
通过对播放转化的分析，我们确认了连续性和加载速度两个体验维度的优化。其中观看连续性指的是错误率与 crash 率的处理。尽量促成播放在 1S 内成功加载。

一）加载速度优化

在加载速度上我们使用 PSR1（Play Success Rate 1 second）也就是 1 秒内成功开播的百分比作为我们的指标。为了找到加载过程的耗时点，我们进行日志打点分析，并把加载时间分为 IO 时间和处理时间两个部分。经过上线分析，发现首帧加载时间为 2s，大部分时间是 IO 消耗。



IO 主要包括网络 IO 和本地 IO，而网络 IO 与本地 IO 对比是很慢的。因此，减少启播过程中网络 IO，是提高首帧时间的有效方法。考虑到用户观看视频的内容筛选行为，很多播放是小于 3s 的，我们对即将播放的视频做了 3s 预先加载



为了减少预加载对播放的影响，我们需要将预加载和播放中的下载统一在一起进行管理，为此对播放器下载模块进行了重构。

对于启播过程中的处理时间，虽然占比小，但是还是有优化的空间。处理过程主要包括解析、解码和控制三个步骤。为了优化处理的速度，对于解析过程，我们采取忽略帧率估计、避免重复解码的策略；对于解码过程，我们采取了解码器异步初始化；对于控制过程，我们采用首帧解码后即显示和 buffer 延迟的策略。

基于漏斗模型的体验优化

有效播放转化率提升-加载速度优化

处理时间

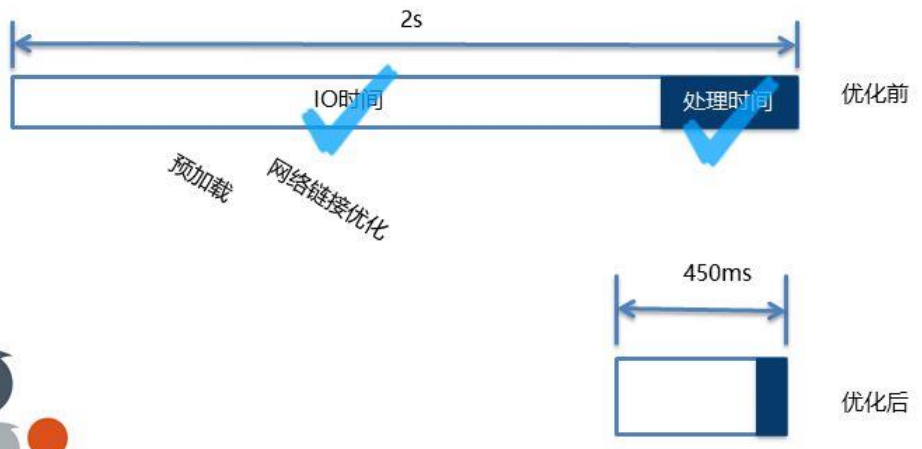
步骤	策略
解析	忽略帧率估计、避免重复解码
解码	解码器异步初始化
控制	首帧解码即显示+延迟buffer

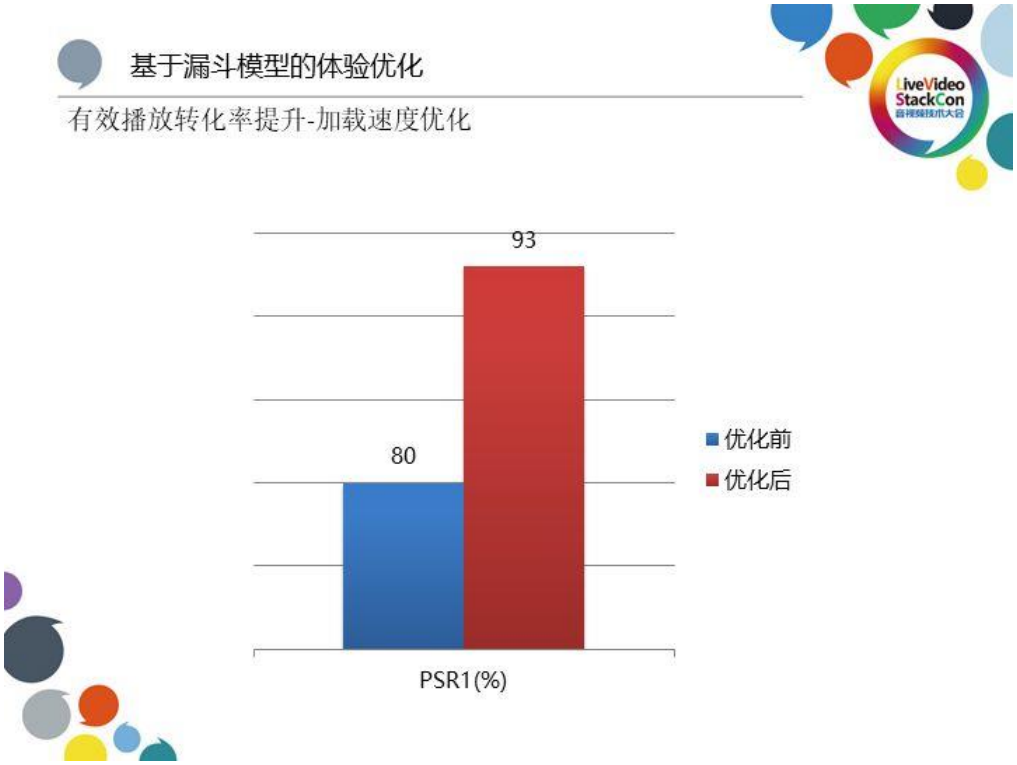
通过对 IO 时间和处理时间的优化，加载时间优化到了 450ms，PSR1 值从 80%提升到了 93%。

基于漏斗模型的体验优化

有效播放转化率提升-加载速度优化

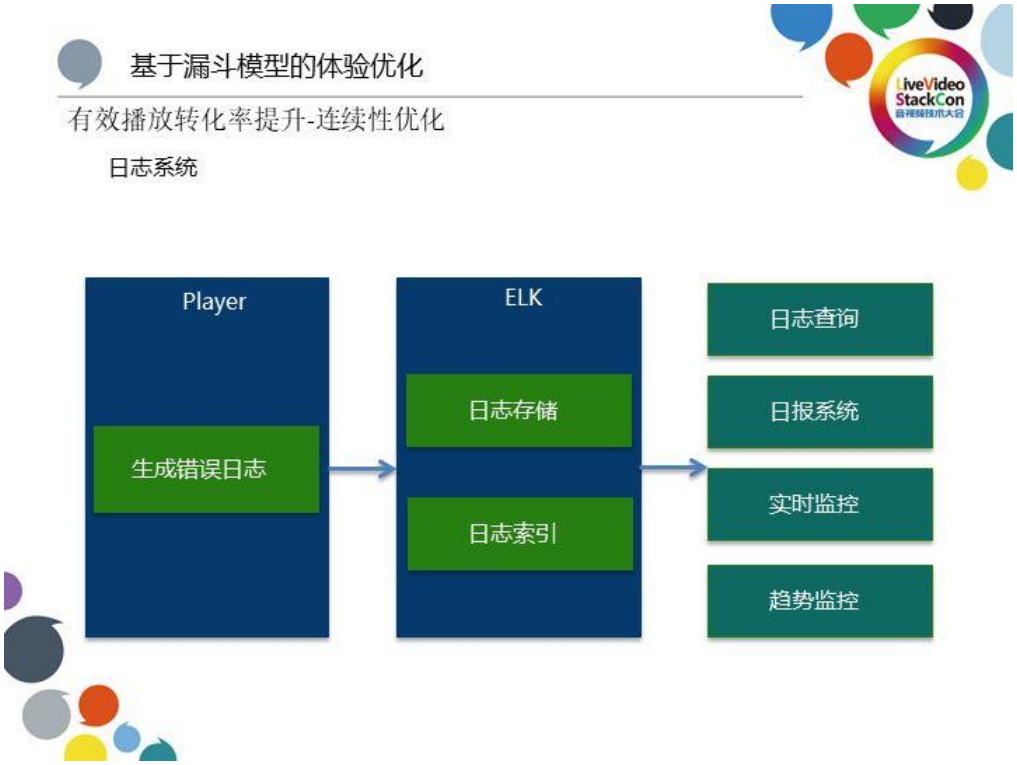
时间分布



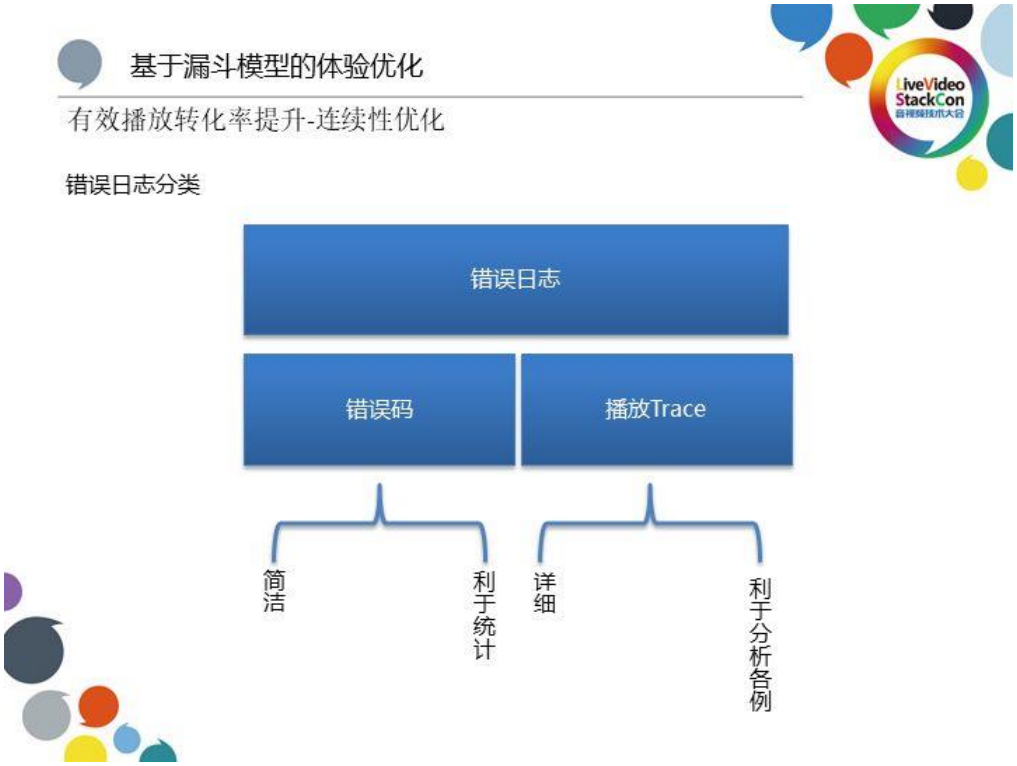


二）连续性优化

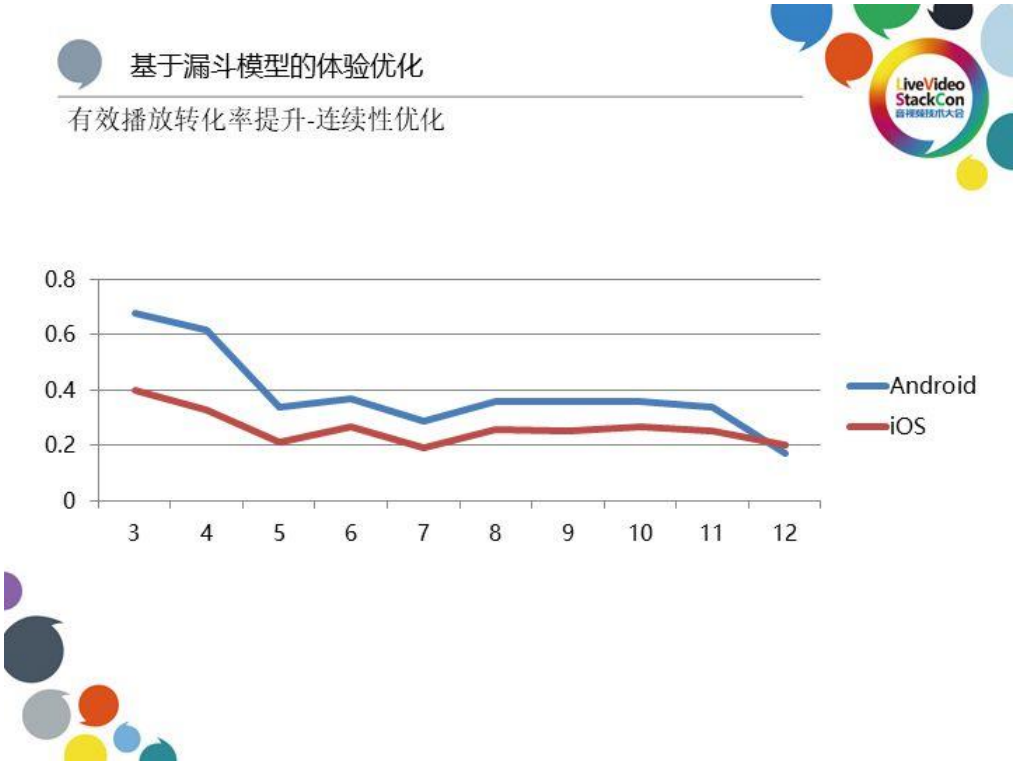
连续性优化主要包括两个方面：错误率和 Crash 率。这两类问题没有什么统一的处理标准和方法，只能一个个解决。为了提升效率，必须依靠一个强大的日志系统来进行分析，如下图所示。



第一，降低错误率；

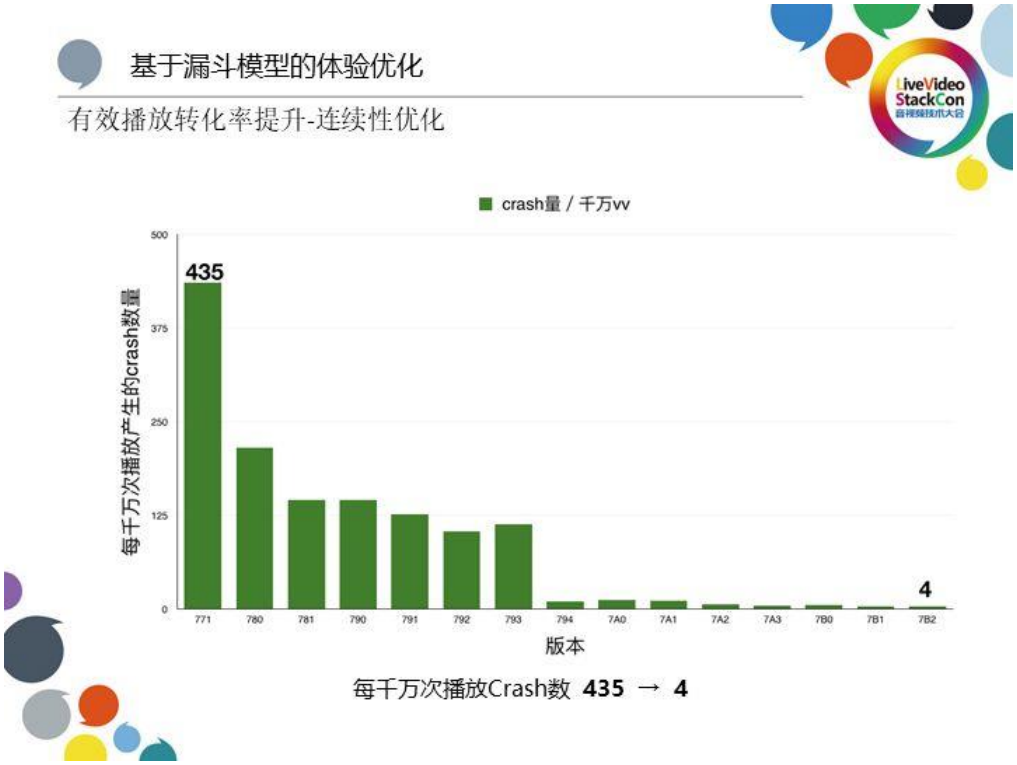


对于错误而言，错误信息的梳理很重要，包含错误码和播放 trace。其中，错误码比较简洁、利于统计。播放 trace 记录了单条播放中，核心步骤的执行情况，对于单个错误的分析非常有效。通过一个个的解决，我们的错误率从开始的平均 0.5%降低到了 0.2%以下。



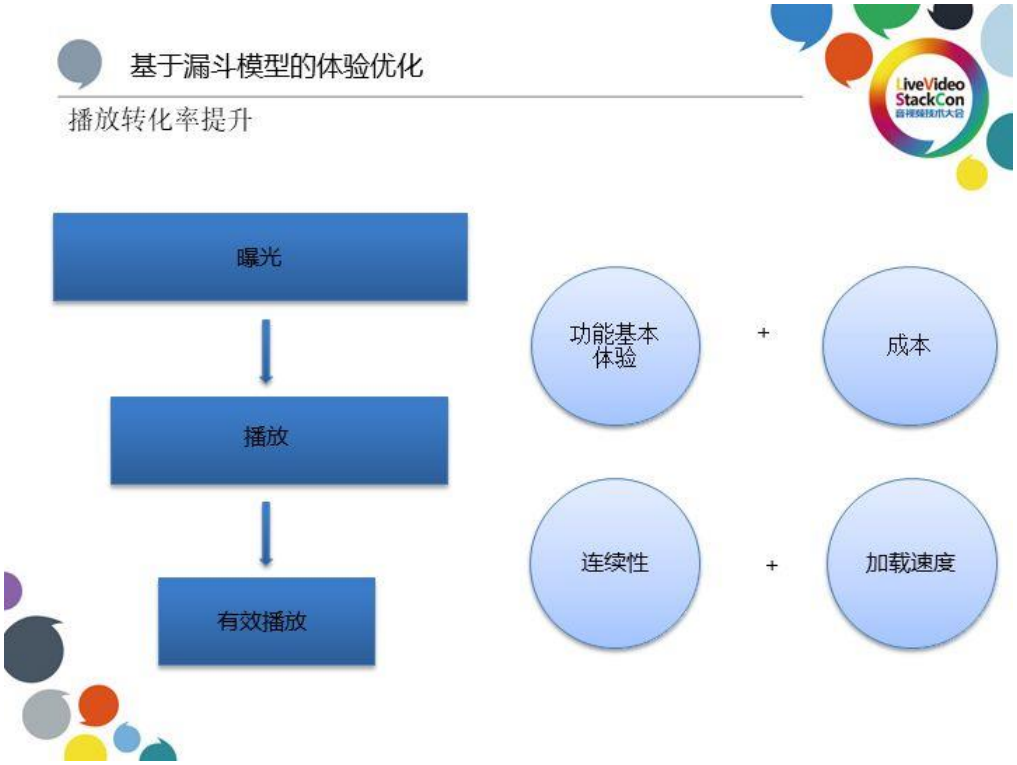
第二，降低 Crash 量；

在 Crash 上，我们花了很多的精力，难点在于由内存问题导致的诡异堆栈。为了快速定位问题所在，一方面我们主要是通过内存分析工具来帮助发现内存的一些隐患；另一方面，对于一些不容易出现的问题，我们通过在压力环境下模拟堆栈发生的逻辑来提高它的复线率。最终，我们的 Crash 率得到了控制，每千万次播放的 Crash 量从 450 多次降到了 4 次。



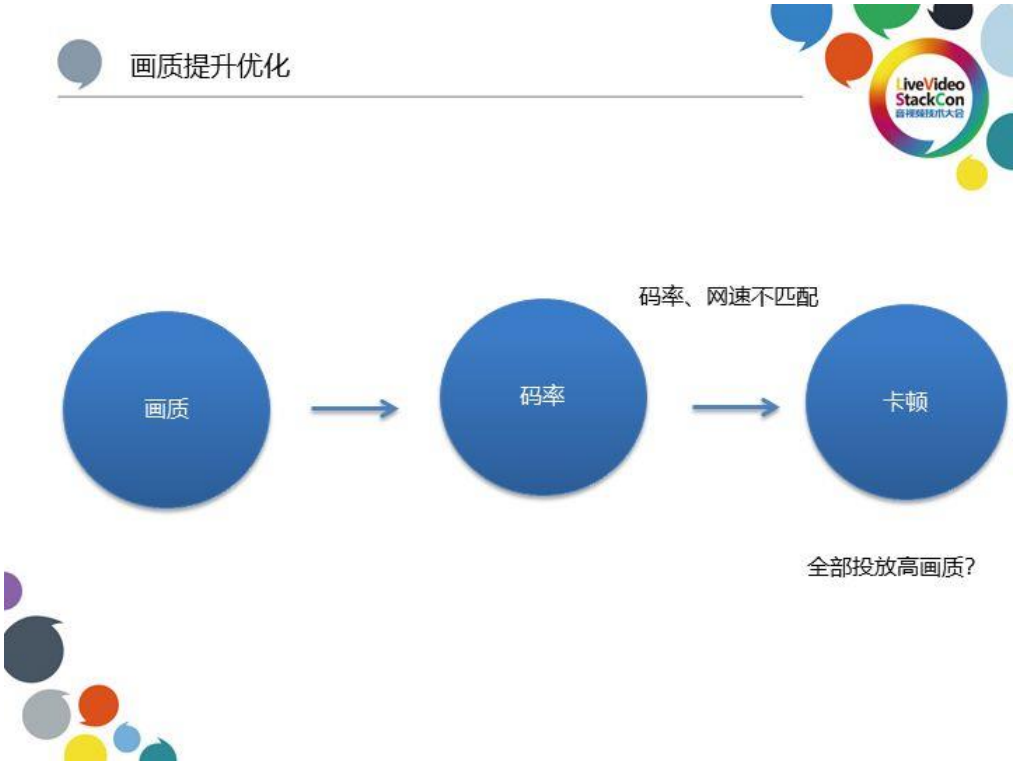
5、小结

总结一下有效播放转化率提升的过程，我们主要是围绕漏斗转化模型来开展优化工作，包括从曝光到播放和从播放到有效播放两个阶段。从曝光到播放过程主要优化的是基本体验和成本过高的问题，从播放到有效播放过程主要优化的是观看连续性和加载速度。



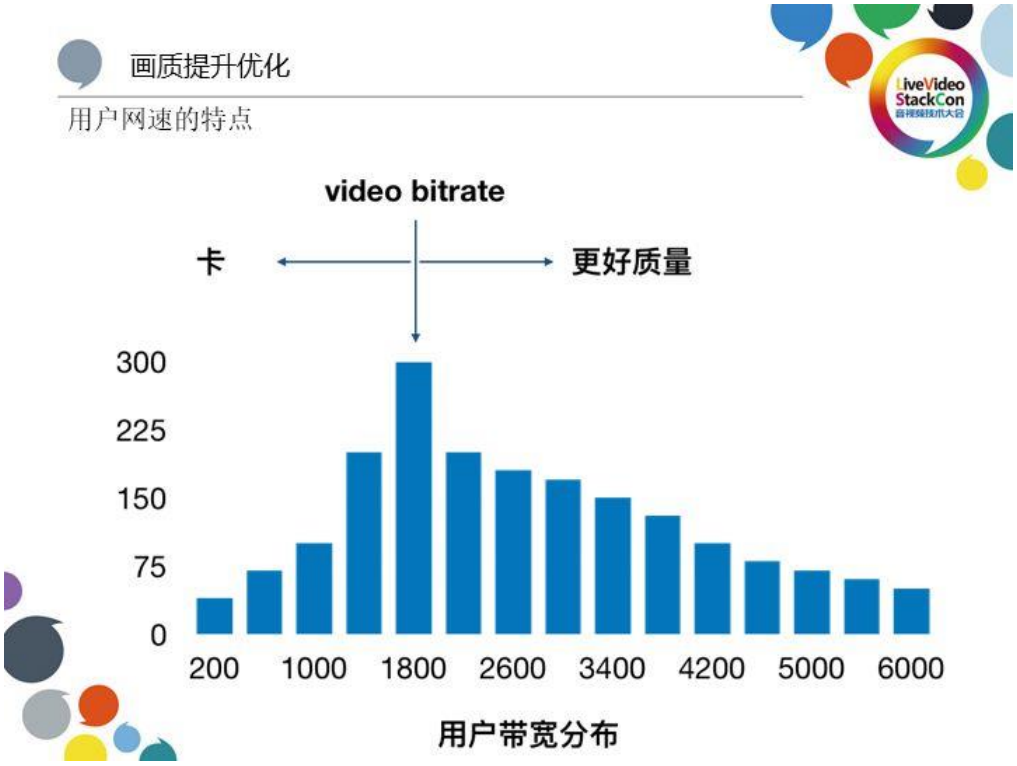
回顾我们之前的思路，为了更好的服务业务，我们主要从用户体验的连续性和加载速度两个维度开展优化。这两年随着带宽条件的改善，客户对画面质量的要求也越来越高，视频画面质量提升将是趋势。 画面质量体验的好坏对业务的影响也会越来越大，所以画面质量优化是我们目前一个重点方向。下面给大家介绍下我们质量提升方案的选择与初步尝试。

三、 画质提升优化

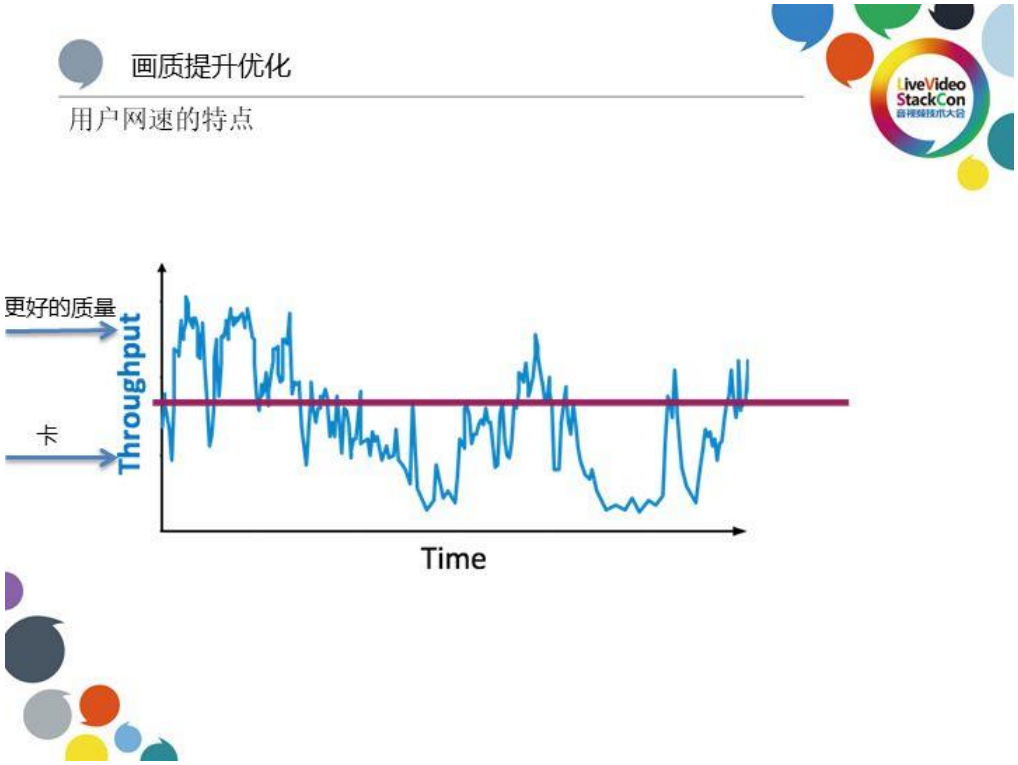


我们都知道提升画质会导致码率上升，要改善画质我们是不是把所有视频内容的码率都提升就可以了呢？答案肯定是不行的。如果我们集中提高码率会导致卡顿上升，这主要是因为码率和网速之间的不匹配造成的。

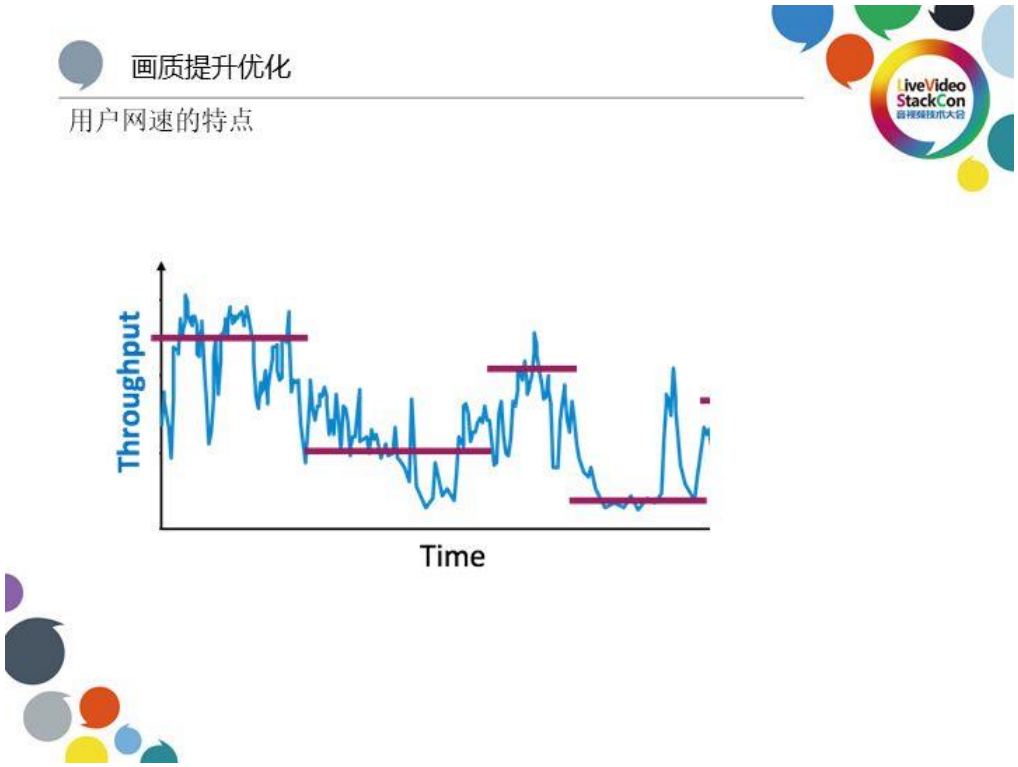
从所有用户的角度看，假设用户的带宽分布如下图所示，一种清晰度不可能适应所有用户的带宽 。



从单个用户的角度看，用户观看视频的过程中，网络是波动的，提供一种清晰度是没法满足单个用户的单次播放的。



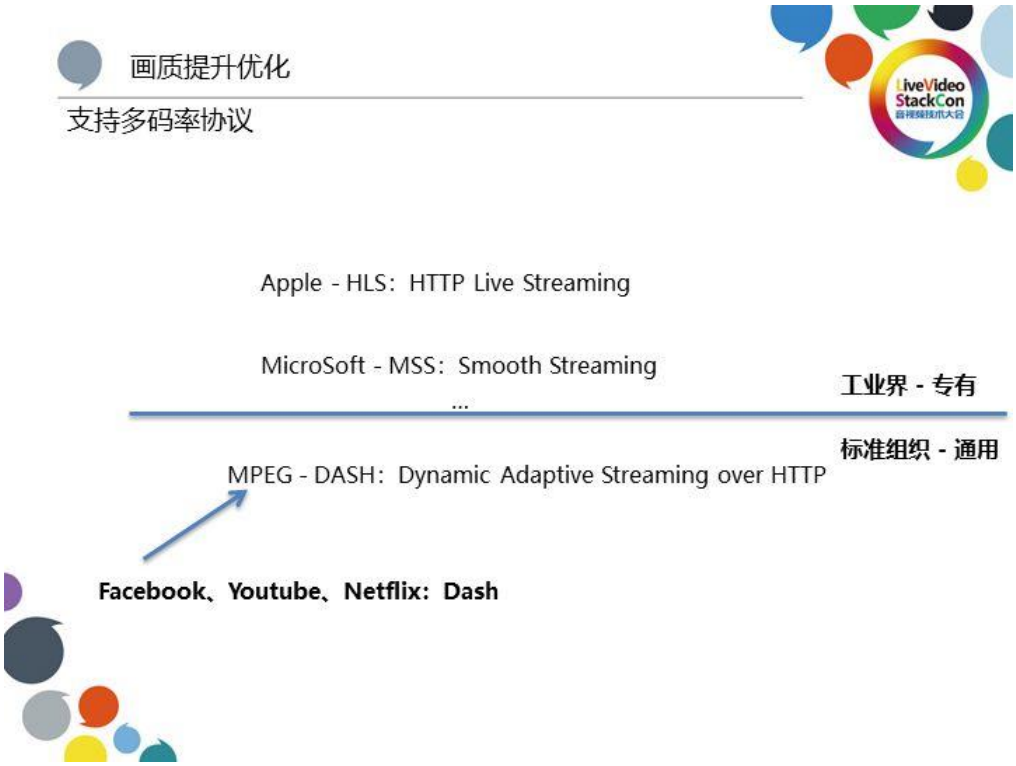
因此，如果我们有多种分辨率，而且我们可以根据用户的带宽给它匹配当时他能播放的最高码率，那么这样的话，我们就可以既保证基本的流畅性体验，又可以尽可能的提升画面质量。



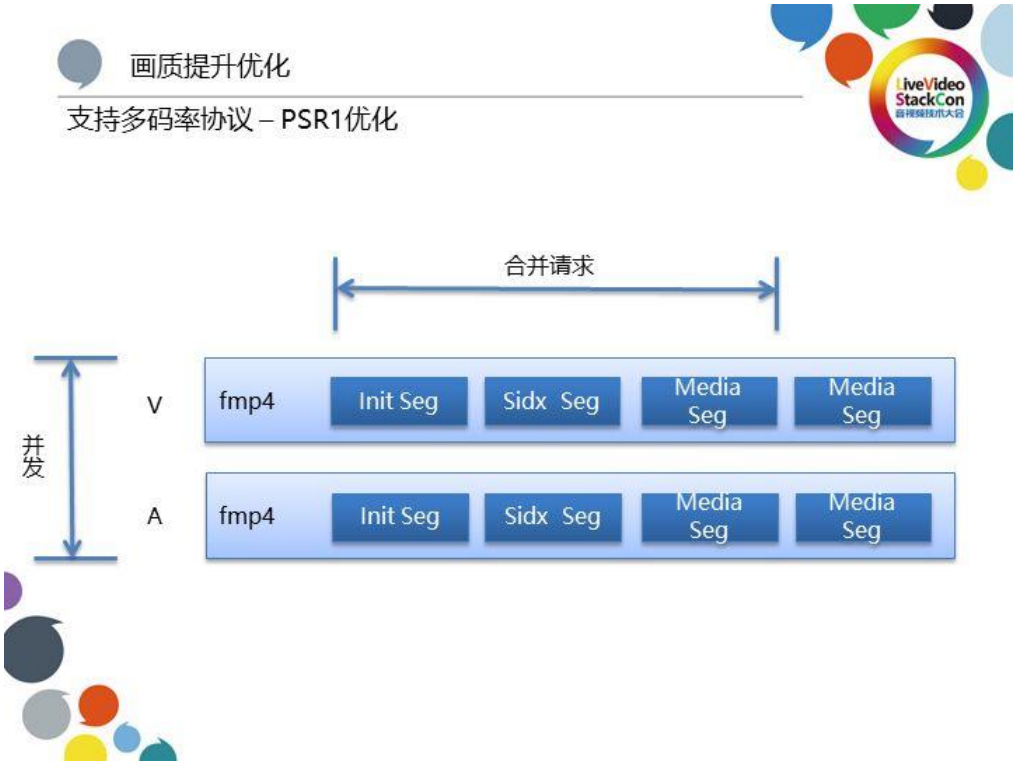
所以，我们的画质提升方案是基于多码率的自适应码率方案，这就需要播放器有自适应码率切换的能力。在这个过程中，我们需要解决两个问题：支持多码率协议播放和码率自适应切换能力。

一）支持多码率协议

现在工业界的多码率协议主要有苹果的 HLS、微软的 MSS 和标准化组织定义的 DASH 等，其中 DASH 相对来说更加通用，是国外很多大厂目前也正在使用这个协议， 所以我们采用 DASH 作为我们支持的对象。



在支持这个协议播放的过程中，我们对加载速度进行了优化。因为 DASH 的起播还是相对比较复杂的，音视频是分离的，单个 fmp4 请求包含了多个 segment 请求。一方面，我们把多个 segment 请求进行合并；另外一方面，我们对音视频的 fmp4 请求进行了并发处理。通过这样的优化，目前为止 Dash 在线上的播放 PSR1 和我们普通视频的 PSR1 的值是相近的。



二）码率自适应切换能力

对于自适应码率切换，现在已经上线了手动切换，而自动切换还处于实验室阶段。这里主要说一下手动切换。

手动切换主要解决切换这个动作平滑性的问题。思想就是在切换实时性和平滑性间做一个折中，损失一定的实时性来保证切换动作的平滑性。 具体而言，其实就是利用正在播放的清晰度的缓存时间，准备下一个清晰度的播放。

四、 总结

总的来说，播放器服务业务的过程，就是对播放器体验优化的过程。体验优化的方向大家的做法都是类似的，包括加载速度、连续性和画面质量。不同的就是我们如何围绕着业务的需求去思考我们什么时间做什么优化。

QoE：一切为了用户体验

于冰：用户导向的音视频体验优化

在 LiveVideoStackCon2018 音视频技术大会上，快手音视频技术负责人于冰分享了题为《用户导向的音视频体验优化》的主题演讲，从用户的角度来去谈音视频的体验优化。

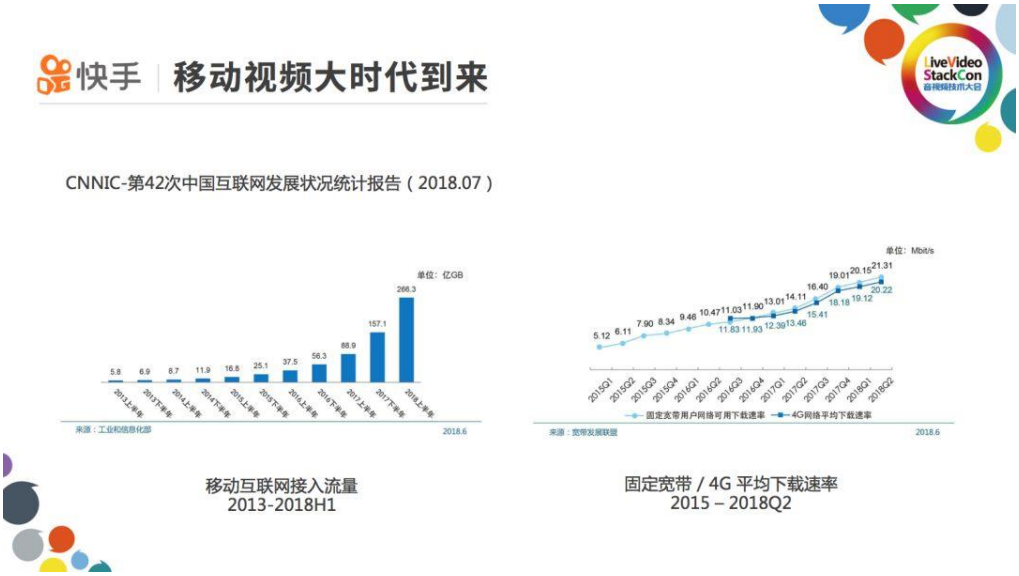
于冰　LiveVideoStackCon 2018 北京　2019-03-06

大家好，这次分享主要从用户的角度来去谈音视频的体验优化，做技术的公司通常都喜欢从技术角度出发去讲问题，如编解码算法、AI 技术等。但是我们不妨反过来想一下，什么是用户最需要的，对用户而言什么是真正的好的体验，这也是快手的一个基本原则。

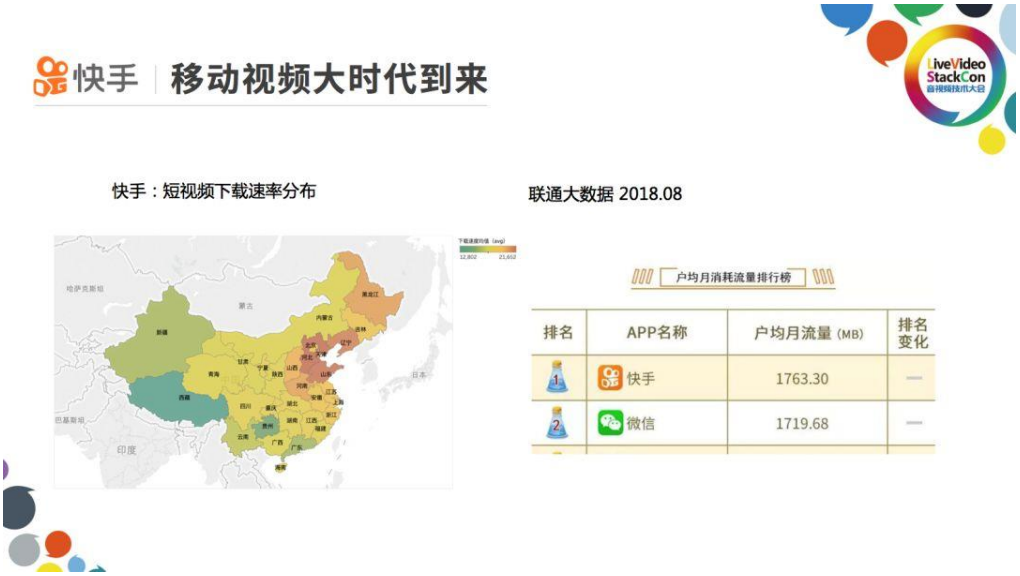
今天的演讲主要分三个部分：

- 一是移动视频大时代的快手；
- 二是用户导向的音视频体验优化；
- 三是移动音视频技术趋势探讨。

一、移动视频大时代的快手



移动视频大时代已经到来，从 CNNIC 最新的互联网发展状况统计报告可以看到，2015 年下半年到现在，移动互联网接入流量已经涨了 5 倍，固定宽带和 4G 平均下载速率已经达到了 21M，21M 是什么样的概念？就是 4K 视频用手机看已经没有太大问题了。现在制约 4G 视频发展的主要因素就是资费问题，而各个运营商也在解决这个问题，推出大王卡、免流套餐等等，相信这会让 4G 的视频消费越来越猛。



这是快手短视频下载速率分布的数据图，和 CNNIC 的数据差不多。其中辽宁、河北、山东这些省份速度是最快的，可以看出这些省份运营商优化的比较好。另外看右侧图，联通大数据 2018 年 8 月份的户均月消耗流量排行榜数据显示，快手排名第一，一位用户一个月要耗掉 1.7GB 的流量。快手数据显示，部分用户一天就要消耗掉 1 个 G 的流量。

在这个移动视频大时代背景下，短视频为什么异军突起，而不是长视频继续独领风骚？这就要分析长视频和短视频之间的巨大区别。

首先，内容摄制方面，长视频通常以头部 PGC 内容为主，由专业设备拍摄、专业人员制作后期，比如由大 IP 改编的作品。而短视频 UGC 内容是移动端拍摄，移动端制作，随时随地上传。

第二，长视频通常是横屏，1080P 或者 4K 的分辨率，适合在大屏幕的电视、电脑上观看。而短视频都是竖屏，540P 或者 720P 的分辨率，更适合手机的分辨率及使用习惯。

第三，长视频以头部内容为主，是集中的热点内容，数量相对较有限但会覆盖到大量的用户，通常人们能看到的、喜欢看的电影差别不会特别大。但是短视频内容是非常长尾的，单个视频播放量会相对比较小，不过会有更多视频被消费者看到。

第四，从消费频次上看，长视频的观看是中低频的，需要用户用固定时间去消费，看一部电影或追剧需要预留大块、持续的时间，一般只能在周末假期集中消费。但是短视频却可以随时随地刷，却非常高频。

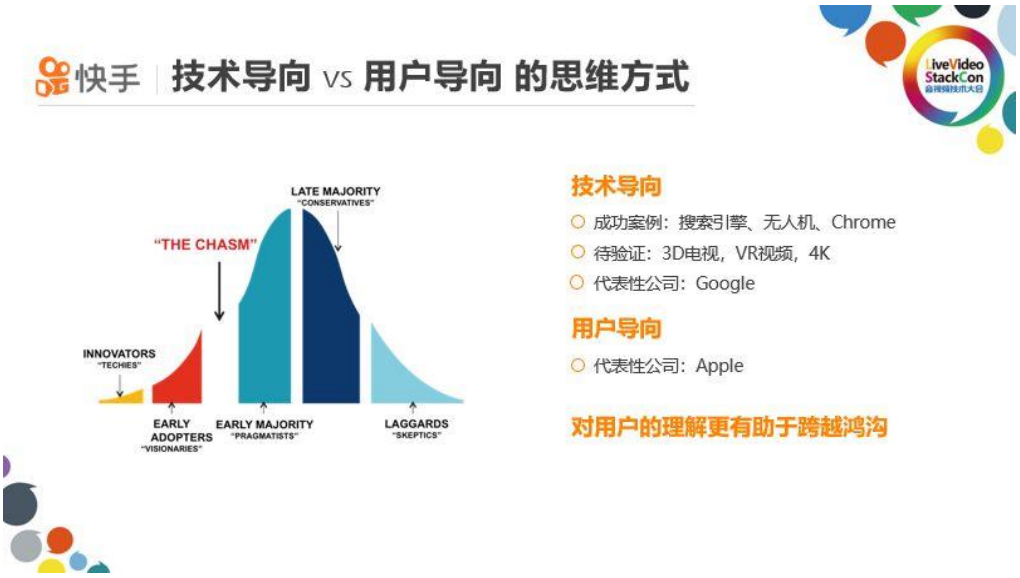
第五，从商业模式上看，长视频一般都采用广告、订阅或付费等相对成熟的商业模式，而短视频则是采用免费观看的模式，并在其中拓展多种商业模式如直播打赏、信息流广告、电商等。

因此，正是由于短视频在体验上的诸多优势，使得更多用户愿意花费更多时间使用快手等短视频 app。

那么快手的总体情况是怎样的呢？快手现在日活跃用户已达 1.6 亿，月活超过 3 亿，原创视频总量为 80 亿条，每日视频播放量 150 亿次，每天上传视频超过 1500 万，一年累计用户点赞数 700 亿次（2018 年 12 月数据）。

快手的使命是“用科技提升每一个人独特的幸福感”。在快手，每个人都能够看到更广阔的世界，也能够被更广阔的世界看到，这就要求平台坚持普惠、去头部化，不然普通用户发布的视频是无法被更多人看到的。

二、用户导向的音视频体验优化



什么是用户导向的思维方式？接下来我们一起了解一下，技术导向与用户导向思维模式的差异。上边这张著名的图来自于《跨越鸿沟（Crossing the Chasm）》这本书，一个技术在处于图中 Early Adopters 阶段与 Early Majority 阶段中会有很大的鸿沟，要想跨越这个鸿沟是很有挑战的，首先需要技术本身非常靠谱，其次要非常理解用户的诉求，能够满足大量用户的刚需。从技术或用户两个角度入手，满足上述要点，都可以取得成功。

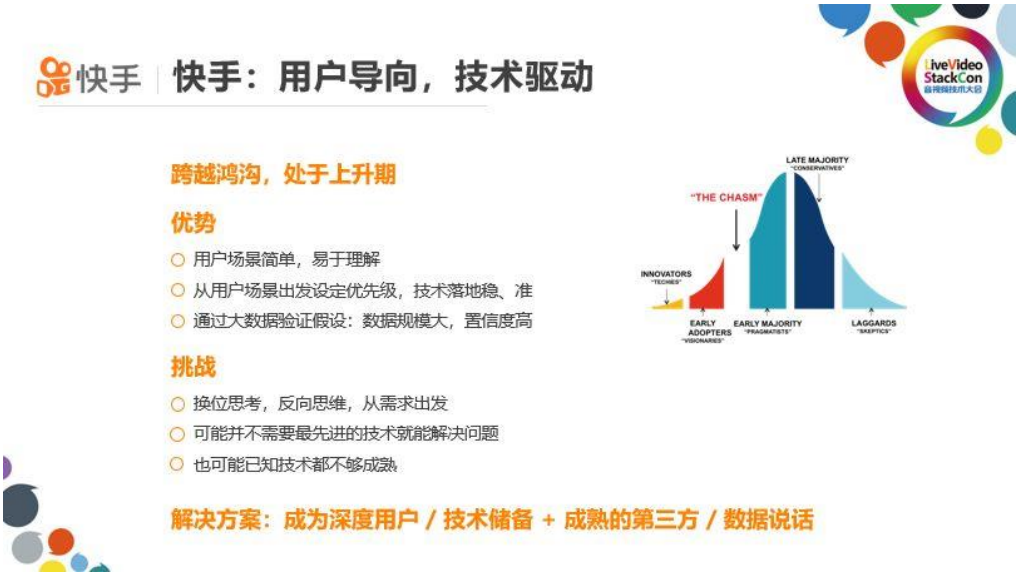
首先看一些技术导向的案例，比如搜索引擎、无人机、谷歌 Chrome 浏览器，都是通过非常强的技术手段实现，并且恰好满足了用户的需求，才会获得成功。代表性公司是 Google，以技术洞见带来的革命性进展推动市场。也有一些待验证的技术，处于 Early Adopters 阶段，比如 3D 电视，需要戴上 3D 眼镜才能看观看，使用率很低；VR 视频，我买了很多 VR 的头显设备，但用过一次后放在抽屉里再没有拿出来；4K 电视也是类似的困境，很多用户并不能看出 4K 和 1080P 的区别。这些技术之所以没有大规模普及，主要是因为没有从用户角度思考问题。

而用户导向代表性的公司是 Apple，其思维方式是从用户需求出发，做出超出用户预期的产品和技术。我个人认为，对用户的理解其实是更有助于跨越鸿沟的。

再举个音视频领域的例子，十多年前我跟优酷合作做过一次大规模的音乐会在线直播，是业界最早采用 H264 FLV 分发技术的案例，支持多机位切换，实时回看。虽然该技术方案与今天所采用的主流方案没什么不同，但由于 web 端大型活动直播的频次低，用户粘性不高，并未引起多大的市场反响。直到七八年以后，随着移动直播体验的普及，相关技术才真正的走入大众市场，变成爆炸性增长的产品。

移动直播最早是从美国兴起的，如 2015 年上线的 Periscope 等 App，这些产品创新地在直播中加入了点赞、评论、打赏等机制，用互动性增强了用户粘性；另外由于移动直播可以随时随地开播，又提升了主播与用户交互的频度。最终通过用户的体验创新来实现了变革。

看完直播，再来看下传统视频点播 App，目前多数影音 App 是类似 YouTube、Netflix 这样界面的传统交互模式，通过分类、搜索、推荐等形式构成了复杂交互体验，对于中低频的影视剧长视频观看是比较适合的。但是在移动端碎片化消费的时代，类似快手这种操作界面简洁、主要通过算法实现个性化推荐的短视频 APP，操作更简便，反而更受用户欢迎。



快手是一家用户导向、技术驱动的公司，这是公司的定位。现在快手属于已跨越鸿沟且处于上升期的阶段，主要优势有三点：

一是用户场景简单，短视频、直播两种形式的内容，通过推荐做个性化分发，十分易于理解。

二是从用户使用场景出发设定优先级，这样技术落地时就会比较稳、比较准。举例而言，假如你先研发了一项技术，然后再去找技术的落地点，就如同拿了一把锤子（技术）到处去敲矿石（用户使用场景），很可能会找不到适合的石头，或者找到的石头里边没有好东西；但假如我们手上有很多优质翡翠的原石，再去找锤子，即使可能找到的不是最好最先进的锤子，甚至是石器时代的锤子，也能够敲开藏着翡翠的原石。

第三个优势是通过数据验证你的假设，快手用户行为样本量大，数据置信度高。算法好与不好，上线 AB 试一下便知道，一切用数据说话。

用户导向也带来了很大挑战，技术人员通常喜欢从技术角度出发想问题，但是现在要换位思考，从用户角度、需求去考虑，反向思维，这是非常难的事，很多技术人员需要经历一段时间的转变才能做到。

另外如刚才我们提及的，一把石器时代的锤子，只要能砸开玉石就是一把好锤子，所以解决问题时并不一定需要最先进的技术。具体而言，解决一个很难的技术问题，可以有不同分数的方案。我们团队很多同学都是成绩优秀的学生，喜欢追求 90 分的方案，投入了大量的时间和资源，却可能因为算法、系统复杂度或其他各种各样的问题，最后迟迟上不了线；而我们很多线上体验一开始可能只有 30 分，采用一个非常简单的方案就能让你达到 60 分，为什么不先去做一个 60 分的方案呢？

我们的解决方案就是让研发人员成为产品的深度用户，自己从用户角度推敲优化。同时做很多的技术储备，当用户需求来了，随时可以拿出来用。如果自己的技术储备不够，又十分着急的话，要考虑采用成熟的第三方技术。技术好不好要用实际数据来说话，看用户是否接受它，而不是盲目上线技术上看起来很牛却曲高和寡的东西。

举个实际例子，快手有个很重要的体验，直播连麦，也就是主播与粉丝实时音视频连线。2016 年，很多直播产品上线了 3 连麦、4 连麦，甚至 8 连麦、16 连麦。快手刚上线直播时，产品团队从用户的角度做了评估，考虑直接上线视频连麦可能会给用户带来社交压力，因为被连麦的用户往往不是主播，不是都喜欢露脸。那么如何最小化用户开通连麦的压力？最易接受的形式就是语音连麦，所以我们最先上线的是技术上简单一些的语音连麦，也有充足时间把体验优化到最好。事实证明用户接受度也特别高，等用户完全接受了语音连麦，并且反馈了需要“露脸”的需求，我们才在 2017 年上线视频连麦；2018 年，视频连麦也无法满足用户需求了，进一步上线了视频直播 PK 的功能，直播 PK 是一个非常复杂的玩法，而此时快手在技术方面已经做足了储备，迅速上线满足了用户的需求。

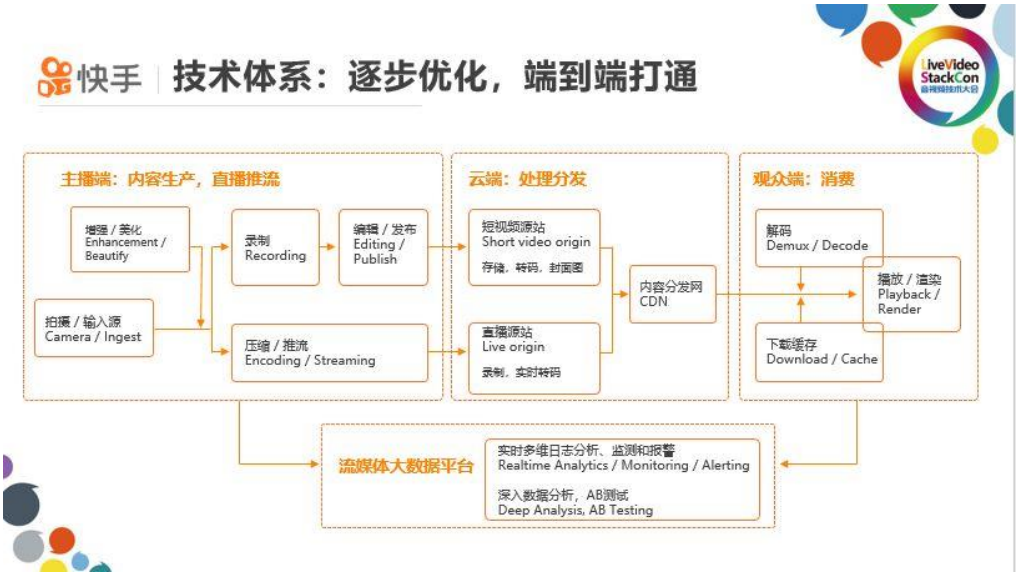
接下来介绍下快手的音视频技术体系建设历程，首先是要从用户场景的痛点出发；第二是端到端的流程打通；第三是做科学化、精细化的流程管理；第四，介绍一下我们是如何抓重点解决若干核心问题的。

对于移动端短视频的体验，从拍摄到编辑、发布，到浏览、播放，用户都有不同的诉求。比如拍摄要真实还原，画质也要优异；编辑要操作顺滑，发布成功率要高；浏览要极速加载、播放流畅，清晰度要高，这是理想的情况。但现实情况是拍摄的时很卡顿，很多性能较差的机型帧率不高，分辨率又低，有偏色的问题，画质也参差不齐；编辑发布响应非常慢，上传失败率高；播放的时候有加载慢、播放卡顿、画质模糊不清楚的问题。这些都是用户的痛点，所以我们设计的方案都是针对这些痛点来做的。

接下来是移动端直播体验，其中直播端涉及推流、连麦，观众端就是简单的观看。理想中的移动端直播，需要清晰流畅、低延时，观看端要极速加载、清晰流畅。

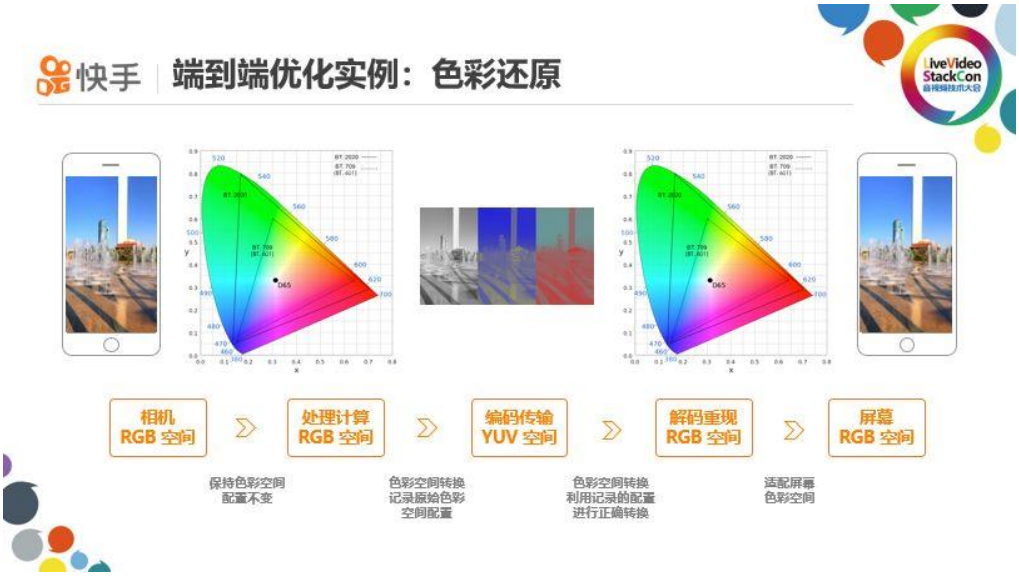
我们在实际优化过程中会遇到很多的挑战，比如连麦的体验，用户一连上就卡，该如何改进？这里放了一张《火星救援》的图，Science the shit out of it，我们相信科学的力量。

怎么样才叫科学的方法呢，就是要做到可描述，可衡量，可验证，具体执行下来必须有分析痛点（确认问题）、提出方案（假设）、本地验证、线上 AB、确认收益几个环节。首先要分析用户的痛点，产品、研发、运营等各个团队都会反馈用户遇到的问题，之后研发团队会深入实际的用户案例，提出通用的解决方案，然后要做本地验证。谁来验证呢？就是医生团队，我们有专门的音视频质量实验室，他们来验证我们的方案是否靠谱。为什么要本地验证？因为经常有算法工程师做了一个算法，自己验证过觉得非常好，结果上线后大家说没效果，这就是缺了一个非常重要的本地验证的环节。因为算法可能在线上系统没有被调用，或者参数输出有问题，要想非常精细化地验证画质、音质是否有改善，需要专业团队专业设备来做。之后才能去做线上做 AB 测试，通过数据分析 QoS 和 QoE 的指标是否改善。QoS 是技术指标比如卡顿，QoE 更复杂些,如用户观看时长有无变化,而影响时长的因素非常多,想要确认 QoE 收益需要大量独立的数据实验。



为了做端到端的体验优化，我们经过两年的时间逐步构建了快手端到端的音视频技术体系，打通了整个快手的直播短视频的技术栈。首先是主播端，就是内容生产和直播推流，包括了拍摄、编辑、直播推流三个大的 SDK，把视频通过文件或实时流的方式到云端。其中短视频源站和直播源站都是我们自建的，短视频的源站是做存储、转码、封面抽取，直播源站是做录制、实时大规模的转码，这些都是业界规模最大的。内容分发是多家一线 CDN 供应商,消费端是我们的播放器 SDK,可以做直播和短视频的播放。

各个环节的 QoS、QoE 数据都会上报到我们的流媒体大数据平台上，平台可以做实时多维的日志分析、监测和报警，任何区域任何运营商有播放、上传、推流的问题，在 5 分钟之内就可以报警。还可以通过深入的数据分析去找出用户的痛点，上线优化方案，做 AB 测试，验证我们的优化是否有效。



为什么要端到端打通技术栈？好处很大，举个色彩还原的例子，从拍摄开始到编辑、压缩、传输，再到解码重现，在屏幕上绘制出来，这个过程中很多地方都会涉及到色彩空间的转换、格式的转换，任何一个地方做错了，都会导致呈现出来的视频清晰度、色彩有偏差。想解决这个问题就需要从头到尾梳理清楚，任何环节有问题都需要修改。

接下来分享一些我们重点解决的核心问题。

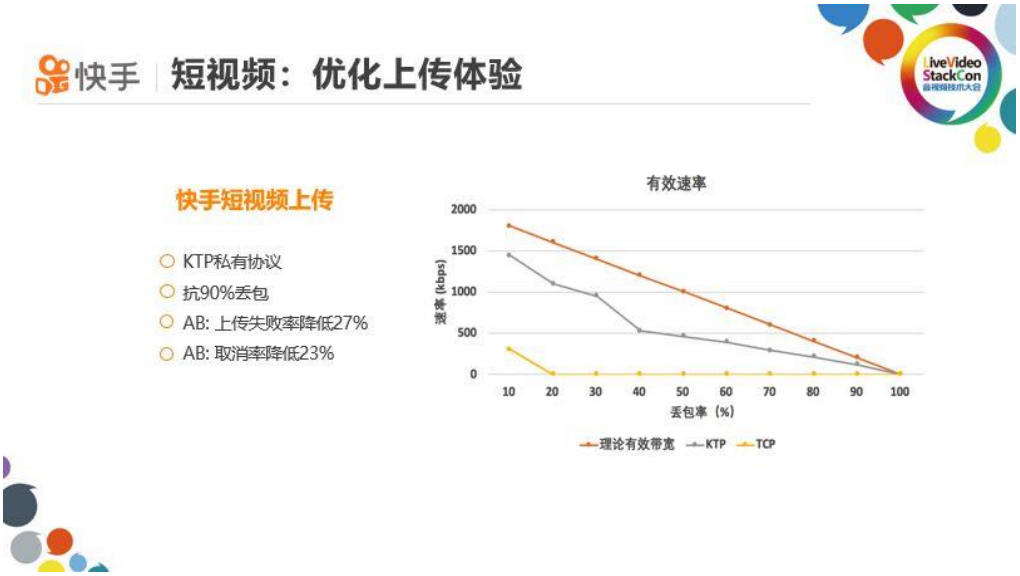
首先我们需要先看下短视频的业务特点，快手是海量用户上传播放视频的平台，用户分布广、机型分布广。其中，相当一部分快手用户处于偏远地区，痛点在于移动端拍摄编辑不流畅，低端机型画质参差不齐，以及移动端上传失败率高，清晰度低、流畅度差等问题。我们主要着力解决的就是这四个问题。

1、优化移动端的拍摄和编辑

快手内部多个团队合作打造了一个性能可伸缩的移动端多媒体引擎，从视频的采集、编辑到上传实现了全流程打通，带有媒体引擎、图形引擎、图像引擎、AR 引擎（由 AI 引擎支撑），再向下是底层的硬件 Video Encoder/Decoder，以及 CPU、GPU、GSP、NPU 的支撑。有了这些引擎、硬件的支持，用户就可以进行短视频的拍摄编辑、视觉特效、魔法表情、图像增强、AR 特效等应用，能够做到高端的手机效果非常好，低端的机型也能够流畅的处理。由于算法复杂度可伸缩，低端手机效果可能相对差一点，但是流畅度没有问题。

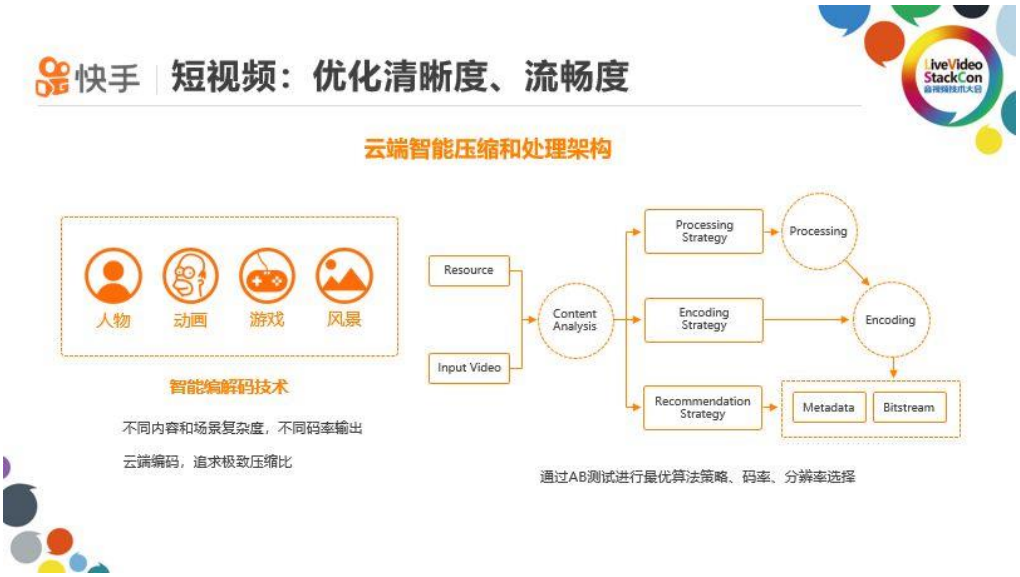
2、优化拍摄画质

优化拍摄画质方面，为了改善低端机的画质，快手开发了自动增强滤镜，能够自动调节色彩亮度，处理画面偏色偏灰的问题，让拍摄功能不太好的国产手机获得跟 iPhone 差媲美的效果。另外一个效果叫暗光增强，是我们跟北大合作的一个算法，本来是光线很差、一团漆黑的视频，暗光增强优化后细节都非常清楚。



3、优化上传体验

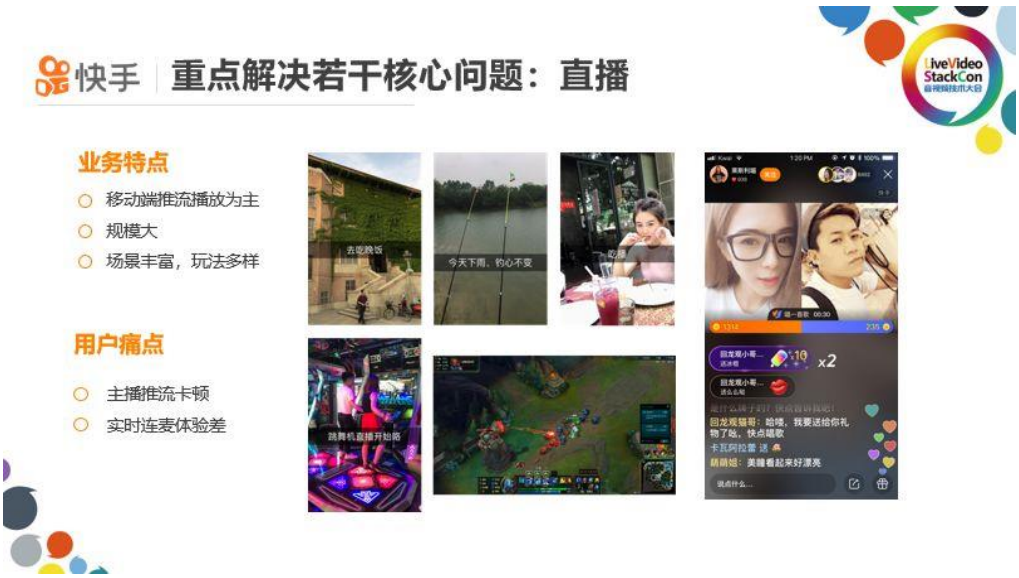
快手拥有自研的 KTP 协议，在短视频 90%的丢包下还可以正常传输数据。理论有效带宽在不同的丢包率下肯定是逐次降低的，例如 90%的丢包率下，可用带宽可能只有 10%，但 10%中我们还能用到 6%；然而用 TCP，丢包率降到 20%就不行了，所以用 TCP 是完全不行的。上线 AB 验证的结果，可以看到 KTP 上传失败率降低了 27%，取消率降低了 23%，还是很有效的。



4、优化清晰度、流畅度

清晰度流畅度的矛盾大家应该都遇到过，如何解决呢？视频压的越小越流畅；而保持同样视频文件的大小，压缩比越高，视频就会越清楚。所以最重要的就是智能压缩的算法。短视频不需要一个恒定的码率，我们可以针对不同的内容和场景复杂度，输出不同的码率，自动根据内容做适配。如果是非常复杂的场景，我们会把码率调高，如果是非常简单的场景码率可以压得低一些。另外，我们选择在云端做非常高复杂度的压缩，追求极致的压缩比。

还有什么手段来优化清晰度呢？可以让系统把不清晰的视频先过滤掉，例如上图最左边的图片系统可判断大概率是很清楚的，中间的图片大概率是模糊的，右边可判断实际上是镜头脏污监测。很多用户的手机拍出来的视频不清楚，主要是因为摄像头不擦，这些我们系统都能检测出来。



下面介绍直播场景。快手直播的特点是以移动端推流、播放为主，我们也支持 PC 端的推流，主要是做游戏直播。还有一个特点是规模大，是世界上最大的直播平台，场景非常丰富，玩法多样，吃饭、钓鱼、玩个跳舞机都可以随时随地直播。这对我们系统的挑战也非常大，因为用户并不只是坐在家里用固定网络直播，移动端推流的网络、位置都是不确定的。

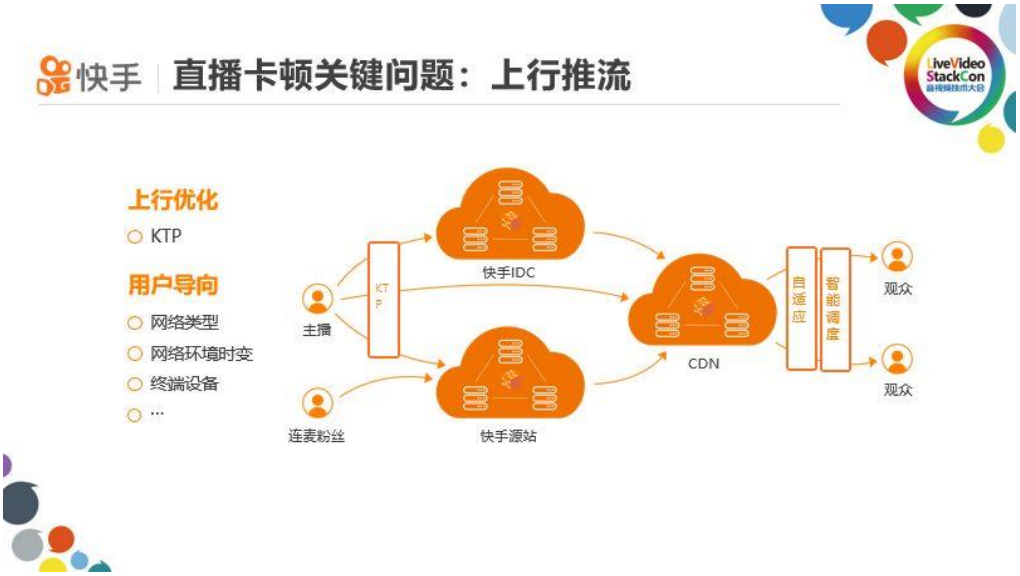
另外直播玩法多种多样，比如最近的 PK 特别火。用户痛点在于，主播端推流卡顿很多，如果主播很卡，所有用户都会卡，所以主播卡顿是我们要重点解决的问题。

直播	延迟	流畅性	可靠性	清晰度
互动直播	< 5 s	高	高	较高
游戏直播	<8～10s	高	高	高
连麦	<400ms	高	一般	一般

直播有几个细分的场景，首先是互动直播，就是我们经常说的观众和主播之间的实时互动，它的延迟要求是小于 5 秒，对流畅性要求特别高，对可靠性要求也非常高，对清晰度要求较高。

其次是游戏直播，对延迟要求更宽松一点，为什么有的主播其实希望延迟高一点？如果游戏主播延迟很低，主播在游戏中就暴露了实时位置，容易被游戏中的敌人伏击，所以游戏主播通常希望延迟在 10 秒以上，这就是基于用户个性化体验的需求，技术人员不做产品可能就理解不了。游戏直播对流畅性、可靠性要求比较高，清晰度则是要求最高的，因为很多游戏直播是 1080P 甚至是蓝光的体验。

最后是 PK 和连麦场景，延迟要求是在 400 毫秒以内，对流畅性要求高，可靠性要求不高，能够容忍一定的丢包。主要是保证低延迟，因为延迟一旦高了，相当于没有实时性，连麦体验很差；连麦清晰度方面其实可以做得更高，但是为了保证延迟低，清晰度可能会做一些牺牲。



基于上述分析，我们重点优化的方向就是直播上行，这个上行是用 KTP 推流来解决的，把直播流通过 UDP 协议推到快手源站，这样在各种弱网下都可以顺畅的直播。然后通过多家 CDN 去做智能调度和码率自适应，让观众端也更流畅。

上图是优化的一个例子，左边是 TCP，右边是 KTP，视频码率大概为 3-6M，带宽限制最高 5M，丢包率是 10%，这时候可以看到 TCP 会经常卡顿，KTP 则非常流畅。

实时连麦的场景是音视频领域工程和算法里面挑战最大的，像 Apple 的 FaceTime，以及微信的实时音视频通话，对传输、编解码、音频处理等相关算法都是要求最高的环节。

首先，传输和编解码要做到延迟低于 400 毫秒，在 400 毫秒以内做所有的算法，要实现 20% 的丢包无感知。另外直播连麦要实现无缝切换，不能连麦以后让所有观众卡一下，所以要做很多工程的优化。

音频处理也非常复杂，要做噪声、回声、增益的控制算法。例如 PK 场景中，两个主播要一起唱歌，就需要支持高音质的 K 歌，混响、音效等很多方面也要做相应处理。所以这个架构看起来比较简单，但是里面有很多非常复杂的算法。快手的这套体系全是自主研发，没有使用开源，便于我们根据业务需求深度定制。

无论直播还是短视频，怎么才能保证线上的体验最好？我们的大数据系统负责实时分析、监测和智能调度，这是快手音视频技术栈的一个突出特点：音视频团队里面有一个大数据团队。我们一开始做的是离线分析系统，做由于数据量太大，做一个实验，需要跑 10 个小时数据才能出结果，效率特别低。于是我们开发了实时多维监测报警平台，能够实时监测线上的数据到底怎么样，任何细分地区都可以看到。我们后面还实现了实时多维 AB 测试，以前做 AB 测试需跑两天才能在线上查数据。现在 AB 测试上了以后，马上就能在线看结果、继续做调优。

下一步我们在做的是实时多 CDN 智能调度。过去我们做实时多维监测报警，如果半夜有报警过来，我们要人工看是哪个区域 CDN 有问题，把它切掉，晚上没法睡觉很痛苦。有了这个系统以后，切量操作或者量级的配比等优化局部卡顿的操作，都可以让系统自动去做。

另外，我们还做了故障自动归因，当一个区域视频发生卡顿了，问题在哪，是运营商的原因还是 CDN 的原因？如果是 CDN 的原因我们就找 CDN 去优化，如果 CDN 没有问题要通过渠道找运营商去优化。归因非常重要，之前都是要人工排查原因，现在用智能的算法就能自动找到原因，直接报警给 CDN 或运营商方面。

三、移动音视频技术趋势探讨

下边跟大家聊一下我们所关注的未来的趋势，大家都能看到，AI 是未来的一个大方向，AI 的算法在编解码算法、智能影像技术、传输和分发算法、智能音频处理上，都会有长足的发展和应用，快手对此也比较关注。

编解码是一个基础技术，我们所有的工程优化都是围绕着编解码在做，编解码可谓是音视频的核心引擎。

首先，硬件的编解码会逐步的成熟，硬件速度快、功耗低、密度高，在移动端、直播等领域应用非常广泛，移动端方面，我们在 iOS、Android 大量使用硬件编解码能力，对提升用户体验非常有效；在 GPU 方面，我们在直播上支持 Intel 和 NVIDIA 两家的高密度硬件编解码方案；快手也在关注 FPGA,评估这个方案在性能和速度上的优势。我们相信硬件编码无疑会越来越好，但在短视频领域，我们主要还是用软件去做，因为我们追求的是压缩比。

后 HEVC 时代，我们比较关注的是 H.266 和 VVC 的国际标准，还有包括 AV1、AVS3、XVC 在内的下一代的编解码技术，快手在美国也建立了标准实验室团队，也在澳门的会议上中了两个提案，我们非常关注下一代标准，希望能贡献我们的力量。

移动端的视频跟标准的通测序列差异是很大的，快手也很关注在这个移动端的视频集里，如何能够针对这些视频的特点在这种序列里面做一些创新的优化算法。

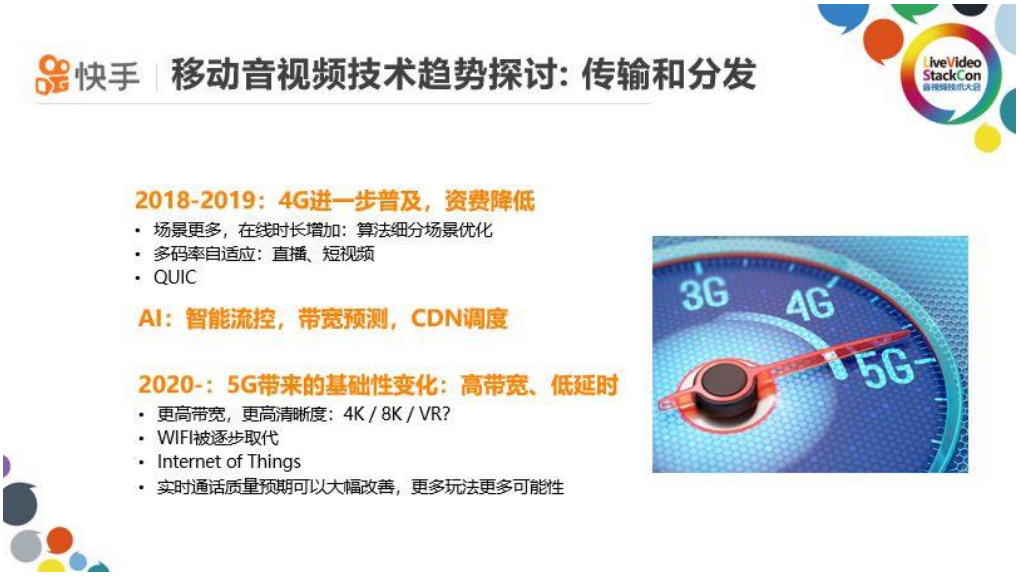
跟 AI 的结合方面，我们比较看好的是通过 AI 来做主观质量评价。现在很多公司用的都是 VMAF，VMAF 其实是用传统图像算法来做 feature extraction（特征提取）、用 SVM 来做预测。有没有办法用深度学习来做特征提取和预测？可不可以比 VMAF 更准更符合人的主观直觉？ 在有了好的主观评价方法基础上，视频的前后处理，也就是视频增强用 AI 实现也会很有收益。

另外就是针对视频的特点，如何精细地做分析，很多做多媒体分析的同学都是从图像角度来分析的，但如果结合视频特征做内容理解收益会很大，也就是结合做编解码的同学对视频模型的理解，以及成熟快速的编解码算法，可能对视频内容的理解会非常有帮助。

再看一下 ROI(region of interest)编码，通过 AI 可以代替人眼找到视频中比较重要的、用户会感兴趣区域，利用 ROI 编码圈定用户主要关注的目标，减少信息量。但我认为这个只能在超低带宽情况下使用，如果带宽足够，视频还是尽量清楚比较好，如果我们把自己认为的一些不重要的细节抹掉了，这些细节对用户而言可能恰恰是视频的亮点、趣味性所在。因此 ROI 在网络不佳的极低码率条件下，才会有用武之地。

下面是智能影像技术。视频的拍摄实际上是一个非常大的领域，跟编解码算法要做非常多的结合。如果在快手 APP 里面做一个普惠的算法，能够覆盖很多的机型，覆盖非常多的普通用户，让计算摄影学这样高深的技术走向大众。这其中有一些推动力量，比如 Camera2 接口，有了 Camera2 的接口实际上能获得更多的信息，包括更丰富的色彩信息、更广的亮度信息，感知空间的深度信息，在此基础之上，可以利用更好的算法替代传统的 ISP。

快手也在构建自己的 Software ISP 引擎，希望能够随着 CPU、NPU 算力的增加，AI 的能力会让手机越来越强，可以用这些计算资源做很多复杂的东西，比如场景识别、物体识别；比如针对不同场景物体对图像做有针对性的调优；比如可以做以前移动端做不了的 Super Resolution，Denoising， Low Light Enhancement。如果这些算法如果能够在移动端跑起来，未来对拍摄体验的改进会非常大。



这两年 4G 在进一步普及，原来很多用户比较担心流量费用较高，出门会手动关掉 4G 流量。最近运营商都在推 4G 免流套餐，让用户能够放心刷视频，我们预计用户在线的时长会显著增加。在更多的场景，比如原来没办刷视频的地铁、高铁、户外、人员密集的地方现在都可以刷视频，这对我们的要求就是针对不同的细分网络场景做算法优化。

我们现在比较关注的是多码率自适应，直播短视频方面都在尝试，尤其在短视频上的多码率自适应业界是没有尝试过的，我们有专门的算法团队做这个优化。还有 Google 做的 QUIC 协议，在用户体验提升上面会有很明显收益的，我们也在关注，各个 CDN 什么时候会能够全面支持。

传输和分发怎么与 AI 结合呢？我们看好智能流控，其中包括带宽预测，还有 CDN 调度，用强化学习等方法做预测。我们更看重这些算法能否上线，上线的效果怎么样。

2019-2020，5G 时代将会正式到来，5G 会带来很多基础性的变化：高带宽、低延时。更高的带宽意味着更高的清晰度，它的带宽会是 4G 的一百倍，所以 4K、8K、VR 都不成问题。在未来的时代要做哪些玩法，进行哪些创新，这是留给大家的思考的问题。还有 Wi-Fi 可能会逐步被取代，很多地区现在就有这样的趋势，如果 4G 资费足够低，用户可能就不装宽带了。还有就是 IoT（Internet of Things），即任何的设备都有一个 IPv6 地址，都可以通信，24x7 的录制和传输视频。再者就是实时通话质量预期可以大幅改善，比如现在微信的语音微信的视频还是不能完全替代电话的，5G 时代，这些改变都可能发生。

智能音频处理。这是快手另外一块很重要的算法，我们会做更高音质的回声消除、噪声抑制和增益控制，这方面我们有专业的团队在做。音频的编码传输需要在实时传输和弱网丢包的情况下做很多的优化。

我们也在利用 AI 技术做音频理解和生成。比如短视频拍摄的一个痛点是给视频的配乐要花很长时间，我们希望算法能够通过用户的视频内容、场景乃至情绪理解，实现自动选取配乐，甚至自动生成配乐。

我的演讲内容如上，感谢 LiveVideoStack 会方的精心组织，这是一年一度音视频技术爱好者的大聚会，也希望今后有更多的同学参与分享音视频优化的经验，谢谢大家！

提升音视频 QoE 的六条建议

音视频服务追求的不仅是单纯 QoS，而是用户最终的极致体验。阿里巴巴信息平台资深技术专家何亚明在 LiveVideoStackCon 2018 音视频技术大会的演讲中针对点播，直播等不同场景结合音视频编码，网络传输，服务器架构等方面实际经验，详细解读了如何从端到端整体提升用户的音视频极致体验。LiveVideoStack 对内容进行了整理。

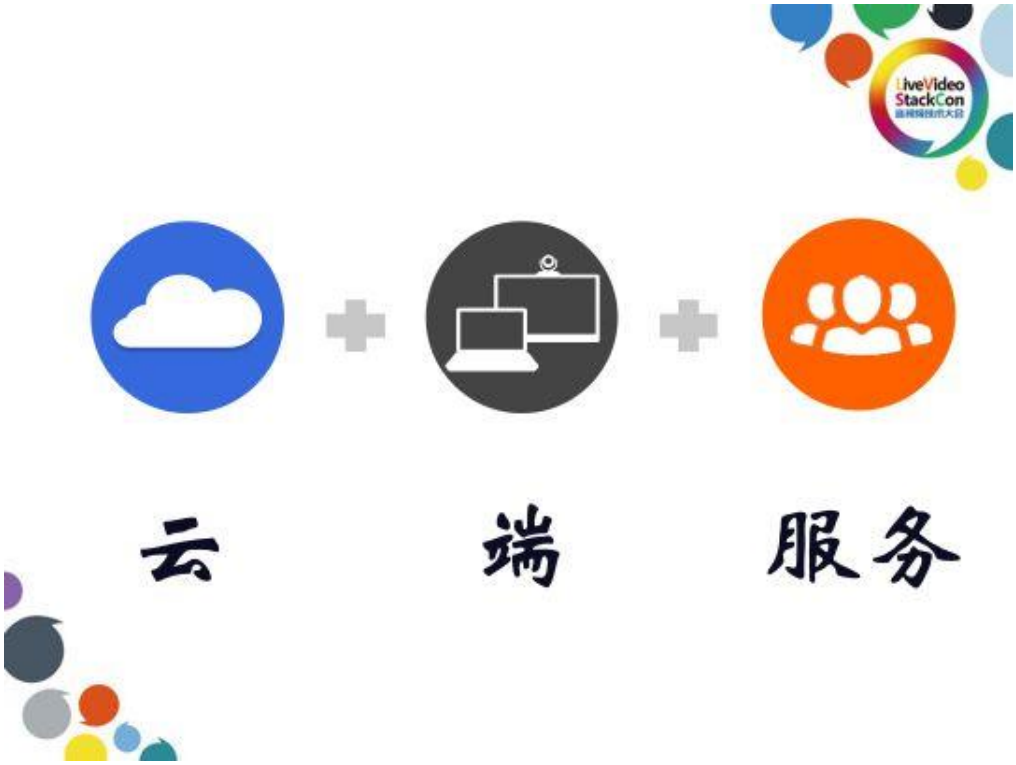
何亚明　LiveVideoStackCon 2018 北京　2019-02-22

大家好，我是来自阿里巴巴的何亚明，主要负责阿里巴巴信息平台在音视频会议业务部分的技术研发，今天我将为大家分享如何打造音视频服务的极致用户体验。

我最早在微软从事底层编解码开发工作，过去作为团队成员为北京奥运会直播提供技术保障；也曾在 Facebook 工作四年并参与了 Messenger 与 Facebook Live 两项产品的开发，Messenger 中的 Video Call 便得利于 WebRTC。相对于国外如 Facebook Live 等直播平台侧重于实时新闻讯息直播的应用场景，国内的直播场景与玩法更复杂多样，这也是我今天分享的关注重点。

1. 视频行业发展方向

回顾视频行业的发展历程我们可以看到，从上个世纪七十年代的黑白电视，八十年代逐渐过渡至数字化到九十年代的 IP 化与标清时代，再到千禧年后的高清与现在的融合通信，总体趋势是不断发生的行业整合与升级。整合体现在以前的电视、互联网、通讯社交等多种通信方式相互独立到现在的全面融合，升级则体现在 4K 超高清与其编解码的新标准，还有背后的硬件资源等逐步实现云化、硬件资源池化、软件化与虚拟化。无论是思科、微软、亚马逊等国内外大型科技企业还是中小规模初创企业都将音视频纳入生产业务流并将服务进一步升级作为关注重点。



总而言之，音视频服务的未来发展趋势便是如上图概括的那样：由云+端+服务三方面元素组成。

其中的“云”是指云平台化。无论是传统的 PaaS 还是 SaaS，亦或者是私有云与公有云，一切功能都是基于云的服务，云服务已经成为我国的一项重要的基础设施。而在“端”方面，除了

我们熟知的手机端、PC 端、Web 端等，对于企业应用场景而言会议室也是需接入云的重要一端；随着 IOT 时代的到来，任何基于 IOT 的设备都可作为云服务的一端实现音视频通话、音视频流播放等功能。

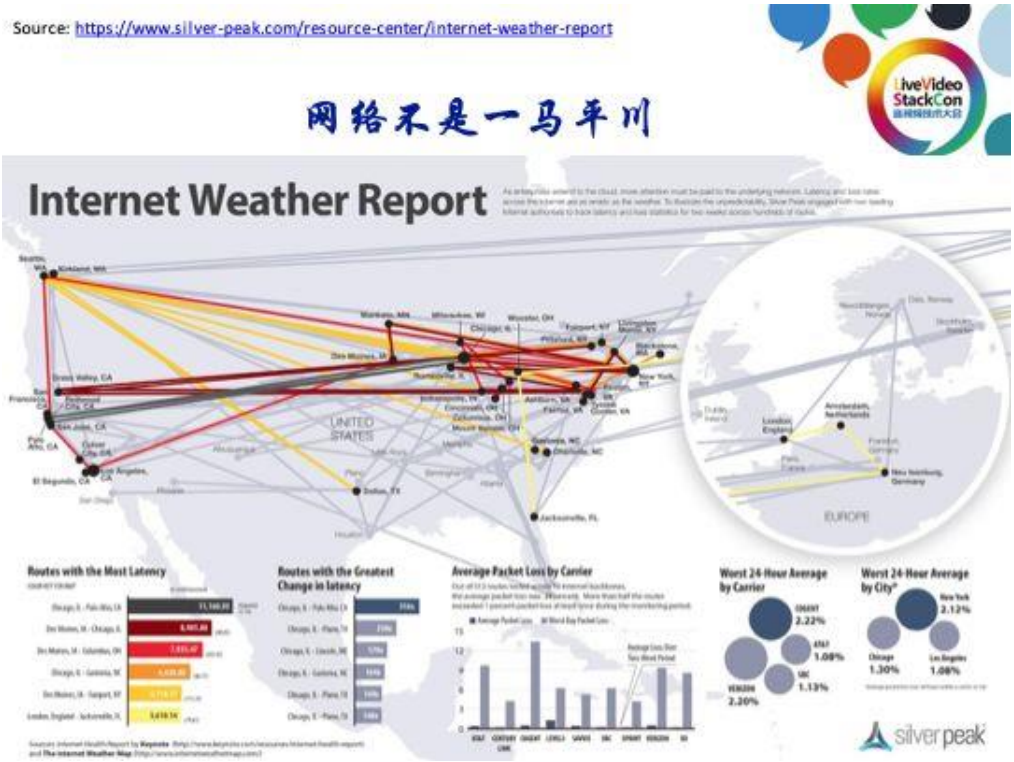
“服务”除了传统的短信、语音、IM、音视频、呼叫中心等，云客服与附加 AI 的服务也是未来音视频服务的重要趋势。

根据由思科出品的 Virtual Network Index 报告，预计到 2021 年全球互联网用户规模可达到 46 亿，这对中国企业而言意味着单纯考虑国内市场是远远不够的，拓展国际化市场势在必行；而随着新一代通信与物联网车联网等技术的不断成熟，预计到 2021 年全球连接至互联网的设备可达 270 亿，而全球互联网流量中有 80%是视频流量，可以看到未来所有互联网服务或多或少都会与音视频相关。十年前我们的信息传播与交流沟通主要依赖文字，近五年我们更频繁地用图片表情包等传递情感，而在不远的将来，基于音视频流的通讯服务毫无疑问将会成为人类传递讯息与表情达意的最常用方式 。于是可以预计在 2021 年，每一秒钟就有一百万分钟的视频内容通过互联网传递，借助 WebRTC 等音视频技术与多种开源解决方案，直播的门槛会越来越低，观看内容也不会再受到时间空间的限制。

画面的提升与分辨率、对比度、色域与色位、帧率等指标紧密相关，而视频分辨率的提升有目共睹。从上个世纪的 CIF、千禧年的 4CIF 到十年前的 720P 与 1080P，再到日趋普及的 4K 与逐渐成熟的 8K，分辨率的提升意味着画面的极致体验，同时也带来了数据量的成倍增长；除了视频分辨率的不断提升，高反差 HDR、广色域、高对比度等可为画面质量与颜色质感带来明显改进的技术也正逐渐受到行业的重视；随着用户对画面的观感与色彩还原程度的要求逐渐提高，传统的 8-bit RGB 已无法满足需求，所以诸如 10-bit RGB、12-bit RGB、16bit-RGB 等可保存更多色彩信息的色位方案不断涌现，随之而来的数据量会在分辨率提升的基础上进一步翻番；高帧率更是绝佳视频体验的必备元素，传统电影大多使用 24 帧拍摄完成，电视则多使用 30 帧；而在未来诸如 60 帧、120 帧甚至 160 帧等会成为媒体行业的标准配置。

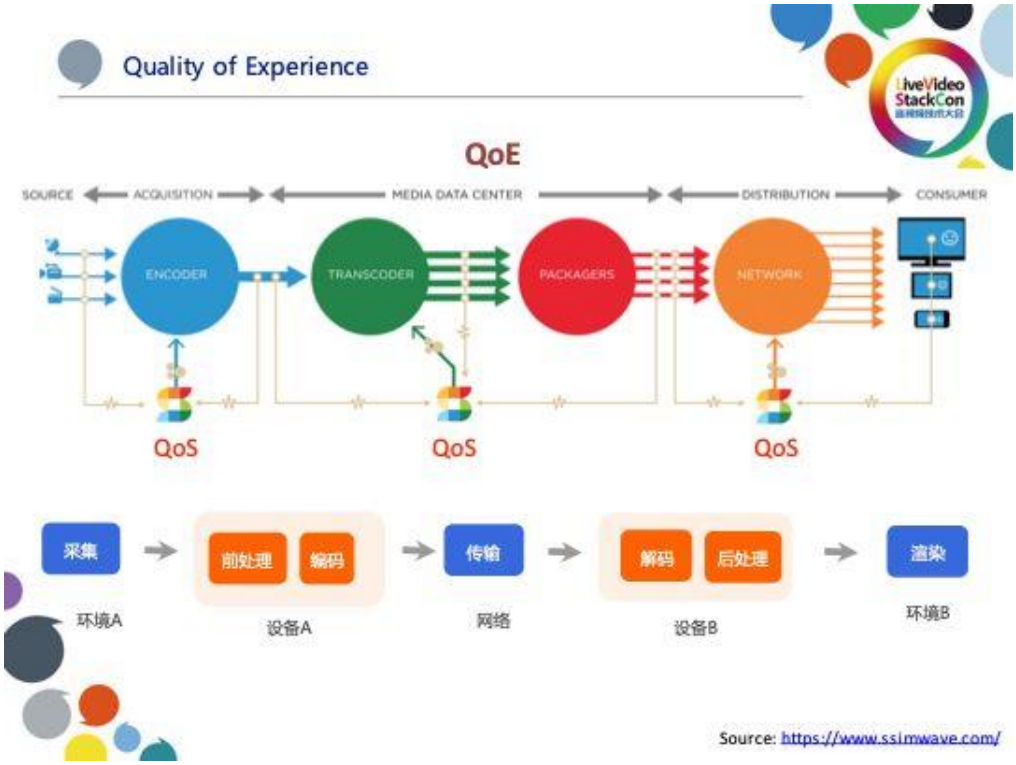
提升声音体验的首要方式是优化声道配置。从传统的单声道、立体声、5.1 与 7.1 到现在的环绕声全景音再到不断成熟的的声音阵列，音频随着场景的不断复杂也需要越来越多的数据量与越来越大的带宽来承载。

总而言之，优异的音视频服务体验一定需要高质量的网络与充裕的带宽作为海量数据传输的基础保障。



但在现实当中，高质量的网络与充裕的带宽并非信手拈来。上图基于 Silver Peak 公司的报告展示了欧洲与北美两个地区网络传输的延迟情况，可以明显看到现实中的网络并非我们期待的那样一马平川。其中发生的大量网络丢包并非由包质量问题所引起而是由于发生网络拥塞时，路由器无法及时处理数据拥塞以至于将所有的包都放至 Buffer 中，导致包的数量超过阈值并出现丢包情况。虽然条形图显示的每两周平均丢包率为 0.34%，但有些天的平均丢包率却远高于这个数字可达到 2.7%，如果以每毫秒为单位统计丢包率其值一定会继续增大。通常的丢包补偿技术有 NACK 和 FEC，以及通过 Codec 本身来适应的 SVC 和 SAC，在某些情况下丢包率可通过对网络架构的调整进行优化。

除了丢包率应受到我们的重视之外，抖动也是一项值得关注的指标。抖动是指信号在某些特定时刻相对于其理想时间位置上的短期偏离。抖动对声音质量有比较大的影响，有时我们可以利用背景噪音弥补抖动带来的影响并基于包前后收到的信息对丢包进行一定程度上的弥补，但类似算法往往无法完全解决丢包问题，采用多种优化与解决方案使传输尽可能适应网络环境是我们当下重点探索的方向。



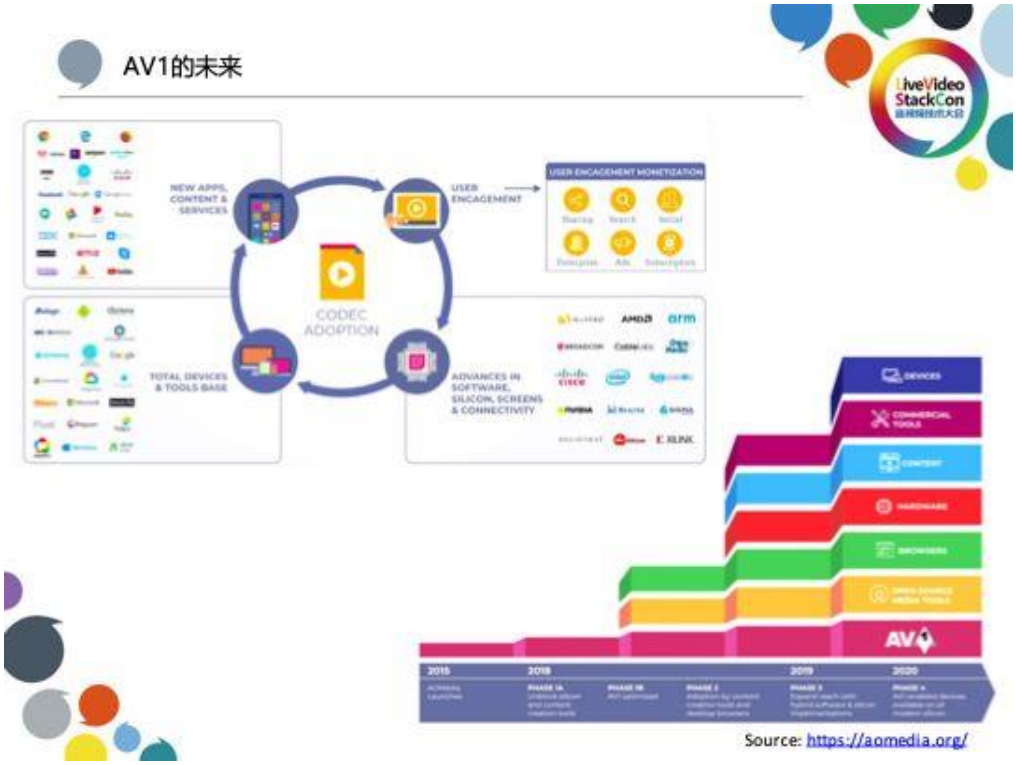
可以说，从采集源头到用户的整条音视频处理链路上的每一个元素都直接影响到用户体验的优劣。采集而来的视频文件经过前处理与编码之后，通过网络传输至解码与后处理设备，在此处理完成后进行渲染并传输至用户端。需要注意的是，编解码转换完成后的数据经由网络传输传递至最后的渲染流程，整个过程需要经过一个包含端、服务器、主干网络、WLAN、Cellular等等元素在内的冗长链路，即使我们能让其中的主干网络满足理想的条件，也难以让复杂多样的支线网络与端设备满足统一的技术标准。

2. 视频编码选择

为优化丢包与网络抖动问题，我们有哪些比较好的思路？与汽车在高速公路上通行类似，提升道路通行效率的可行方案是缩小每辆汽车的体积与完善路面铺装情况。对应到音视频传输上来说，缩小每辆车的体积就是进一步优化编解码方案，其中的关键是选择合适而高效的视频编码方案。与过去几年各种良莠不齐的编码方案鱼龙混杂不同，现在的编码方案更为标准化，这样最大的好处是其在不同设备上的兼容性相较于以往提升明显。虽然现在业界较为主流且兼容性最好的编码方案是 H.264，但诸如 VP8、VP9 等更高性能编码方案也越来越受到如 Facebook 等公司的青睐，这也与不同国家地区的主流设备特性相关。如我国现在市场上销售的绝大多数手机都支持 H.264 及以上的编码方案，而如非洲等一些国家与地区的用户所用的设备则无法满足这一要求。根据实际应用场景选用合适的 Codec 是选择视频编码的最重要思路，有些场景下选择 H.264 可能相对于选择 VP8、VP9 等效果更好。虽然硬件水平无法改变，但我们可通过基于应用场景的算法优化提升软件的适应能力，从而为尽可能多的用户提供良好的使用体验。

2.1 AV1

作为值得期待的新一代视频编码标准之一，AV1 的性能提升可通过上图数据清晰体现。经过汇编级的优化还可实现进一步提升。



AV1 最大的优势在于其开源而不受专利权的限制。相较于 Google 一家独大而缺乏其他厂商硬件支持的 VP9，AV1 在 Codec 层面结合了谷歌的 VP10 以及思科、Mozilla 等多个厂商的 Codec 工具，并实现了主流硬件、软件、内容生产厂商的全面参与从而建立生态闭环。相关的最新结果是 AV1 的码流标准已经确立，而 Encode 和 Decode 也正在优化当中，预计到 2020 年基于 AV1 的硬件芯片即可亮相。

2.2 场景对编码的影响

不同的用户场景对视频编码的需求

	点播	直播	实时通话
视频编码效率	★★★★	★★	★
视频编码速度	★	★★	★★★★
码率稳定性	★	★★	★★★★
容错能力	★	★★	★★★★

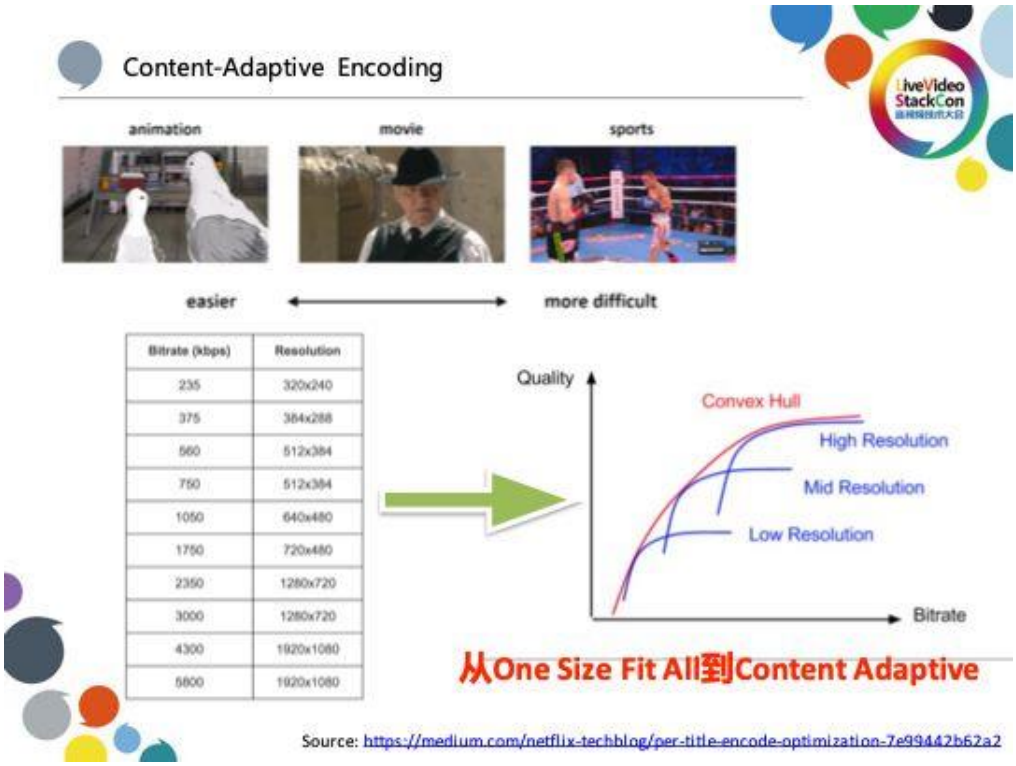
不同的用户场景对视频编码的需求也有所不同，这就要求我们需要分别针对点播、直播、实时通话等不同场景定制参数不同的编码方案。

对点播场景来说视频编码的效率是至关重要，因为点播场景要求采集端使用尽可能少的时间上传尽可能多的数据，而编码速度、码率稳定性与容错能力的考察权重则低于视频编码效率。

至于直播场景，由于允许其存在几秒的延迟，我们对视频编码效率的要求不如点播场景严苛，但由于直播对传输的要求高于点播，其对视频编码速度、码率稳定性与容错能力的要求相较于点播进一步提升。

实时通话场景无疑需要非常高的视频编码速度、码率稳定性与容错能力，才能在尽可能降低延迟的同时保证数据传输的稳定；我们努力控制延迟不高于 300ms 以降低人对延迟的主观感知，为此可以牺牲一定的视频编码效率以换取更高的视频编码速度。

3. 内容自适应编码



Content-Adaptive Encoding 也就是内容自适应编码则是当下一个较为流行的技术趋势。大家知道无论是点播、直播还是实时通讯应用场景，视频的分辨率、码率、帧率等参数会随着应用场景的不同而做出变化，影响这些参数的因素包括视频内容、编码方案、硬件水平、网络带宽环境等等。根据上图右下角的曲线我们可以看到，在分辨率不变的条件下增加码率，视频画面质量会在初期明显提升，达到拐点之后增长趋于平缓，这个时候就应该提升视频分辨率来提高视频质量，所以单纯地增加视频码率不会带来视频质量持续的线性增长。如果我们希望通过少量的带宽资源稳定传输高清甚至全高清视频流，即可基于此图像采用码率和分辨率自适应编码的方案，在低码率条件下使用低分辨率而在高码率条件下使用高分辨率，每当码率增大至对应分辨率图像拐点位置时切换分辨率至更高值，从而实现如图中红色曲线 Convex Hull 展示的随着码率提升，视频质量不断提升的效果。而实现此方案的关键在于基于视频内容精确找到每种分辨率所对应的拐点，解决此问题除了可基于大量的测试和经验值，我们也可引入机器学习来进一步强化拐点确定的精准与快速。

3.1 主观评价指标——MOS

视频质量同样是极致音视频服务体验不可或缺的一部分，众所周知的与 HVS 人类视觉系统模型最为接近的视频质量评价指标便是 MOS，其原理便是采用人眼观看视频并主观评价的方式为不同目标视频赋分。为保证准确性与有效性 MOS 需要大量人力投入以获取足够多的测试结果，其缺点为昂贵且耗时并容易受到环境、心情等无关变量的干扰，我们还是希望能够使用更为客观高效的方法衡量主观视觉体验的优劣。

3.2 客观评价指标

这里有多种客观指标可用于视频质量评价，最早的 PSNR 是累积视频每一帧画面每个像素点压缩前后的差异值再平均化得到的结果。首先，由于人眼的视觉对于误差的敏感度并不是绝对的，其感知结果会受到许多因素的影响而产生变化，加上人眼不会关注每一个像素点，这就使得 PSNR 的分数无法与人眼看到的画面质量完全匹配，有可能 PSNR 较高者人眼观察反而比 PSNR 较低者差；例如，如果我们在视频像素中随意插入一些黑白噪点，使用算法处理这些黑白噪点并不困难；处理后输出的视频虽然理论上画质更好但其 PSNR 值却较低，原因是其与带有黑白噪点的原视频画面存在一定差异，这就意味着人眼可观察到的主观画质提升会被 PSNR 判断为画质劣化，这就是为什么业界随后出现了 SSIM 与 MS-SSIM。其中的 MS-SSIM 会在多种分辨率下对图像进行 SSIM 评价，达到模拟视频在不同大小的显示设备上播放的效果。

更进一步的方案便是 VMAF，其原理是在支持向量机回归因子中使用下列基本指标进行融合：

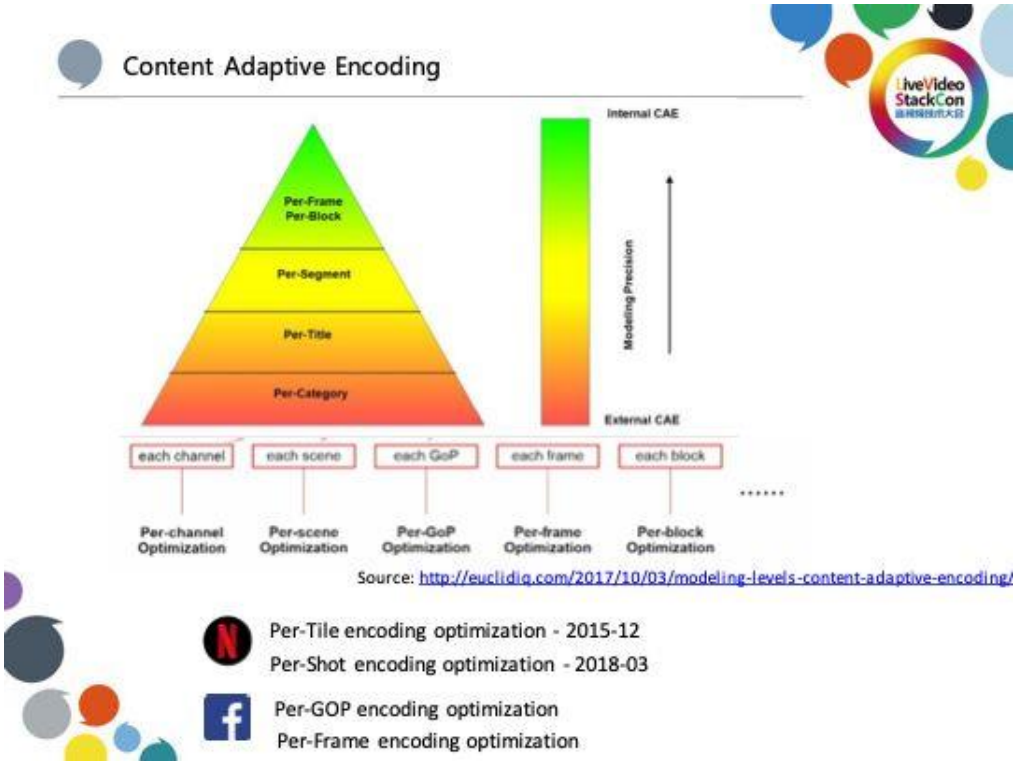
- 视觉信息保真度（Visual Information Fidelity，VIF）：综合四个 Scale 下保真度的丢失情况得出。
- 细节丢失指标（Detail Loss Metric，DLM）：分别衡量可能影响到内容可见性的细节丢失情况与可能分散观众注意力的不必要损失并与 Additive Impairment Measure 结合从而算出最终分数。VMAF 中仅使用 DLM 作为基本指标。
- 运动量：由于 VIF 和 DLM 都是衡量图片质量的指标，考虑到视频的 Temporal characteristic，我们引入运动量这一衡量相邻帧之间时域差分的有效措施，通过计算像素亮度分量的均值反差即可得到该值。

通过获取上述三项指标并加权计算得出基于 VMAF 的视频质量评价结果，将此结果与同一目标文件基于 MOS 得到的评价结果进行对比，分析二者差异与影响因素；同时借助 CNN 等深度学习技术，利用对比分析反馈出的数据训练并优化 VMAF 使其尽可能逼近真实场景下的评价结果。由于 VMAF 中的 VIF 与 DLM 多用于静态图像质量评价，如果我们在视频中间隔性零散插入若干个坏帧，虽然坏帧存在但人眼所感受到的视频画面质量几乎不会出现明显劣化；而如果我们将相同数量的坏帧集中连续地插入视频帧序列的某一个区域，虽然在此区域前后视频质量非常出色，但人眼观看此视频时的主观感受尤其到了插入坏帧的部分一定十分糟糕。诸如此类的问题还需业界进一步探索。

Facebook 曾提出一种被称为 FBMOS 的视频质量客观指标，此客观指标不仅考虑到用户端播放视频时呈现出的视觉体验，更考虑了当网络环境发生变化时缓存端的性能与状态变化情况，此方案更多针对 Facebook 业务形态与产品使用场景而设立。

需要注意的是，上述介绍了一系列评价视频质量好坏的客观指标，并不意味着某一种指标就绝对适合或性能优秀。针对不同使用场景与应用环境的各种视频质量客观指标百花齐放，我们应当根据应用场景特点与视频类型选取最适合的评价指标并随时做出优化与调整才能获取最为客观准确的质量评价结果。

3.3 精度提升



为了进一步提升内容自适应编码的准确，我们需要细化模型分析与处理的精度。从对每一类别视频进行大致考量到对每一种场景与主题内容进行归纳分析，从视频段继续细化至块甚至帧级

别，随着模型精度的不断提升与内容自适应编码的不断深入，其对视频质量与用户体验的优化也在不断提升。上图下方展示的两篇文章分别介绍了 Netflix 分别在 Per-Tile 与 Per-Shot 级别做出的优化，其过程并不修改 Encoder 内核，仅在码率控制上做出优化；而 Facebook 则在 Per-GOP 与 Per-Frame 级别做出了进一步优化，文章内容会对这些内容作出更加详细的介绍，这里我就不再赘述。

Facebook 于 2014 年收购了一家致力于实现 GOP 级别编码优化的公司，其思路主要是通过机器学习为不同类型的视频 GOP 训练多组通用 Encoding 参数，X.264 中的参数大约有一百个左右，这样便组成了一个数据集；通过使用某一种指标（PSNR 或 SSIM）对其进行训练即可获得多组参数变量，通过分析变量间差异与总体变化找到影响视频质量的几个最关键参数；根据视频内容与应用场景的不同选取合适的参数，采用并行压缩的方式压缩视频若干次，每一次仅取一种参数作为变量并观察压缩后的最终结果，压缩效果最好质量最高的即为所需最佳编码调参方案。

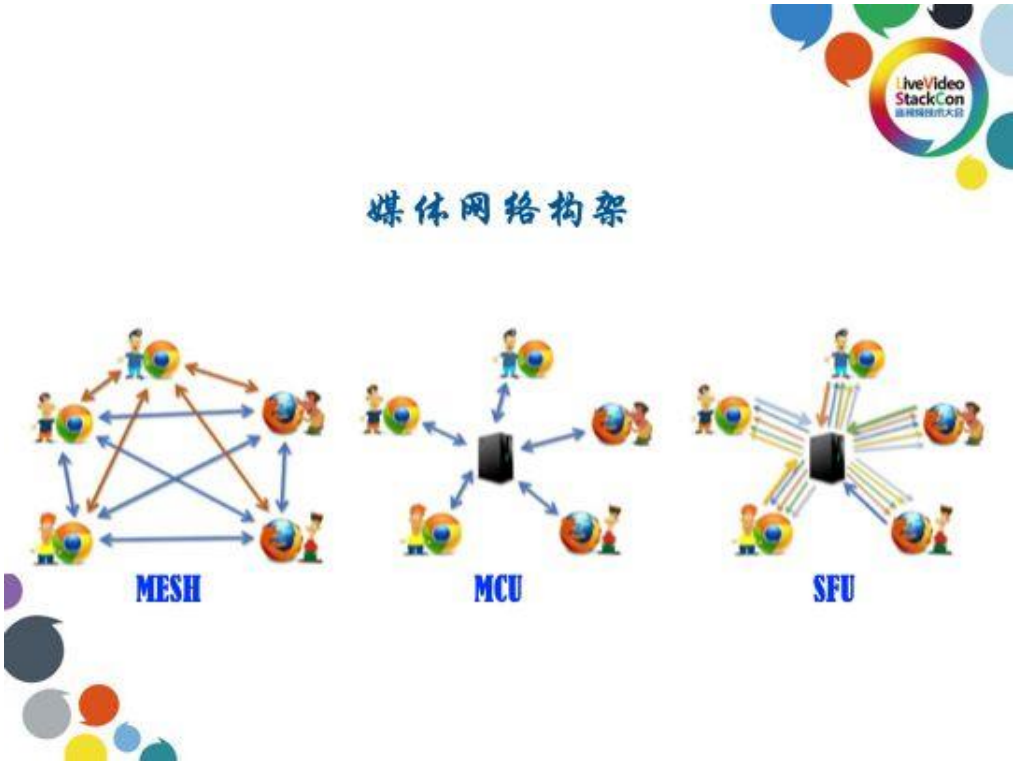
4. 编码优化

进一步探索帧级别的编码优化，我们研究了 Video Saliency Encoding 视频显著度对编码的影响。大家知道，虽然大脑所占人的体积并不多，但却能够消耗人体每日总需能量的 1/3；从基因角度来说大脑会尽量控制过度的能量消耗，反馈出的结果便是人在观看视频时不会注意画面当中的所有细节。换句话说人在观看视频一定会专注于画面的某一区域，而这部分区域一定是与视频内容相关并直接关系到其所传递核心信息与大脑对视频内容的理解。

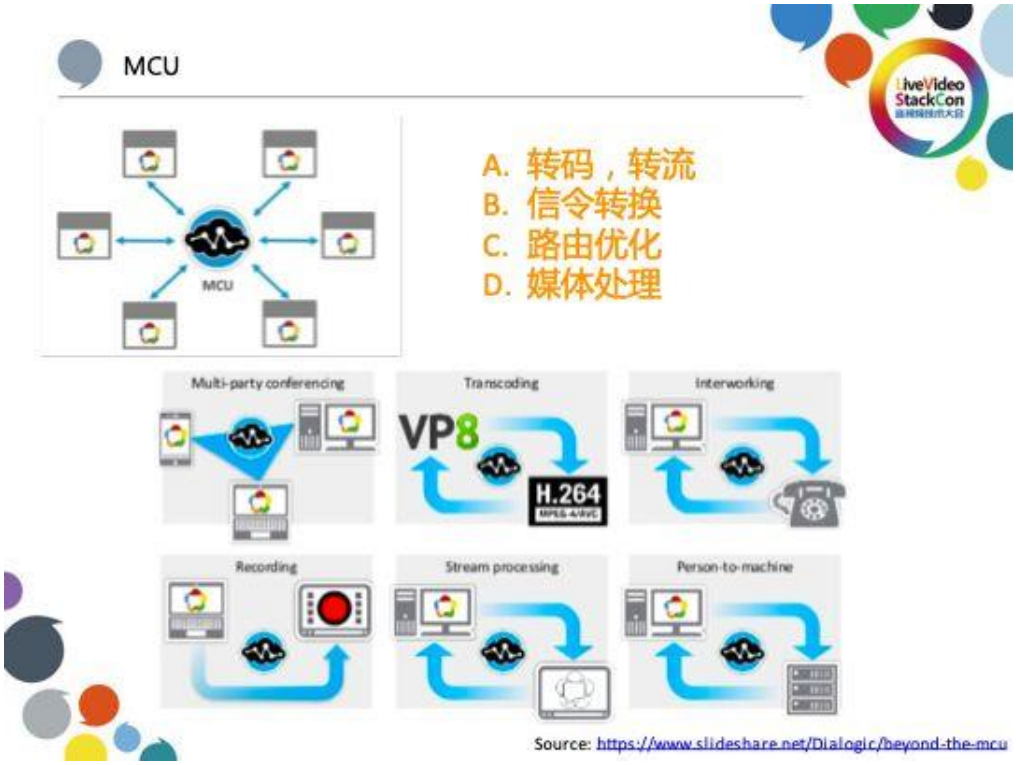
我们通过研究上千位试验者观看同一内容视频时眼球的运动轨迹来确定每一帧视频画面的显著程度，从而得到可反映人眼观察视频画面时专注度的热力图。可以看到上图右上红色区域便是人眼重点观察的区域对象，这一部分也是我们在编解码时需要重点优化以保证其画面质量与流畅度的区域。此技术多用于 VR、全景视频等视频规格更高且对实时性要求更严苛的应用场景，通过对高显著区域的编码重点优化、视频质量强化以及码率的科学调配，优先保证视觉重点区域画质与流畅度，从而提高带宽资源利用效率，使用有限的网络资源提供最优质的音视频服务体验。

Video ROI Encoding 同样是值得我们讨论的技术趋势。相较于 Video Saliency Encoding ，Video ROI Encoding 更多是基于主观判断结合机器学习实现的视频重点区域编码优化。例如上图中下展示的那样，画面的视觉重点一定是两位主人公的人脸，我们可通过优化人脸所在区域的编码来增强其质量与流畅度，但最后依旧需要保证视频整体码率的平衡，这就会造成不在视觉重点区域的元素例如人物背景等的质量会有所下降。编码器拥有多种增强质量方法供开发者选择，例如针对视觉重点区域，通过评估区域块的重要性调整编码器编码所选用的函数，需要保证质量的区域块使用高质量函数编码，需要妥协质量的区域块使用低质量函数编码，这是未来编码端打造音视频服务极致用户体验的发展方向。

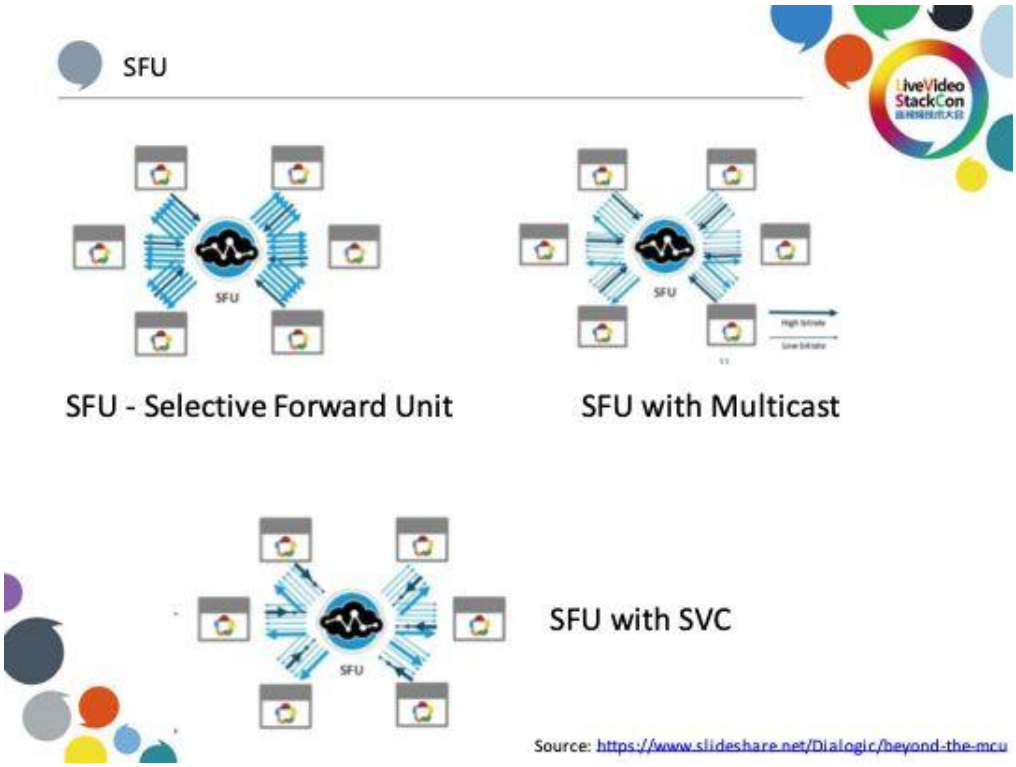
5. 媒体网络架构



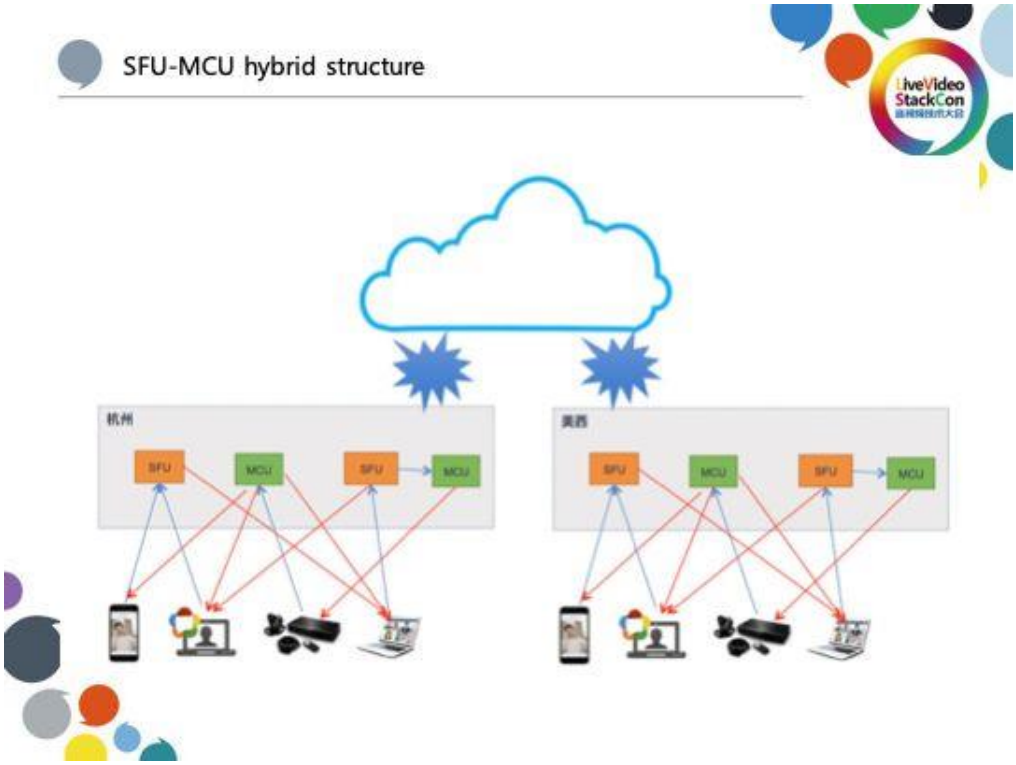
常用的媒体网络构架分为 MESH、MCU 与 SFU 三种。点对点的 MESH 在多人通讯场景中几乎没有被使用，MCU 的优势在于其在传输过程中可针对不同内容与场景进行转码、转流与信令转换、路由优化、媒体处理等操作。



除此之外，通过将流分为大流与小流并配置不同的码率，MCU 可为网络环境较好带宽质量优良的客户端提供高质量视频内容，为网络环境较差带宽质量一般的客户端提供一般质量的内容。这种针对不同用户的个性化传输服务能确保视频体验在不同网络环境下的一致性。MCU 的路由优化也可让数据包经过不同的路径传输，从而减少丢包现象的发生。

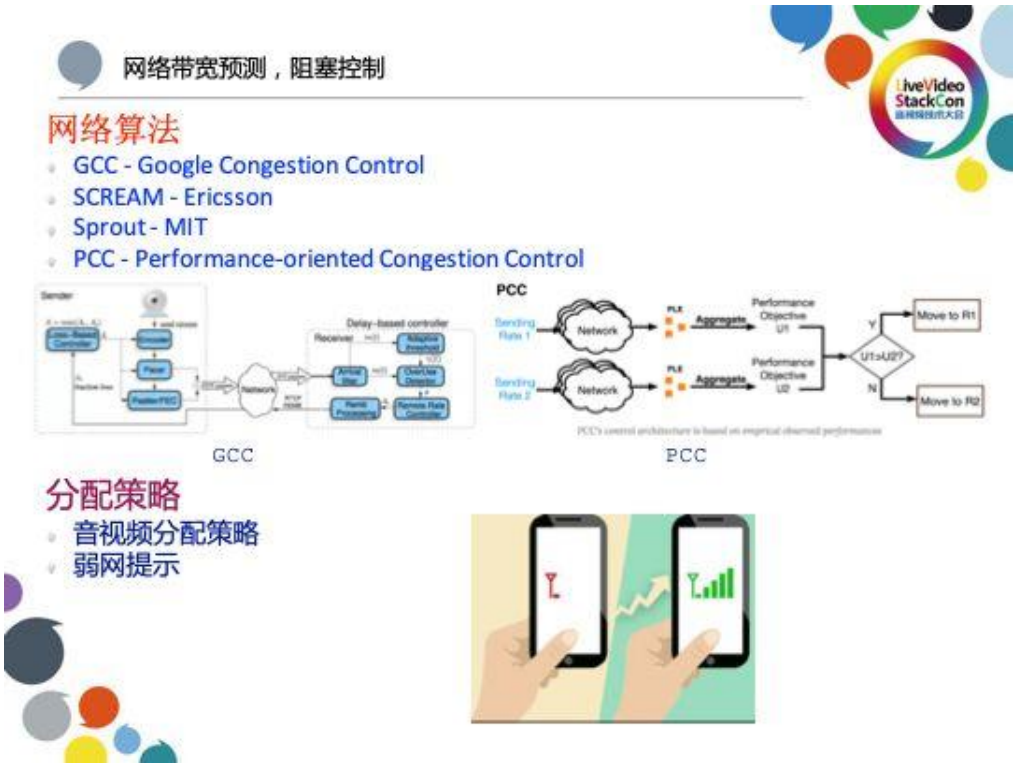


如果对分发速度有更高要求，SFU 是一个可选方案。但 SFU 的分法策略相对于 MCU 更复杂多样，需要特别关注。



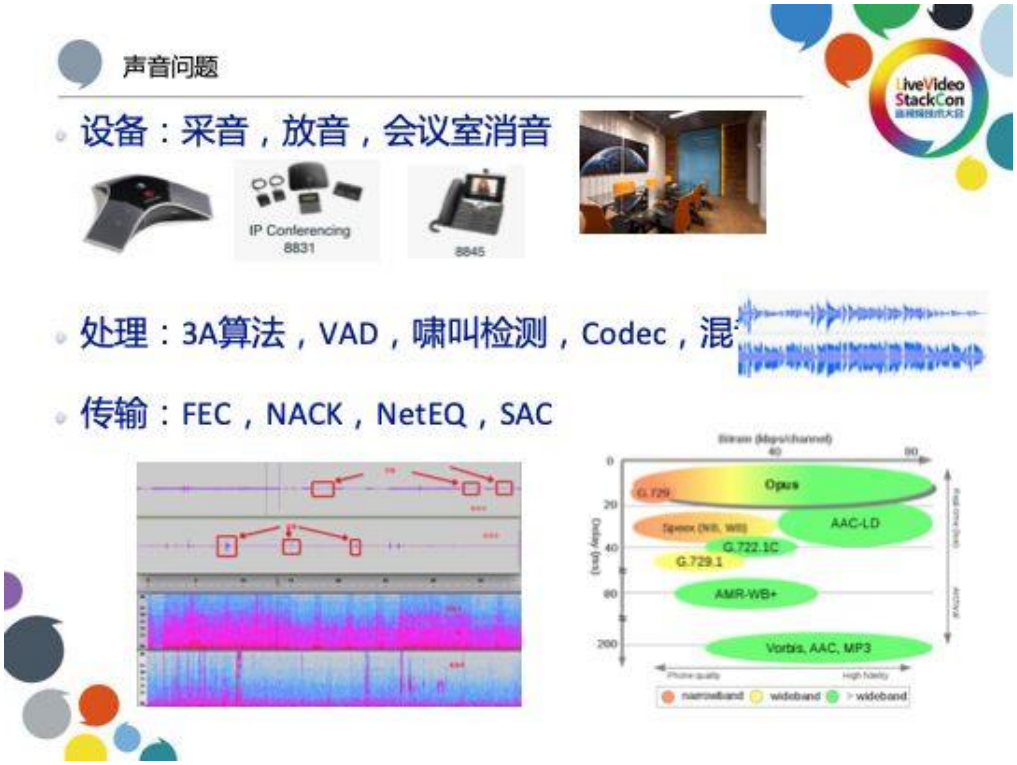
我们更推荐采用这种 SFU-MCU 混合网络架构，对无法分流仅支持一路流的端（多为商业端或对码流格式有特定要求）采用 MCU；而对服务器性能优异，但网络带宽或接收端质量十分糟糕的情形，我们推荐使用混合方案，具体还需要针对实际应用场景做出特别优化。

6. 网络带宽与拥塞控制

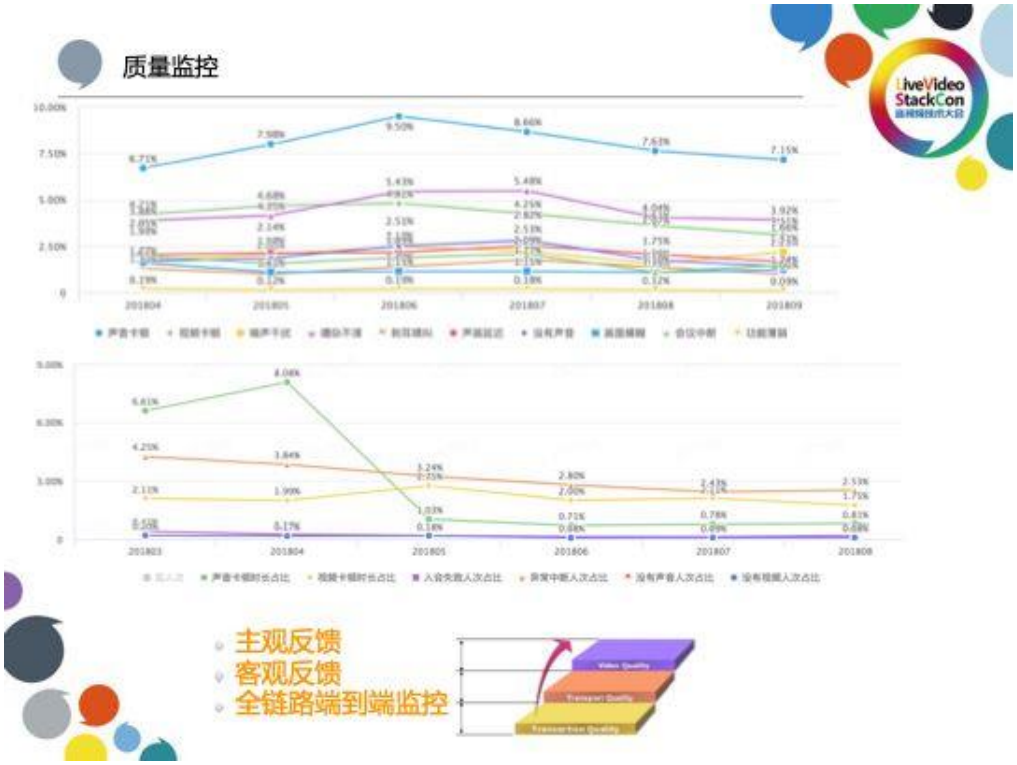


网络带宽预测与阻塞控制也是打造音视频服务极致用户体验不可或缺的一环。除了上图展示的 GCC、SCREAM、Sprout 等传统网络算法，在 WebRTC 中 Google 开发了一种被称为 PCC 的网络算法，也就是根据丢包率与抖动两项指标对传输带来的影响判断拥塞控制方案，上图中展示了其大致原理与流程。除包括音视频分配策略与弱网提示在内被老生常谈的分配策略话题之外，我想补充一点心理学对音视频服务体验的影响。通过实际数据我们发现了一个有趣的现象，在通讯或播放页面添加“HD”标识可让用户停留在产品上的时间增加 1%，不得不说心理暗示也是一项影响产品体验的重要元素。

7. 声音保真与质量监控



声音当然是我们关注的重点，尤其对于商业级别远程会议而言，音频收集与传输的质量优劣直接关系到所传达信息的准确性。这也就是为什么一些大型公司为保证视频会议的声音采集，愿意花重金改造整个会议室的声学结构。这里我想强调的是，音视频服务的良好体验无法完全依赖算法优化完美解决，达成最为理想的极致体验需要设备、软件、算法、工程、网络、心理学等多方面的共同配合努力。



而基于全链路的主观反馈数据实现的端到端质量监控可以为每一项主观与客观指标提供必要的数据基础。

8. 指导性建议

我曾在 Facebook 工作的感受是 Facebook 利用数据指导开发工作。这需要足够数量的灰度测试与高效的 AB Testing 比较，通过监控工具对关键数据进行实时监控，基于数据随时对产品进行调整与优化；与此同时，根据数据工程师也能快速定位问题，根据用户反馈数据对产品进行升级，我想这会对我们提升音视频服务用户体验带来一定指导意义。

AI+多媒体，不止于技术升级

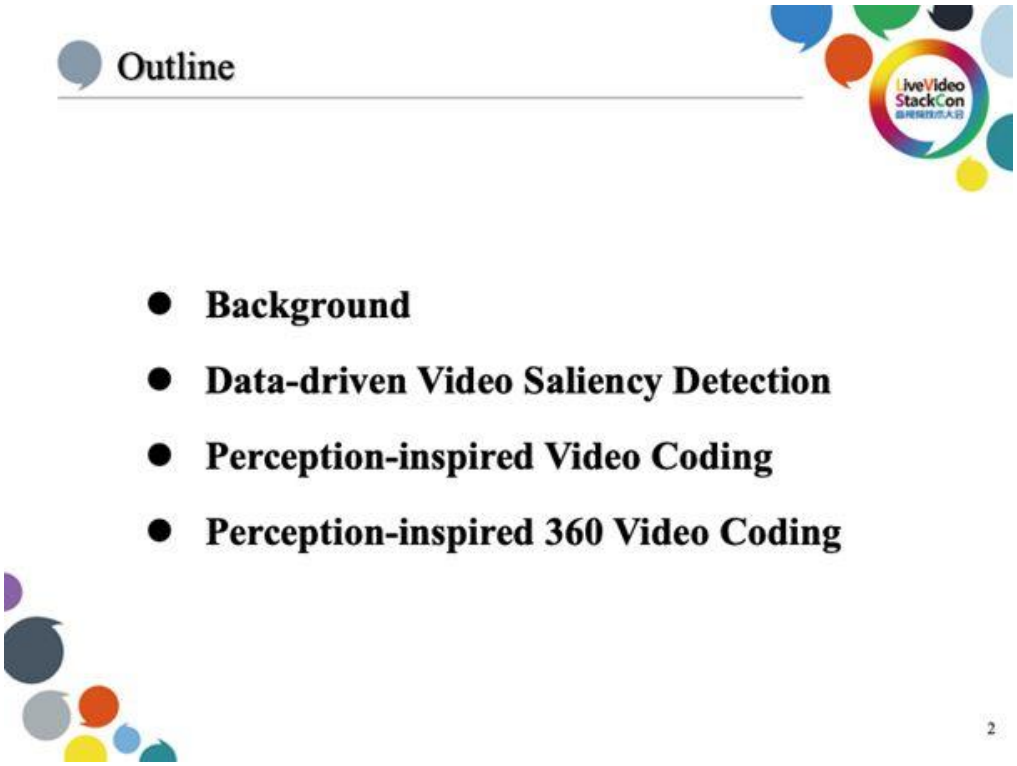
编码压缩新思路:面向 QoE 的感知视频编码

面向用户体验的感知视频编码即通过机器学习检测用户感兴趣的视觉感知区域，并重新分配以更多的码率与复杂度。本文来自北京航空航天大学副教授、博士生导师 徐迈在 LiveVideoStack 线上交流分享，并由 LiveVideoStack 整理而成。

徐迈　LiveVideoStack 线上分享　2018-12-17

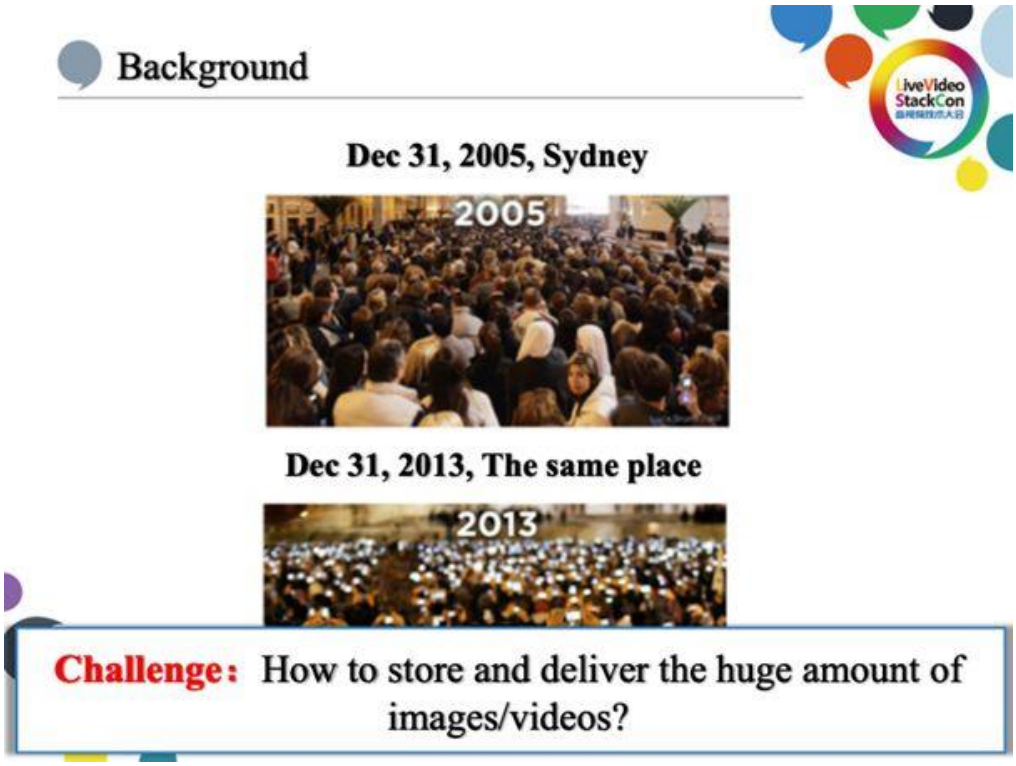
大家好，我是来自北京航空航天大学的徐迈。本次我将为大家分享面向 QoE 的感知视频编码。我们希望通过基于感知模型的视频压缩编码技术，优化产品用户体验。

本次的分享将围绕以下四个方面展开：



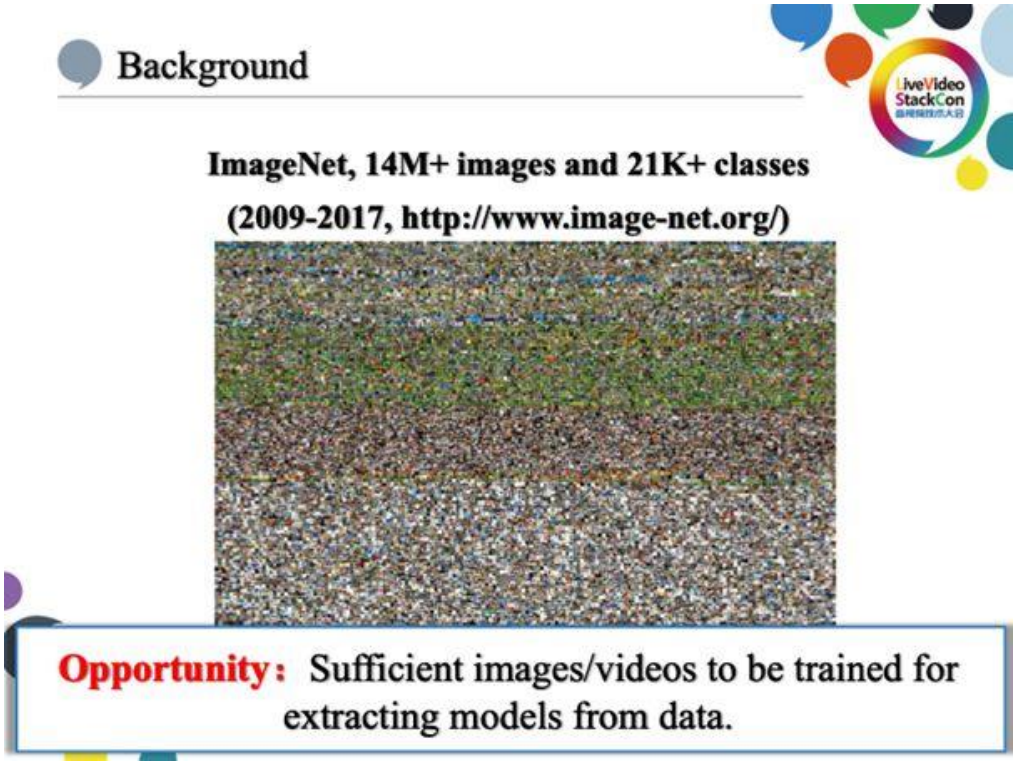
1. 技术发展背景

用户的需求升级推动技术的不断前行，技术的进步也在不断优化产品用户体验。

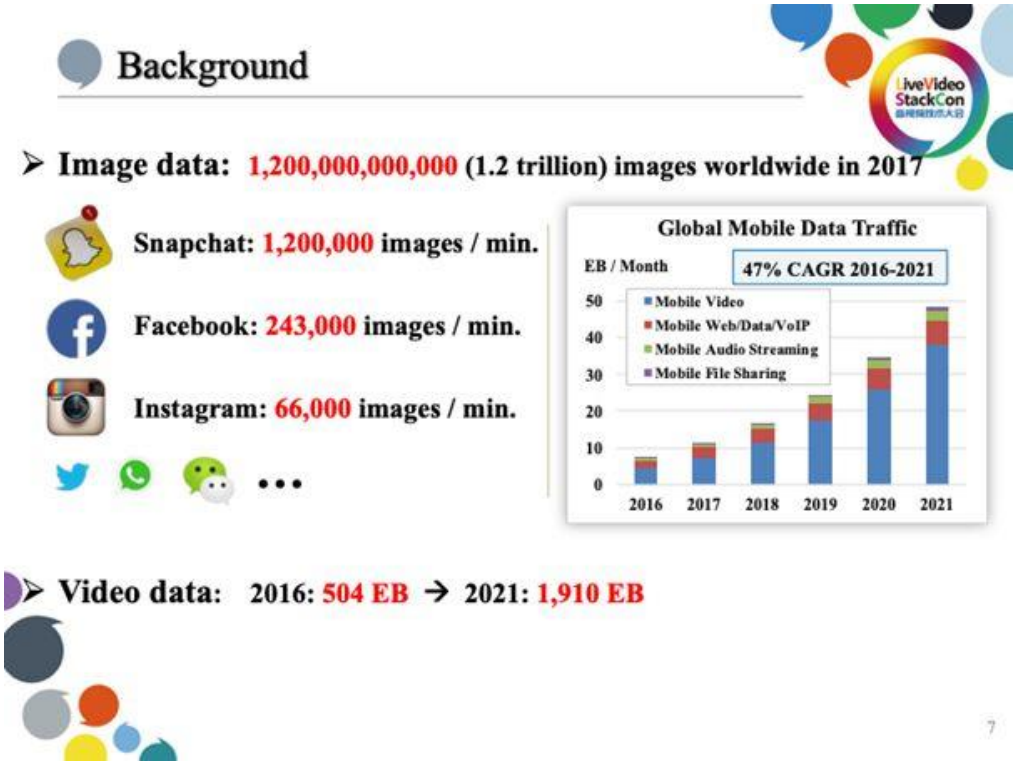


十几年前，人们聚在一起庆祝新年，借助电话或短信嘘寒问暖，一起在温馨氛围中为新年到来而欢呼；而现在，人们相聚在一起庆祝新年，更多的是拿出智能手机、平板电脑等移动智能终端记录这样一个美好的瞬间。十几年的发展带来的首要影响就是数据量的激增，如何稳定高效传输大量用户随时随地采集到的音视频数据成为我们亟待解决的问题。

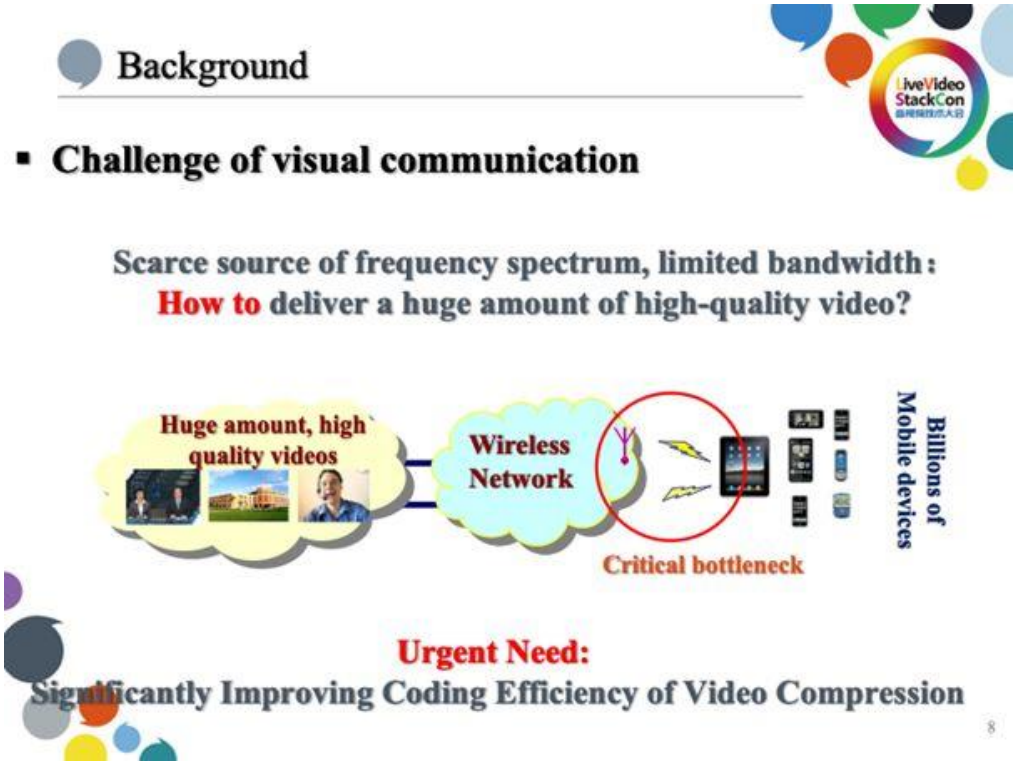
数据量的激增虽然为网络传输带来了巨大挑战，但也为人工智能等高新技术带来了需求与发展契机。



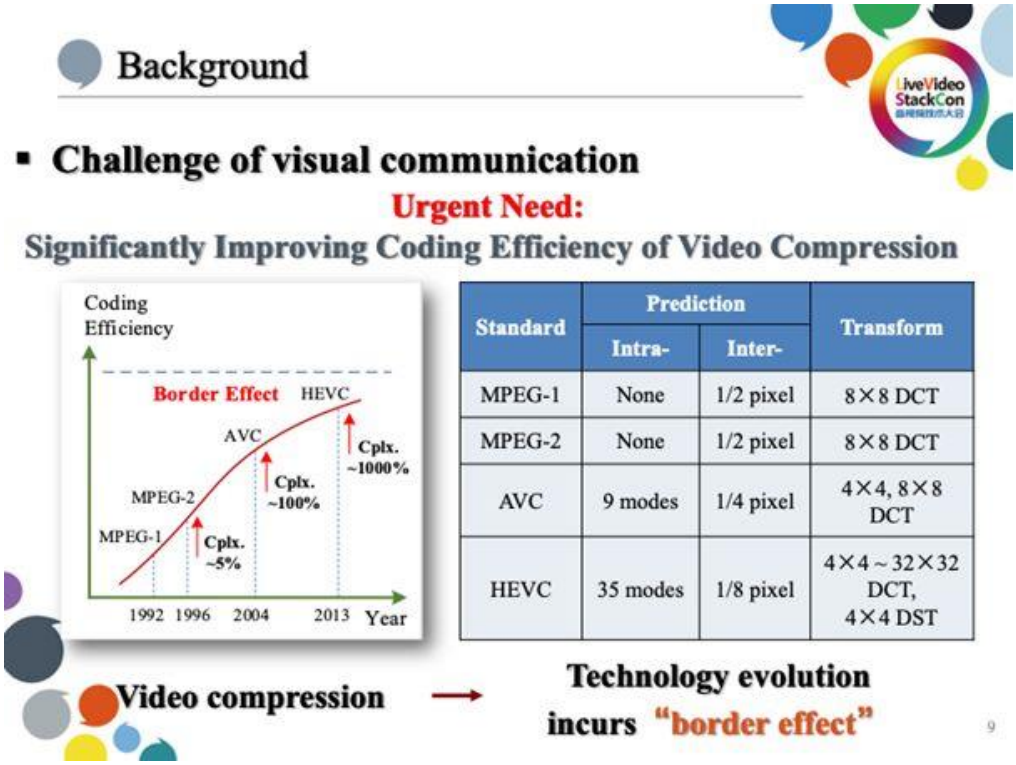
于是在 2009 年，斯坦福的李飞飞等科学家一起构筑了用于测试视觉识别性能的 ImageNet 数据库。初期 ImageNet 包含了四千多个类别的四百多万张图像，而到了 2017 年底其已包含两万多个类别的 1400～1500 万张图像。2009 年 ImageNet 数据库的建立与当时互联网上出现的大量图像数据密切相关，直到 2018 年的 ImageNet 中已包括了 5400 万余张图片，不得不说这加速了机器学习在视觉识别领域的运用进程。视觉识别离不开通过大量的图片训练增强其对相似视觉元素特性的规律总结能力，我们可以将这一思路运用在编码压缩领域，通过大量的视频压缩训练使机器学习掌握洞悉视频压缩结构规律的能力，极大程度优化视频编码性能，提升用户体验。



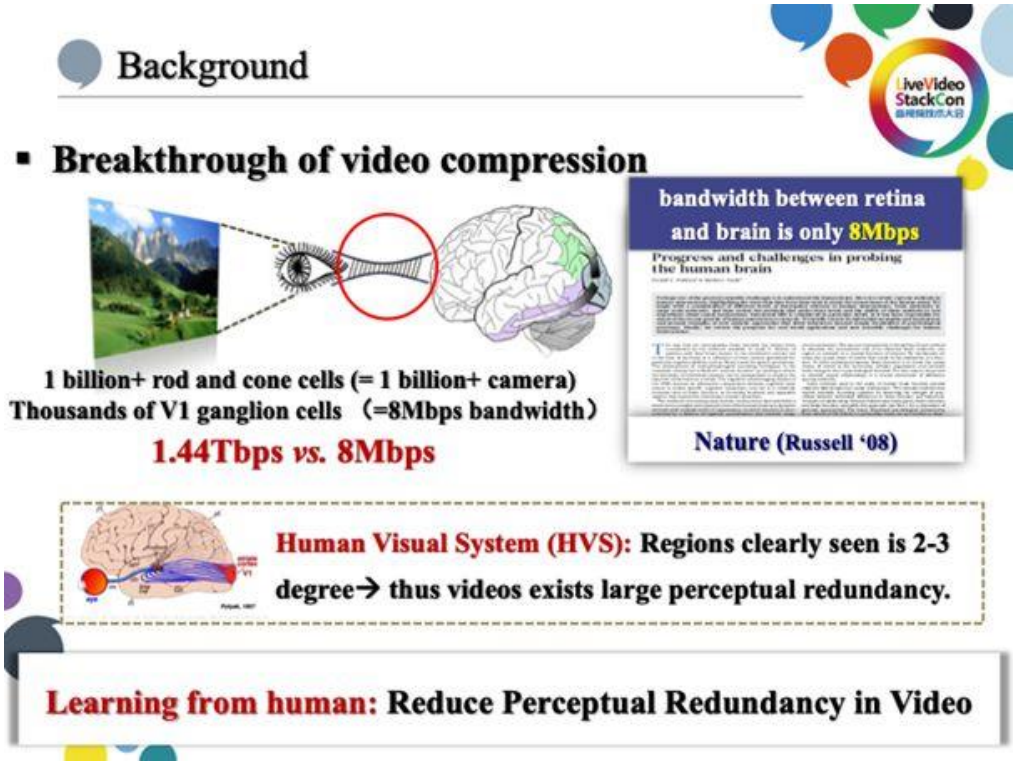
根据统计，过去的 2017 年全球互联网上有 1.2 万亿幅图像产生，每一分钟就有几百万张图片被上传至包括 Facebook、Snapchat 在内的各大互联网平台；而视频的数据量则更为庞大，预计全球互联网视频数据总量将在 2021 年达到近 2000EB。



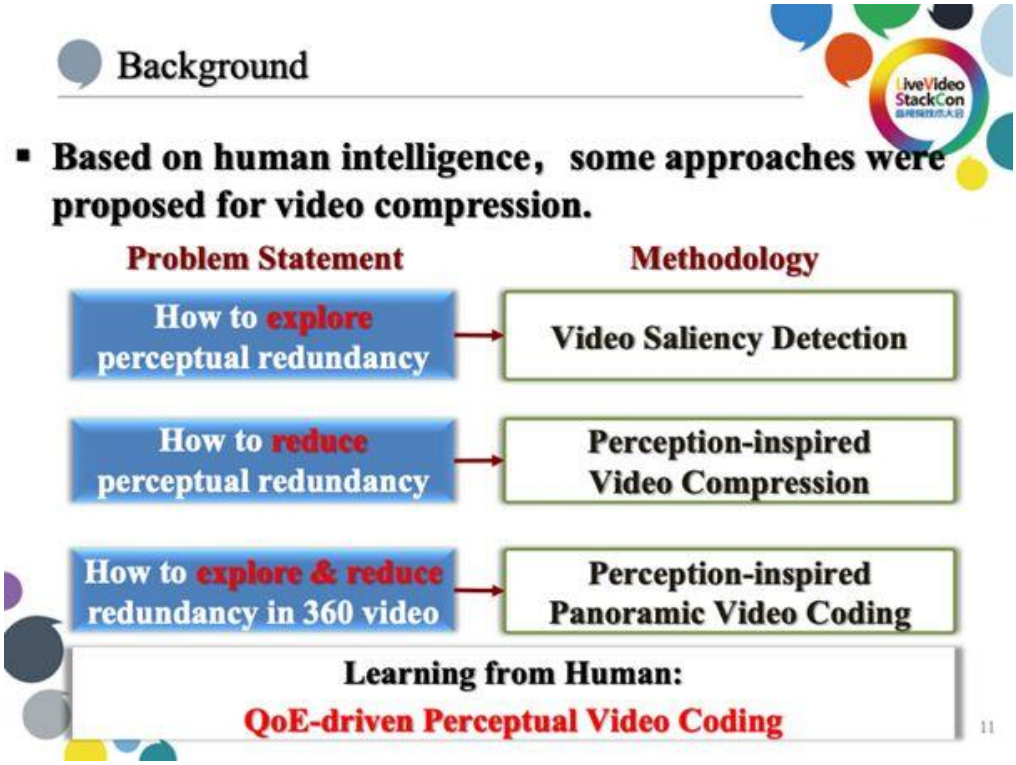
如此庞大的数据量无疑会为图像识别与通讯网络的发展带来巨大挑战，受限于通讯资源，我们的实际传输带宽资源远没有视频数据量所要求的那么充裕；若想借助有限的带宽资源快速稳定传输大量的图像与视频数据，则离不开高效的视频编码解决方案。



讲到这里，我想我们需要回顾一下视频编码的发展历程。早期视频编码变革较快：四年间 MPEG-1 发展到 MPEG-2，所带来的码率节省约为 50%，编码效率翻倍，复杂度增长为 5%左右；而 H.264（AVC）发展到 H.265（HEVC），虽然编码效率仍有部分提高，但其背后复杂度增长却十分显著达到了二至十倍，实测可能更高。复杂度增长是现在编码发展的一个明显趋势，而从右侧数据中我们可以看出，随着编码标准的演进，编码增益的成长也十分显著：从初期 AVC 的 9 个 Modes 发展到 HEVC 的 35 个 Modes；除此之外，早期的 MPEG-1 与 MPEG-2 是基于 8x8 的 DCT 变换，而发展到了 AVC 则实现了 4x4 与 8x8 的 DCT，HEVC 更是实现了 4x4 ~ 32x32 DCT 与 4x4 DST；至于 H.266 还引入了预测模式，60 多种预测模式使得相关参数复杂程度进一步提升。从中我们不难看出，视频压缩一直基于信号处理技术并不断进行演进，而信号处理技术发展到现在已经很难再产生颠覆性革新。随着技术的发展，边际效应愈发明显，技术突破愈发困难，因此我们迫切需要一种编码压缩的新思路。

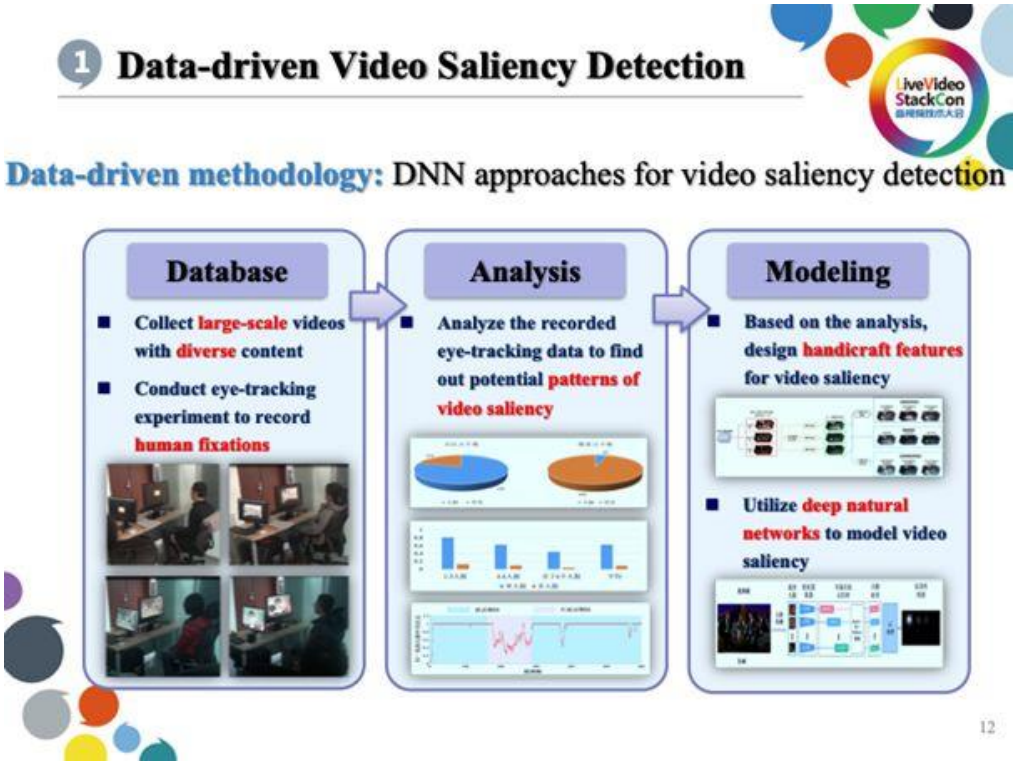


这种新思路就是结合用户感知对编码过程进行优化。用户感知与 QoE 紧密相关，人类视网膜大约拥有十亿视觉细胞，这使得人眼成为一台十亿像素的高清相机；大脑皮层会识别分析处理这些视觉信号，但连接人眼与大脑皮层的神经细胞仅有一万个左右，这就像一个资源十分有限的窄带带宽，那么人眼是如何利用这样一个窄带带宽传输像素高达十亿的高清视觉信号呢？这就是人作为高级动物的智慧所在：研究人的视觉感知模型我们可以发现，在人眼可感知的视角内，真正会引起大脑皮层明显兴奋的区域仅为 2～3 度；换句话说，人的视觉会将感知重点放在感兴趣的目标区域。由此启发，我们可通过降低感知冗余进一步提升视频压缩效率。

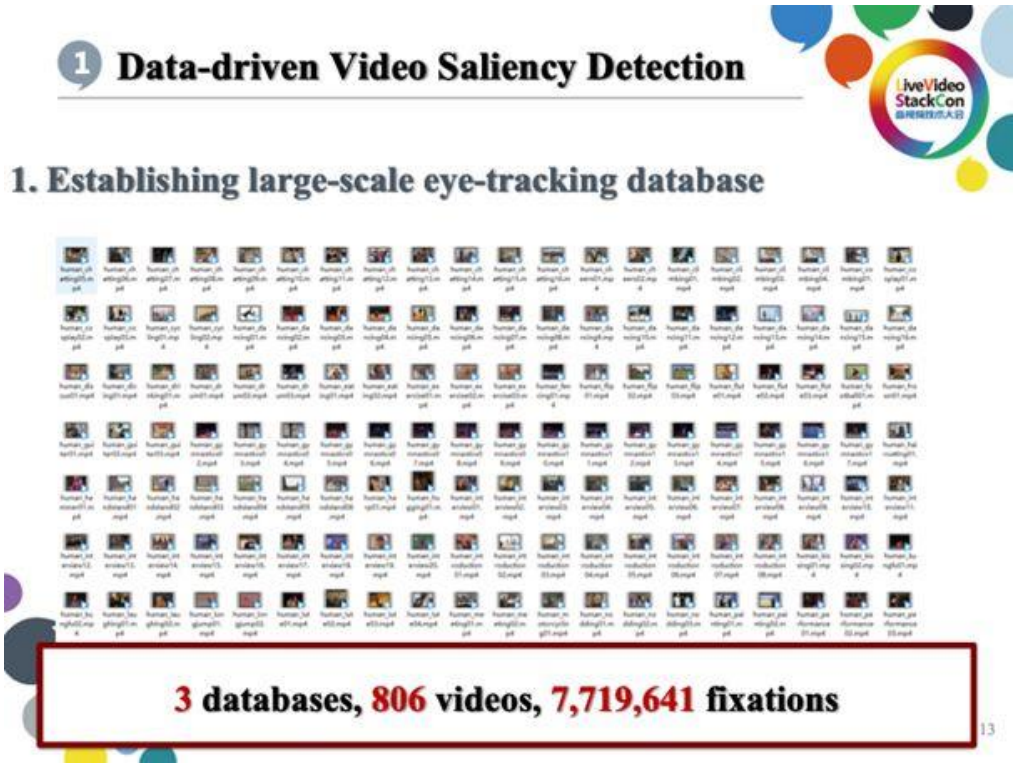


基于人类智慧我们提出了感知视频压缩并努力降低感知冗余。首先我们明确了如何察觉到视频感知冗余的出现，解决方案是借助机器学习与计算机视觉检测出视频画面里用户会重点关注的部分；当监测到感知冗余出现之后，我们尝试减少感知冗余与其影响，解决方案是重新分配资源，将更多码率与复杂度分配到用户感兴趣的视觉感知区域；接下来我们尝试将视频感知冗余的优化运用在全景视频之上，使得视频压缩更加契合人类的视觉习惯，也就是我们所说的面向 QoE 的感知视频编码。

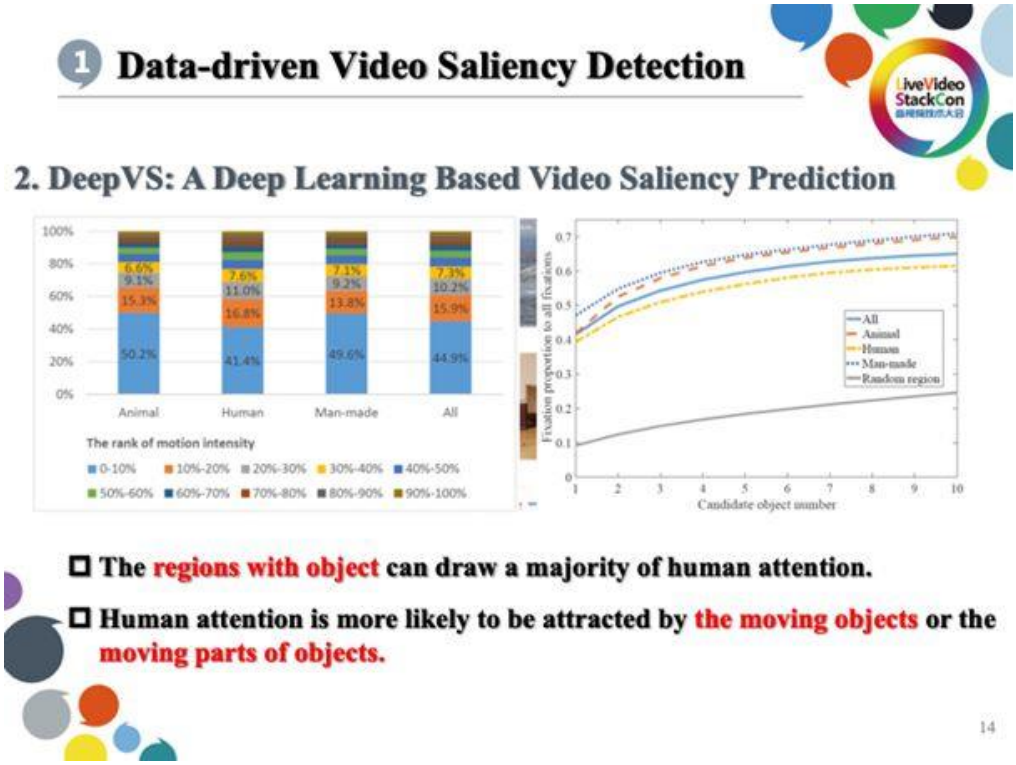
2. 基于数据驱动实现视频显著性检测



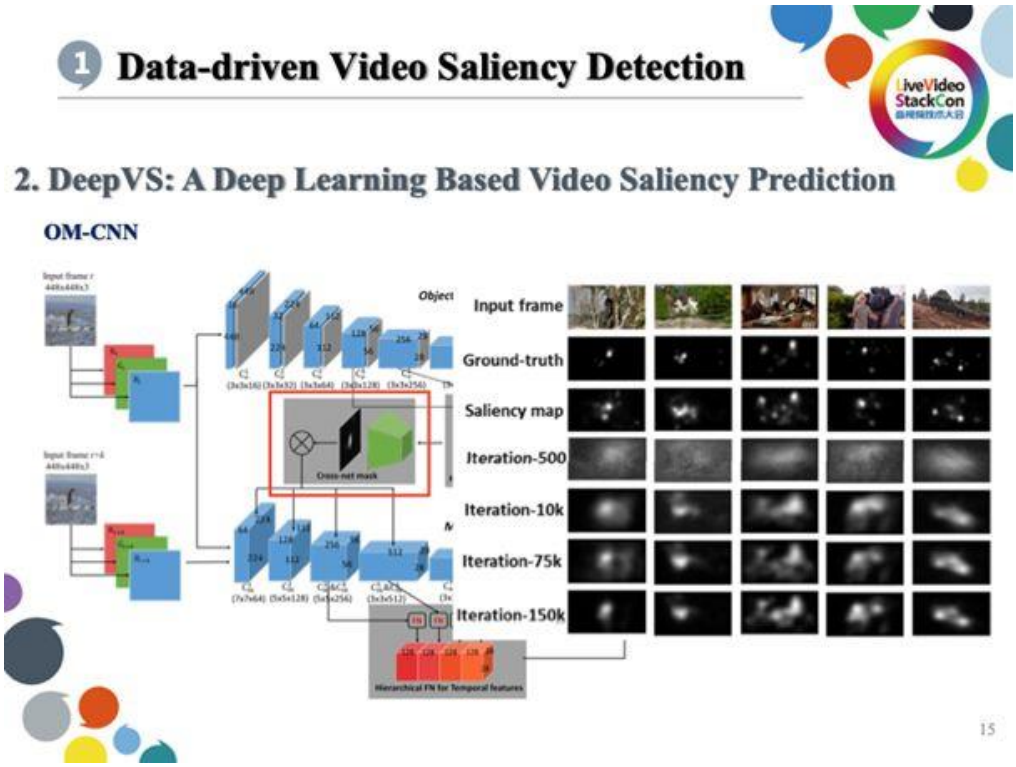
视频显著性检测是指通过计算机视觉技术推断用户观看视频时视觉感知重点集中的区域，主要借助基于机器学习的 CV。其大致步骤为建库、分析与建模；我们的建库工作开始于 2015 年，初期主要依靠眼动仪检测志愿者观看视频时视觉关注重点的变化从而生成热点图数据库；建库之后我们对热点数据进行了特征分析，随后依据特征分析结果建立深度学习模型从而初步实现了视频显著性检测。



现在我们针对视频已建立三个包括八百多个视频的数据库，共积累七百多万个关注点。

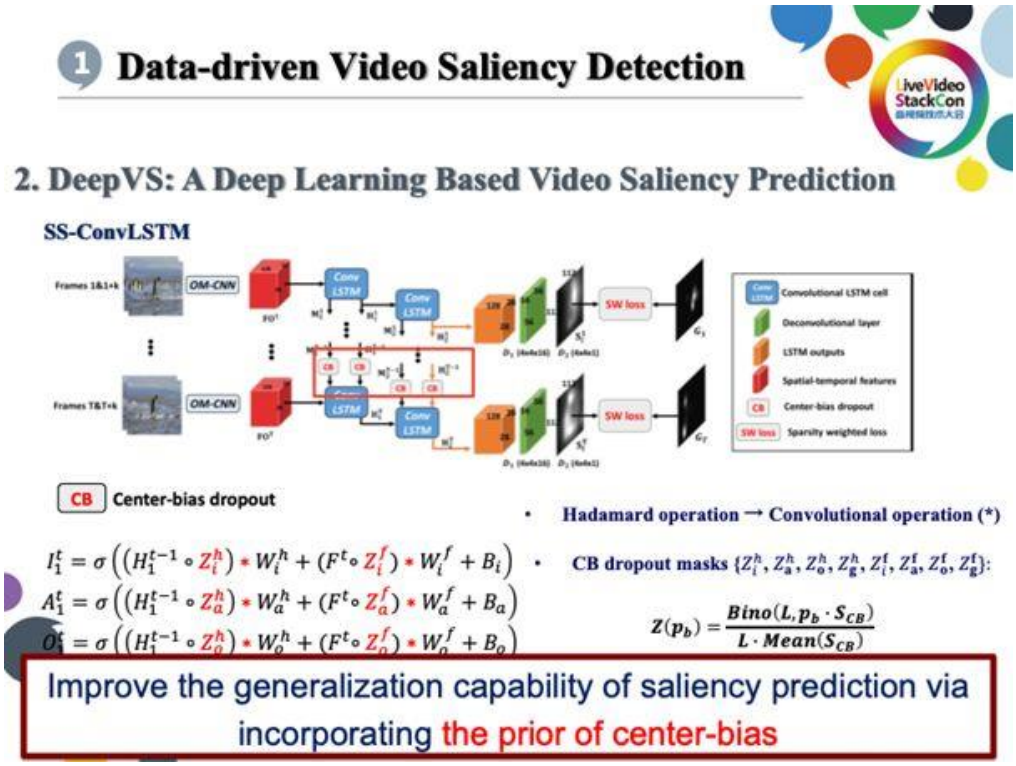


在完成数据库的建立之后，我们对数据库进行分析，并对显著性进行可视化。上图展示的就是多位志愿者观看视频时视觉重点关注的位置变化数据，借助此数据可生成视觉热点图，偏向红色的区域代表视觉关注重点而偏向蓝色的区域则代表视觉关注程度较少，没有颜色则代表不被关注。通过上图展示的实验数据我们可以发现，人在观看视频时视觉感知系统不会过多关注视频画面的背景部分而会重点关注目标元素尤其是运动的人与物。

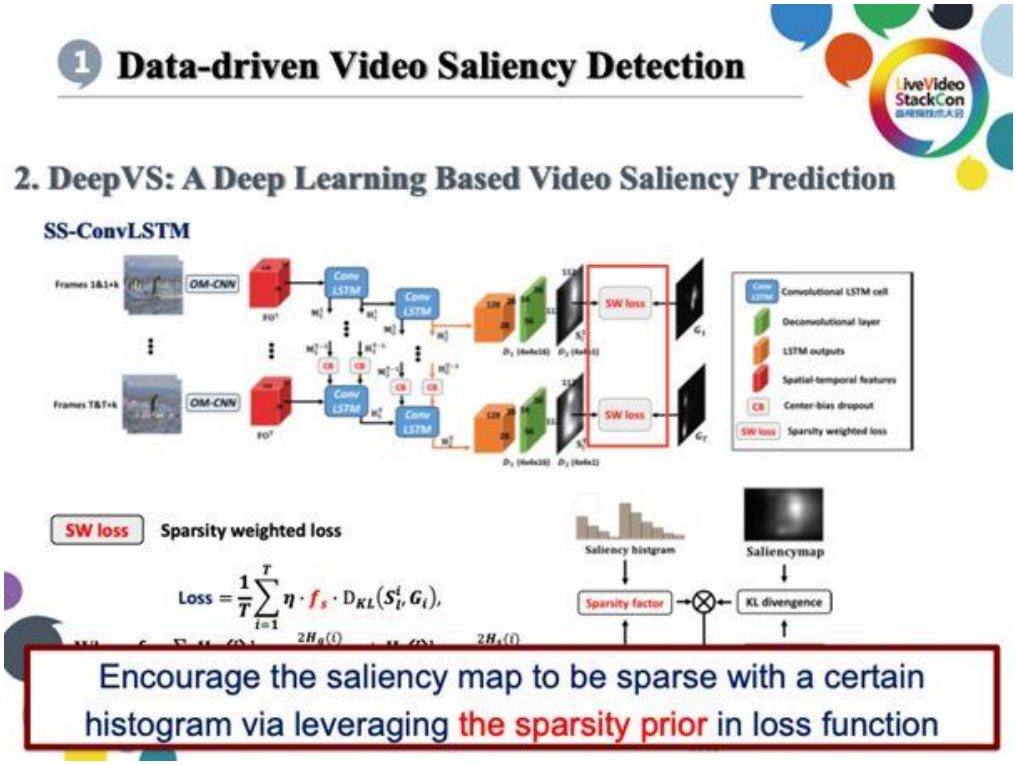


为了找到能够支撑上述结论的有力依据，我们尝试通过深度学习模型预测视觉关注重点。

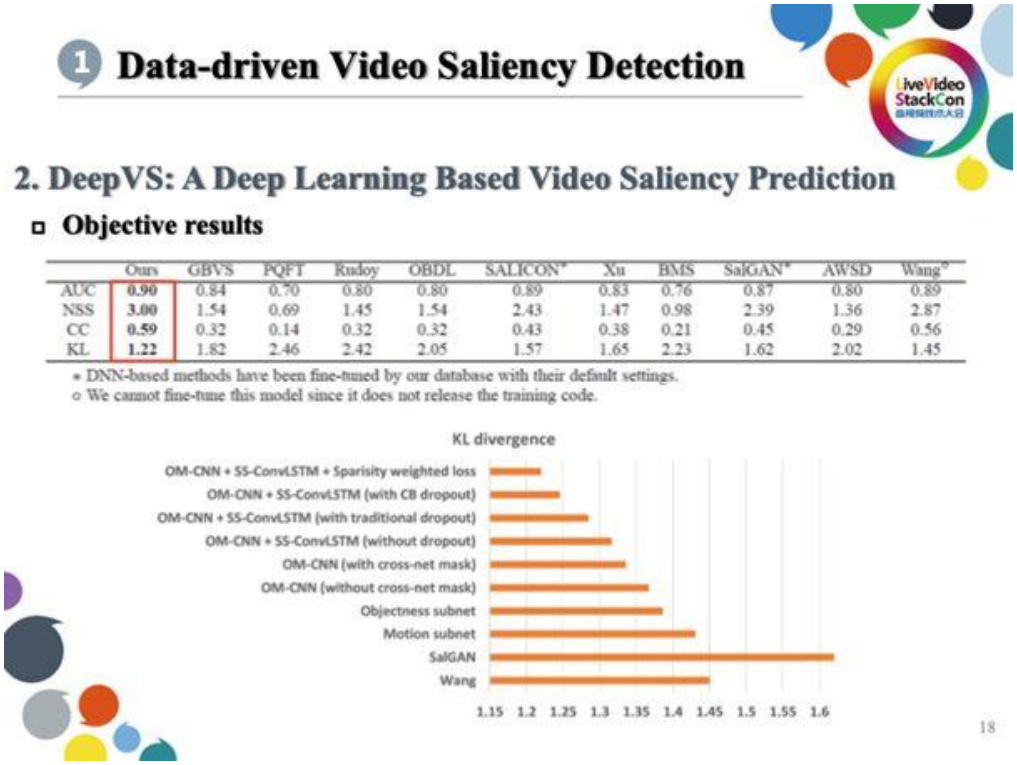
首先我们基于对目标物体的检测来构建网络，接下来针对容易吸引视觉关注的运动物体我们构建了 Motion Subnet 用以优化对运动物体的检测识别；在分别准确识别了物体与其运动状态之后，我们借助借助 Cross-net mask 实现综合分析二者数据得到运动物体的动态参数，并通过经历多次迭代的深度学习得出具备预测视觉关注重点的深度学习网络。初期此网络检测精度依旧有限只能检测识别整个运动的物体，但经过多达 150K 代的迭代以后，现已可实现对物体局部的某一运动部分（如汽车正在旋转的车轮）进行准确识别检测。



完成了 CNN 网络的构建，随后我们又将 CNN 网络特征输入进 SS-ConvLSTM 从而使得系统能够准确推测帧与帧间视觉关注重点的相关性。为了实现这样的功能，我们提出了 Center-bias Dropout ，其依据在于人眼更容易将画面的中心区域作为视觉关注重点，除了人类遗传导致人的视觉系统会将视觉中央附近的物体作为感知重点之外，主要的原因还有摄影师拍摄画面时自然会把拍摄主体放在画面中央。这就使得即使系统不对视频进行内容分析，也能大致确定某视频用户的视觉关注重点会分布在画面中央附近。

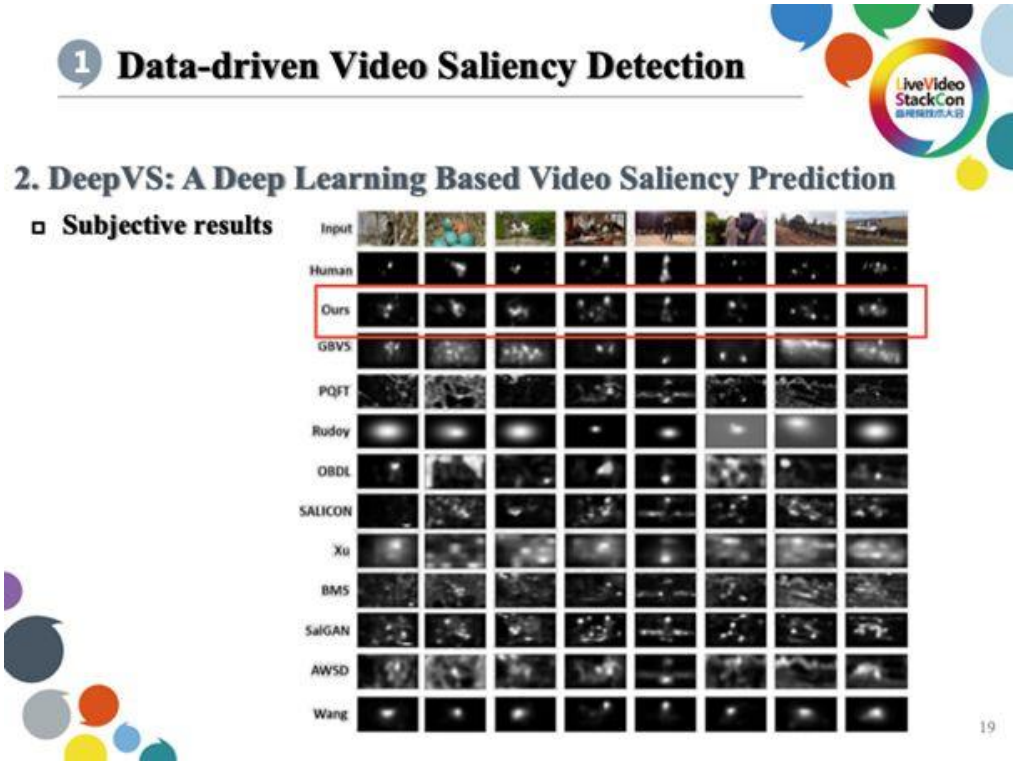


除此之外我们也提出了 Sparsity-weighted loss（稀疏加权损失函数），主要用于对目标物体与关注重点区域的定量分析，下图展示的是我们的定量结果，可以看到性能好于其他多种算法。



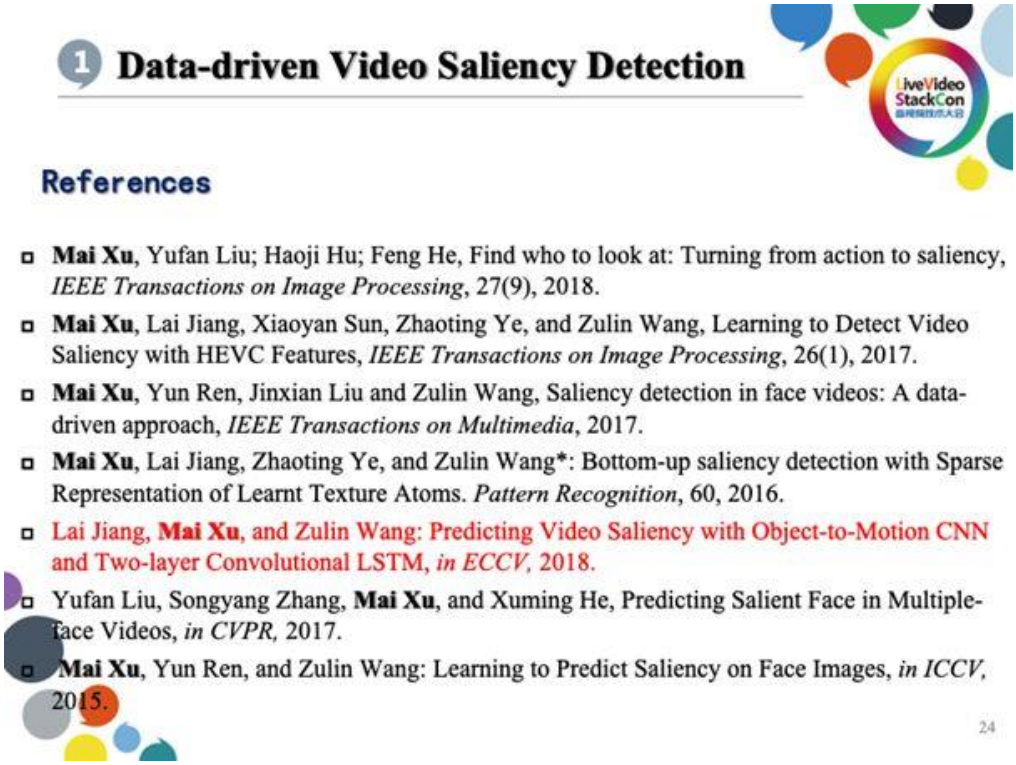
	Ours	GBVS	PQFT	Rudoy	OBDL	SALICON*	Xu	BMS	SalGAN*	AWSD	Wang°
AUC	**0.90**	0.84	0.70	0.80	0.80	0.89	0.83	0.76	0.87	0.80	0.89
NSS	**3.00**	1.54	0.69	1.45	1.54	2.43	1.47	0.98	2.39	1.36	2.87
CC	**0.59**	0.32	0.14	0.32	0.32	0.43	0.38	0.21	0.45	0.29	0.56
KL	**1.22**	1.82	2.46	2.42	2.05	1.57	1.65	2.23	1.62	2.02	1.45

* DNN-based methods have been fine-tuned by our database with their default settings.
° We cannot fine-tune this model since it does not release the training code.



通过测试上述的分析流程我们得到了上图展示的测试数据：以人脸检测为例，“Human”行代表人类视觉实际的重点关注区域，“Ours”行代表计算机预测的视觉显著性结果，可以看到与真实区域基本重叠，而相对于下面几行代表的传统方法在准确度上优势明显。

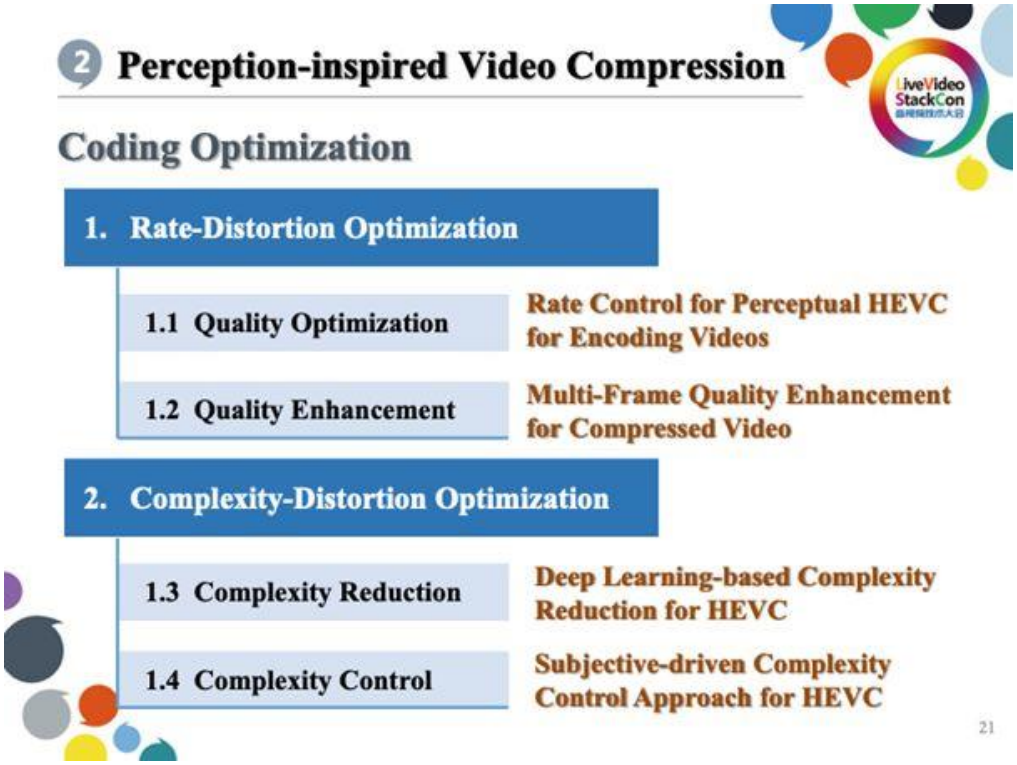
下图展示的是我们在此研究上主要发表的论文。



3. 面向一般视频的感知视频压缩编码

接下来将重点介绍有关感知视频编码在一般视频场景中的应用。

3.1 编码优化



编码优化是必不可少的优化思路。

速率失真优化

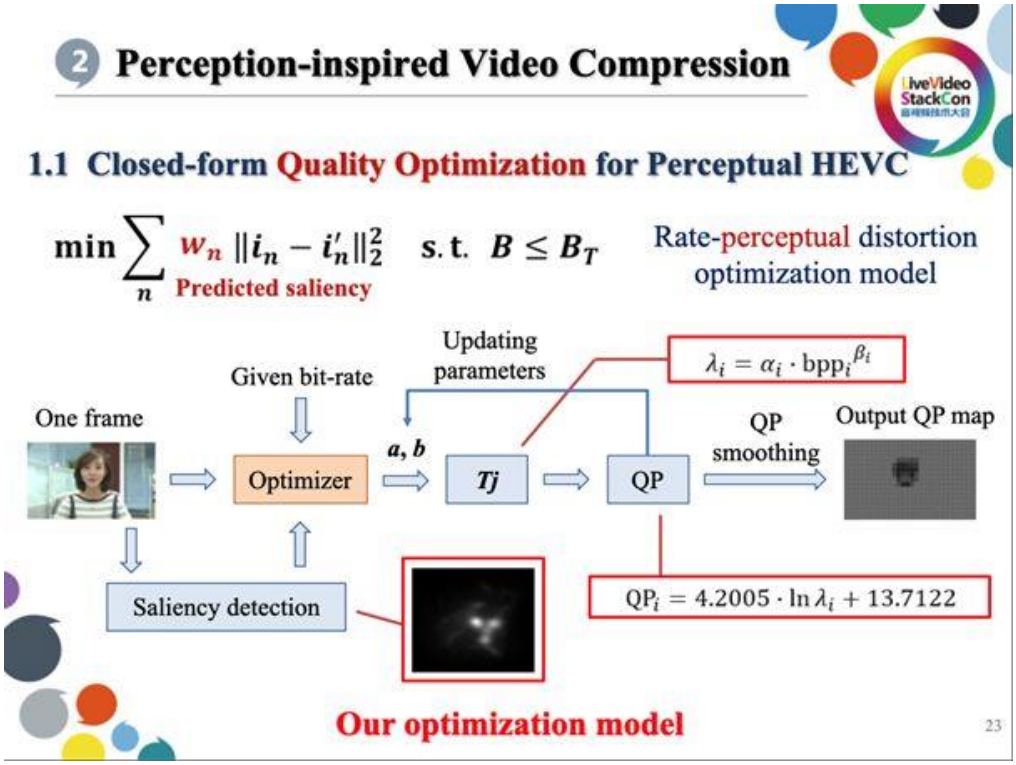
首先我们进行了速率失真优化，也就是在控制一定码率的同时尽可能降低失真或在控制失真的同时尽可能降低码率。速率失真优化主要包括借助基于感知模型的码率控制实现指标优化与基于机器学习预测压缩失真的位置与特性实现质量增强。

复杂度失真优化

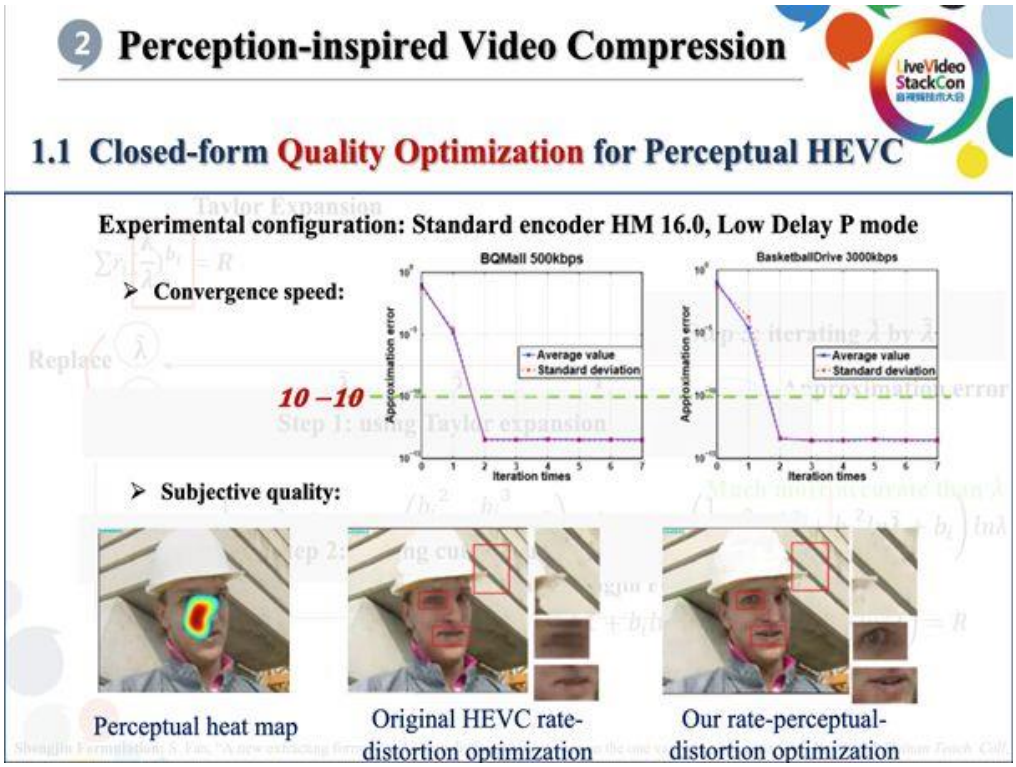
复杂度失真优化主要包括以下两方面工作：首先是在保证质量的同时尽可能降低复杂度，其次是将复杂度控制在目标值之下并确保失真尽可能少。

进一步落实优化算法，我们期待通过算法尽可能减少失真与其带来的影响。

1）针对 HEVC 感知视频编码的闭型质量优化

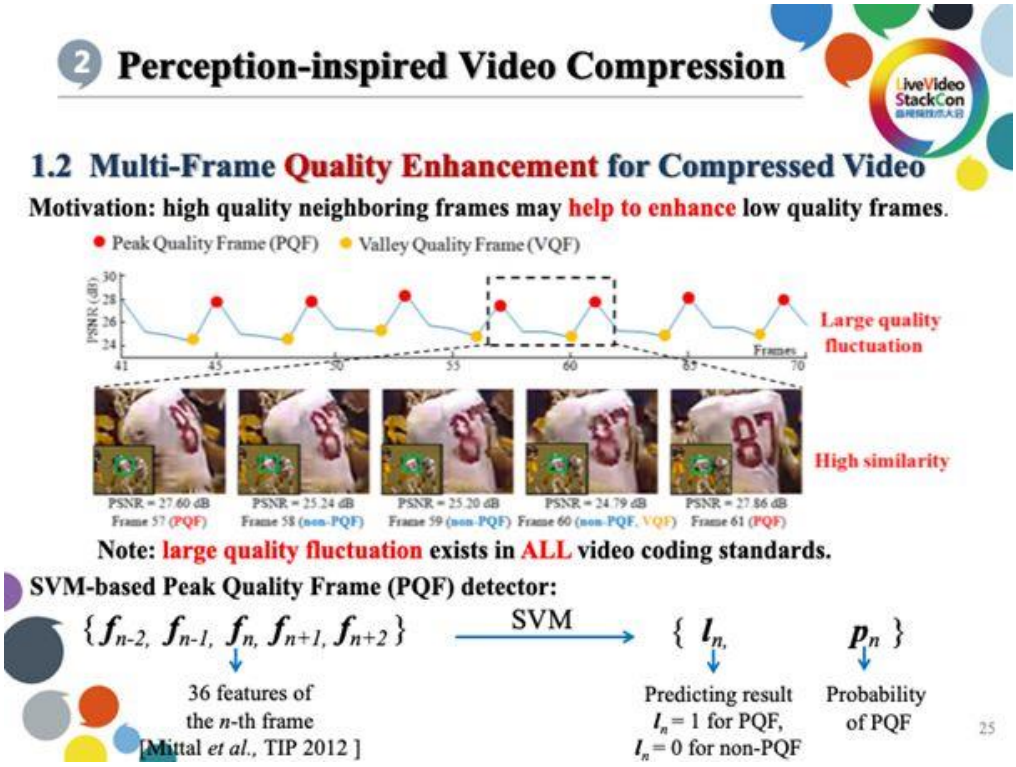


根据上图展示的公式我们可以看到算法的原理是尽可能（减少第 n 块的码率分配）并引入 Wn（Predicted Saliency）显著性预测，将其作为权重加入失真优化当中，显著性越高失真影响越大那么系统对其优化越明显。如果不引入 Wn 仅依赖（it 模型），经过多次迭代之后客观情况下可达到 5%的码率节省而在主观情况下基本可达到一半左右的码率节省。

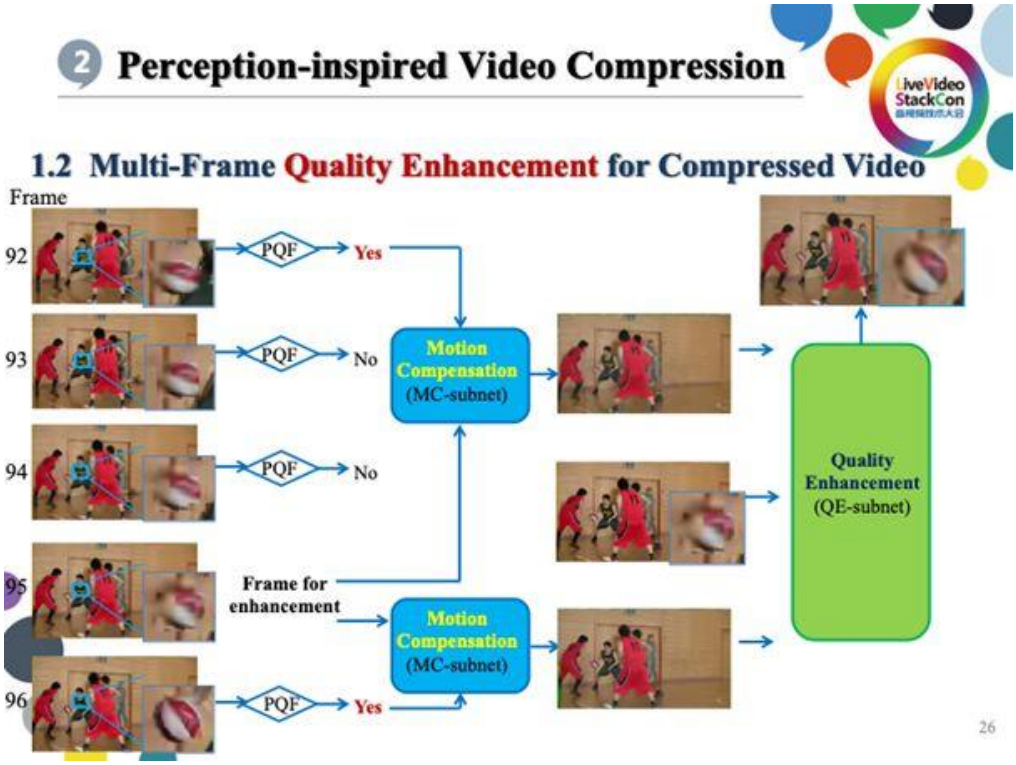


例如上图中下半部分展示的视频画面中左侧热力图表示观众对人脸的关注重点发布，但可以比较明显地看到人物眼睛与嘴巴的位置较为模糊，严重影响观看体验；此时我们就会着重对眼睛与嘴巴区域的画面进行优化，将更多码率调整至人脸部位尤其眼睛和嘴巴的位置从而尽可能降低眼睛与嘴巴所在画面位置的失真。这种码率调整会将更多码率资源用在人脸的眼睛与嘴巴等视觉重点区域，也会同时降低背景部分的码率，这种主观保证视觉重点区域编码质量的方案可使编码效率与性能提升一倍。

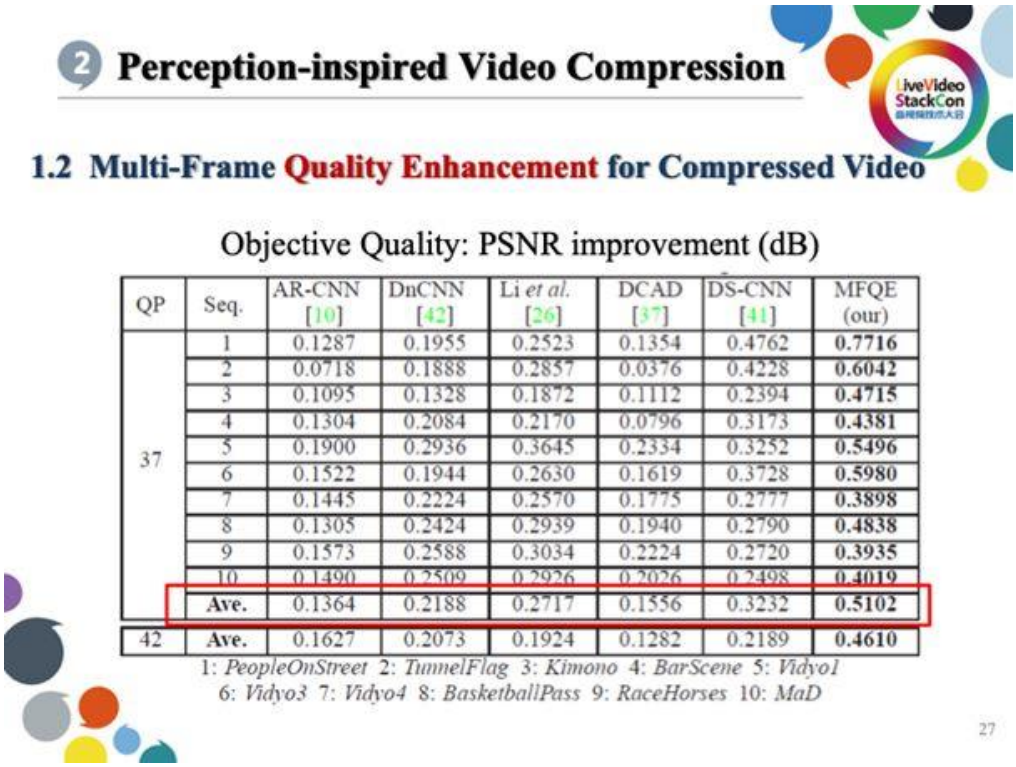
2）针对视频压缩的多帧质量提升



接下来我们需要将此技术用于提升多帧画面的编码质量。通过实验我们发现，几乎所有的编码标准都会使视频质量出现明显波动，尤其 HEVC 编码会导致帧与帧之间的编码质量差异过大。体现在画面上的结果可能是第 50、61 帧质量较高画面较为清晰而到了第 87 帧则被遭到丢弃使其难以恢复原有质量。此时我们需要依赖 87 帧前后质量较高帧的帮助，进一步还原提升 87 帧的质量，这就是多帧质量提升操作。进行多帧质量提升之前，我们需要借助 SVM 对每一帧进行质量评估并选取其中质量不一的帧构造其特征并分类：ln=1 代表高质量帧 ln=0 代表低质量帧。

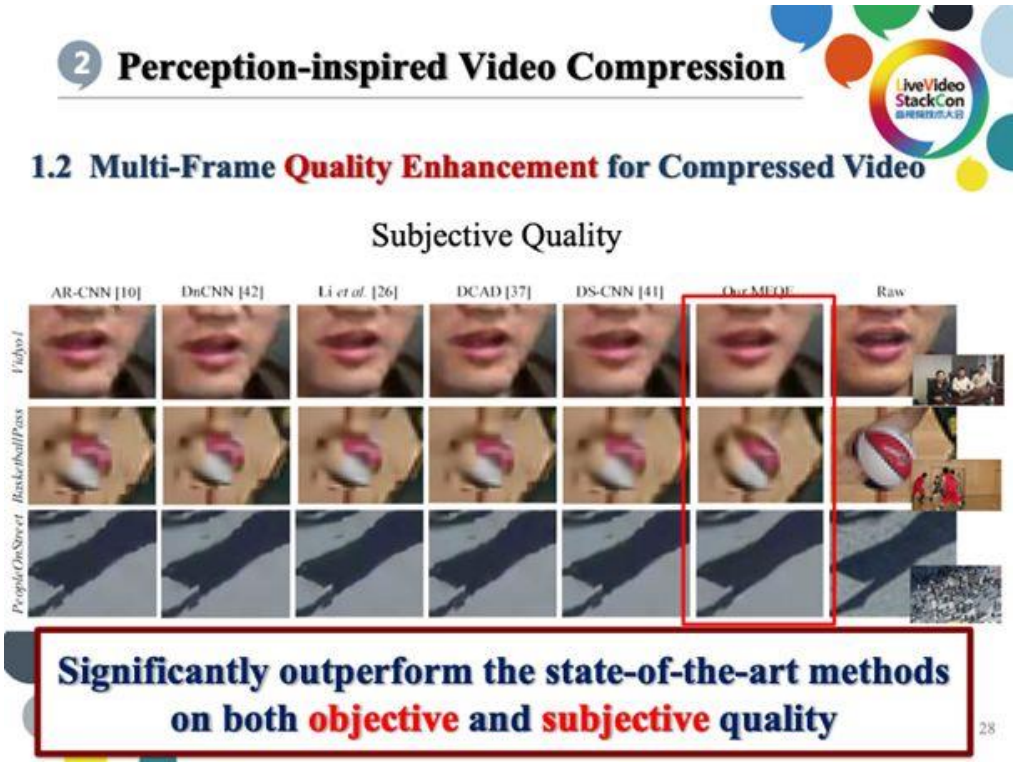


在确定高质量帧与低质量帧后，系统会在处理视频的同时将低质量目标帧与其前后相邻高质量帧一并输入神经网络，借助运动补全方法使得目标帧与其前后相邻两帧像素内容一致，编码近似，凭借机器学习得到的优化算法处理这三帧从而显著提高其压缩质量。如画面当中球所在的区域，画面质量较高。

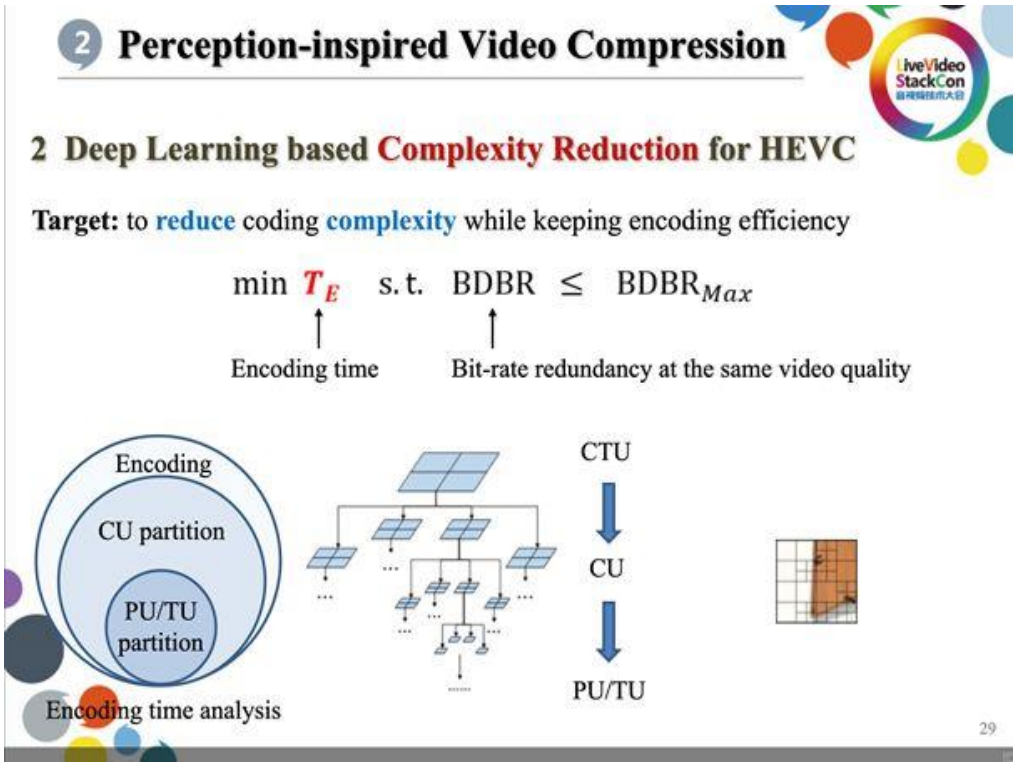


QP	Seq.	AR-CNN [10]	DnCNN [42]	Li *et al.* [26]	DCAD [37]	DS-CNN [41]	MFQE (our)
37	1	0.1287	0.1955	0.2523	0.1354	0.4762	**0.7716**
	2	0.0718	0.1888	0.2857	0.0376	0.4228	**0.6042**
	3	0.1095	0.1328	0.1872	0.1112	0.2394	**0.4715**
	4	0.1304	0.2084	0.2170	0.0796	0.3173	**0.4381**
	5	0.1900	0.2936	0.3645	0.2334	0.3252	**0.5496**
	6	0.1522	0.1944	0.2630	0.1619	0.3728	**0.5980**
	7	0.1445	0.2224	0.2570	0.1775	0.2777	**0.3898**
	8	0.1305	0.2424	0.2939	0.1940	0.2790	**0.4838**
	9	0.1573	0.2588	0.3034	0.2224	0.2720	**0.3935**
	10	0.1490	0.2509	0.2926	0.2026	0.2498	**0.4019**
	Ave.	0.1364	0.2188	0.2717	0.1556	0.3232	**0.5102**
42	**Ave.**	0.1627	0.2073	0.1924	0.1282	0.2189	**0.4610**

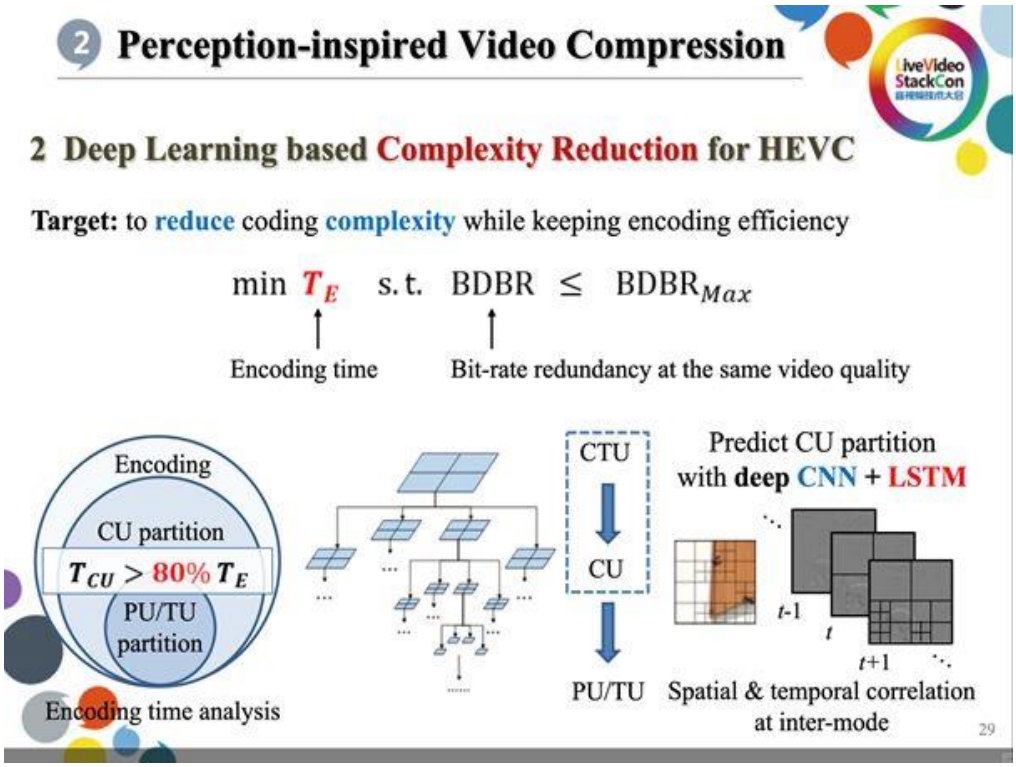
上图展示的是我们测试得到的主观指标，Ave 一行代表平均测试结果。可以看到我们 MFQE 的质量提升平均值达到了 0.5102dB，领先于前者多种算法。下图展示的是直观带来的优化体验，可以看到画面中篮球所在画面区域的质量重建提升十分明显。



3.2 复杂度降低

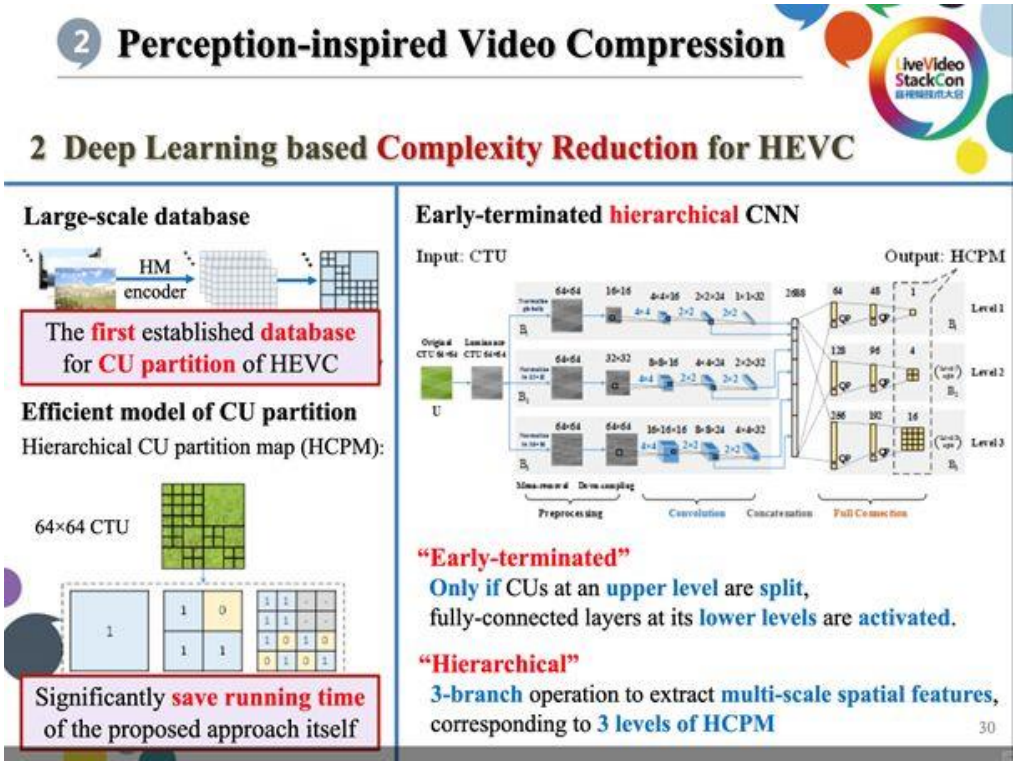


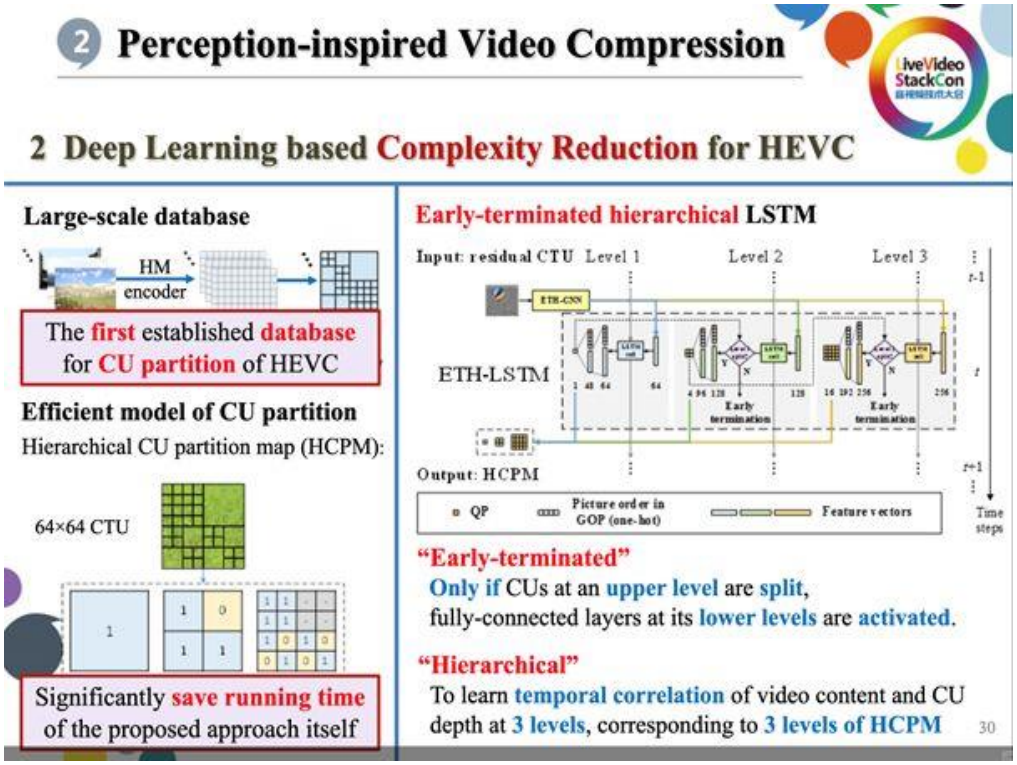
大家知道，H.265、H.266 的最大贡献是 CTU 分割，并且 H.266 引入了二叉树与三叉树，其背后复杂程度十分惊人。例如在 CU 分割时，由于无法准确预判分割块的编码性能优劣，只能通过预编码也就是对 RDO 进行全局搜索的形式确定分割块的编码性能；由此导致的编译次数增多会使其复杂度占总体高达 80%。



为了进一步降低复杂程度，我们可借助内容分析推测合适的 CU 分割策略。对于单帧可通过 CNN 挖掘其在空间上的相关性；

而对于多个相连帧可通过 CNN+LSTM 分析得出其在分割上的相关性与一致性。





之后我们的探索主要分为三步完成：首先是建立一个包括115个视频与两千多幅图像的数据库，接下来我们运用特征分析方法构建结构性输出，最后利用卷积神经网络预测块分割策略。

2 Perception-inspired Video Compression

2 Deep Learning based Complexity Reduction for HEVC

Performance Comparison on All 18 Test Videos and 200 Test Images

(Average on QPs: 22, 27, 32, 37)

Mode	Media type	Method	BDBR(%)	BDPSNR (dB)	ΔT (%)
Inter-mode	Standard test videos	Zhang's	1.799	-0.054	-44.97
		Correa's	5.051	-0.163	-44.57
		Mallikarachchi's	3.616	-0.108	-43.57
		Proposed	**1.495**	**-0.046**	**-54.20**
Intra-mode	Standard test videos	D. Liu's	8.559	-0.419	-46.22
		Z. Liu's	6.189	-0.316	-61.08
		Proposed	**2.247**	**-0.104**	**-61.85**
	Images from CPH test set	D. Liu's	4.945	-0.284	-56.14
		Z. Liu's	2.353	-0.155	-63.05
		Proposed	**1.386**	**-0.081**	**-67.18**

31

上图展示的是降低复杂度带来的性能提升。在帧间模式上我们的复杂度可降低约 54%，与此同时 BDBR 增加约 1.459%，BDPSNR 损失约 0.046%；对于帧内模式而言同时测试图像与视频，无论是视频还是图像其复杂度都会降低约 60%，与此同时 BDBR 增加约 2%。下图展示的是我们与此技术相关的部分技术专利。

2 Perception-inspired Video Compression

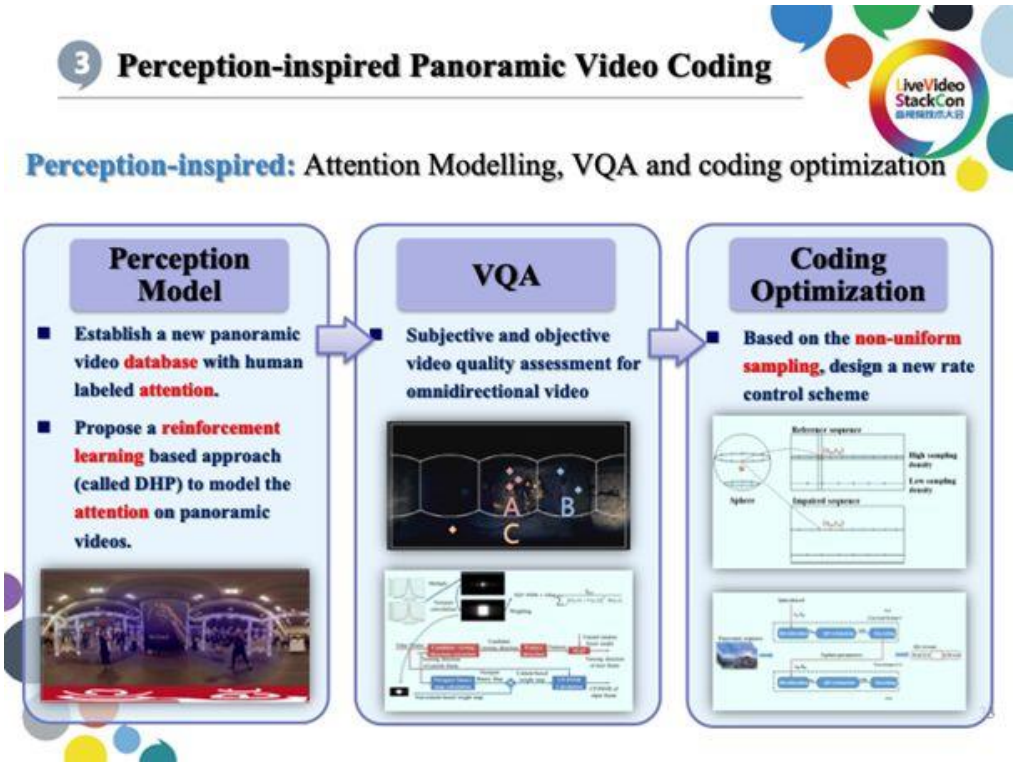
References

- **Mai Xu**, Tianyi Li, Zulin Wang, Xin Deng, Ren Yang and Zhenyu Guan, "Reducing Complexity of HEVC: A Deep Learning Approach," *IEEE Transactions on Image Processing*, Jun. 2018.
- Shengxi Li, **Mai Xu**, and Zulin Wang, "Optimal Bit Allocation for CTU Level Rate Control in HEVC," in *IEEE Transactions on Multimedia*, Jan. 2018.
- Shengxi Li, **Mai Xu**, Zulin Wang and Xiaoyan Sun, "Optimal Bit Allocation for CTU Level Rate Control in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2409-2424, Nov. 2017.
- Xin Deng, **Mai Xu**, Lai Jiang, Xiaoyan Sun and Zulin Wang, "Subjective-Driven Complexity Control Approach for HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 91-106, Jan. 2016.
- **Mai Xu**, Xin Deng, Shengxi Li and Zulin Wang, "Weight-based R-λ rate control for perceptual HEVC coding on conversational videos", *IEEE Journal of Selected Topics in Signal Processing* 8 (3), 475-489, 2014.
- Ren Yang, **Mai Xu**, and Zulin Wang, Multi-Frame Quality Enhancement for Compressed Video, *In CVPR*, 2018.
- Tianyi Li, **Mai Xu** and Xin Deng, "A deep convolutional neural network approach for complexity reduction on intra-mode HEVC," *in ICME*, 2017.
- Shengxi Li, **Mai Xu**, and Zulin Wang, "Optimal Bit Allocation for CTU Level Rate Control in HEVC," *in DCC*, 2016.
- Shengxi Li, **Mai Xu** and Zulin Wang, "A novel method on optimal bit allocation at LCU level for rate control in HEVC," *in ICME*, 2015.
- Shengxi Li, **Mai Xu**, Xin Deng and Zulin Wang, "A novel weight-based URQ scheme for perceptual video coding of conversational video in HEVC," *in ICME*, 2014.

32

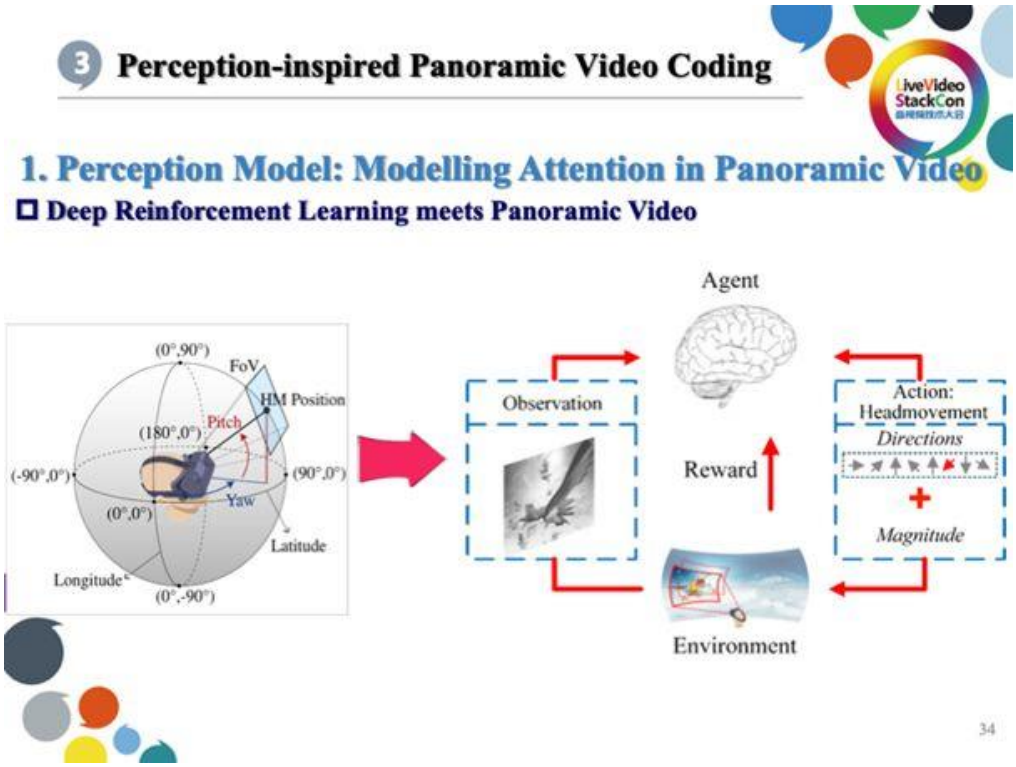
4. 面向全景视频的感知视频压缩编码

接下来将为大家介绍我们针对全景视频进行的感知压缩编码优化。

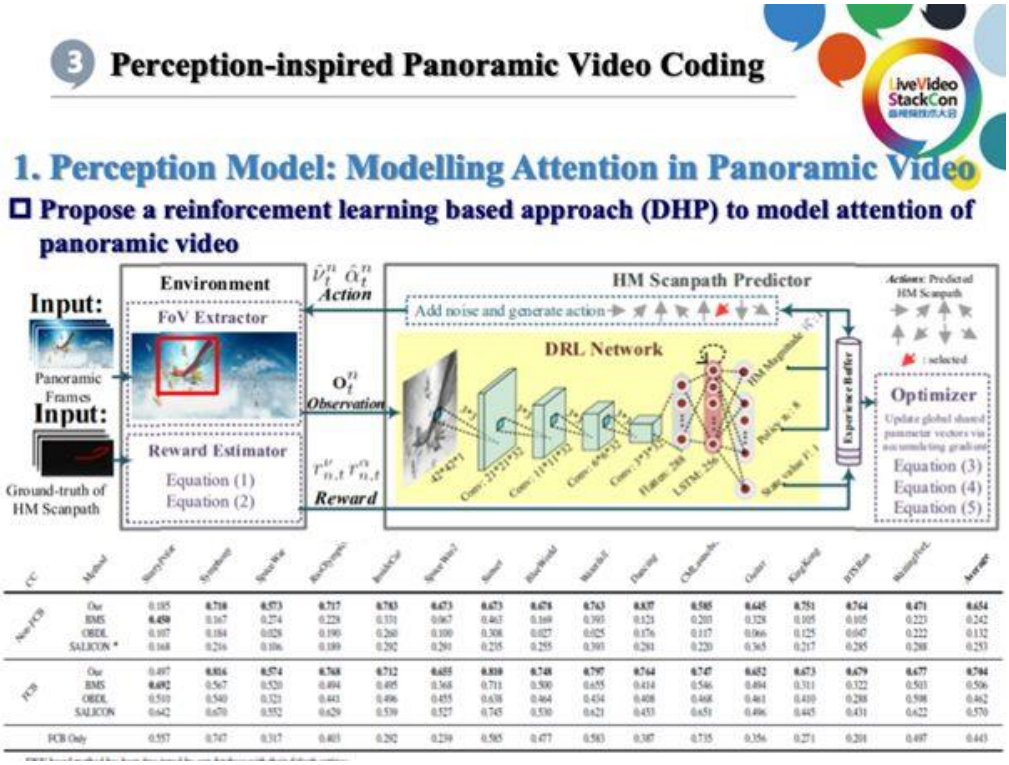


全景视频的感知压缩编码主要分为以下三个步骤：第一步进行的 Model 预测主要是用于判断用户观看全景视频时人眼视野关注到的画面区域；第二步进行全景视频质量评估。最后进行视频编码优化。

4.1 Model 预测

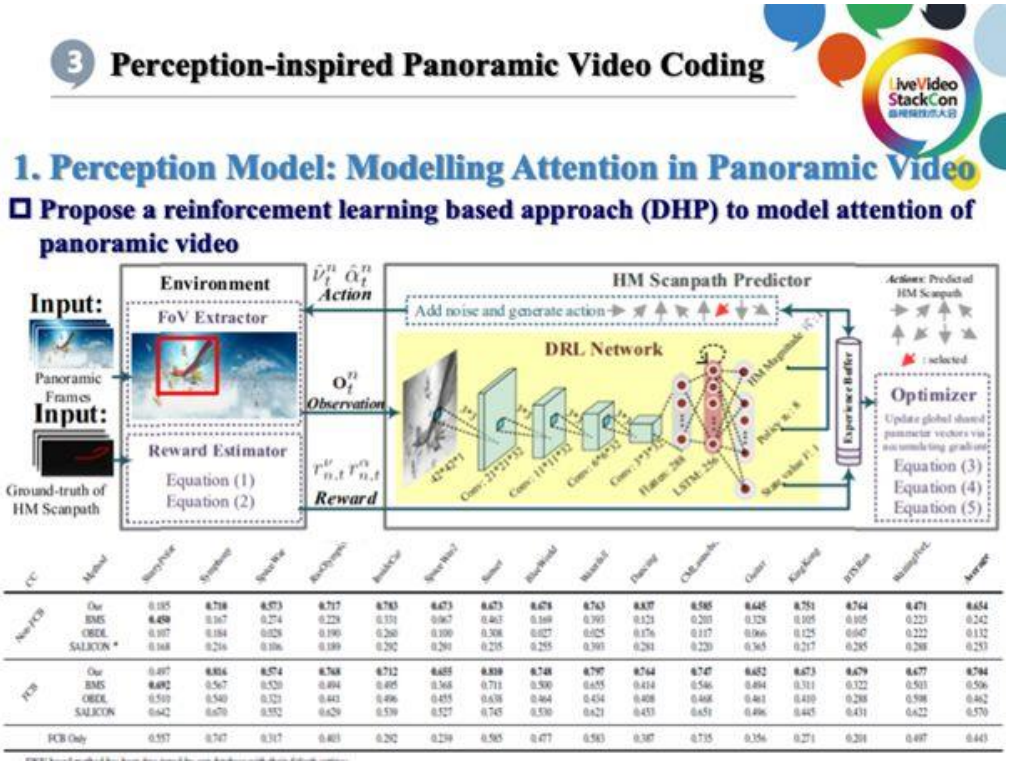


人类视觉系统对全景视频的感知与普通视频存在明显区别：用户在观看全景视频时视觉不会受到画幅的限制，但人眼 110 度的视觉角度只占全景视频 180 度甚至 360 度广幅画面中的一部分，而人眼真正关注的视觉重点可能最多在 60 度左右；通俗来说用户在观看全景视频时画面中的很多内容是无法感知的。这就是我们在实现全景视频的感知压缩编码时首要解决的问题：哪些区域是用户视觉重点关注的？全景视频的视角变化与用户头部运动的状态有关，我们可从用户头部运动入手，借助机器学习对用户头部运动与设备的交互所决策的视野角度进行强化学习训练；而人类头部的运动状态与交互行为是根据全景视频内容展现的环境而改变的；这种根据环境变化得到的头部运动状态与交互反馈信息能让我们准确得到经优化构造而成的最优回报函数（Reward），让用户在自己最为感兴趣的目标画面上得到最佳的用户体验；最后我们通过深度学习进一步优化此回报函数得出其最佳策略，再将此策略运用于对用户头部运动状态的预测判断；拥有了用户头部运动状态的预测信息，我们就可以预测推断用户视野的变化并将其用于视频画面的感知压缩编码。

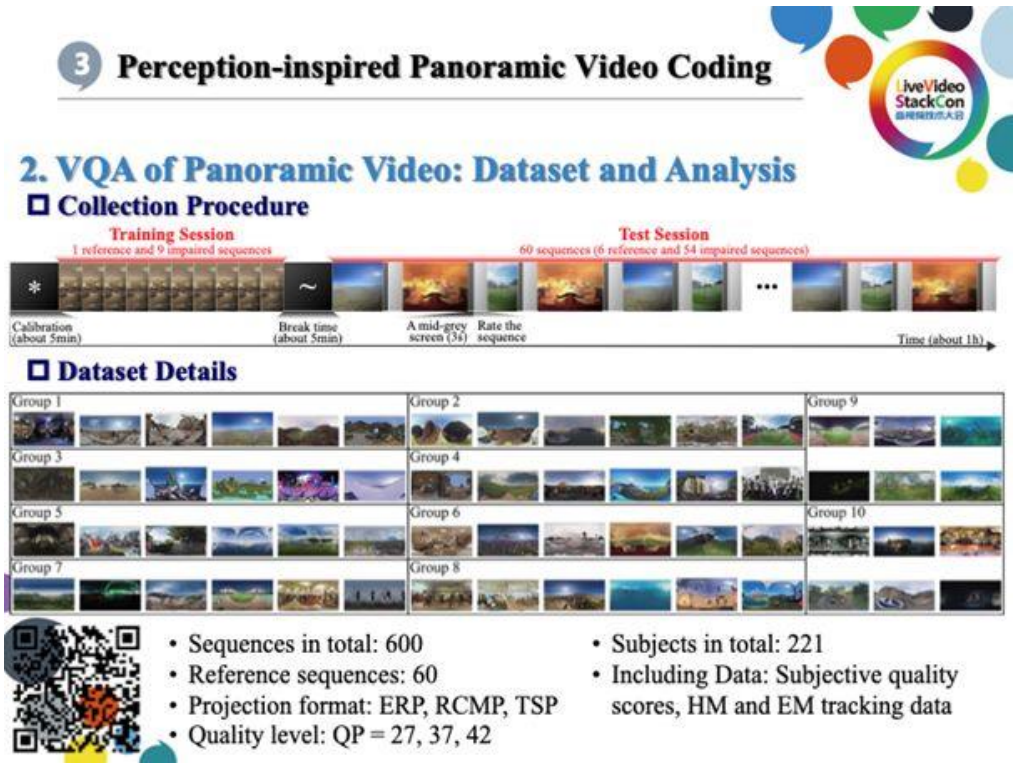


图中上半部分展示的是我们通过多人实验得到的可用于全景识别感知预测的 DRL Network 模型的架构。根据图中下半部分展示的测试结果，我们能从中得出视觉热点图。

下图同样展示了我们的定量测试结果。



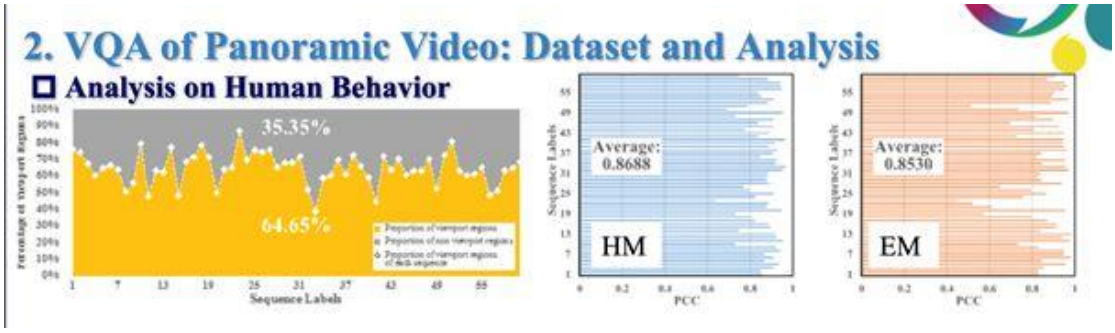
4.2 全景识别与质量评估



为了实现质量评估，我们首先构建了有包括 60 个无失真的基准视频在内的 600 个全景视频的全景识别数据库。其中的基准视频采用了 ERP、RCMP、TSP 三种映射模式并使用 27、37、42

三种 QP 规格的压缩方式。我们请大约 221 个观测对象为多个全景视频集评分并使每个全景视频拥有约 20 个评分数据；评分同时我们也会收集被测对象头部的运动状态、视野位置、眼部运动状态等关键数据。

整理分析这些信息我们可得到多项结论：

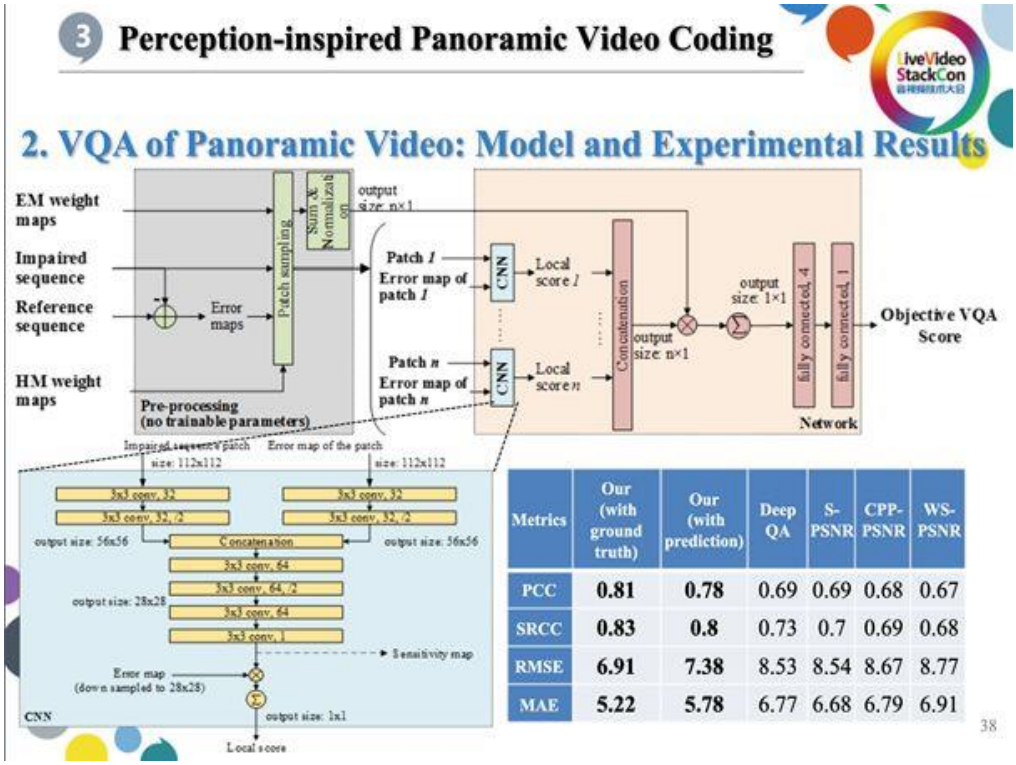


1）左侧图线代表是人们视野不关注的比例变化，平均有 35%的区域是用户从未关注到的；侧图线代表头部与眼睛运动的一致性，纵坐标代表视频而横坐标代表相关系数，可以看到头部运动状态与眼部运动状态相关性保持较高，由此我们可得出结论：用户观看全景视频时的行为习惯相似，画面中有约 30%的内容被视觉系统忽视。

Impact of Human Behavior on VQA

Metrics	Without HM and EM						With HM and EM					
	SSIM		PSNR		S-PSNR		$PSNR_{O\text{-}HM}$		$PSNR_{I\text{-}HM}$		$PSNR_{I\text{-}EM}$	
	Mean	All Data	Mean	All Data	Mean	All Data	Mean	All Data	Mean	All Data	Mean	All Data
PCC	0.60	0.44	0.56	0.49	0.61	0.55	0.66	0.60	0.67	0.61	**0.70**	**0.67**
SROCC	0.62	0.49	0.55	0.51	0.62	0.58	0.67	0.62	0.67	0.63	**0.70**	**0.69**
RMSE	9.25	10.63	9.70	10.36	9.29	9.95	8.65	9.51	8.48	9.43	**8.09**	**8.84**
MAE	7.27	8.31	7.77	8.11	7.36	7.75	6.87	7.46	6.73	7.38	**6.30**	**6.90**

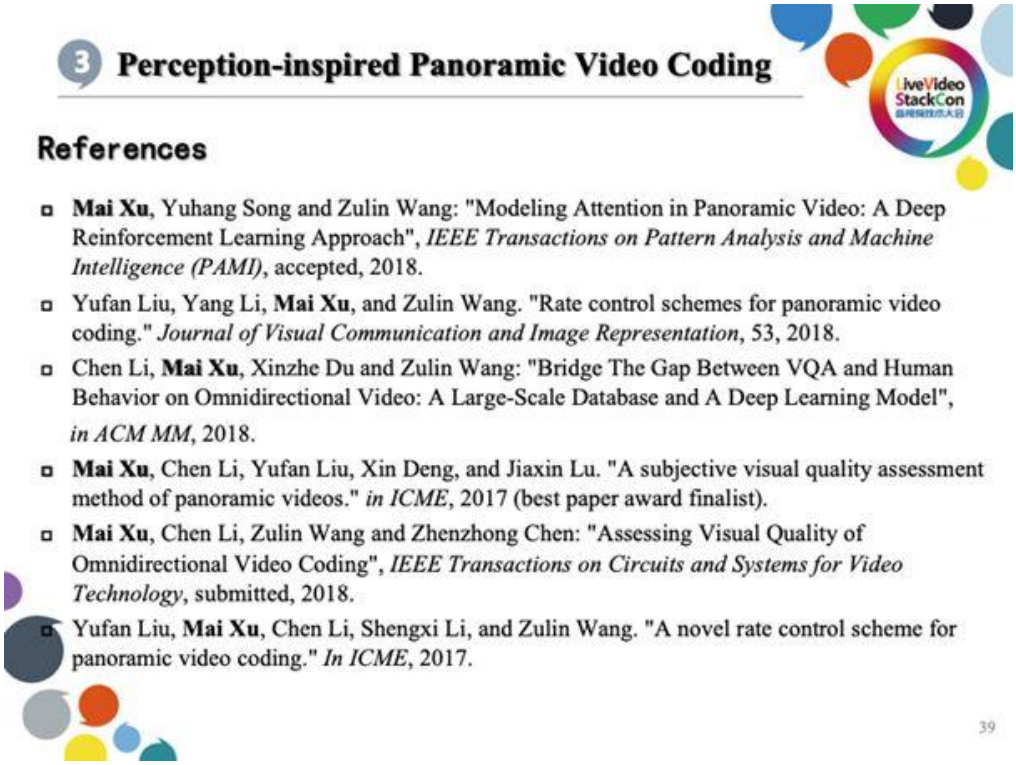
2）用户行为与全景视频质量的关系：画面中用户看不到或者不愿意观看的非重点部分的质量变化对用户观看体验的影响微乎其微；将用户头部运动状态、视野变化、眼动状态与 PSNR 相结合可大幅度提升 PSNR 性能，即可推断出头部运动状态、眼动状态等用户行为可被用于提升全景视频质量评估性能。



Metrics	Our (with ground truth)	Our (with prediction)	Deep QA	S-PSNR	CPP-PSNR	WS-PSNR
PCC	**0.81**	**0.78**	0.69	0.69	0.68	0.67
SRCC	**0.83**	**0.8**	0.73	0.7	0.69	0.68
RMSE	**6.91**	**7.38**	8.53	8.54	8.67	8.77
MAE	**5.22**	**5.78**	6.77	6.68	6.79	6.91

基于以上研究，我们可针对全景视频提出基于深度学习的质量预测方法，也就是将头部运动状态检测与眼动状态检测得到的数据用于评估不同优化模型得出的全景视频质量并将输出的质量分数与其相关性进行比较。如传统方法得到的 PCC 模型预测值约为 0.78，而 SRCC 模型则可达到 0.81，从而进一步提升全景视频质量评估结果。

下图展示的是与此项研究相关的一些 References。



5. 总结

最后总结一下，首先我们的这些技术离不开数据的澎湃力量；其次全新的感知模型拥有非凡的编码优化潜力，最后是未来层出不穷的创新媒体渠道与应用模式离不开技术的强大力量。

实时远程医学影像服务质量保障与网络优化

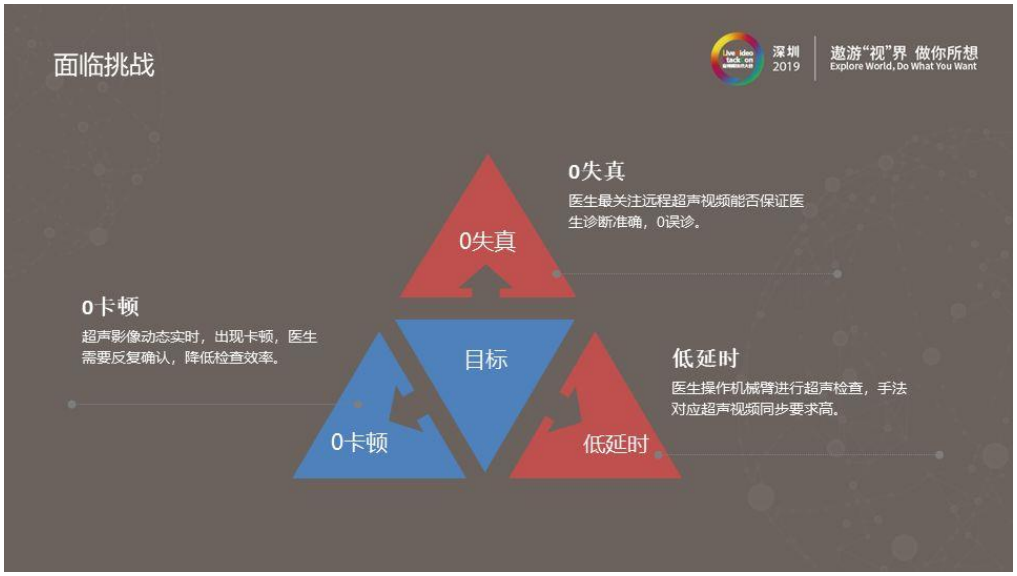
实时远程医学影像产品对图像传输的失真、卡顿、延时的要求尤为严苛。在LiveVideoStackCon2019 深圳大会上，华大智造音视频技术专家 黄翠萍详细了介绍如何在现有网络保障实时远程医学影像的服务质量及在网络方面所做的优化。

黄翠萍　LiveVideoStackCon 2019 深圳　2020-02-20

1.产品介绍

华大影像团队成立的意义：通过集成机器人技术、实时远程控制技术及超声影像技术，解决偏远地区、基层医疗机构缺少超声医生、以及现有医生超负荷工作的现状；打破传统超声诊疗方式的局限，克服时空的障碍，改善医疗资源分布不均衡的现状；使全民平等的享受优质的医疗服务。

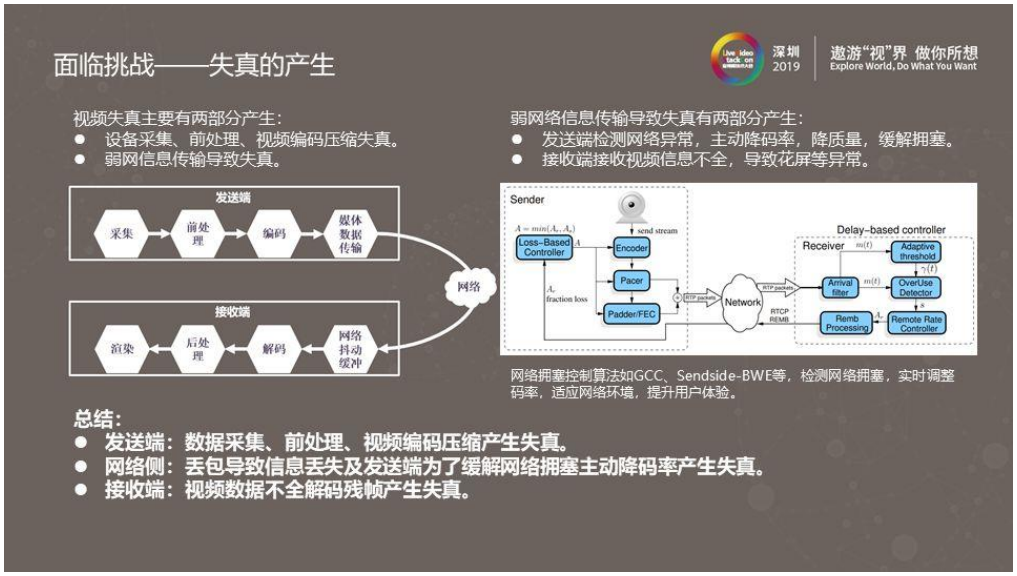
2.面临挑战



在远程医疗视频方面的挑战主要有三方面：失真、卡顿、延时。

- 失真：医生最关注远程超声视频能否保证医生诊断准确，0 误诊
- 卡顿：超声影像为动态实时，出现卡顿，会导致医生需要反复确认，降低检查效率
- 延时：医生通过操作远程机械臂进行超声检查，手法对应超声视频同步要求高

为了应对挑战，首要任务是分析产品产生失真、卡顿、延时的原因是什么?



失真主要由以下三个模块产生：

- 发送端：数据采集、前处理、视频编码压缩产生失真；
- 网络侧：丢包导致信息丢失及发送端为了缓解网络拥塞主动降码率产生失真；
- 接收端：视频数据不全解码残帧产生失真。

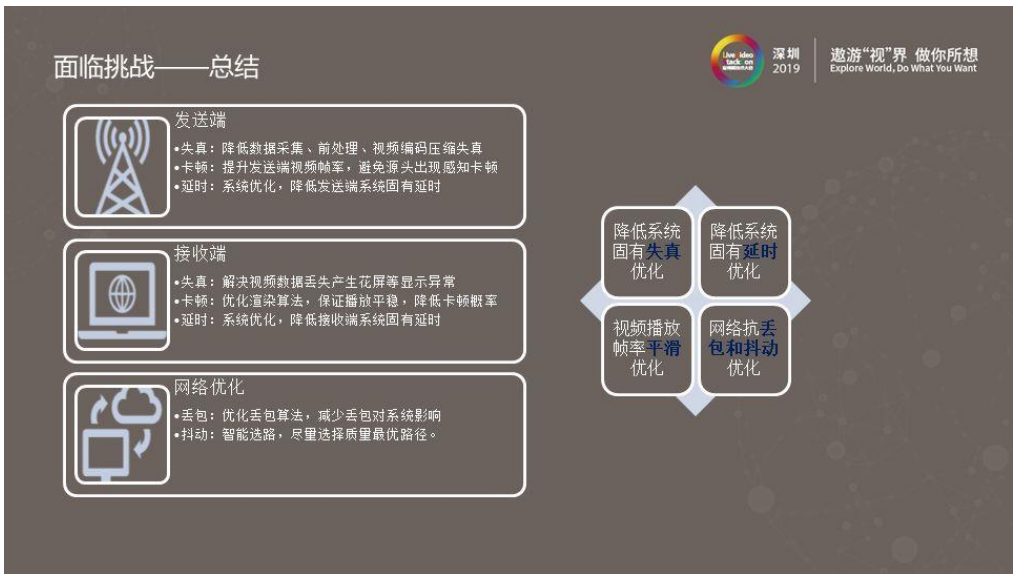
卡顿主要由三部分产生，人眼最高捕获帧率为 30 帧每秒，当帧间间隔分布不均、渲染帧率过低，人会感知卡顿。

- 发送端：如果发送端发送的帧率偏低，例如通讯软件的帧率一般在 15 帧每秒，帧率的流畅性不符合医学要求；
- 网络端：弱网延时过大、或视频帧丢失，会产生冥想卡顿；
- 接收端：如果帧率为 30 帧每秒的帧率可以稳定的情况下，不会感觉明显卡顿，但是如果帧率不稳定，忽高忽低情况下，用户就会感知卡顿，会降低用户体验。

根据 ITU-T G.114 国际标准规定，端到端的延时控制在 200 毫秒以内时，则用户体验良好，整体感知不到明显延时，对于我们产品的延时主要有两大类：固有延时和网络延时。

- 系统固有延时：仅仅指端的采集、视频编解码的延时；
- 网络延时：不仅仅指网络传输的延时，会增加抗网络丢包和抖动的手段，例如 NACK、FEC 本身引入的缓存所导致的延时，可以归类为网络延时。

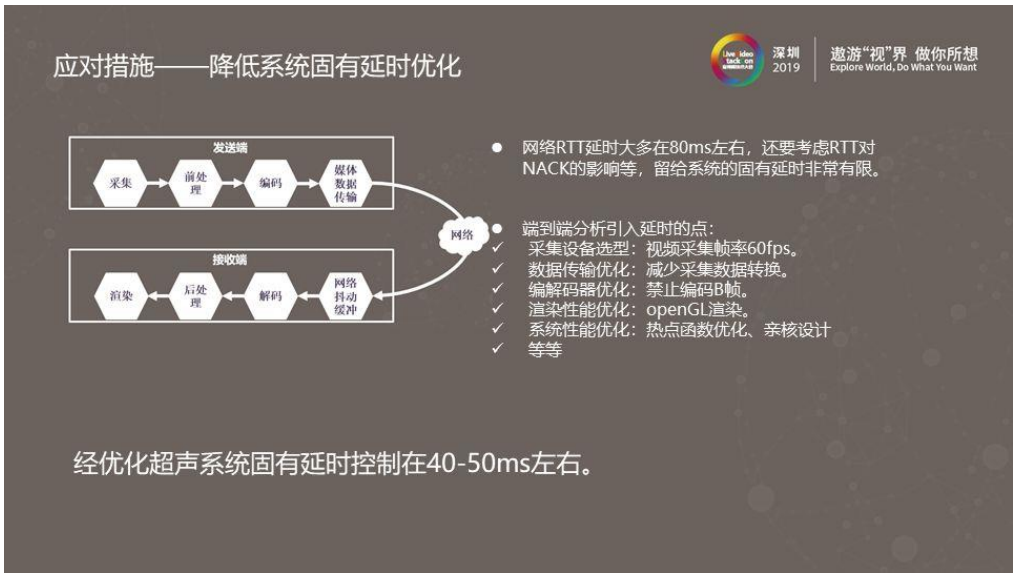
根据通讯行业标准规定，网络质量分为良好、较差、恶劣三个等级，实际在互联网应用中发现，大多数网络并不满足如图所示行业标准规定。一般网络环路延时大概在 70、80 毫秒左右，丢包在 8%左右，因此如果想要提供良好的远程服务，控制系统的固有延时和网络延时需要在 200 毫秒以内。



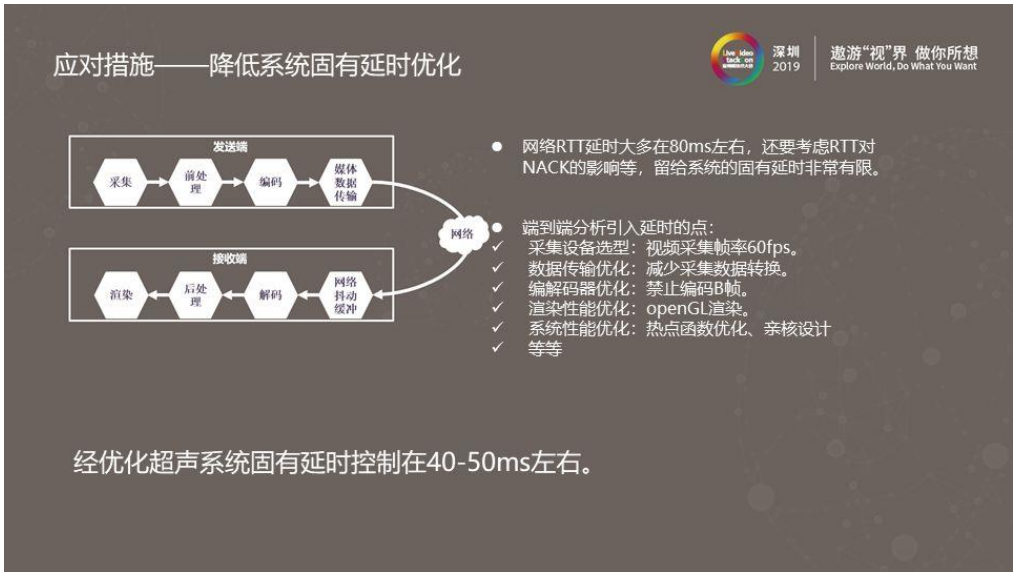
通过分析总结出 4 个模块内容可进行优化：

- 降低系统固有失真优化
- 降低系统固有延时优化
- 视频播放帧率平滑优化
- 网络抗丢包和抖动优化

3.应对措施



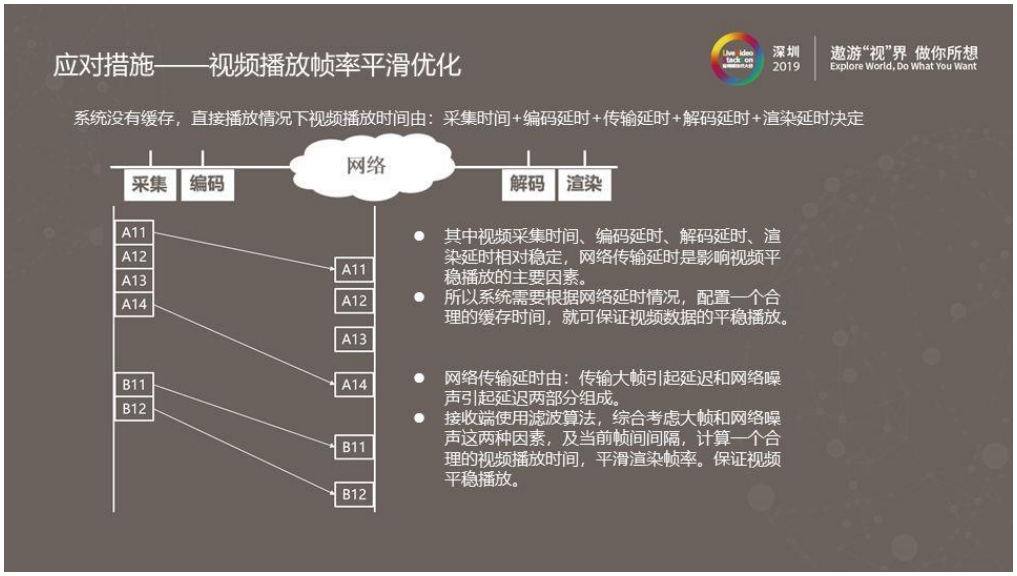
对于医生面对产品所产生的顾虑：是否会导致误诊？使用体模方式把关超声视频质量，针对扫描不同的脏器器官购买了不同的体模，进行 0 失真验证；再增加 psnr+ssim+vamf 方式把关视频质量，以保证在任何情况下，在发送端发送 0 失真数据，完善把关方法后，就可以分析发送端到接收端中所有可优化的点，进而采取相应优化措施。结合网络质量动态优化编码算法，在高码率的情况下保证速度优先，低码率情况下保证质量优先，中等码率情况下均衡二者的关系。对于接收端，使用私有协议对视频报文、帧、帧间参考关系进行完整性判断，其中首先对 RTP 报文的完整性进行判断，如若不完整则使用 HARQ 进行重传修复，如若单帧不完整则采取丢帧操作等对应的操作，以保证接收端接受的信息无异常并且高质量及完整。



网络 RTT 延时大多在 80 毫秒左右，同时要考虑 RTT 对 NACK 的影响等，留给系统的固有延时非常有限。因此进行端到端分析引入延时的点有：

- 采集设备选型：视频采集帧率 60fps。在医学实际应用中，很多情况要求采集帧率高，例如心脏科等。
- 数据传输优化：减少采集数据转换。视频采集可用 MGP 或 YUV 格式，针对不同情况，选用对视频传输最有利的格式。
- 编解码器优化：禁止编码 B 帧。
- 渲染性能优化：openGL 渲染。
- 系统性能优化：热点函数优化、亲核设计。

经过优化的超声系统的固有延时控制在 40-50 毫秒左右。



如若没有经过任何缓存，直接播放情况下视频播放时间由：采集时间、编码延时、解码延时、渲染延时共同决定，对于以上几种延时情况较为稳定，而网络传输延时是影响视频能否平稳播放的主要因素，因此系统就需要根据网络延时情况配置合理的缓存时间，同时，接收端使用滤波算法，综合考虑传输大帧引起的延时、网络噪声引起的延时以及帧间间隔，计算合理的视频播放时间，平滑渲染帧率，以保证视频平稳播放。

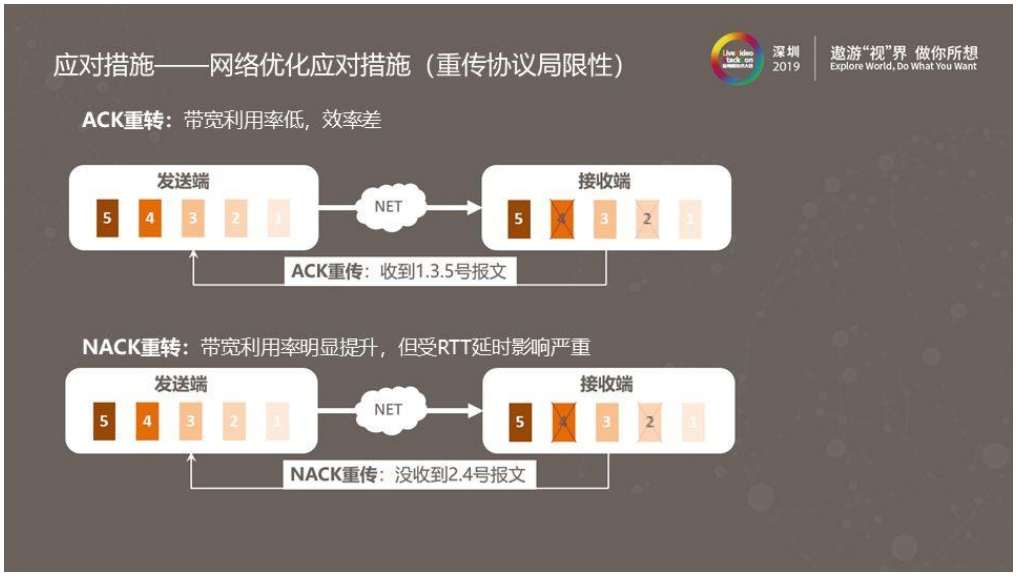
- 冗余协议局限性及优化

常用的抗网络丢包方法有冗余、重传和 SVC 编码，但这三种方法各有优缺点。例如 RFC 2198 冗余协议的性价比低；XOR_FEC 协议性价比略高，但恢复能力有限；RS_FEC 冗余算法的恢复能力强，但合适的冗余度配置度较难。

常规算法：当前丢包率、丢包率加权平均值（恢复能力差）、窗口期最大丢包率（带宽利用率低）。

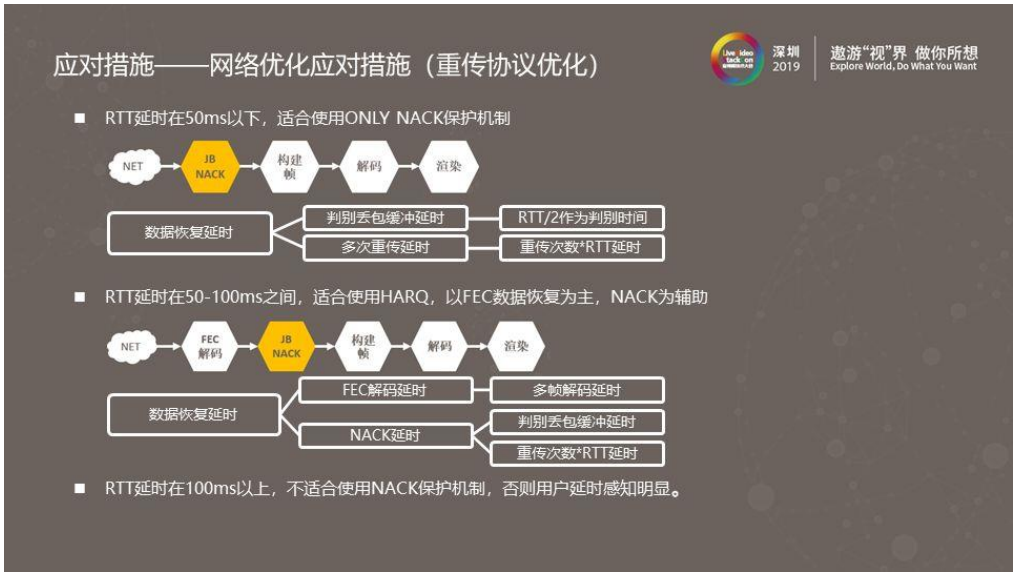
远程超声冗余度配置方法：卡尔曼+窗口期最大丢包率加权平均值，同时使用多帧 FEC 冗余机制和深度优化参数，根据视频帧率、RTT 环路延时，动态调整 FEC 冗余帧数，以保证视频传输实时性。

- 重传协议局限性及优化



ACK 重传：TCP 常用，收到报文则进行 ACK 响应，其缺点是带宽利用率低，效率差

NACK 重传：音视频领域常用，带宽利用率明显提升，其缺点是受 RTT 延时影响严重。在环路延时较大时，会引入系统延时高。



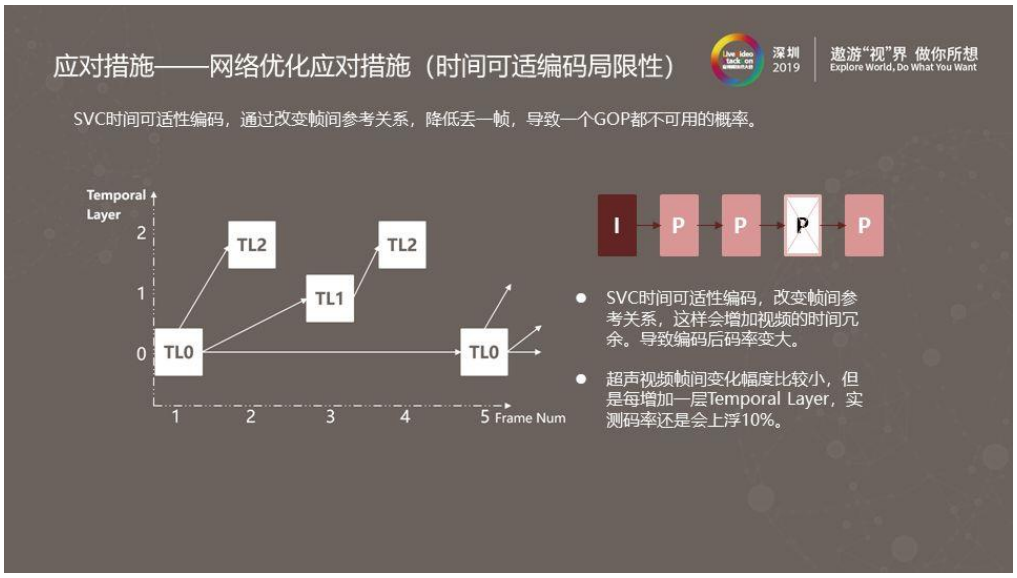
针对 RTT 延时对 NACK 的影响较严重的问题，针对不同情况提出三种应对策略：

A. RTT 延时在 50ms 以下，适合使用 ONLY NACK 保护机制。即使用 RTT/2 作为是否进行丢包缓冲延时的判别标志；使用重传次数的 RTT 倍的时长作为是否进行多次重传延时的判别标志。

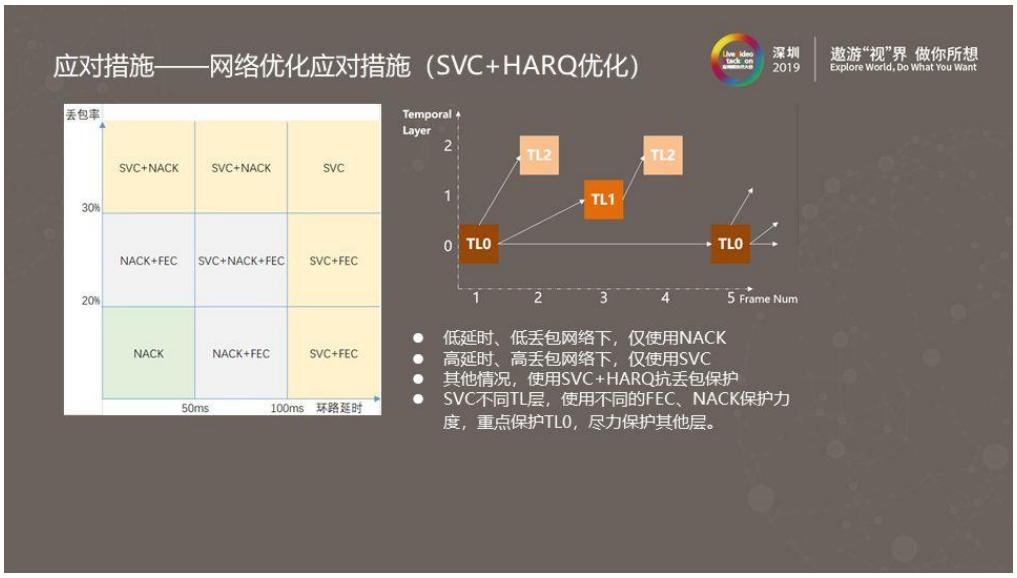
B. RTT 延时在 50-100ms 之间，适合使用 HARQ，以 FEC 数据恢复为主，NACK 辅助。

C. RTT 延时在 100ms 以上，则不适合使用 NACK 保护机制，否则用户延时感知明显。

- 时间可适编码局限性及优化

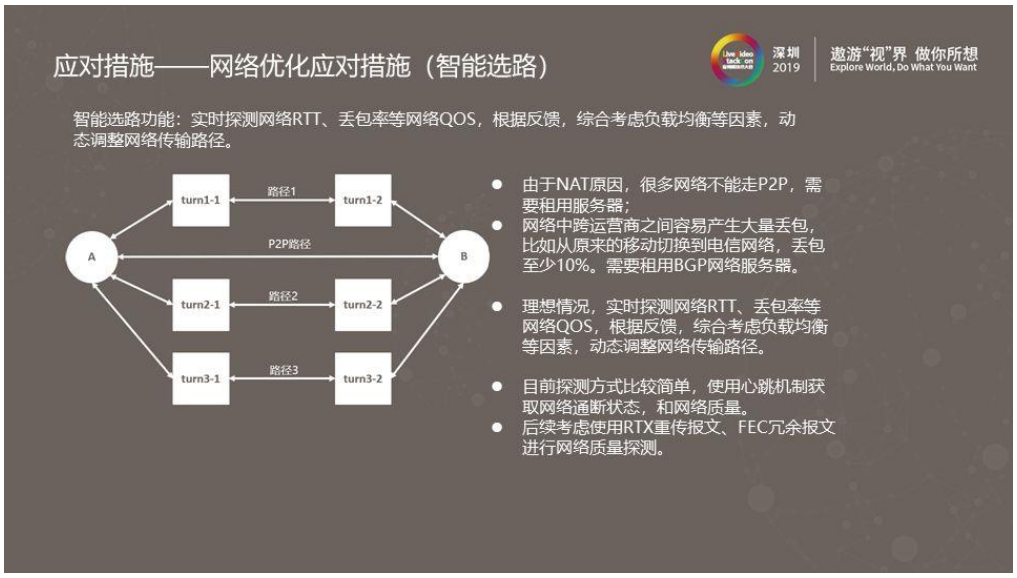


SVC 时间可适性编码：通过改变帧间参考关系，降低丢一帧导致整个 GOP 都不可用的概率，但是增加了时间冗余度信息，导致编码后的码率上升。对于超声影像，帧间幅度的改变较小，但每增加一层 Temporal Layer，实测码率会上浮 10%。



低延时、低丢包网络下，仅使用 NACK 抗丢包保护。
高延时、高丢包网络下，仅使用 SVC 抗丢包保护。
其他情况，使用 SVC+HARQ 抗丢包保护，策略是，在 SVC 不同 TL 层，使用不同的 FEC、NACK 保护力度，重点保护 TL0 层，尽力保护其余层。

- 智能选路



由于 NAT 原因，很多网络不能丢 P2P，需要租用服务器，但是网络中跨运营商之间易产生大量丢包，例如从移动切换到电信网络，丢包至少 10%，因此需要租用多台 BGP 网络服务器。

对于多台 BGP 网络服务器的路径选择问题，使用心跳机制（目前所使用的方式）或使用 RTX 重传报文、FEC 冗余报文实时探测网络 RTT、丢包率等网络 QOS，根据反馈，综合考虑负载均衡等因素，动态调整网络传输路径。

4.成果展示

如图是项目应用情况，经过若干优化工作后，目前远程超声项目可以在 RTT 延时 100 毫秒以内，网络丢包 30%以内，实现 0 失真、0 卡顿、低延时传输。

5.未来展望

华大智造是为了解决就医难而成立，未来有以下几方面的展望：

- 远程计移动超声诊断：借助 5G 网络，进一步将远程计移动超声覆盖更多有需要的地方；
- 医学影像云服务：综合 5G+AI+云技术，为医生和患者提供更快捷、更准确的诊断体验。

通过不断技术更新，进而实现智慧医院整体方案和智慧医疗平台。

音频技术之美

开源声码器 WORLD 在语音合成中的应用

语音合成(TTS)是语音 AI 平台的基础设施，而声码器则决定着其中的声学模型以及合成质量。喜马拉雅 FM 音视频高级工程师 马力在 LiveVideoStack 线上交流分享中详细介绍了新一代合成音质更高，速度更快的开源免费 WORLD 声码器的原理架构，并结合 Tacotron 模型，演示中文语音合成的应用。本文由 LiveVideoStack 整理而成。

马力　LiveVideoStack 线上分享　2019-01-29

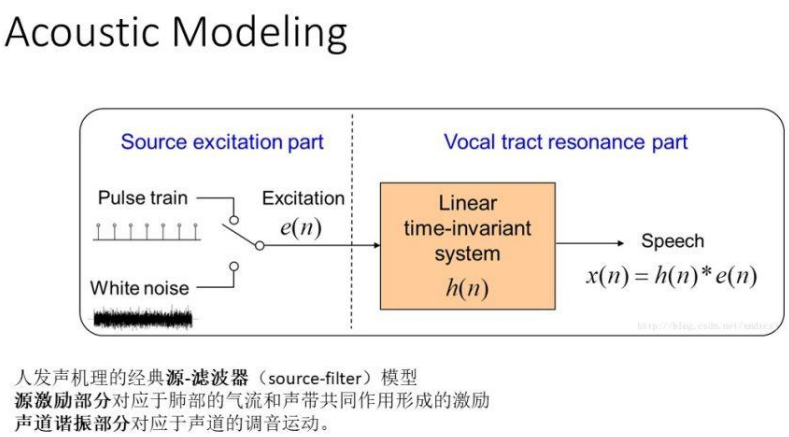
大家好，我是马力，现就职于喜马拉雅 FM 担任音视频高级工程师，本次我将为大家介绍开源声码器 WORLD 在语音合成中的应用。

1. Tacotron+WORLD

WORLD 声码器可与谷歌的开源语音合成模型 Tacotron 结合使用，如何实现这一过程呢？Tacotron 是一种端到端的 TTS 深度学习模型，所谓“端到端”是指利用前端文本直接预测后端声谱，整合了之前的独立子模块，达到系统整体最优；WORLD 则是一种声码器，与 Tacotron 结合可基于人类发音频谱将文字转化为与人类发音相似的声音：将每个文字转化为拼音之后，声码器会把每个拼音看作为一个序列而 Tacotron 会在此基础上预测每段需要合成语音的序列，随后 WORLD 声码器再将预测出的声谱转换为原始的声音波形。Tacotron 主要负责确定此声谱特征能否使用 WORLD 将其恢复为声音波形并评估语音质量是否符合要求。

1.1 WORLD Vocoder

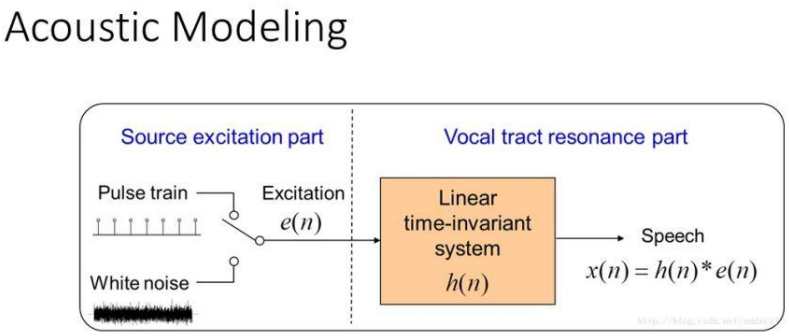
WORLD Vocoder 基于声音特征实现，主要分为三个部分：Analysis 分析、Manipulation 操作与最后的 Synthesis 合成。Analysis 分析过程主要是指从一段原始声音波形中提取声学特征如线性谱、低频、MFCC；Manipulation 操作过程是指对提取出的原始声学特征进行压缩等降维处理使其表征能力进一步提升；最后的 Synthesis 合成过程是把此声学特征恢复至原始的声音波形。



人发声机理的经典**源-滤波器**（source-filter）模型
源激励部分对应于肺部的气流和声带共同作用形成的激励
声道谐振部分对应于声道的调音运动。

既然 WORLD Vocoder 基于声学特征实现，那么就需要相应的声学建模。实际上所有的 Vocoder 都可通过一种简单的模型实现，此模型就是线性时不变系统。如上图展示的那样，左侧代表声音输入，右侧代表声音输出。输入部分被称为 Source Excitation Part 也就是激励部分，右侧输出结果被称为 Vocal Tract resonance Part 也就是声道谐振部分。整体过程为一个基于人发声机理的经典源-滤波器模型，也就是输入的激励部分通过线性时不变系统进行操作；随后输出的声道谐振部分作为合成语音。如果对应人体发声结构进行分析，激励部分对应肺部气流与声带共同作用形成的激励，而声道谐振部分对应于声道的调音运动。

1）激励部分

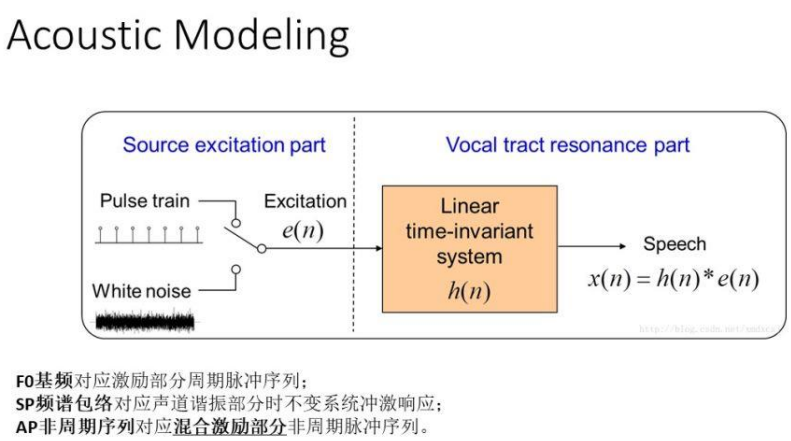


浊音(voiced)：气流通过紧绷的声带，对声带进行冲击而产生振动，使声门处形成准周期性的脉冲串，激励信号简化为**周期性的脉冲激励**。
清音(unvoiced)：声带处于松弛状态，不发生振动，气流通过声门直接进入声道，激励信号简化为**随机白噪声**。

激励分为两种：Pulse train 脉冲激励与 White noise 随机白噪声。脉冲激励对应我们说话声音当中的浊音，也就是气流通过紧绷的声带时，对声带进行冲击所产生的声音；此时声门处会形成准周期性的脉冲串，我们把这种激励信号简化为周期性的脉冲激励。

随机白噪声对应我们说话声音当中的清音，也就是声带处于松驰状态不发生振动时气流通过声门直接进入声道所产生的声音；我们把这种激励信号简化为随机白噪声。

2）声学特征



WORLD 对应以下三种声学特征：F0 基频、SP 频谱包络与 AP 非周期序列。F0 基频对应激励部分的周期脉冲序列，如果我们将声学信号分为周期性信号与非周期信号的话；SP 频谱包络对应声道谐振部分时不变系统的冲激响应，大家可以看到经过此线性时不变系统之后 Vocoder 会对激励与响应进行卷积；AP 非周期序列对应混合激励部分的非周期脉冲序列，因为所处理的信号包括周期性信号与非周期信号，虽然可以在混合时仅处理周期性信号并将其作为特征提取，随后再进行系统处理也能够合成相应的目标信号，但其由于缺乏非周期性信号的输入，合成的自然度难以符合要求，因此我们会混合提取非周期信号并同样将其作为特征之一进行提取与处理。

Acoustic Features

- F0(fundamental frequency)
  - 一组正弦波组成原始信号，频率最低的正弦波为基频，其它为泛音。
- SP(spectral envelop)
  - 将不同频率的振幅最高点通过平滑的曲线连接起来就是频谱包络。
- AP(aperiodicity)
  - 语音的非周期信号参数
  - 混合激励可以通过aperiodicity来控制浊音段中的周期激励和噪声（非周期）成分的相对比重

对比三种声学特征，正弦波组成的原始信号中频率最低的正弦波即为基频，其它正弦波则被称为谐波(泛音)；频谱包络是指将不同频率的振幅最高点通过平滑的曲线连接起来得到的包络线；非周期序列对应混合激励部分的非周期脉冲序列，其中的混合激励是指通过多种参数控制周期激励、噪声与非周期信号，最后混合周期与非周期信号并通过 aperiodicity 调控二者权重。

1.2 优势

那么我们为什么选择 WORLD 作为 Vocoder 呢？WORLD 相对于常见的如 STS 等其它传统声码器的优势有以下三点：

首先是声音质量，WORLD Vocoder 输出声音的质量优于其它传统声码器，其输出结果有人类自然语言的听感效果；而传统声码器输出声音不但质量欠佳，如机器般冰冷生硬的听感也非常不好。

其次 WORLD 的处理速度也高于传统声码器，处理速度是语音合成中一项十分重要的指标，我所知的一些基于神经网络的声码器虽然可以实现高质量的声音合成但速度却十分低下，究其原因在于神经网络需要对每一个样本点进行处理；假设一段 48k 采样的音频，其每一秒中包括 48000 个采样，如果对每个采样都进行处理无疑会大大拖慢处理效率；而 WORLD 声码器有其独特的处理效果如无损提高语音合成速度等，同时语音质量不会出现任何衰减。

最后是开源，WORLD 的算法没有专利性而且开放给任何人使用。相对于其它需要额外花费购买商业版权的声码器，WORLD 的优势十分明显。

2. WORLD 分析功能

WORLD Analysis

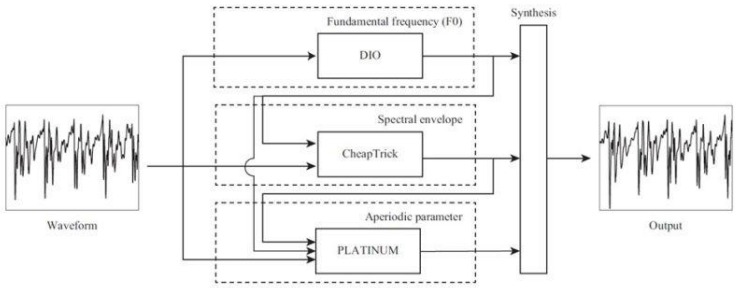
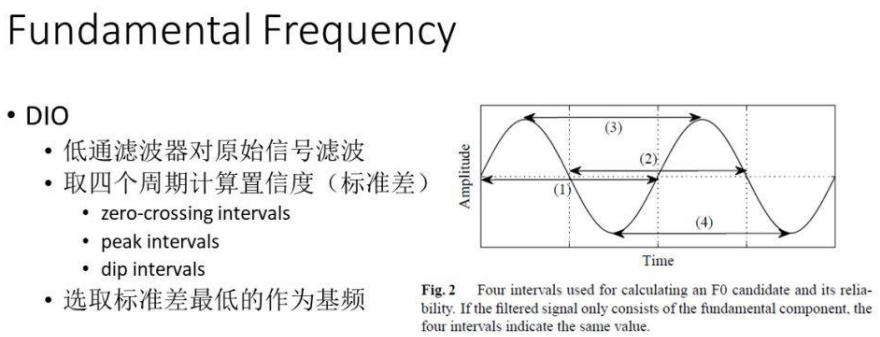


Fig. 1 Overview of the developed system. WORLD consists of three analysis algorithms for determining the F0, spectral envelope, and aperiodic parameters and a synthesis algorithm incorporating these parameters.

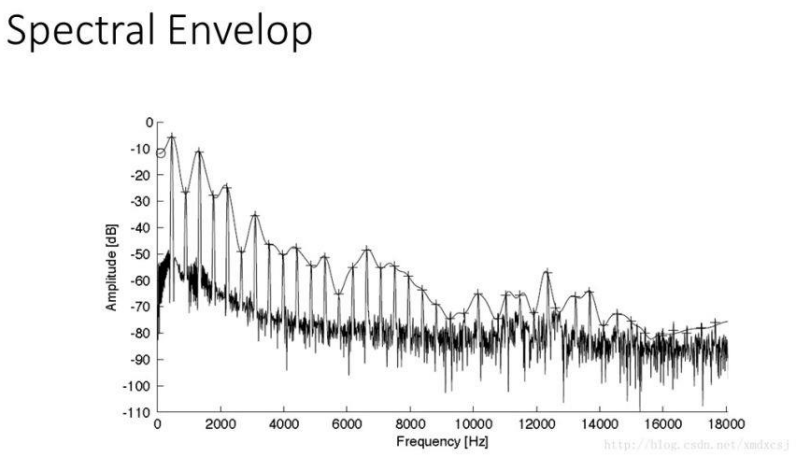
WORLD 分析功能的整体架构图如上，我们将一段语音输入系统并得到此段声音的一个特征体系，我们可以存储这部分特征并在需要时读取；语音信号经过一系列合成最终输出目标信号，WORLD 会重点提取原始波形中的基频、频谱包络与非周期信号，这三种声学特征对应三种提取算法：负责提取基频的 DIO、将提取的 F0 基频同波形一块输入提取频谱包络 SP 的 CheapTrick，以及将 F0、SP 和波形一同输入提取非周期信号的 D4C，最终将三种声学特征通过合成算法输出恢复其原始波形信号。

2.1 Fundamental Frequency——DIO 算法



DIO 算法通过一个低通滤波器对原始信号进行滤波得以实现对基频的提取，具体过程为从某个时间点的正弦波中提取出四个周期信号并计算置信度也就是标准差，随后选择标准差最低的作为基频。

2.2 Spectral Envelop——CheapTrick



上图展示了声学特征中的包络。我们的声音包含不同频率的信号，覆盖 0 到 18000 赫兹，每个频率都有其振幅（以分贝为单位），我们定义每种频率波的振幅最高点连线组成的图形为包络。

Spectral Envelop

- LPC
  - 一个语音取样值可以用若干个语音取样过去值的加权线性组合来逼近
- Cepstrum
  - 复数倒谱拥有频谱幅度跟相位的信息
  - 信号 -> FFT -> 取绝对值 -> 取对数 -> 相位展开 -> IFFT -> 倒频谱
- CheapTrick (pitch synchronous analysis)
  - F0 adaptive windowing
  - Smoothing of the power spectrum
  - Liftering in the quefrency domain

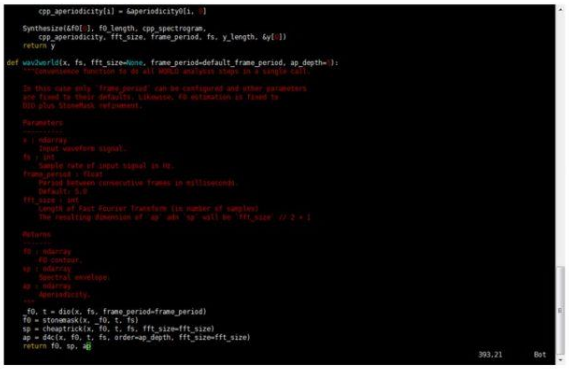
我们有三种提取包络声学特征的方法：LPC、Cepstrum 与 CheapTrick。LPC 就是线性预测编码，其原理为用若干个语音取样过去值的加权线性组合不断逼近一个语音取样值；Cepstrum 则是基于复数倒谱拥有频谱幅度与相位信息的原理，通过对一个信号进行 FFT、取绝对值、取对数、相位展开、IFFT 的变化处理，从而得到其相应的倒谱图实现。

而 WORLD 中采取的 CheapTrick 也被称为音高同步分析的方法，其过程是先将不同基频进行适配加仓操作与平衡以及功率平滑操作，随后将信号在频域进行同态滤波操作。

2.3 Aperiodicity——D4C

非周期信号对应两种提取算法：Plantinum 算法与 D4C 算法，在 WORLD 中我们主要采用 D4C 算法，基本可分为计算、修正参数与估计 Band-aperiodicity。

2.4 应用



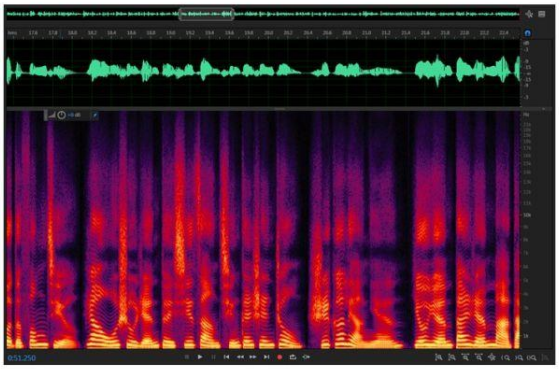
我们可以在一些开源的工程项目中见到对上述内容的应用如 Python Wrapper。借助 Python 实现整个特征提取过程：首先通过 DIO 提取基频，再通过基频中的 CheapTrick 提取包络；得到基频与包络后再通过 D4C 得到一个非周期信号，我们将原始波形作为输入信号，利用其采样率、快速傅里叶变换的次数，以及帧移（frame period）；最后返回这三个音频特征。

3. 神经网络优化

需要强调的是，单纯的基频、频谱包络与非周期信号结合基于神经网络训练的深度学习技术并不能达到我们期待的理想效果，其原因在于维度过高，尤其是频谱包络与非周期信号的高维特点使得未经优化的特征可达一千多纬度，这为神经网络的训练带来很大压力。

而 Merlin-WORLD 通过实现维度转换有效降低维度，首先是 MGC 也就是将提取到的 MFCC 特征降低到 60 维度从而方便神经网络的训练；其次是将一个非周期信号转变为一个 band 非周期信号，以 48KHz 采样为例，经过此操作可实现将一千维度降低至 5 维；基频则保持原来维度不变。这样我们就可将以上三种特征维度连接起来，变成小于一百维的特征输入 Tacotron 模型从而实现理想的训练与深度学习效果。

Resynth



上图展示了我们提取声学特征后恢复原声音波形得到的频谱，可以看到图像基本清晰，整体方法是利用 Tacotron 端对端 TTS 深度学习模型预测文本的声谱并通过 WORLD 声码器还原声谱。相信下图展示的参考资料会帮助大家进一步了解 Tacotron+WORLD 的无限魅力。

参考资料

- 论文PDF：https://github.com/tuanad121/Python-WORLD/tree/master/doc/lit
- 科普博文：https://me.csdn.net/xmdxcsj
- Merlin工具脚本：https://github.com/CSTR-Edinburgh/merlin/tree/master/misc/scripts/vocoder/world
- Merlin中文手册：https://mtts.readthedocs.io/zh_CN/latest/merlin.html#merlin-vocoder
- Tacotron整合：https://github.com/Rayhane-mamah/Tacotron-2/issues/304
- 标贝开源中文语音合成数据下载：http://www.data-baker.com/open_source.html

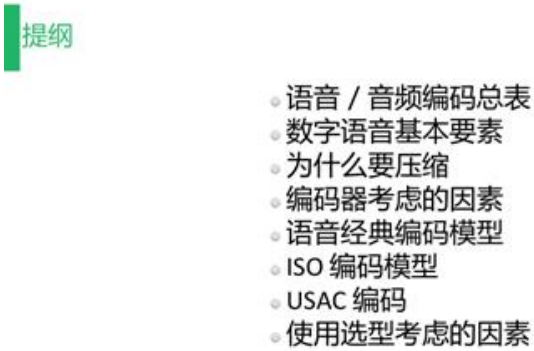
语音编解码技术演进和应用选型

本文来自现网易云音乐音视频实验室负责人刘华平在 LiveVideoStackCon 2017 大会上的分享，并由 LiveVideoStack 整理而成。分享中刘华平以时间为主线，讲述了语音编解码技术的演进路线及实际应用中的技术选型。

刘华平　LiveVideoStackCon 2017 北京　2018-11-22

大家好，我是刘华平，从毕业到现在我一直在从事音视频领域相关工作，也有一些自己的创业项目，曾为早期 Google Android SDK 多媒体架构的构建作出贡献。

就音频而言，无论是算法多样性，Codec 种类还是音频编解码复杂程度都远远比视频要高。视频的 Codec 目前还主要是以宏块为处理单元，预测加变换的混合编码框架，例如 H.264 和 H.265 都是在这一框架下，另外在应用中，某一时间往往单的视频编码能主导，当前 H.264 就占据了 90%以上的市场份额。，而音频则相当复杂，且不同的场景必须要选择不同的音频编解码器。以下就是本次为大家分享的主要内容，希望通过此次分享可以使大家对音频编解码有一个整体的认识，并在实际应用中有参考的依据。



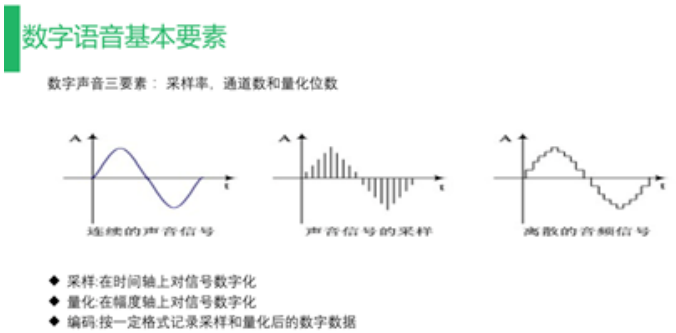
1. 语言/音频编码总表

语音 / 音频编码总表

Audio compression	ISO/IEC	MPEG-1 Layer III (MP3) · MPEG-1 Layer II (Multichannel) · MPEG-1 Layer I · AAC (HE-AAC · AAC-LD) · MPEG Surround · MPEG-4 ALS · MPEG-4 SLS · MPEG-4 DST · MPEG-4 HVXC · MPEG-4 CELP · MPEG-D USAC · MPEG-H 3D Audio
	ITU-T	G.711 (A-law, μ-law) · G.718 · G.719 · G.722 · G.722.1 · G.722.2 · **G.723** · G.723.1 · G.726 · G.728 · G.729 · G.729.1
	IETF	Opus · iLBC
	3GPP	AMR · AMR-WB · AMR-WB+ · EVRC · EVRC-B · EVS · GSM-HR · GSM-FR · GSM-EFR
	Others	ACELP · AC-3 · AC-4 · ALAC · Asao · ATRAC · CELT · Codec2 · DRA · DTS · FLAC · iSAC · Monkey's Audio · TTA (True Audio) · MT9 · Musepack · OptimFROG · OSQ · QCELP · RCELP · RealAudio · RTAudio · SD2 · SHN · SILK · **Siren** · SMV · Speex · SVOPC · TwinVQ · VMR-WB · Vorbis · VSELP · WavPack · WMA · MQA · aptX

上图展示的是语音/音频编码总表，可以看到其比视频编码要复杂得多，单纯的算法也远远比视频要更加复杂。

2. 数字语言基本要素



数字声音具有三个要素：采样率、通道数、量化位数。

3. 为什么要压缩

为什么要压缩

长度为4分钟，采样频率为44100Hz，采样深度为16bits，双声音 Wav文件大小： 44100Hz * 16 bits * 4 minutes * 2 = (44100 / 1 second) * 16 bits * (4 minutes * (60 seconds / 1 minutes) * 2 = 705600 bits / second * 240 seconds = 169344000 bits = 169344000 / (8 bits / 1 byte) * 2 = 42336000 bytes = 42336000 / (1048576 / 1M) bytes ≒ 40.37MB

MP3 128kbps压缩后文件大小：
128kbps * 4 minutes = (128k bits / 1 second) * (4 minutes * (60 seconds / 1 minutes)) = (128k bits / 1 second) * 240 seconds = 30720k bits = 30720k bits / (8 bits / 1 byte) = 3840k bytes = 3840k / (1024k / 1M) bytes = 3.75M bytes = 3.75MB

1、存储和带宽二大因素决定了语音压缩的必要性；

2、压缩理论上的支持是语音压缩的充分条件；

压缩音频，主要是为了在降低带宽负担的同时为视频腾出更多带宽空间。

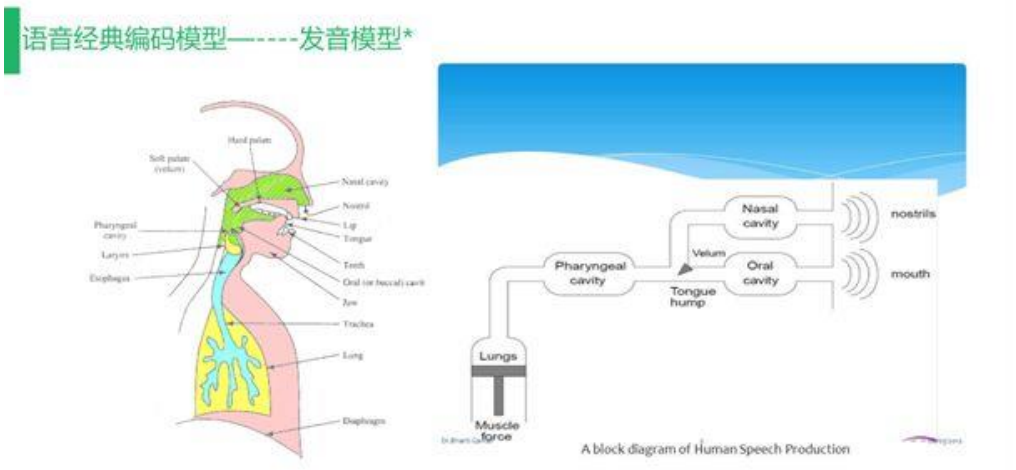
4. 编码器考虑因素

编码器考虑的因素

- 最佳压缩比
- 算法的复杂度
- 算法延时
- 针对特殊场景下的特定设计
- 兼容性

通过一些特定的压缩算法，可以压缩音频文件至原来的 1/10，同时人耳也无法分辨压缩前后的声音质量差异，需要满足多种条件才能实现这种效果；而对于编码器，无论是设计阶段还是使用阶段，我们都需要考虑最佳压缩效果、算法的复杂度与算法的延时，结合特殊场景进行特定的设计；而兼容性也是我们不能不考虑的重点。

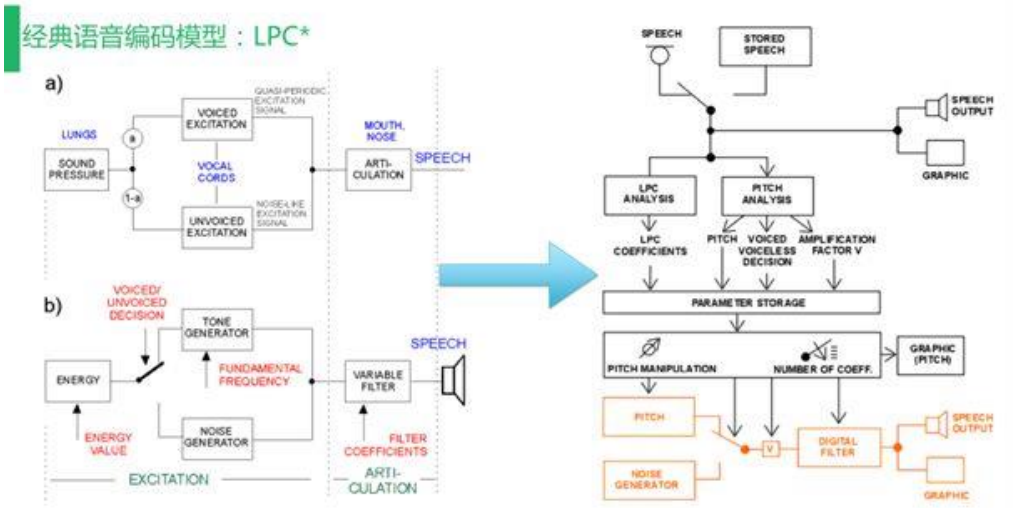
4.1 语音经典编码模型——发音模型



* http://mirlab.org/jang/books/audioSignalProcessing/humanVoiceProduction.asp?title=3-3%20Human%20Voice%20Production%20(%A4H%C1n%

我们的很多编解码器都是基于综合人的发音模型与一些和听觉相关的理论支持研究提出的特定编解码算法。初期我们通过研究人的发音原理来设计音频编解码的算法，包括端到端的滤波或轻浊音等，只有充分理解人的发声原理我们才能在编解码端做出有价值的优化。

1）LPC



* https://www2.spsc.tugraz.at/add_material/courses/scl/vocoder/tech.html

LPC 数学表达：

思想x[n]之前的N个样本的线性组合来预测x[n],

$$y[n]=\sum_{i=1}^{N}a_i x[n-i]$$

其中y[n]就是x[n]的线性预测。预测误差为:

$$e[n]=x[n]-y[n]=x[n]-\sum_{i=1}^{N}a_i x[n-i]$$

线性预测的目的就是寻找使得误差平方和函数最小的最优预测系数：

$$E=\sum_{n=0}^{L-1}[e[n]]^2=\sum_{n=0}^{L-1}\left[x[n]-\sum_{i=1}^{N}a_i x[n-i]\right]^2$$

对于一个N阶滤波器，滤波器系数ai通过求解一个N*N线性系统Ra=r求得，其中：
如右上角公式：

R（m）为信号x[n]的自相关：

$$R(m)=\sum_{i=0}^{N-1}x[i]x[i-m]$$

$$\mathbf{R}=\begin{bmatrix} R(0) & R(1) & \cdots & R(N-1) \\ R(1) & R(0) & \cdots & R(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(N-1) & R(N-2) & \cdots & R(0) \end{bmatrix}$$

$$\mathbf{r}=\begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(N) \end{bmatrix}$$

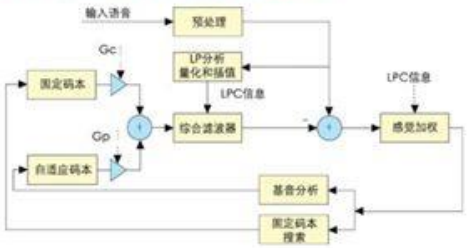
因为R是Hermitian Toeplitz矩阵，
所以可以利用Levinson-Durbin算法，使得复杂度由O(N^3)降低为O(N^2)

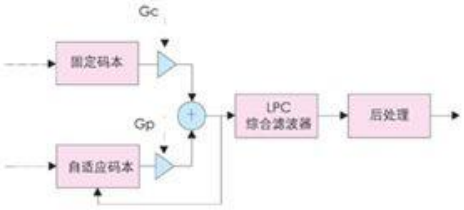
LPC以及改进算法是很多Codec基石，例如：SILK，MPEG-4 ALS，FLAC，G7xx

LPC 作为经典语音编码模式，其本质是一个线性预测的过程。早期的 G.7 系列编码模型便是通过此模型对整个语音进行编码，上图展示的过程可与之前的人发声过程进行匹配，每个环节都有一个相应的模块用来支撑人发声的过程。其中使用了 AR 数学模型进行线性预测，此算法也是现在很多语音编码的重要组成模块。

2）G.729

经典语音编码模型: G.729(CELP)

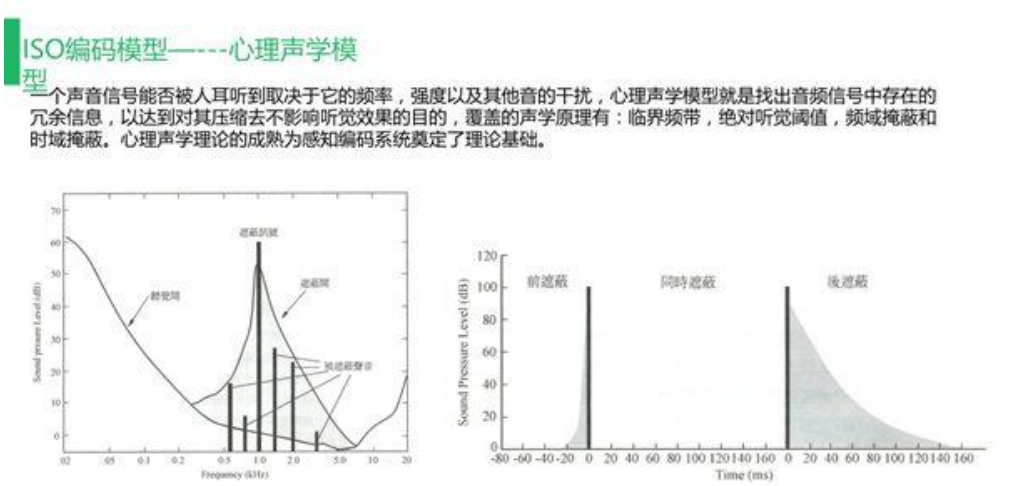




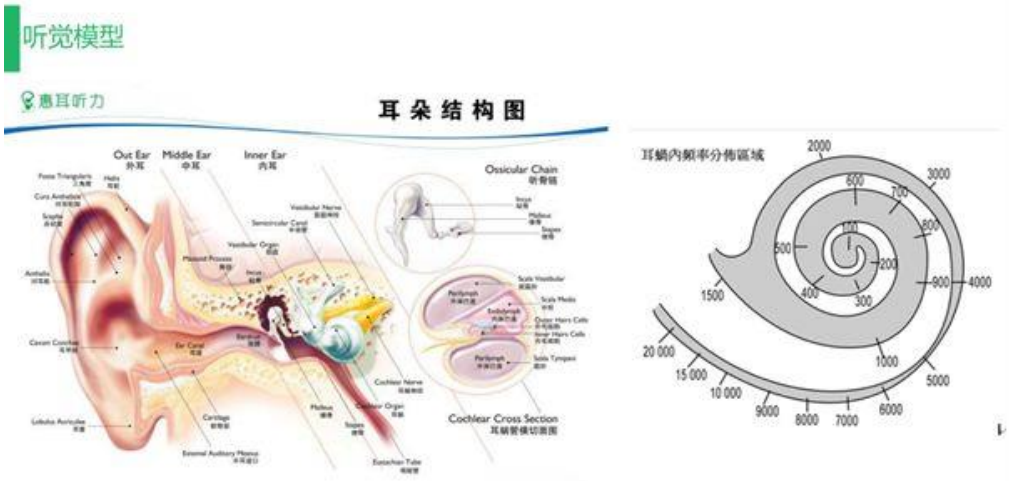
G.729 建议了共轭结构的算术码本激励线性预测(CS-ACELP)编码方案。
G.729算法的帧长为 10ms, 编码器含 5ms 前瞻，算法时延15ms，语音质量 MOS分可达4.0.

G.729 同样是经典的语音编码模型之一，也是我们学习语音编码的一个入门级 Codec。G.729 的文档十分完善，包括每个模块的源代码在内都可直接下载。G.729 可以说是在早期发声模型基础上的改进，需要关注的性能指标是帧长与算法上的延时，包括语音质量的 MOS 分。
G.729 也有很多变种，由于语音需要考虑系统兼容性，不同的系统指定携带的 Codec 也不同，音频编码的复杂程度要远高于视频编码。

4.2 语音经典编码模型——听觉模型



除了研究人发声的原理，我们还需要研究人听声的原理，从而更好实现声音的收集与处理。一个声音信号是否能被人耳听见主要取决于声音信号的频率、强度与其他音的干扰。心理声学模型便是用来找出音频信号中存在的冗余信息从而实现在压缩声音信号的同时不影响听觉的目的。心理声学理论的成熟为感知编码系统奠定了理论基础，这里的感知编码主要是 ISO 编码模型，主要覆盖的声学原理有临界频带、绝对听觉阈值、频域掩蔽、时域掩蔽等。



无论是 MP3 还是 AAC 以至于到后面的杜比音效都是基于听觉模型进行的探索与创新。

1）临界频带

临界频带

由于声音频率与掩蔽曲线不是线性关系，为从感知上来统一度量声音频率，引入了"临界频带"的概念。通常认为，在20Hz到16kHz范围内有24个监界频带。临界频带的单位叫Bark(巴克)

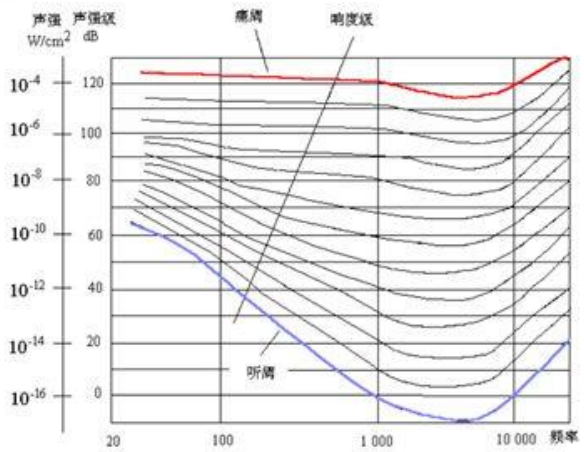
$$B(f) = 13\tan^{-1}(0.76\frac{f}{1000}) + 3.5\tan^{-1}(\frac{f}{7500})^2$$

临界频带	频率 (Hz) 低端	高端	宽度	临界频带	频率 (Hz) 低端	高端	宽度
0	0	100	100	13	2000	2320	320
1	100	200	100	14	2320	2700	380
2	200	300	100	15	2700	3150	450
3	300	400	100	16	3150	3700	550
4	400	510	110	17	3700	4400	700
5	510	630	120	18	4400	5300	900
6	630	770	140	19	5300	6400	1100
7	770	920	150	20	6400	7700	1300
8	920	1080	160	21	7700	9500	1800
9	1080	1270	190	22	9500	12000	2500
10	1270	1480	210	23	12000	15500	3500
11	1480	1720	240	24	15500	22050	6550
12	1720	2000	280				

临界频带主要用于心理声学模型。由于声音频率与掩蔽曲线并非线性关系，为从感知上来统一度量声音频率，我们引入了“临界频带”的概念。人耳对每段的某个频率的灵敏度不同，二者关系是非线性的。通常我们会将人可以听到的整个频率也就是从 20Hz 到 16KHz 分为 24 个频带，可在其中进行时域或频域类的掩蔽，将一些冗余信息从编码中去除从而有效提升压缩率。

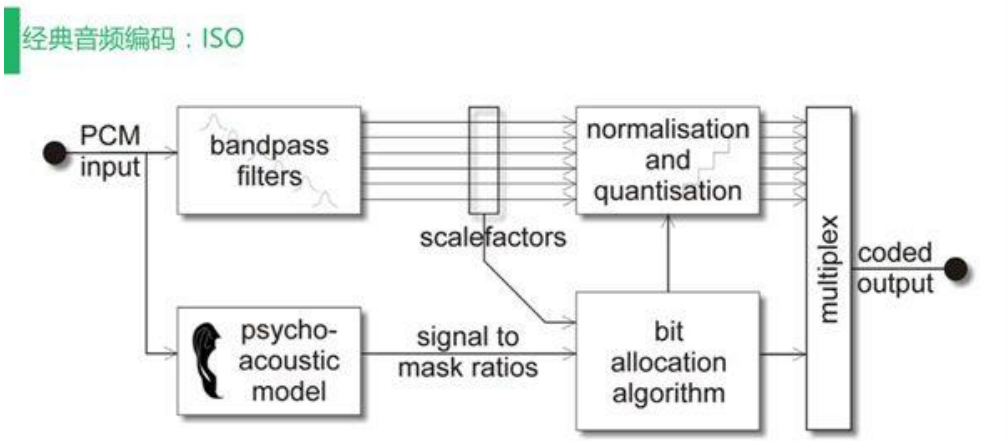
2）绝对听觉阈值

绝对听觉阈值

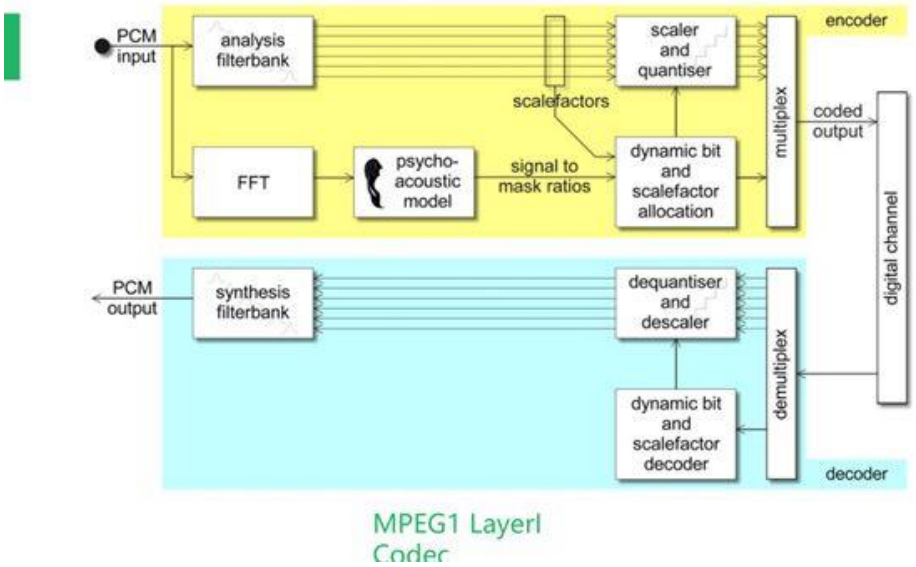


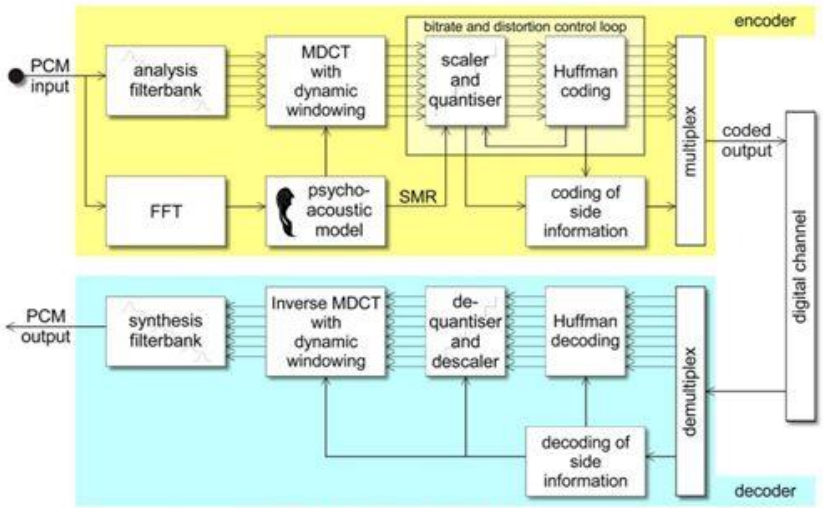
绝对听觉阈值也可有效提升压缩率，基于心理声学模型，可去除编码中的冗余部分。

4.3 经典音频编码：ISO



我们可将最早的 MP3 Layer1 理解为第一代的 ISO 感知编码，随后的一些纯量化内容更多的是在压缩上进行改进而核心一直未改变。从 MP3 Layer1 到 Layer2 与 Layer3，主要的改变是心理声学模型的迭代。

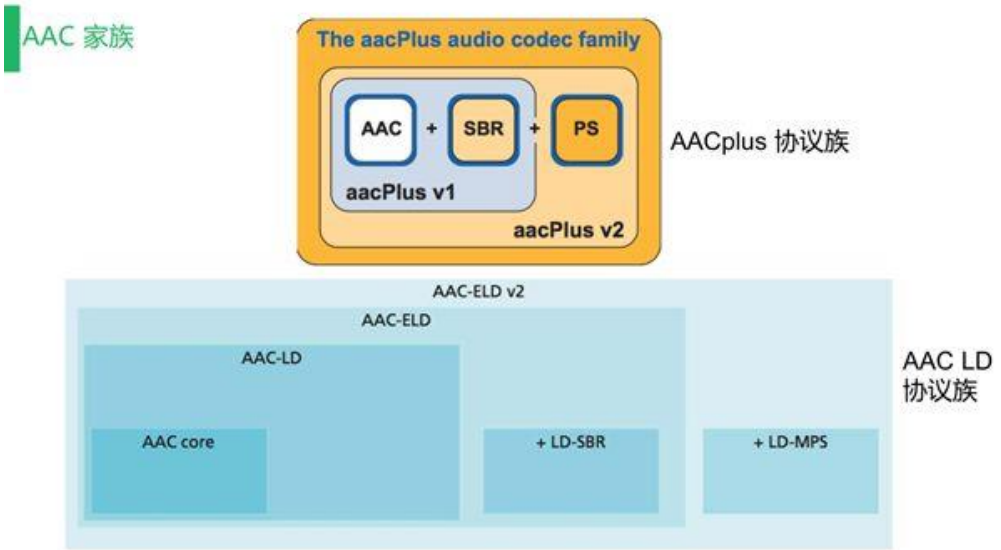




MPEG1 LayerIII Codec

上图展示的是 Encode 与 Decode 的回路。输入的 PCM 首先会经过多子带分析与频域中的心理声学模型冗余处理，而后进行量化编码；Layer III 中的是我们现在常说的 MP3 的 Codec：Encode 与 Decode 之间的整体回路，相比于 Layer1 多了几个处理环节以及霍夫曼编码。

4.4 AAC 协议族

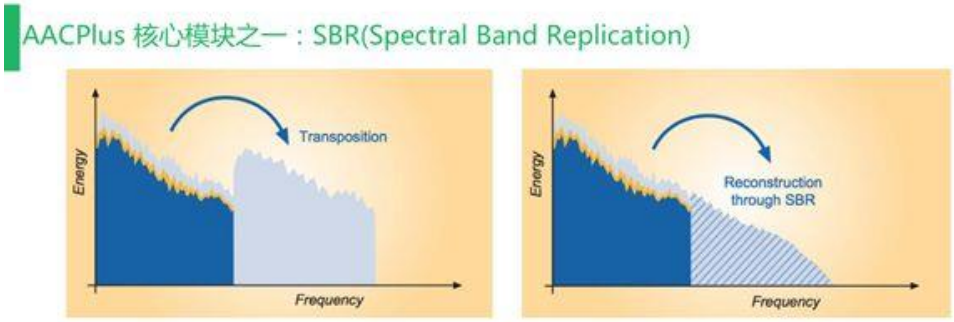


AAC 与 G.719 一样包括很多系列，但 AAC 的巧妙之处在于向下兼容的特性。开始时我们就强调，所有 Codec 在设计时都需要考虑兼容性，瑞典的 Coding Technology 公司曾提出在兼容性上特别优化的方案。AAC Plus V1 包括 AAC 与 SBR，AAC Plus V2 包括 AAC+SBR+PS，现在常见的很多音乐类或直播音频编码都是基于 AAC Plus 协议族进行的。

德国的霍朗浦学院曾在 AAC 低延时协议扩展方面做出一些探索并得到了 AAC LD 协议族，其原理仍基于传统的 AAC 模块，但在后端会对处理长度进行调整，例如之前是以 1024bit 为一

个处理单位，那改进后则以 960bit 为一个处理单位。除此之外 AAC LD 加入了 LD-SBR 与 LD-MPS 等，从而形成一个规模较大的 AAC-ELD V2 模块，可以说是十分巧妙。

1）AACPlus 核心模块——SBR（Spectral Band Replication）

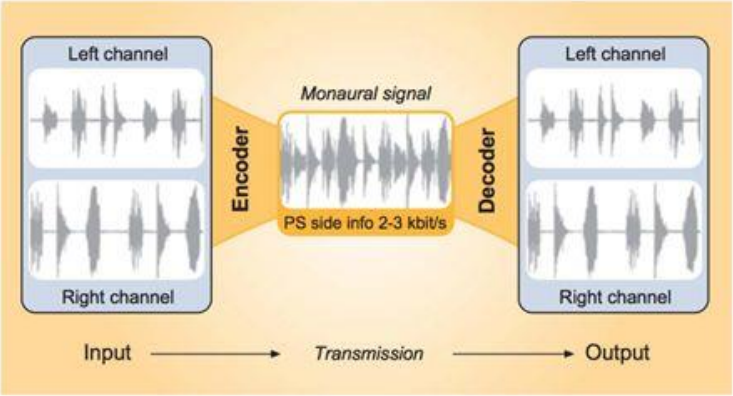


Stereo bitrate (bit/s)	AAC frequency range (Hz)	SBR frequency range (Hz)
20 000	0 - 4 500	4 500 - 15 400
32 000	0 - 6 800	6 800 - 16 900
48 000	0 - 8 300	8 300 - 16 900

我们可以看到，AAC 可以说充分利用了频域扩展，用很小的代价实现诸多功能优化。AAC 的核心之一是 SBR，这是一种使用极少位数就可描述高频部分并在解码时进行特殊优化从而实现频域扩展的模块。上图展示的是不同压缩率模块所覆盖的频率取值范围，而使用 AAC 时需要注意一个被称为“甜点码率”的指标。无论是采样率还是码率都是变化的，在应用时选择何种码率十分关键。例如直播时采用 64Kbps 即可在覆盖整个频段的同时保持良好音质。

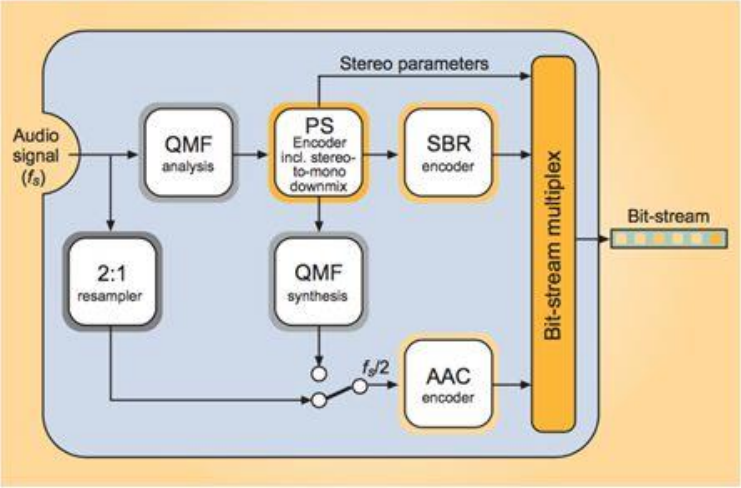
2）AACPlus 核心模块——PS（Parametric Stereo）

AACPlus 核心模块之二：PS(Parametric Stereo)

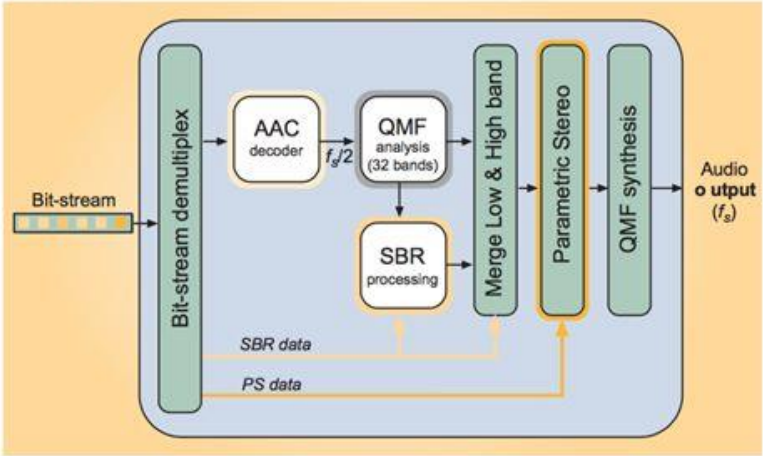


PS 描述参数：IID(Inter-channel Intensity Difference), ICC(Inter-channel Cross-Correlation) IPD(Inter-channel Phase Difference)

AACPlus v2编码框图



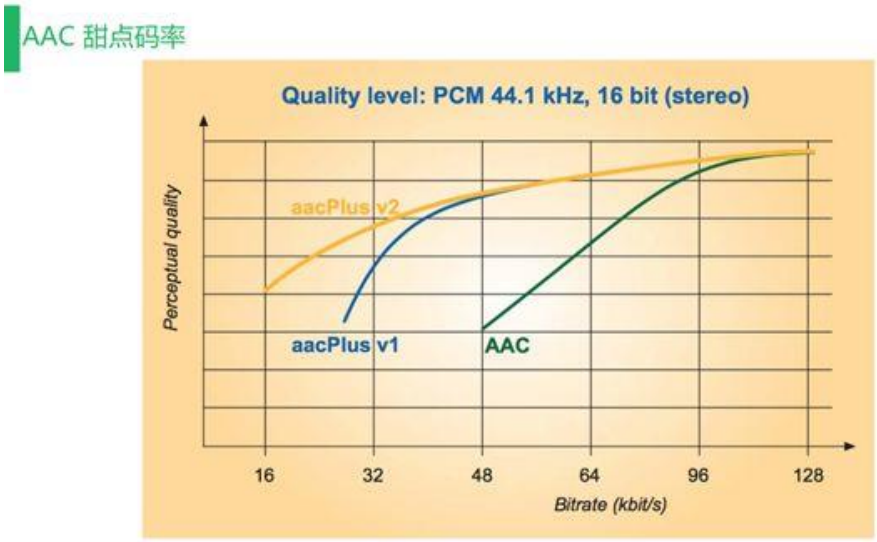
AACPlus v2解码框图



PS 模块也是 AAC 的核心模块之一，主要用于分析左右声道属性并使用非常少的位数表示左右声道相关性，而后在解码端将左右声道分离。这里比较巧妙的是 PS 的向下兼容特性，整体数

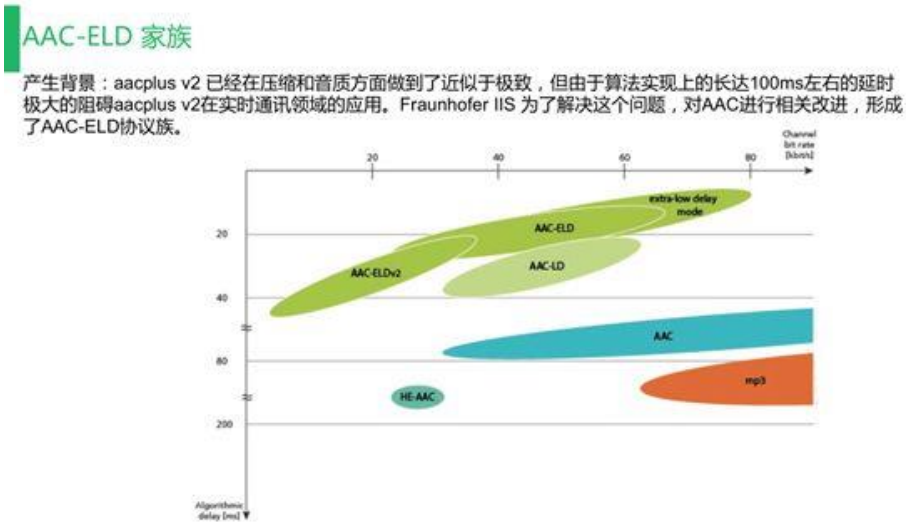
据打包是分开进行的。如果获取到 AAC、SBR、PS 三者的基本数据包后，在解码阶段我们就只需 AAC—LC。上图展示的就是 AAC 的解码框架，如果大家读过 3GPP 的代码就可发现其每一个模块都相当清楚。我们可根据文档读取代码并对应到每一个环节。

3）甜点码率



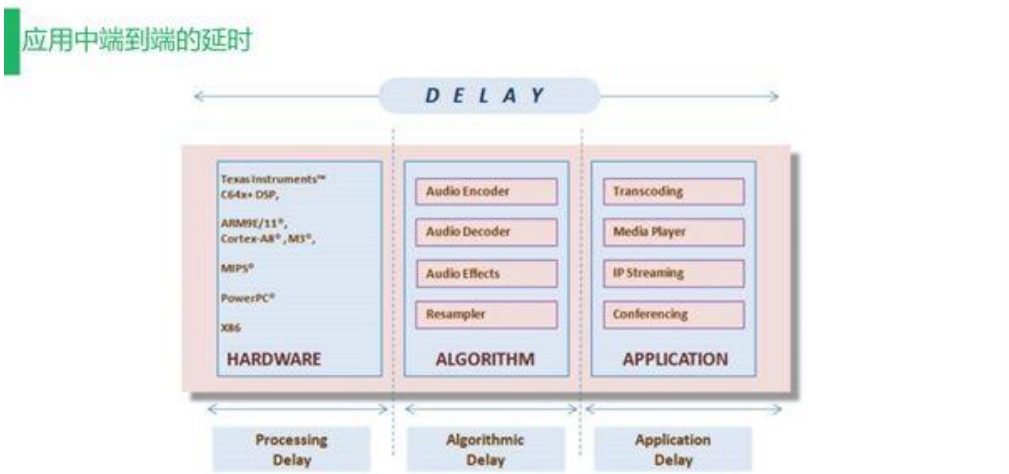
甜点码率是一项很关键的指标。例如在手机直播应用场景中，一般的视频分辨率为 640×360，音频码率大约在 800K 左右。如果音频码率过大则会直接影响视频质量，因而我们需要控制音频码率在一个较为合适的范围内从而实现最佳的音画效果。在很多应用场景中可能需要系统根据不同的网络环境下载不同音质的文件，例如在 2G 环境中下载较小的文件，这样做主要是为了节省带宽并提高音频文件的播放流畅程度。

4.5 AAC-ELD 家族



AAC-ELD 家族带来的主要改进是低延迟。如果 Codec 的延迟太长便无法在一些特定场景中被使用。例如早期 AAC Plus V2 的整体延迟可达 100ms，如此高的延迟肯定无法被应用于语音通话等对实时性要求极高的应用场景。霍朗普学院推出的 AAC-ELD 可在保持音质的前提下将延迟降低至 15ms，相对于 MP3 最高长达 200ms 的延迟而言提升巨大。

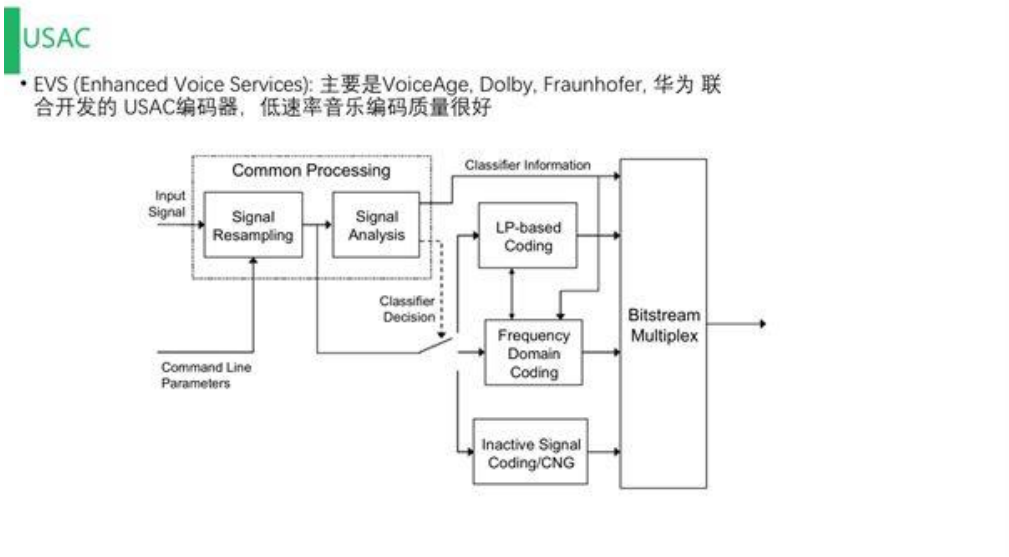
4.6 应用中端到端的延迟



编解码过程也存在延时问题，这也是我们选择编解码器时需要考虑的最主要因素之一，编解码的延时主要由处理延时与算法延时组成，例如 G.729 的算法延时为 15ms，而 AAC-LC 可达到一百毫秒以上。另外，播放端或采集端的长帧数量太多，播放时缓存太多等也会直接影响延时，我们在选择编解码器时需要考虑延时带来的影响。

编解码器已经历了两个发展方向：一个是以 G.7（G.729）为例，根据发声模型设计的一套主要集中于语音方面的编解码算法，另一个是以 ISO 的 MP3 和 AAC 为例，根据心理声学模型设计的一套感知编码。最近的趋势是编码的统一，原来在语音场景下我们使用 8K 或 16K 进行采样，音乐场景下则需使用覆盖到全频带的 44.1K 进行采样，每个 Codec 都有一个频域覆盖的范围。在之前的开发中，如果应用场景仅针对压缩语音那么需要选择语音编码方案，如果应用场景针对压缩音乐则需要选择音乐编码方案，而现在的发展方向是通过一套编码从容应对语音与音乐两个应用场景，这就是接下来将要被提到的 USAC。

这里介绍两个比较典型的 Codec：一个是 Opus，通过其中集成的模块可实现根据传入音频文件的采样率等属性自动选择语音编码或音乐编码；另一个是 EVS 这也是霍朗普等组织推行的方案，已经尝试用于 4G 或 5G 之中。



由框图我们可以了解到 USAC 向下兼容的特性。编解码器可总结为经历了三个时代：发声模型、听觉感知、融合方案。接下来我将展示目前所有的 Codec 情况并整理为表格以方便大家检索查阅。

5. Codec 总结

5.1 IETF 系列

IETF系列

	子系列	支持采样率	通道数	支持码率	甜点压缩比	备注
Opus	窄带单声语音 (SILK)	8kHz (128kbps)	1chs	变码率, 最佳码率: 11kbps	8.5%	Opus特点： 1. 支持码率：6~510kbps 2. 采样率 8kHz(Narrowband) to 48kHz(fullband) 3. 帧大小：2.5ms ~ 60ms 4. 支持CBR/VBR 5. 动态调整 6. 支持单声道和双声道，最多支持255个音轨 7. 抗丢包
	宽带单声语音 (SILK)	16kHz (256kbps)	1chs	变码率，最佳码率：20kbps	7.8%	
	全带单声语音 (CELT)	48kHz (768kbps)	1chs	变码率，最佳码率：32kbps	4.1%	
iLBC	20ms Frame	8kHz (128kbps)	1chs	15.2kbps	11.8%	iLBC特点： 1. 抗丢包 2. 音质较好，15.2kbps Mos 4.14
	30ms Frame	8kHz (128kbps)	1chs	13.3kbps	10.3%	
GSM	FR (LPC-PRE)	8kHz (128kbps)	1chs	13kbps	10.1%	帧块大小20ms, 蜂窝电话用的编码器
	HR (VSELP)	8kHz (128kbps)	1chs	5.6kbps	4.3%	

IETF 作为标准协议联盟组织之一推出了以上 Codec：Opus 包括采样率为 8kHz、甜点码率为 11kbps 的窄带单声语音（SILK），采样率为 16kHz、甜点码率为 20kbps 的宽带单声语音与采样率为 48kHz、甜点码率为 32kbps 的全带单声语音（CELT），采用甜点码率意味着将压缩率和音质保持在一个良好的平衡状态。在一些窄带单声语音应用场景例如常见的微信语音聊天，其压缩率可达到原来的 8.5%。

Opus 没有技术专利和源代码的门槛，使得其受到现在很多流媒体厂商的欢迎，Opus 支持更广的码率范围，具备丰富采样率选择，可实现极低延迟与可变帧大小，也具备以往一些 Codec 的许多特性如 CBR、VBR、动态调整等，支持的通道数量也更多。除此之外，Opus 同样具备许多从 SILK 移植而来的特性或功能。如在 VUIB 传输上集成了扛丢包模式等。

iLBC 早在 SILK 未出现时就被提出同样具备抗丢包。的特性，高达 15.2kbps 的甜点码率与 4.14 的 Mos 使其音质较为良好，超过 G.729 的相关指标；GSM 就是最早手机网络仍停留在 2G 时代时流行的编码形式，主要用于蜂窝电话的编码任务。

5.2 AMR 系列

AMR系列

	原始码率	压缩码率	采样率	通道数	最大压缩比	备注
AMR-NB	128kbps	4.75kbps, 5.15kbps 5.9kbps, 6.7kbps 7.4kbps, 7.95kbps 10.2kbps, 12.2kbps	8kHz	1	3.7%	AMR-NB 是最流行的语音编码器，因为压缩效果好，且支持多种码率形式，同时满足GSM和3G网，且是Android系统最早支持语音编码器
AMR-WB	256kbps	6.6kbps, 8.85kbps 12.65kbps, 14.25kbps 18.25kbps,19.85kbps 23.05kbps, 23.85kbps	16kHz	1	2.5%	AMR-WB 是AMR-NB 向宽带的扩展版，主要用在3G和4G通话标准协议中。
AMR-WB+		融合AMR-NB, AMR-WB 外，还有扩展	支持到48kHz	2		第一个支持语音和音频的混合编码，原本3GPP是想通过AMR-WB+ 来对抗AAC，但因为各种原因，没有推广开来

AMR 早在 3G 时期就被广泛应用，AMR-NB 是最流行的语音编码器，具有压缩效果好，支持多种码率形式的特点；与此同时，这也是 GSM 与 3G 时期 Android 平台最早支持的窄带语音编码方案。AMR-WB 作为 AMR-NB 向宽带的扩展版，主要用于 3G 和 4G 通话标准协议中，其甜点码率为 12.65kbps。在实践中我们将码率参数调整为此值即可实现压缩率与质量的平衡。AMR-WB+则是上述两者的融合，三者共同构成 AMR 系列。

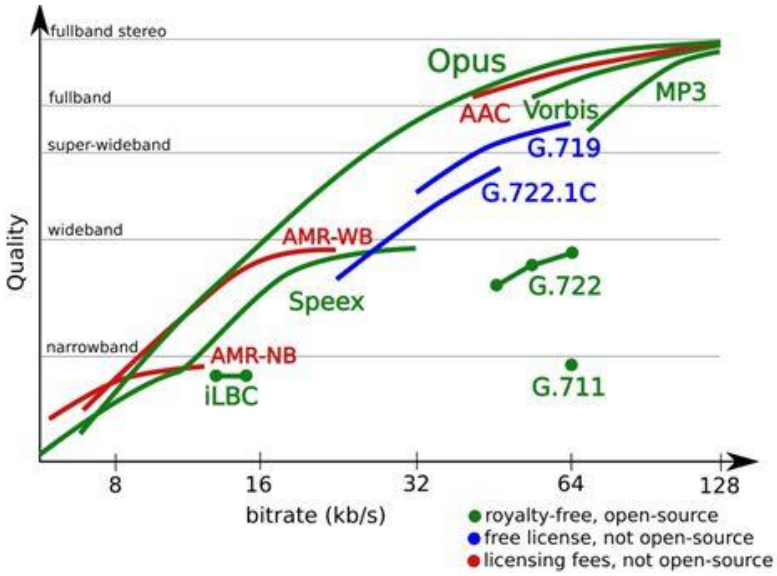
5.3 ITU-T G 系列

ITU-T G系列

G系列	原始码率	压缩码率	压缩比	备 注	SR/Stereo
G711	128kbps	64kbps	2:1(50%)	最早电话语音压缩算法，目前基本上不用，但会作为VOIP项目互连互通必选项	8kHz/No
G721	128kbps	32kbps	4:1(25%)	波形编码ADPCM，音质略次于PCM编码，可懂度和自然度都很不错	8kHz/No
G722	256kbps	64/56/48kbps	5.3:1(18.75%)	G.722最后被G.726收编	16kHz/No
G.722.1 Siren7	256kbps	32/24/16kbps	16:1(6.25%)	Polycom 第三代Siren7 压缩技术，20ms 封帧，提供40ms的算法延迟 Microsoft office communications Server(audio conferencing)	16kHz/No
G.722.1C Siren14	512kbps	48/32/24kbps Mono	21:1(4.68%)	2005年单声道Siren14变成ITU-T G.722.1C, 20ms封装，提供40ms的算法延迟	32kHz/Yes
G.719 Siren22	768kbps	64/48/32kbps Mono	24:1(4.16%)	2008年Siren22变成ITU-T G.719 全频带Codec	48kHz/Yes
G.723.1	128kbps	6.3/5.3kbps	24:1(4.16%)	H.324 语音编码器，30ms 封帧，37.5ms 算法延迟	8kHz/No
G.726	128kbps	40/32/24/16kbps	8:1(12.5%)	在G.711基础上进一步压缩 早期的视频会议用得较多	8kHz/No
G.729/G.729A/ G.729AB	128kbps	8kbps	16:1(6.25%)	高效的窄带语音编码器，CS-ACELP， 10ms 封装，Algorithmic delay is 15ms per frame, with 5ms look-ahead delay	8kHz/No

注：平时视频会议Polycom系统一般是用Siren14或22

ITU-T G 系列包括最早的波形编码 G711 到现在大家熟悉的 G.729 这里我想强调的是 G722.1 Siren7、G722.1c Siren14 与 G719 Siren22，例如 G.719 可覆盖整个前频带且支持立体声。即使都属于老协议，但由于其优秀的兼容性，不应被我们忽略。



将 Opus 与其他一些 Codec 进行对比我们可以看到，无论是质量还是延时控制，Opus 的优势十分明显；加之 Opus 作为开源的免费方案，不存在专利限制，受到业界追捧也不足为奇。

5.4 ISO 系列

ISO 系列

	子系列	支持采样率	通道数	支持码率	甜点压缩比	备注
MP3	MPEG-1/MPEG-2 audio Layer III	32kHz 44.1kHz 48kHz	2chs	32,40,48,56,64,80,96,112,128,160,192,224,256,320kbps	11:1	128kbps 达到CD音质
	MP3 Pro (MP3+SBR)	32kHz 44.1kHz 48kHz	2chs	码率能达到上面的一半	22:1	因为高频是通过算法生成，会在听觉上产生不自然。另外因为专利和应用上推广的限制，MP3 Pro实际上没有大规模应用。
AAC	AAC-LC	8～96kHz	最多支持48	48kbps ～ 400kbps	甜点码率：96kbps 6.8%	96kbps aac 质量相当于MP3 128kbps, 同样达到CD音质
	HE-AAC	8～96kHz	最多支持48	20～192kbps	甜点码率：32kbps 2.3%	主要用于<80kbps 的编码，由于采用SBR+PS, 在同质量情况下，压缩率可以做到最低
	AAC-LD/ELD	8～96kHz	最多支持48	32~64kbps	4.6%	主要是做到了AAC的低延时，最小延时做到7.5ms, 达到了延时和压缩比的最佳平台

ISO 里我想强调的是 MP3 与 AAC，二者同样支持很多码率。MP3 的甜点码率为 128kbps，MP3 Pro 的码率可达到 MP3 的一半；AAC 支持 8～96khz 的采样率，AAC-LC 的甜点码率为 96kbps，HE-AAC 的甜点码率为 32kbps，AAC-LD 与 ELD 做到了 AAC 的低延时，实现了延时与压缩比的最佳平衡。

5.5 3GPP 系列：EVRC

3GPP系列:EVRC

	EVRC	EVRC-B	EVRC-C	EVRC-D	EVRC-E
Bandwidth	200~3400Hz	200~3400Hz	50~7000Hz	50~7000Hz	50~7000Hz
Standardized	TIA 1997	3GPP2 2006	3GPP2 2007	3GPP2 2009	3GPP2 2011
Technology	RCELP/ACELP	CELP/PPP/NELP	CELP/LPC/PPP/NELP	CELP/LPC/PPP/NELP	CELP/LPC/PPP/2K/NELP
Bitrate(kbps)	6	4 to 8.3	4 to 7.5	4 to 7.5	2 to 7.5
Delay(ms)	23	23	23	23	23
Frame(ms)	20	20	20	20	20
Lookahead(ms)	3	3	3	3	3
Quality	Narrowband	Narrowband	Wideband	Wideband	Wideband
Complexity	30mis	40mips			
ROM(Bytes)	200k	200k			
RAM(bytes)	20k + 10k/ch	20k + 10k/ch			

EVRC 是CDMA 中使用的语音编解码器，由高通公司1995年提出目标是取代QCELP

高通公司主推的 3GPP 是 CDMA 中使用的语音编解码器，在未来选择编解码器类型时我们需要特别考虑延时与帧长。由于语音编码种类很多，帧长也是复杂多变的，其背后的算法复杂程度，RAM、ROM 占用等都是在实践当中需要着重考虑的。

5.6 极低码率

极低码率

	原始码率	压缩码率	压缩比	备 注
MELP	128kbps	2.4kbps	1.8%	MELP 美国军方语音编码标准, 22.5ms 帧封装
MPEG-HVXC	128kbps	2kbps, 4kbps	1.5%	MPEG4 中的低极码充语音编码算法，用于对讲机
IMBE	128kbps	4.15kbps	3.2%	IMBE是MBE改进算法，个别洲的对讲机codec协议组成员
AMBE	128kbps	4.8kbps	3.75%	目前对讲机主要的codec算法，DVSI公司专利算法，算法延时32ms

注：1) 极低码率应用场景：对讲机 / 卫星通讯 / 军工
2) 目前最极端极端低码率可以做到300bps, 相关于原数据的：0.2%， 这时只能传达语音的内容，很多声色部分都丢掉了。

极低码率主要的应用场景是对讲机、卫星通讯、军工等。图表中的 MELP 最早由美国军方开发，现在绝大多数的对讲机都基于此模型进行扩展开发，压缩后的码率可达到 2.4kbps 而目前最极端的极低码率可实现 300bps，相当于压缩为原数据的 0.2%，此时的音频文件仅能被用于传达语音内容而丢失了很多声色。

5.7 全频带

全频带

	分级	支持码率（kbps）	延时
EVS	Narrowband(NB) 8kHz	5.9, 7.2, 8, 9.6, 13.2, 16.4, 24.4	20ms 帧封装 编码延时：20ms + 0.9375ms (Resampling filter) + 8.75 (look-ahead) + 2.3125(time-domain bandwidth extension) 解码延时：30.9375ms (8kHz)
	Wideband(WB) 16kHz	5.9, 7.2, 8, 9.6, 13.2 , 13.2(通道感知) 16.4, 24.4, 32, 48, 64, 96, 128 (6.6~23.85 for AMR-WB IO)	
	Super-Wideband(SWB) 32kHz	9.6, 13.2, 13.2(通道感知) 16.4, 24.4, 32, 48, 64, 96, 128	
	Fullband (FB)	16.4, 24.4, 32, 48, 64, 96, 128	

全频带中的组合也是多种多样。

6. 编解码使用注意

6.1 License

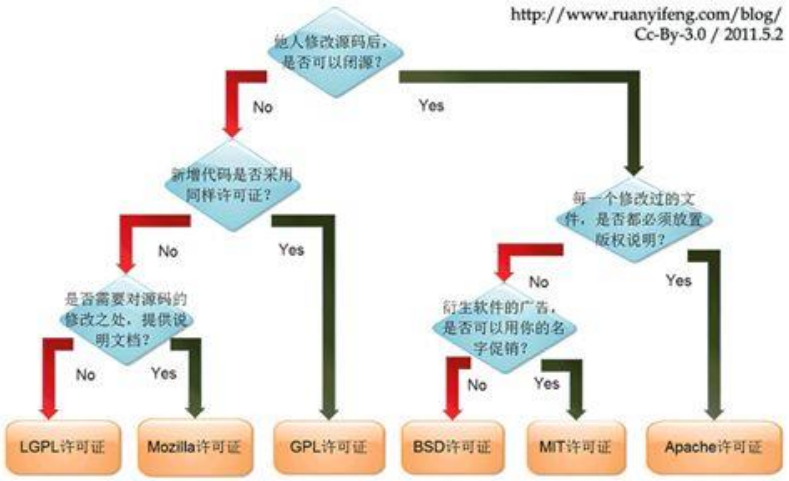
编解码使用注意License

开源项目常用License 协议:

Top 20 Most Commonly Used Licenses in Open Source Projects

Rank	License	%
1.	GNU General Public License (GPL) 2.0	42.49%
2.	MIT License	11.49%
3.	Artistic License (Perl)	7.84%
4.	GNU Lesser General Public License (LGPL) 2.1	7.13%
5.	BSD License 2.0	6.82%
6.	GNU General Public License (GPL) 3.0	6.41%
7.	Apache License 2.0	5.47%
8.	Code Project Open 1.02 License	2.14%
9.	Microsoft Public License (Ms-PL)	1.79%
10.	Mozilla Public License (MPL) 1.1	1.03%
11.	GNU Lesser General Public License (LGPL) 3.0	0.87%
12.	Eclipse Public License (EPL)	0.71%
13.	Common Public License (CPL)	0.42%
14.	zlib/libpng License	0.36%
15.	BSD Two Clause License	0.34%
16.	Common Development and Distribution License (CDDL)	0.32%
17.	Academic Free License	0.32%
18.	Open Software License (OSL)	0.22%
19.	Microsoft Reciprocal License (Ms-RL)	0.20%
20.	Ruby License	0.20%

国内大部分企业在开发时容易忽视包括专利安全性在内的与 License 相关的内容。如果企业计划得比较长远，需要长期使用某项技术或企业规模不断扩大时则不能不考虑专利问题。专利费用包括 Open Source 与算法专利，二者完全独立互不干涉，如果我们从某家专利公司购买了 AAC 的专利算法，并不能获得此 AAC 专利的源代码，仅能获得与此技术相关的专利使用授权。专利公司会给予需要下载的文件列表，通过这种方式实现技术的授权使用。



上面的二叉树图比较清晰地展示了代码授权的具体流程，随着企业的规模化发展日趋成熟，企业应当规范自身的技术使用行为，尽可能避免专利纠纷带来的不利影响。

6.2 专利

开源代码使用注意事项：
Source code files copyright and Technologies patent 区别
多媒体方面的技术专利简介:
二个多媒体技术的专利池:

MPEG-LA:
world's leading packager of patent pools for standards and other technology platforms used in consumer electronics, as well as chemical, eCommerce, education, energy, environment, healthcare and biotechnology, manufacturing and materials, transportation and wireless technology

Via Licensing:
Via Licensing 是致力于知识产权、富于经验的专家组，它代表第三方公司组织管理授权专案。Via Licensing 公司是杜比实验室公司 (NYSE:DLB) 的独立子公司，是一家拥有 40 多年技术授权经验的公司

主流语音编解码技术拥有两个专利池：MPEG-LA 与 Via Licensing。很多非常复杂的 Codec 涉及高达上千个专利，与之相关的企业或组织多达几十个，为专利授权而与每一个企业或组织进行洽谈显然是不现实的，因而专利池的出现使得技术授权更加规范清晰，方便企业统一处理技术授权问题。

6.3 常见 Codec Patent License

常见Codec Patent License

Proposed AVC License Terms, MPEG-LA Pool

Field	Internet	"Over-the-Air" Broadcast	Subscription Content	Packaged Media or Pay-per-view Content
Encoder and/or Decoder Product	$0.20/product $0.10/product above 5M units annually 100K Unit floor Paid by manufacturer of end-user product or PC operating system Caps: Total Enterprise/Entity: $3.5-5M/yr			
Use	Free for free content May be raised to local broadcast rate at license renewal	$2.5K per encoder used (one-time fee) – or – Per Market fee: Free < 100K subs. $2.5K/year < 500K $5K/year < 1M $10K/year > 1M	Free <100K subs. $25K/year < 250K $50K/year < 500K $75K/year < 1M $100K/year > 1M	Lesser of $0.02 per title sold or 2% of first sale price Free for short (<12min.) content Paid by content replicator or service/content provider
	Cap: Total Enterprise/Entity: $3.5-5M/yr			

- License through 2010, grace period through 2004 (Encoding and Decoding) or 2005 (Use)
  - In the U.S. this would exempt only the 40 smallest markets out of 210 Nielsen DMA's, the largest of these being Billings, MT or Dothan, AL.

Proposed AVC License Terms, Via Pool

Field	Internet	"OTA Broadcast"	Subscription Content	Packaged Media or Pay-per-view Content
Encoder and/or Decoder Product	$0.25/product $0.0025/product for temporary decoder provided with content Exemption for free 30-day demos of PC software Paid by manufacturer of end-user product Caps: Total Enterprise/Entity: $2.5M/yr except PC OEM (non-retail) software at $4M/yr A PC software OEM can pay product royalties that cover the PC Hardware the software is shipped on.			
Use	Free			Per title sold (Free for free content) $0.005 <30 Min $0.015 <90 Min $0.025 >90 Min $0.0025 for < 30 days, 20 plays temporary content Paid by party that completes financial transaction and authorizes content delivery
Overall Floor	Less than 50K Units and $500K revenue – no fees or reporting			

- Initial Fee of $15K at start of license, "hardship terms" for entities with <$2M annual revenues
- License term 5 years, grace period through 2004

Codec	File Format/Type
H.264	mp4, 3gp, m4v, ts, h264, 264
Mpeg4	mp4, 3gp, mpeg4
H.263 (Count together with MPEG4)	mp4, 3gp, h263, 263
WMV (Talk to Microsoft)	avi, asf, wmv
Motion Jpeg	avi
DivX	avi (DivX3 and before. For DivX4 and after, only if Mpeg4 compliant)
MP3 (Talk to Thomson)	mp3
AAC/AAC+/eAAC+	mp4, 3gp, aac, m4a, m4v
AMR NB/WB/WB+	mp4, 3gp, amr
WMA (Talk to Micrsoft)	avi, asf, wma, wmv
MIDI	mid
JPEG	jpg

Red word – Need to pay license and each license fee detail will be describe in the below.
Codec portion include H.264, Mpeg4(H.263) , AAC , AAC+ , eAAC+ , AMR
File format portion include **mp4, 3gp, m4V** (Generally called **MPEG4 file format**).

Blue word – Need to pay license but Royaltek should have the experience, so I will skip this portion.

希望大家在使用技术的同时尊重知识产权，助力技术创新可持续发展。

多媒体开源技术栈

开源流媒体服务器：为何一定得再撸个新的

作为开发者，我们需要有一个服务器来支持新视频行业的互联网化，有哪个开源方案能支持新爆发的业务？该方案需要支持哪些关键的能力或需求？本文由自阿里云 RTC 服务器团队负责人杨成立在 LiveVideoStack 线上分享的内容整理而成。

杨成立　LiveVideoStack 线上分享　2020-06-02

大家好，我是来自阿里云的杨成立，本次分享将会和大家详细介绍开源流媒体服务器的关键技术及未来发展。

我从 2009 年开始从事 FFmpeg 流媒体相关工作，2012 年开始参与流媒体服务器的开发，2013 年开始做开源流媒体服务器 SRS，至今也有七年多的时间了。在这短短几年间，随着视频直播的爆发，SRS 也迎来了快速增长。2017 年我来到阿里云之后换成了 WebRTC 方向。我们可以看到目前 Live WebRTC 在整个视频的应用非常广，包括在线办公、在线教育、在线娱乐等各个行业都大展拳脚，音视频已经成为当前互联网交流与信息传播不可或缺的媒介手段。

这次的分享内容将主要围绕 SRS 的诞生与历程、SRS 接下来的发展计划等，带领大家深入研究 SRS 存在的价值与意义。

得益于我国通信基础设施的日趋完善，尤其是 Wi-Fi、4G 网络的下沉普及，我国互联网市场音视频产品服务在 2015～2018 年经历了爆发式增长。当时消费者普遍拥有的可以观看视频的带宽大约为 1M，网络环境较为稳定。

直播背后的技术早在功能机时代就落地成熟。例如 2010～2012 年，消费者主要通过 PC 上的 Flash 来观看网络直播视频，因为 Flash 可以跨主流 PC 端浏览器。而移动端尽管也支持 Flash，但效果很不好。移动端如 Android 或 iOS 则主要支持 HLS，早期 Android 对 HLS 的支持效果并不佳，后面有明显改善。

关于视频行业的发展

PC时代，平台是Adobe Flash(1996~2017)
RTMP/FLV，和Apple HLS，流媒体服务器有Red5，
NGINX-RTMP, CRTMP, Wowza, AMS, Helix...

现在，移动端和IoT/5G时代来临，实时通信互联，我们需要SRT做远距离传输，GB28181支持物联网接入，WebRTC开展互动和在线沟通。

那么，有哪个开源方案能支持新爆发的业务？

无论是传统 PC 时代还是现在的移动互联网时代，流媒体中主要使用的协议都是 RTMP/FLV 与 Apple 的 HLS，流媒体播放器主要有 Red5、NGINX-RTMP、CRTMP、Wowza、AMS 等。大约是从 2017 年开始，Chrome 中的 Flash 播放器就已经处于默认禁用的状态，后续 Flash 也会逐渐退出互联网历史舞台。

随着互联网的发展，移动端直播逐渐兴起并且成为主流。例如当前移动办公平台中的互联网直播：主要调用的是 Native 播放器，使用 FLV 协议。浏览器端则多采用 HLS，二者共同构成了较为成熟完善的互联网直播协议体系。

随着移动基础网络的建设不断升级，移动端和 IoT/5G 时代来临，实时通信互联的需求日趋旺盛，Flash 被禁用之后出现了更加完善的替代方案——H5 播放器，其所使用的技术规范是 MSE。H5 播放器现可被绝大多数 PC 浏览器支持，同时 H5 也能播放 FLV 等格式。MSE 扩展和 Flash 比较相似，提供的是 JS 接口，将 FLV 或 HLS 等解封装，然后打包为 MP4 后，送到 MSE 接口中播放。H5 是替代 Flash 的标准方案，FLV、HLS、DASH 等都可以直接通过 MSE 播放。

除此之外，大家所追求的另一个方向是低延迟直播，一般传输协议其延迟可达十几秒，而 RTMP 可以将延迟降低到 3 ~ 5 秒，公网上的 TCP 有时会出现抖动，此时延迟会变大。
目前大家都在探索更好的降低直播延迟的方法，在此方面 WebRTC 是大家公认的理想解决方案。尽管 5G 可以带来更低的延迟，但从通信角度来说可用性更加重要。5G 网络的盛行意味着整个网络基础设施的稳定，有更多的通信设备可以达到相应要求。例如疫情期间使用视频直播的用户出现井喷式增长，而现有网络直播服务依旧未因此而出现重大宕机，这主要是得益于过去十多年的通信网络基础设施建设，以及整个开源环境、商业、云计算等领域的保障与进步相关。

目前 SRT、IoT 等的发展仍需面临很大挑战，特别是现在国内互联网的可能性不断丰富，直播产业生态环境也会趋于向好，新场景层出不穷。

首先，不同场景对网络基础设施以及整个商业环境的要求是完全不一样；其次，业务与开源往往是相互促进的，业务驱动新开源方案的不断落地实施，而开源方案也为业务提供技术支持；最后，不同行业之间往往存在很深的代沟，例如监控行业往往不会需要 APP，而直播行业也不会用到私有协议。我们需要 SRT 做远距离传输、GB28181 支持物联网接入，WebRTC 开展互动和在线沟通。

我们希望有一套开源的解决方案能满足不同行业不同场景下的低延迟直播需求。现在的云计算有融合的趋势，无论是 CDN 还是云计算都开始逐渐满足在线直播的需求。作为开发者，我们需要有一个服务器来支持这些新视频行业的互联网化，有哪个开源方案能支持新爆发的业务？该方案需要支持哪些关键的能力或需求？

若想实现此开源流媒体服务器，我们需要考虑诸多关键约束和能力。

首先就是该平台需要具有一定伸缩性，也就是足够的弹性。互联网业务可以从局部扩展到很大的领域，如果我们使用开源方案则需要清晰意识到如果业务规模变大之后，现有资源与经验能否支撑起如此大规模的服务运行，这需要很多开发者的维护与云厂商的支持。如果没有开源平台和云厂商的支持，那么我们只能自主搭建平台并部署服务器。对于很多企业来说，他们不可能有能力和资源开展这么多业务，所以开源方案至关重要。

开源的前提是必须要有云计算的支持，现在能看到的 CDN，包括阿里云和腾讯云等其实都支持 RTMP、FLV、HLS，并且现在也开始支持 WebRTC，在此基础上扩充生成了诸多商业落地应用，具备大规模应用的能力。我们自己基于开源方案搭建平台并将其对接到 CDN 上，即可妥善解决弹性问题。如果没有云服务的加持，开源平台的价值也无从谈起。

低延迟是我们需要注意的第二点。现在视频发展的一大趋势是低延迟，例如 TCP 类的协议其延迟可达 3~5 秒，这不仅仅是 TCP 协议本身所致。而像 HLS 切片、播放器延迟、编码延迟等都可能会提高延迟至 8～10 秒甚至更多。WebRTC 通讯场景延迟一般小于一秒甚至可达 400 毫秒。常见的语音沟通场景延迟高于 400 毫秒就需要人工对两个人的讲话进行同步。

第三点是搭建的服务平台需要具备较为出色的易用性。如 Red5、NGINX-RTMP、CRTMP、Wowza、AMS、Helix 等。还有一项关键是协议之间的互通，一个业务可能需要基于多个协议，打通其中的隔阂至关重要。若想快速部署该方案，以上三点至关重要。

1. Scenario

1.1 互联网直播与连麦

Domain	Tech	OpenSource	Commercial
Camera	Camera	Camera, Mobile Phone	
Video Codec	H.264	x264, openh264, FFmpeg	Hardware
Audio Codec	AAC	fdkaac, FFmpeg	Hardware
Encoder	System	OBS, FFmpeg	Haivision
Transport	RTMP, FLV, HLS	NGINX-RTMP, SRS	Wowza, AMS
Player	System	Mobile Phone, Chrome, Safari, Edge	
Interactive	RTC	WebRTC	

互联网直播和连麦的应用场景想必大家并不陌生，其中一些技术细节值得我们重点关注。例如在编解码方面，H.264 相对比较完善，而 PC 等设备上则有硬件编解码。商用编解码方面，比如国内的虹软、国外的 HaiVision 等，包括一些广电行业也有其自己的编解码器。除了编解码，再往上如推流 OBS、FFmpeg 等则主要被集成在系统当中。如果从主播端直接推流，那么基于 OBS 修改的方案比较多。

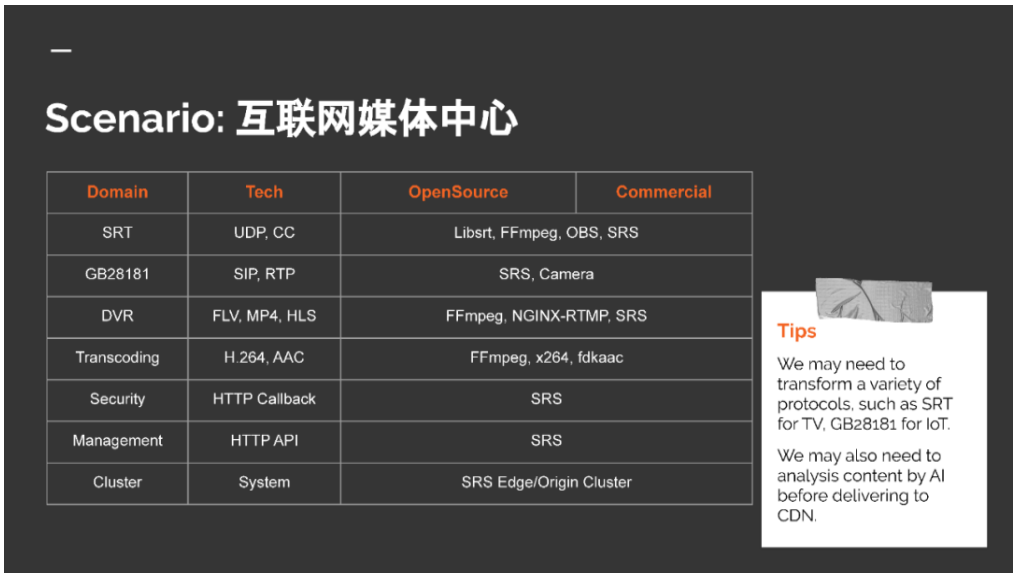
传输方面，我们需要把内容分发给许多观众，这一块的开源方案有 NGINX-RTMP 与 SRS 等，商业解决方案有 Wowza 和 AMS 等，商业解决方案更多是直接通过 CDN 网络直接进行分发。

播放器中的方案则主要是 H5 播放器，设备中大多会集成播放器来实现编解码，当然现在还有开源 SDK 来实现这一需求。直播连麦主要通过 RTC 与 WebRTC 的交叉功能来实现。

1.2 互联网实时通信

互联网实时通信的典型应用范例是视频会议，视频编解码方面与上一场景类似。但音频编解码方面，互联网直播多采用 AAC 而互联网实时通信则使用 Opus，因为 Opus 的延迟更低。客户端包括推流与播放主要是 WebRTC 框架，推流与播放需要服务器，才能把流分发给很多人。此时大家会发现这里的服务器和前文提到的直播服务器完全不同，其支持 Janus、Mediasoup、OWT 与 SRS 等。在线会议场景中有一个特殊的应用就是等格式，我们希望实现电话之间的互联互通，这一块的开源方案是 FreeSWITCH，其本身就是一个庞大的系统。

1.3 互联网媒体中心



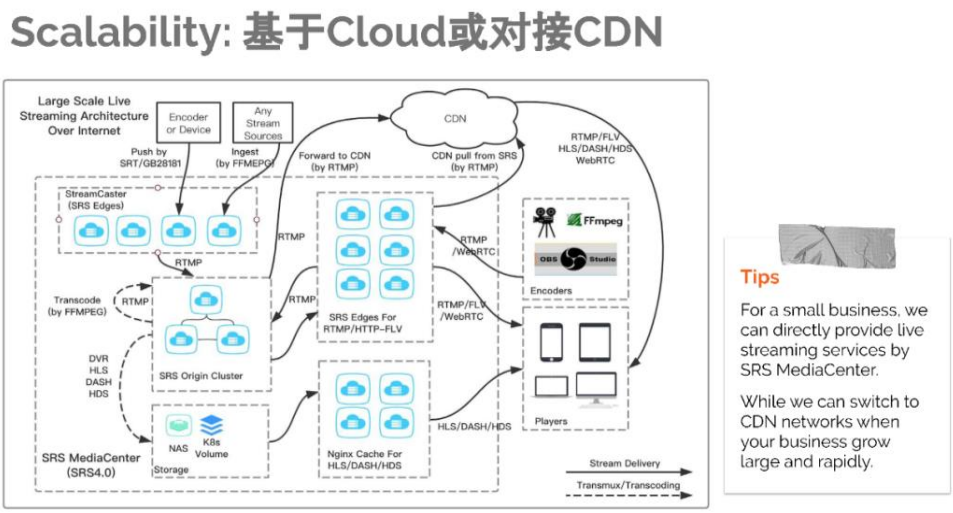
互联网媒体中心作为一大应用场景，主要用于对内容的管控。例如当需要录制视频时，我们希望该视频可以被反复观看，如录制好的培训课程。还有一些内容并不会被频繁重复观看，如国

庆直播、足球赛直播等，这里就需要设计对于已录制内容的妥善管控，如不良内容鉴定、自动剪辑等。媒体中心的设计和内容强相关，其需要包括转码、编码、存储等全流程的加持。传统方案是将媒体流传输给多个 CDN 或者借助 CDN 将流分发给多个 CDN。此方案本身非常浪费资源，更好的方案是建立一个媒体中心。

安全方面，CDN 也有播放的鉴权，例如限制一定的参与人数，对内容进行加密等，Token 也是一种鉴权方式。除此之外，我们还需要一种接入标准如 GB28181《安全防范视频监控联网系统信息传输、交换、控制技术要求》，尽管其属于一种标准，但非常私有，CDN 不太好去支持该标准。云计算 CDN 更适合去做标准的事情，基础设施、分发等都需要标准来规范。如果接入协议非常私有，那么自建一套媒体中心更符合企业需求。将内容转换成标准协议发送至 CDN 或者其他企业，相对而言更加容易实现互联网化。

在特殊场景下，如为跨国直播设计的远距离传输当中，数据大多是通过专项网络进行传输，也可以走互联网或 SRT。这些特殊的场景与业务强相关，并不太适合用统一的标准来框定，其规模不足够大也不能够成为标准。

2. Scalability——基于 Cloud 或对接 CDN



上图展现了基于云部署或对接 CDN 的部署图，SRS 的 Demo 网络就是这么部署的。其主要用 K8s 来部署，也可以用二进制来部署，包括边缘集群、媒体中心、源站等。流的输入端会退回非标准协议下的传输流，并将标准的 RTMP 流推送到源站。随后沿边缘 CDN，经过 RTMP、FLV 等标准协议进行分发，如果规模不够大则直接从云机房当中播放分发，即便是切片协议也可以通过 NGINX 分发。因为数据可以对堆积到 CDN，所以该系统具备伸缩能力。协议主要通过 RTMP，也可以通过 CDN，CDN 现在也支持 WebRTC，也可以通过 RTC 来对接，但 RTC 中有很多私有的东西，未来 RTC 可以走 CDN，但还需要一定时间才能实现。

3. Latency

3.1 实时流媒体直播

关于延迟，SRS 现在支持了 WebRTC 的播放，推流很快会被支持。上图视频画面显示的是一个时钟，OBS 抓取时钟运行的画面。OBS 本身有 100 毫秒左右的延迟，通过 RTMP 和 WebRTC 播放器播放该时钟运行画面，可以看到时钟指示数字出现明显不同，这也体现了二者的延迟差异。

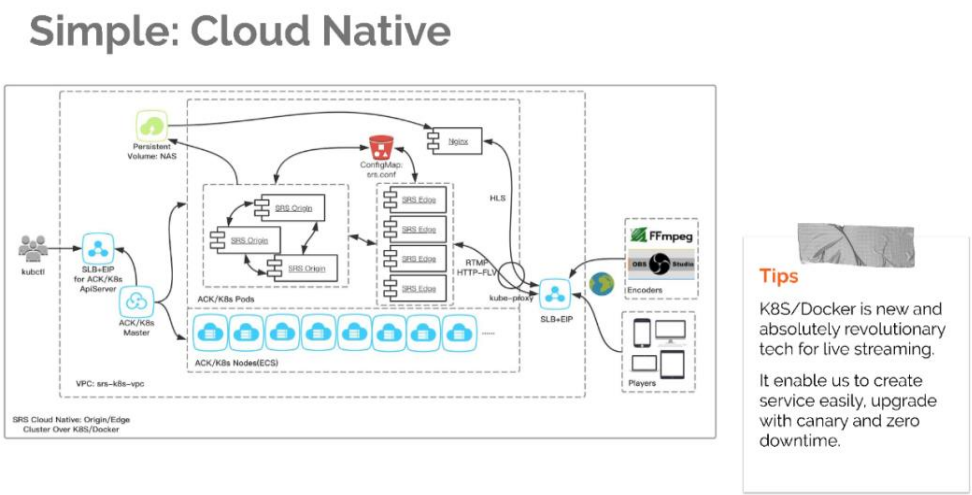
3.2 实时流媒体系统

对 GB28181 进行测试，从上图实验结果我们可以发现，HIKVISION 监控内网摄像头延迟为 280 毫秒、阿里云服务器 WebRTC 延迟约为 210 毫秒、阿里云服务器 RTMP 延迟 1100 毫秒。我们可以看到 WebRTC 服务器比内网监控摄像头的延迟还要低，出现这种情况主要是因

为延迟并不单纯是网络问题。该场景下 WebRTC 延迟比监控还要低，且具备场景下载的能力。监控大多会走私有传输路线，传统方案若想正常播放则需要安装 IE 插件。而通过标准协议大家想要从手机端看到，手机只需要直接集成 SDK，同时浏览器也可以直接看到画面，这样不需要安装任何插件，各个摄像头的流都可以看得到。

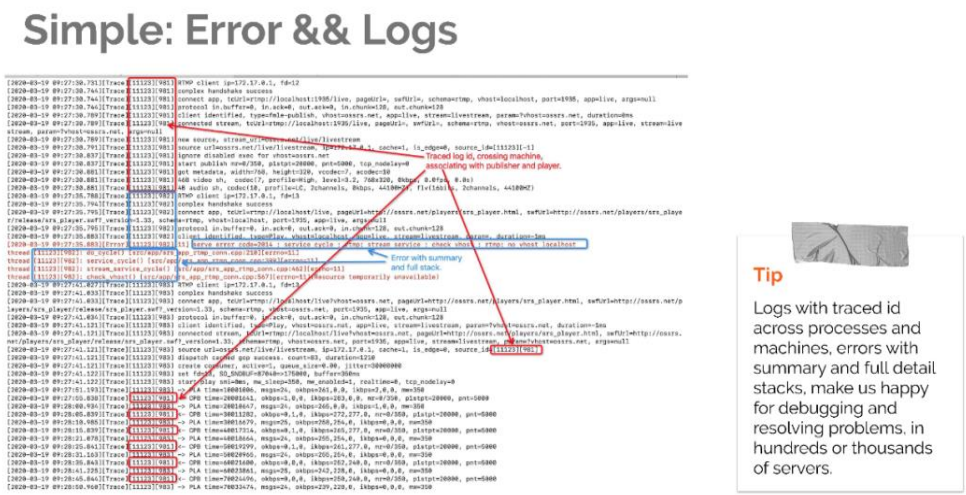
4. 举例

4.1 Cloud Native



第三部分是部署，SRS 支持 K8S 和 Docker 部署，包括我们每个新发布版本都会支持 Docker。上图主要展现了如何部署 K8S，这里我就不再赘述，大家可以仔细观看相应文档。以前我们主要通过二进制安装包来部署，其实 SRS 有多个镜像仓库，可以加速代码下载。仓库小下载速度也很快，编译、安装、启动等相对容易，其实 Docker 的部署方式更容易。最近陆陆续续有一些朋友反馈说编译存在一些问题，但 Docker 实际上可以在任何平台上部署，例如 windows 就完全可以部署 Docker 并运行，ARM 平台上则可以交叉编译。有时需要我们解决的问题会比较多，但如果使用 ARM 的 Docker 编译就没有问题。因为 Docker 的环境是不变的，Docker 是将环境、编译等问题统一解决，包括 k8s 等都可以在发布的时候实现不中断服务升级，业务低峰期时就可以发布新版本。

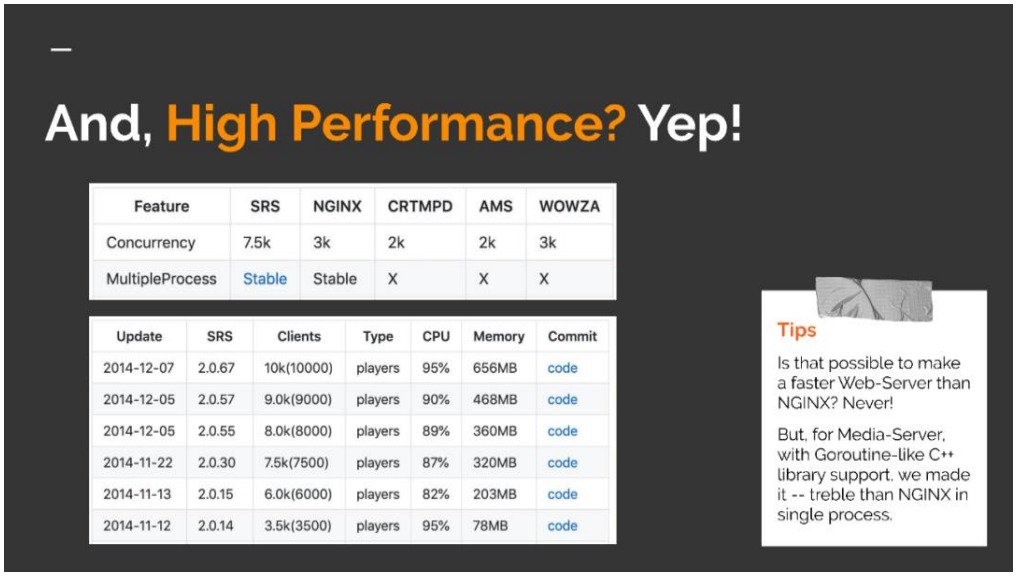
4.2 错误&日志



上图展示了 SRS 的日志，其中存在进程号与 ID。一个 ID 代表服务器上的一个连接，一个服务器为成百上千个用户与进程提供服务，ID 用于定位问题出现的位置与所属上下文日志。流媒体与 HTTP 不同，作为传输流存在上下文。长时间的数据交换使得其日志不仅仅只有一条，中间发生的事情都会通过日志来呈现。特别是 RTC 的日志非常多，如何从服务器中摘取关键信息？其实 SRS 设计了一套机制，也就是知道每个用户的日志究竟是什么，并及时从中摘取出。

除了日志之外，上图还展现了 SRS 中的错误反馈，错误参考了 Go 的机制，因为 Go 中出现错误可以 Wrap 打包错误，这样大家在反馈错误时就可以粘贴相应日志，就可以知道堆栈是什么。常规情况下出现错误只会呈现一个错误码，开发者并不知道究竟发生了什么。但如果有错误相应的堆栈以及给出每一层堆栈的变量，查询定位错误的过程就会变的非常方便。一般在关注一个新的开源项目时大家不太会关注这个问题。但当问题出现需要大家去查找问题源头时，堆栈的作用非常关键。这意味着我们不仅能够确定该问题的源头，也能妥善解决问题。

5. 高性能



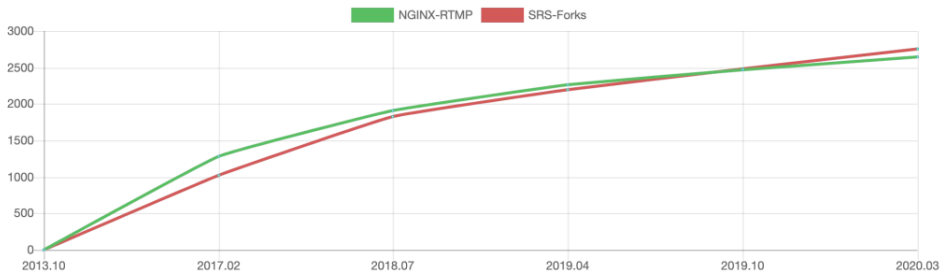
Feature	SRS	NGINX	CRTMPD	AMS	WOWZA
Concurrency	7.5k	3k	2k	2k	3k
MultipleProcess	Stable	Stable	X	X	X

Update	SRS	Clients	Type	CPU	Memory	Commit
2014-12-07	2.0.67	10k(10000)	players	95%	656MB	code
2014-12-05	2.0.57	9.0k(9000)	players	90%	468MB	code
2014-12-05	2.0.55	8.0k(8000)	players	89%	360MB	code
2014-11-22	2.0.30	7.5k(7500)	players	87%	320MB	code
2014-11-13	2.0.15	6.0k(6000)	players	82%	203MB	code
2014-11-12	2.0.14	3.5k(3500)	players	95%	78MB	code

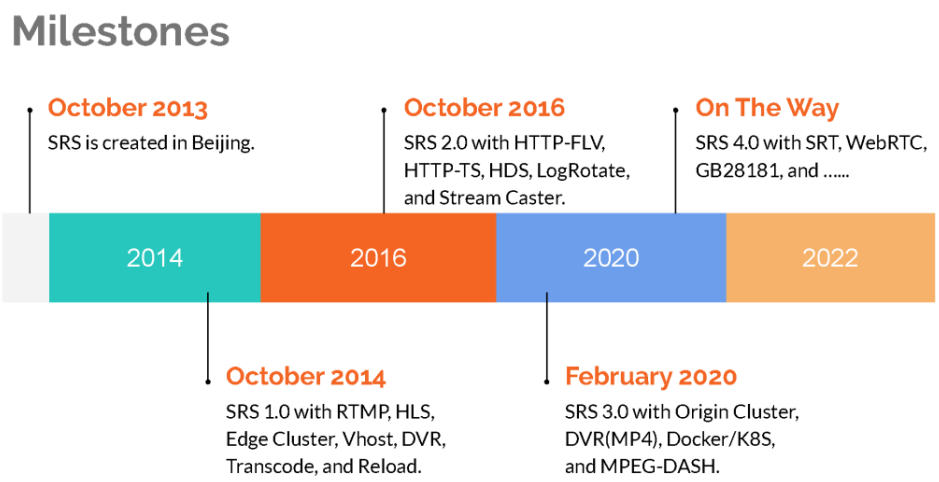
性能是一项基础要求，一般情况下 SRS 的性能是其它服务器的两倍左右。对性能的要求在 RTC 中其实会更严苛，因为 RTC 的性能消耗更多。

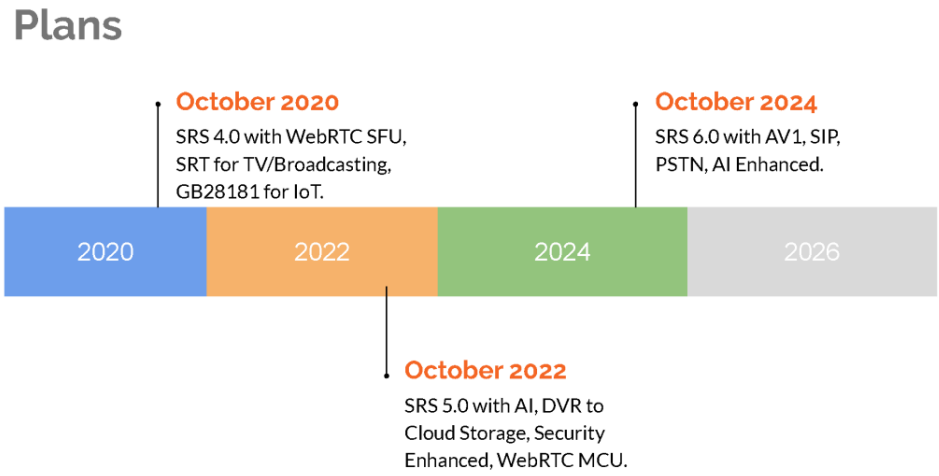
6. SRS 发展

关于 SRS 的发展，从 2013 年开始 SRS 一直都在稳步发展。初期由于应用场景相对固定，更新力度并不大。而现在，随着 Original Cluster 边缘集群相继得到支持，直播场景的覆盖也愈发完善，RTC 也在不断发展。大家可以看到最近各个视频行业都在互联网化，因此最近 SRS 的活跃度也非常高。



2019 年左右，SRS-Forks 超越了 NGINX-RTMP，预计未来 SRS-Forks 的增长是 NGINX-RTMP 的两倍。

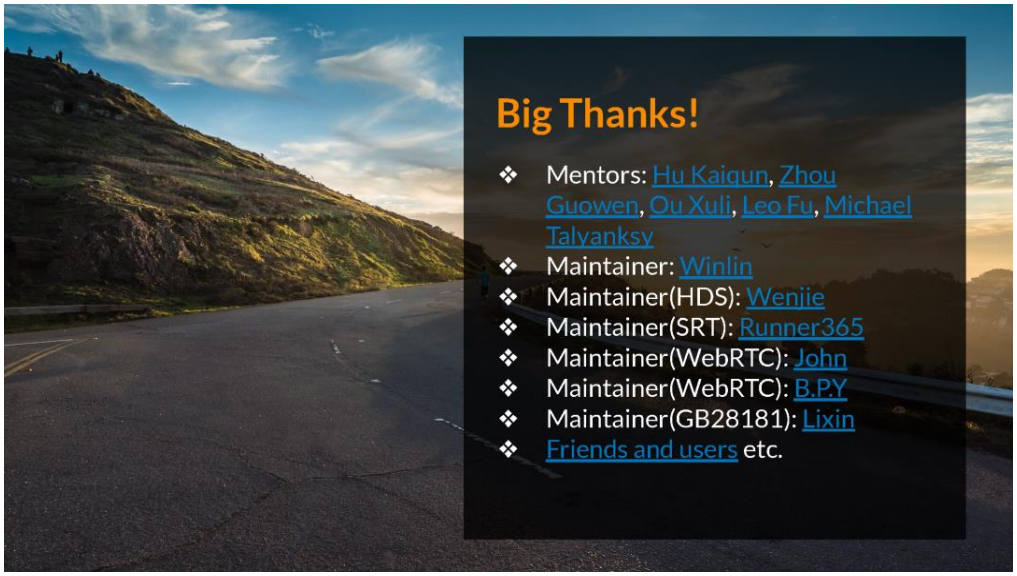




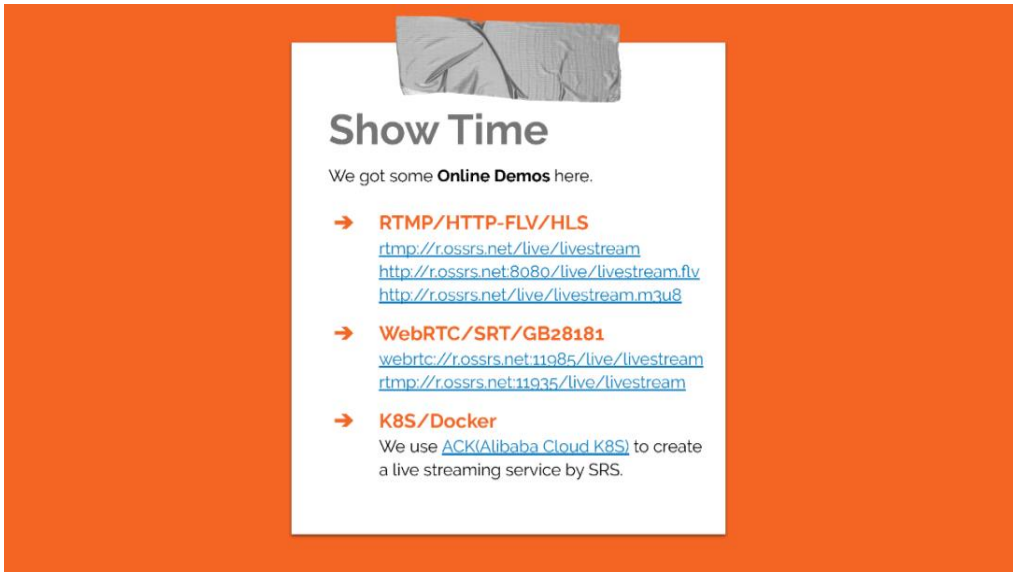
回顾 SRS 的发展脉络，2013 年 v1.0 实现了对直播基础协议的支持如 RTMP、HLS 等。随后的 v2.0 则主要支持 FLV 等以及移动互联网应用，v3.0 则提供了对 Original Cluster 的支持，同时很早就提供了对边缘集群的支持，边缘集群主要应对很多人播放的场景。而 Original Cluster 则主要用于支持流播，例如监控摄像头等。边缘不会存储流而 Original Cluster 则会存储流，所以需要集群的存在，目前对直播场景的支持相对完善。

2020 年初 SRS 支持了 SRT，SRT 主要用于解决远距离传输。同样也是用于直播与广电互联网化的综合场景，例如一些专业赛事、海外直播推流等。包括最近的 GB28181、WebRTC 等都能得到 SRS 的支持。

未来我们需要满足更广泛的互联网直播场景与需求，例如支持 SFU、IoT、AI 能力与云存储录制、安全以及 MCU、SFU、AV1、SIP 等。希望能够在 2024 年具备基本满足以上所有场景的能力。



感谢以上伙伴的卓越贡献



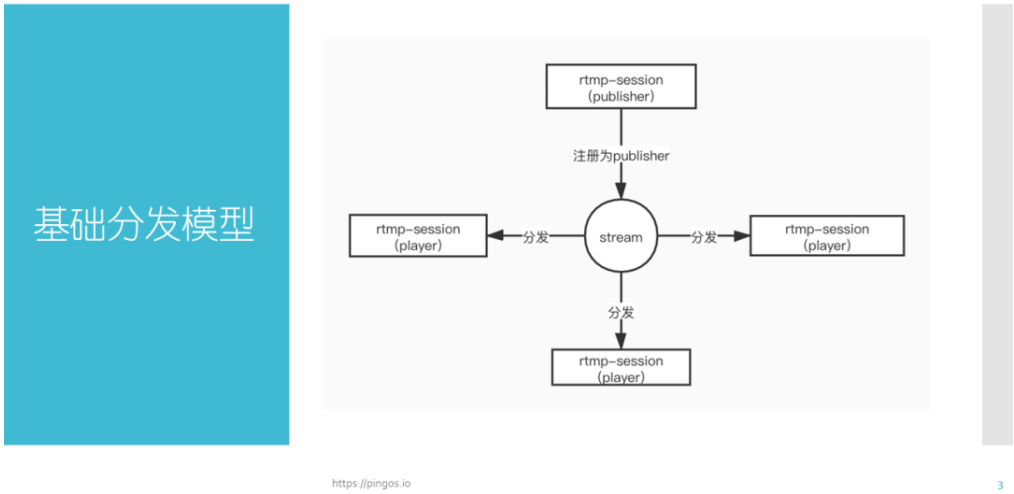
上图展现了我们现有的线上 Demo，欢迎大家前来访问。

基于 Nginx 的媒体服务器技术

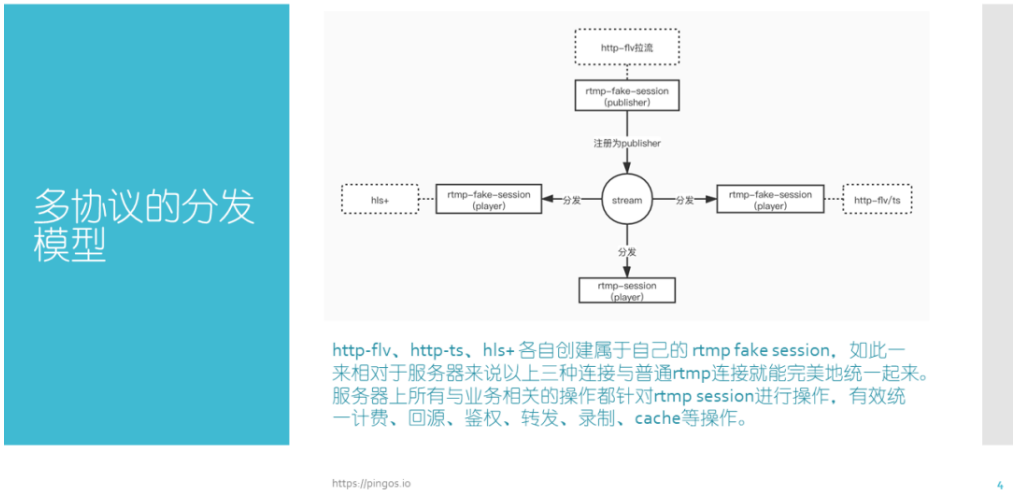
国内应用比较多的开源流媒体服务器 nginx-rtmp-module 一直存在功能少、集群化难度大等问题。在 LiveVideoStack 线上分享中，PingOS 开源项目组开发工程师、UCloud RTC 研发工程师朱建平详细介绍了基于 nginx-rtmp-module 的 PingOS 流媒体服务器在 http-flv、http-ts、hls+、多进程、转推、回源以及集群化部署方面的技术实现细节。

朱建平　LiveVideoStack 线上分享　2020-03-25

1. Nginx 流媒体扩展：http-flv、http-ts、hls+



最初始的 nginx-rtmp-module 相关模型与包括 SRS 在内的多数流媒体服务器实际上是一样的（1 个生产者，n 个消费者）。Nginx 存一个问题：它仅仅做了 RTMP 的消费模型，如果想扩展 http-flv 或 http-ts 的形式会较为困难。由于 rtmp-session 仅供 RTMP 协议使用，如果想扩展 http-flv，首先我们需要了解其基础分发模型（如上图所示）：所有的生产者与消费者都会被挂载到同一个 stream 中，生产者负责从网络端接收数据，消费者从 buffer 中获取数据对外发送。



如果是发送 flv 数据，那么可以保留原有 rtmp-session，当服务器收到一个 HTTP 请求时，创建一个 rtmp-session，此 session 与网络不相关，仅仅是逻辑上的 session。然后将这个 session 注入 stream 当中，如果是以消费者的角色注入进 stream 当中，则可以实现获取数据并往外分发。

假如此时服务器收到的是 http-flv 的请求，就可以创建一个逻辑上的 session，并把它注入 stream 中，此时理论上我们可以获得的是 rtmp 的数据。但我们需要的是 flv 的数据，由于 flv 数据与 rtmp 数据相似，我们可以通过 tag-header 的方式非常简单的将 rtmp 数据还原成 flv 数据。

根据上述思路，在生产者和消费者模型中，消费者可以通过创建 http-fake-session 的形式来复用以前的分发流程并实现 http-flv 协议。我们对其进行扩展，创建一个 http-fake-session 作为生产者，并让 http-fake-session 与一个 http client 进行关联，关联之后 http client 负责从远程服务器端下载数据传递给生产者，生产者就可以把这些数据通过分发模型分发给下面的 rtmp-session。这样也就间接实现了一个 http 回源的功能。通过上述思路我们就能够快速地实现 http-flv 的播放与拉流。

同样，我们可以根据上述思路继续扩展协议。假如我们在收到一个 http 请求之后，创建一个同样的 rtmp-fake-session（逻辑上的 session，与网络不相关），我们把它以消费者的角色插入到 stream 当中。这样就可以从 stream 当中获取到需要向下分发的数据。需要注意的是：stream 中最初保存的是 rtmp 数据而不是 ts 数据，无法直接获取 ts 数据。

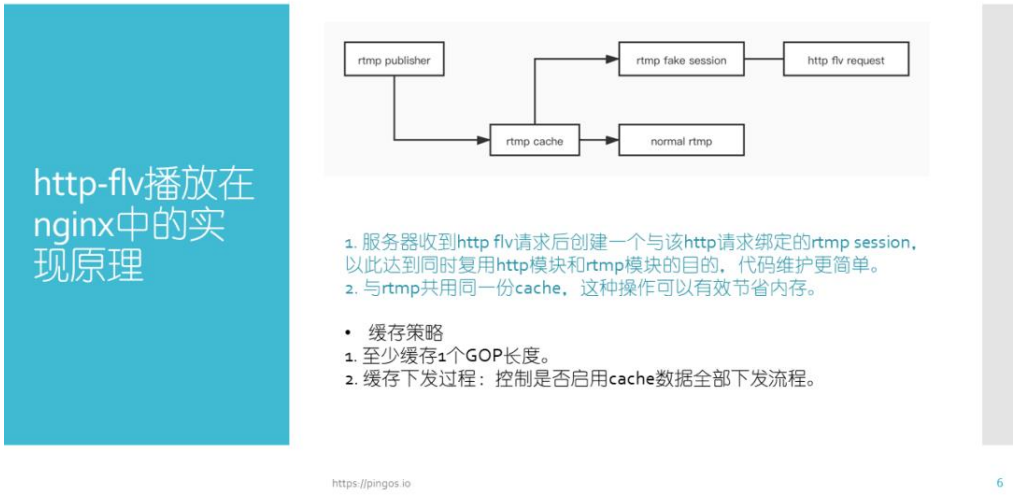
1.1 http-flv 在 Nginx 中的实现

基于 Nginx 实现 http-flv 需要注意以下几点细节：首先该实现复用了 Nginx 的分发模型以及 http 功能模块。（Nginx 对 http 协议栈的支持更加完善，包括 http1.0、http1.1 协议）

在部分线上业务中，客户可能需要在下载 http-flv 时添加后缀，按照以往的实践逻辑我们会在代码当中过滤后缀。如果遇见更为复杂，如修改是否需要开启 http chunked 编码的需求，我们就只能修改代码。而如果是基于 Nginx 通过复用 http 的现有模块来实现 http-flv，我们就可以通过 nginx-http-rewrite 功能来实现这些操作。因此使用 nginx-http 的原生功能来开发 http-flv 可以带来更多好处，如显著降低代码量。

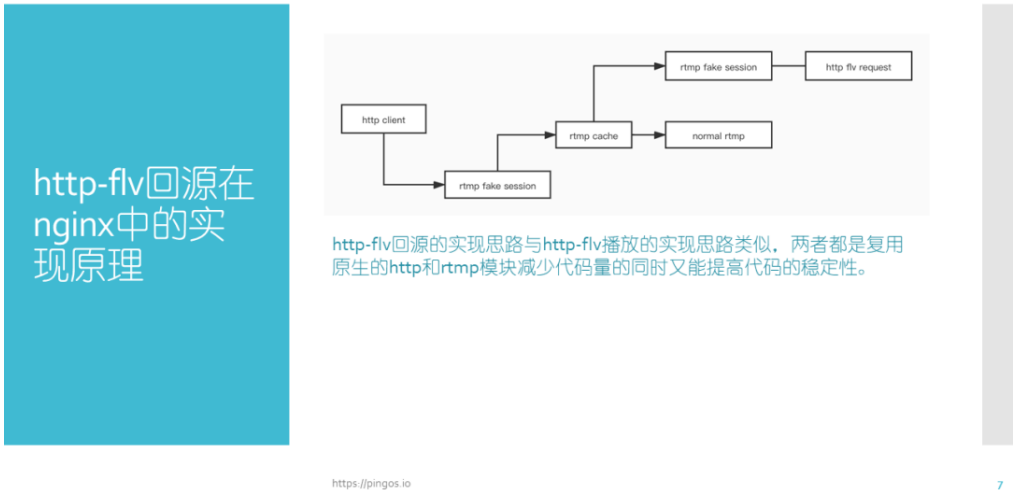
在这里我曾经看到过一种情况：即复用了 http 模块，但没有复用 rtmp 的分发流程。这样就会导致我们需要将分发流程在 http-flv 中重新再做一遍，对业务的控制就会变得非常复杂。举个例子，假如此时有人请求播放，需要将消息通知给业务服务器。此时，如果 rtmp 与 http-flv 两种协议的实现是分开的，那么意味着如果两者都被触发，就需要分别向业务服务器进行汇报。于是我们就需要付出双倍的代码与逻辑维护工作，这无疑会显著增加开发与维护成本。

因此，最简单的实现方案就是 flv 不做任何与业务相关的处理，仅在下发的时候进行格式转换，相当于 rtmp 分发时只发 rtmp 格式的数据，而 flv 分发时只需要将 rtmp 的数据打上 flv 的 tag-header，然后再进行下发，这样就省去了业务层的开发。http-flv 播放实现



图中展示的是 rtmp 的缓存对于 rtmp 和 http-flv 这两个协议的支持。http-flv 和 rtmp 二者共用一套缓存，其实 rtmp 本身传输的就是 flv 的数据，只不过是把 tag-header 给抛掉了。http-flv 的下发与 rtmp 的下发唯一的区别点在于 send 函数不同：http-flv 调用的是 http 的 send 函数，rtmp 下发时调用的是原生的 send 函数，在下发前需要添加各自的协议头。二者共用一块内存可以达到节省内存的效果，并且实现业务统一，降低开发成本。

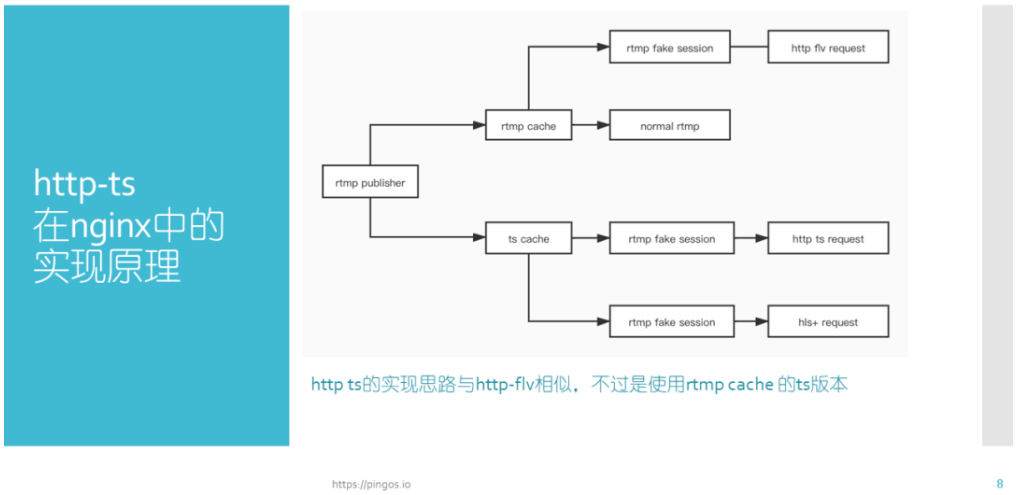
http-flv 回源的实现



图中展示的是 http-flv 回源在 nginx 中的实现。http-flv 回源实现的思路与 http-flv 的播放实现思路类似：即在需要回源的时候创建一个 http client，http client 所做的事情就是把 http 数据下载到本地。在下载数据到本地之前 http client 需要先创建一个 rtmp fake session 并将其作为生产者注入 stream 当中。而后 http client 开始从网络上下载数据并且将下载到的 flv 数据拆成 rtmp 数据。

为什么要拆成 rtmp 数据？这是因为 rtmp 的推流过来的缓存数据类型是 rtmp，因此从网上下载到的 flv 数据需要做一次拆分，拆成 rtmp 的数据，然后放入缓存。最终根据实际要求将数据转成 rtmp 或 flv 的格式。这样按照 http-flv 播放中 rtmp fake session 的逻辑，也就能够快速的实现 http-flv 的回源操作。

1.2 http-ts 在 Nginx 中的实现



图中展示的是 http-ts 在 Nginx 中的实现。其实现思想与 http-flv 的实现基本一致，仅仅是在操作上有所不同，不同点在于 http-ts 需要一个独立的 buffer 进行缓存。由于 http-ts 与 http-flv 的数据格式相差较大，对于 flv 数据到 rtmp 来说，只需要将数据拆成一个个小块，并在前面添加一个 header。即使 flv 数据的最一帧或者一个分块缺少也不用补齐。

但是 ts 数据不同，它的要求比较严格，每一分块必须为 188 字节，其中包括 ts header 以及有效载荷部分。并且如果数据库大小不足 188 个字节，则需要补齐。而 rtmp 的数据块没有严格固定要求其长度大小。对于 ts 数据来说，要想将 flv 数据转成 ts 数据，这个过程是需要消耗一些计算量的。

由于 ts 数据和 flv 的数据格式相差太大，因此在这里我们将 ts 的 buffer 与 rtmp 的 buffer 完全独立开。但此操作并不是默认开启的，需要在服务器中进行配置。开启配置后，才会将 rtmp 的 buffer 生成一份镜像的 ts 数据，这一部分的 ts 数据仅会供 http-ts 和 hls 两个协议使用。服务器中还涉及到一个原生的 hls 服务，在这里我们没有做任何的改动，而是加入了 hls+的服务来使用这个 buffer。

无论是 ts 还是 hls+，它们都注册了自己的 fake session，这样做的目的是为了统一业务。例如在有播放请求进入时，我们需要让业务服务器知道当前有请求产生。类似这种网络通知、事件通知的接口，在开发的过程中大家都希望只需要编写一份业务数据，而不是说做 hls 协议要针对

hls 播放写一个通知，做 ts 协议还要针对 ts 再写一份通知，这样业务代码会越来越庞大，最后导致服务几乎就很难维护。因此 fakesession 的作用是非常大的，其会把网络层与业务层完全隔离开。即使服务器本身的下发协议不是 rtmp，创建一个 rtmp-session 并挂载到业务服务器中即可。

总的来说，http-ts 与 http-flv 唯一实现区别就是获取 buffer 的位置不同。http-flv 需要从 rtmp buffer 获取，http-ts 则是从 ts buffer 中获取。如果能理解 http-flv 的协议流程，那么也就不难理解 http-ts 的实现流程。

1.3 hls+在 Nginx 中的实现

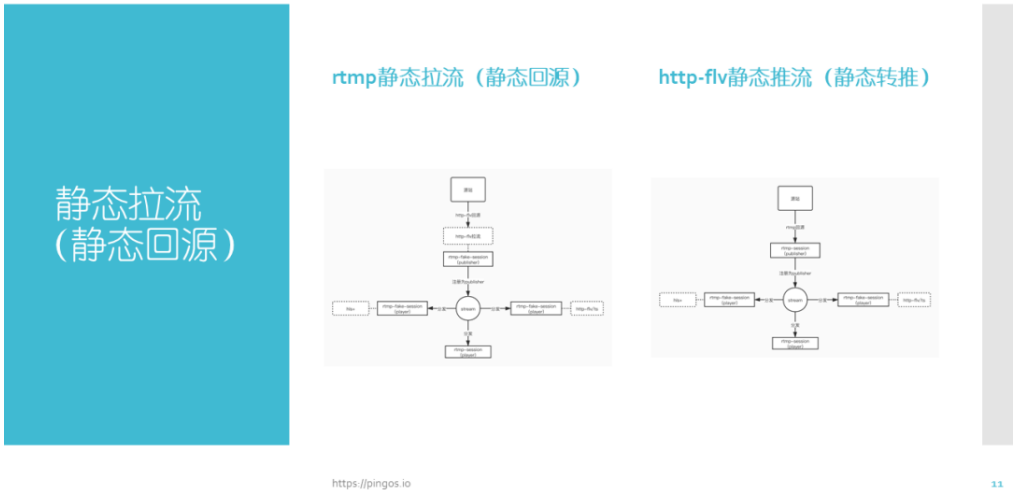
图中展示的是 hls+在 nginx 中的实现。hls+与传统 hls 不同，传统 hls 在服务端没有状态，服务端包含大量碎数据，客户端在不断执行下载，而 hls+则会记录每一个客户端的状态。

对于如何记录每个客户端的状态，之前我曾尝试通过对 hls+的连接创建一个虚拟连接用来记录状态。但是发现业务会比较复杂，并且后期会存在很多问题，包括代码量、bug 以及维护成本等。于是更换另外一种思路，还是用 fake session 的方式来实现。利用 fake session 作为消费者放入，根据每次进入的 http，连接，通过 session ID 进行绑定。由于第 1 次发送 hls 请求时客户端是不知道 sessionID 的，如果服务器获取到一个没有 session ID 的连接，则认为此客户端为第 1 次进入。客户端会接收到一个 302 的回复，302 回复中会告诉客户端一个新的地址，其中包含一个 session ID。客户端得到 session ID 之后，再次请求 m3u8 时，会加入 session ID，服务器就可获取相应 session ID 并对客户端进行身份区分。这样就能够通过 session ID 记录每一个客户端的播放状态。

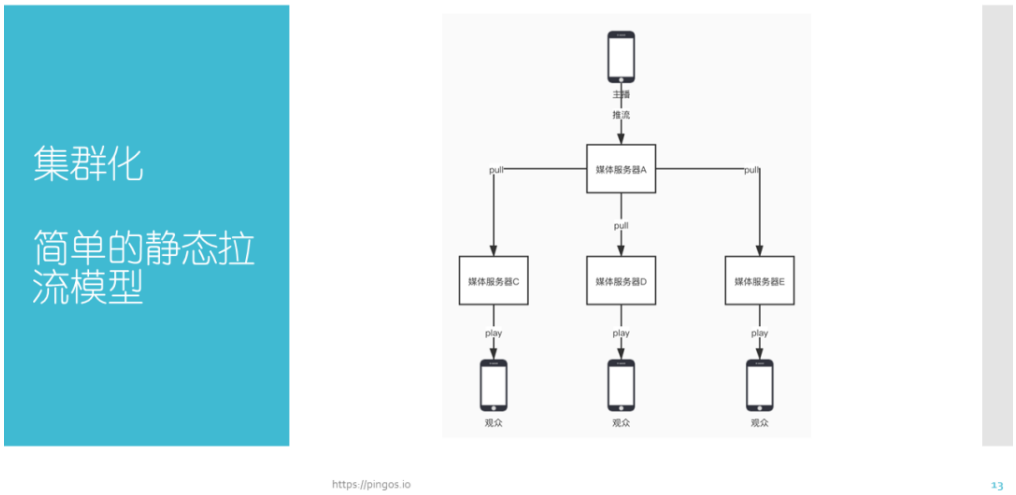
为什么要记录这个状态？这主要是因为服务器不是将数据直接写入硬盘而是放进内存，它需要知道每一个用户、每一个客户端的下载进度，并根据不同的进度从内存中定位 ts 数据。hls+和 http-ts 它们共用了一个 ts buffer，并且 hls+是实时的从 buffer 中定位 ts 内容。所以对于 hls+来说，并没有真正的 ts 数据产生，只是记录每一个文件在内存里面的偏移量。因此 hls+不存在读写的问题，在做 hls 服务时，以前可能会遇到过一个问题——读写硬盘的瓶颈。机械硬盘的读写速度比较慢，普遍的解决思路就是挂载一个虚拟硬盘，将内存映射到目录中进行读写。如果采用的是 hls+的方案，就可以省去挂载的操作，对于内存也并没有太多的消耗。而且如果同时有 hls+以及 http-ts 的需求，此时对于内存的利用率是非常高的。

2. 静态推拉流

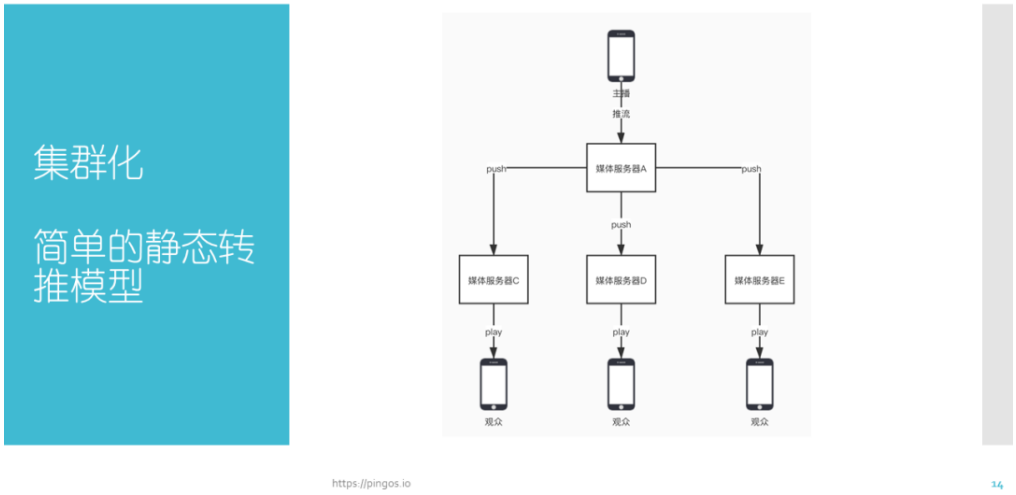
静态推拉流主要是为了满足集群化的需求。如果单台服务器不足以支撑服务的高并发量，那么我们就需要考虑服务器的扩展性。除此之外如果用户分散在全国各地，还需要进行服务器之间的打通。但是如果业务没有那么复杂就可以选择使用静态推拉流。



静态推拉流服务配置如上图所示，首先看静态拉流：首先存在一个目标源站，如果使用静态回源，那么目标地址会被配置在配置文件当中，目标源站能随意更改。



图中展示的是一个简单的静态拉流模型：如果来自主播的数据被推流到源站 A，那么我们需要保证服务器 A 的地址不会改变。



除此之外，如果想要构建一套完善的流媒体系统，则需要包含静态拉流与静态推流。假如有观众向服务器 C 请求播放，那么服务器 C 就会向服务器 A 拉流，无论服务器 A 是否存在视频流，服务器 C 都会拉取。因此该模型只适用于较为简单的业务场景。

3. 动态控制：动态回源、动态转推、鉴权

相对于静态推拉流的“无脑”推拉流，更适用于多数人需求的则是动态推拉流。

名称	触发阶段	描述
oclp_proc	start	针对worker进程生命周期
oclp_play	start、update、done	针对player的生命周期
oclp_publish	start、update、done	针对publisher的生命周期
oclp_pull	start、update、done	针对pull操作的生命周期
oclp_push	start、update、done	针对push操作的生命周期
oclp_stream	start、update、done	针对stream的生命周期

Nginx 的 RTMP 服务针对每一项功能都做了不同的触发阶段。以 oclp_play 为例，当有人启动播放时会触发 play 消息，play 消息会携带一项 start 参数。在播放过程中，play 消息依旧会被触发，只不过此时还会携带 update 参数。在 play 结束时也会触发一个 play 消息，所携带的参数是 down。借助这些参数，我们可以实现向业务服务器通知请求播放以及播放的具体阶段。

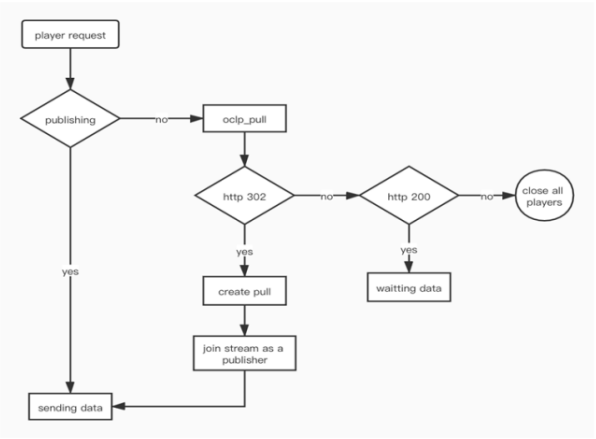
能力控制
（result）

名称	返回结果	描述
oclp_proc	200	任意，不必参考返回值
oclp_play	200 | 403	200：允许播放 403：不允许播放
oclp_publish	200 | 403	200：允许推流 403：不允许推流
oclp_pull	200 | 302 | 4xx | 5xx	200：服务器什么也不做 302：服务器向location字段的地址回源 4xx | 5xx：服务器终止该条stream下的所有订阅者
oclp_push	200 | 302 | 4xx | 5xx	200：服务器什么也不做 302：服务器向location字段的地址回源 4xx | 5xx：服务器终止该流的推流操作
oclp_stream	忽略返回结果	不对响应结果做处理

16

3.1 动态回源

动态拉流
（动态回源）

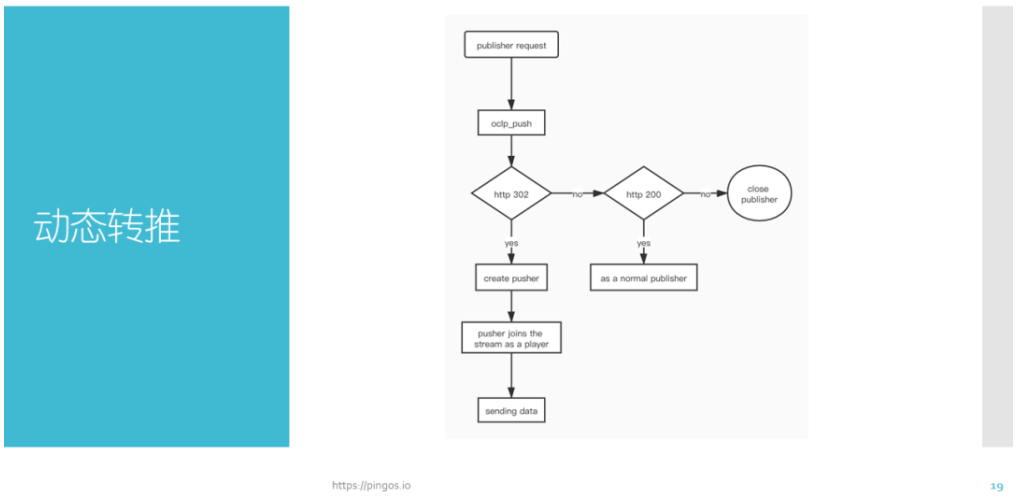


https://pingos.io

18

推流过程也存在类似操作，推流中存在 publish，同样分为三个阶段，play 和 publish 主要应用在鉴权操作中。如果在 start 阶段，业务服务器返回了一个 404 或者非 200 的结果，服务器就会中断当前的 play 请求，publish 亦是如此。除此之外， pull 与 push 主要应用于动态拉流阶段。当服务器接收到 play 请求，并且发现当前服务器里面没有目标流，也就是说 publish 的流不存在，就会触发 pull 的 start 阶段。在发送 start 请求之后如果业务服务器返回结果为 302，并且在 location 中又写了一个新的 rtmp 地址或 http-flv 地址，这台服务器就会向标记的那一台目标服务器拉取 rtmp 流或 flv 流，这个过程就被称为动态拉流。

3.2 动态转推

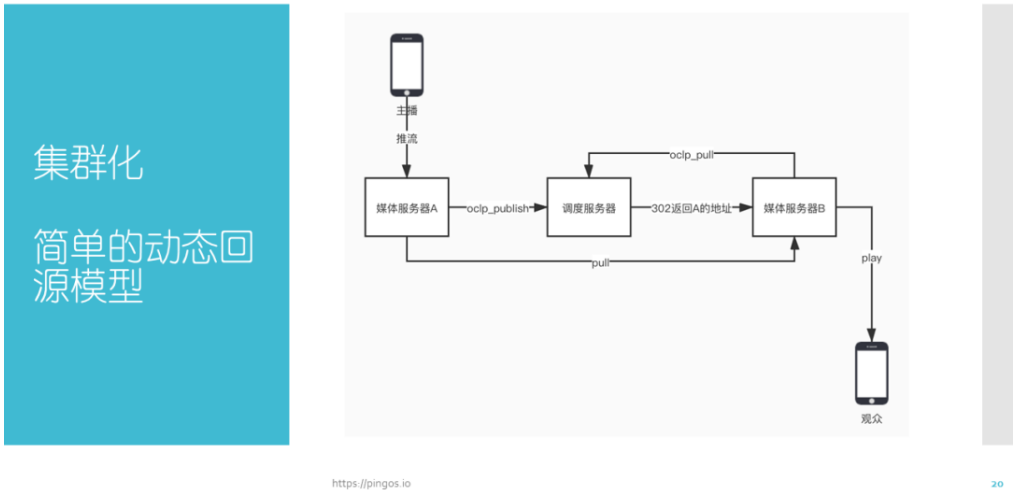


与动态拉流相对应的是动态推流，其理解方式与动态拉流大致相同。如果你向服务器推流，服务器会向配置好的目标地址发送 start 请求。如果在返回结果当中加入一个新的 rtmp 地址，这一台媒体服务器就会向新的 rtmp 地址推流，这也就是动态推流的操作。

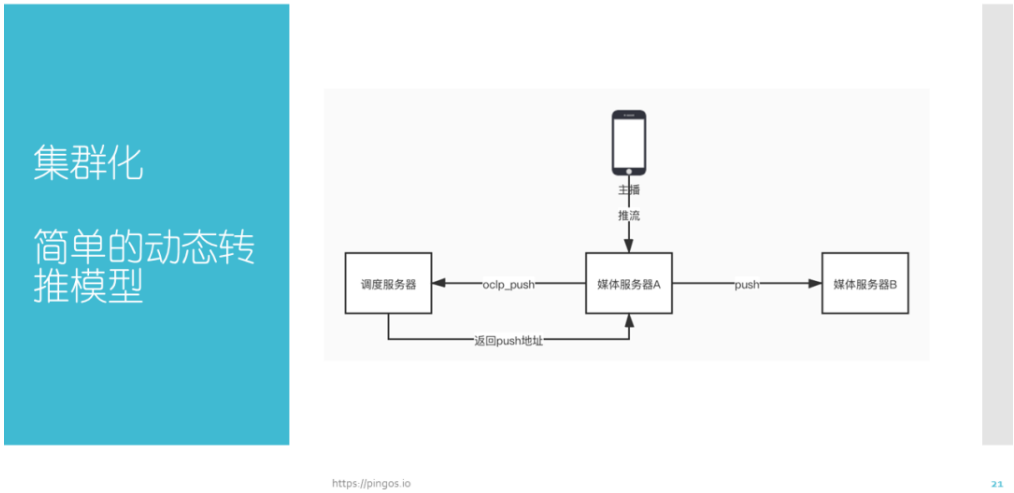
这一切的前提是返回 302 的结果，如果不想将流推出，那么反馈给服务器 400 或其他非 200，该流就会被中断。Oclp_stream 用的比较少，仅仅在这路流创建与消失时被触发。不管是 play 还是 publish，如果只有 play 或 publish 存在，都会认为这路流的生命周期还没有结束，只有当二者全部消失时才会被认定该路流生命周期已结束。同样的，如果一路流没有被发布过而是仅仅第一次有人请求，此时也会触发 start 并认为是该路流被创建，只不过没有生产者而已。这种场景的应用比较少，只有对业务要求比较高的系统可能会用到这一条消息。

名称	触发阶段	描述
oclp_proc	start	针对worker进程生命周期
oclp_play	start、update、done	针对player的生命周期
oclp_publish	start、update、done	针对publisher的生命周期
oclp_pull	start、update、done	针对pull操作的生命周期
oclp_push	start、update、done	针对push操作的生命周期
oclp_stream	start、update、done	针对stream的生命周期

上图展示了一个配置事例，主要包括查询服务器的 IP、查询服务器 play 操作希望支持哪些阶段等。

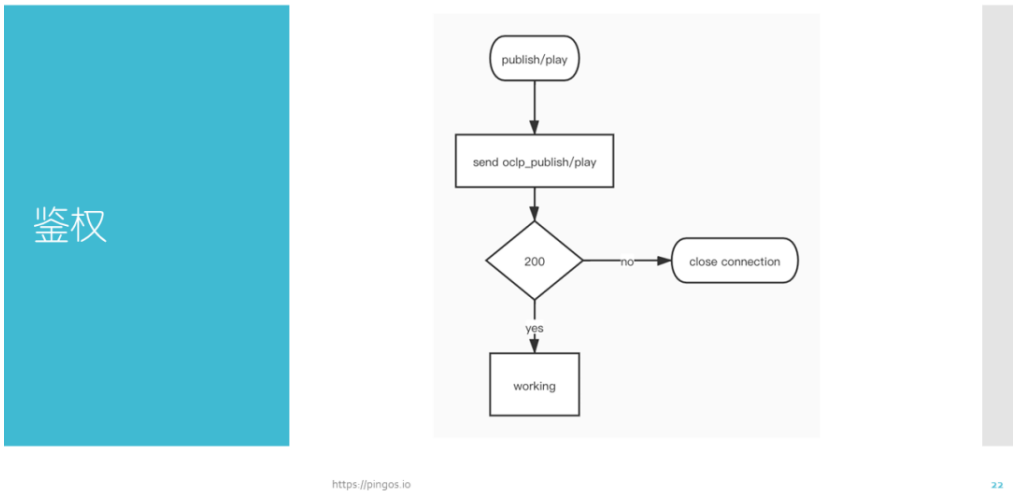


集群化部署依赖业务（调度）服务器，如果有回源需求则让边缘服务器 B 在 oclp_hold 阶段向业务服务器查询，此时业务服务器会告诉边缘服务器 B 一个 302 地址，其中包含源地址。边缘服务器 B 就会从标记出来的这一台（媒体服务器 A）拉流，从而实现动态回源。



动态转推主要是为了把本地的流推出去。在 CDN 的服务中，不同集群负责不同的职能。例如有些集群负责录制，有些则仅负责转码，此时我们希望核心机器能够把这些需要转码或需要录制的流按照需求转接到相应集群。动态转推非常重要，如果业务中包含这些不同的类型，就需要添加配置 oclp_push 去实现动态转推。

3.3 鉴权



鉴权操作中，我们只会对 publish 或 play 进行鉴权。

能力控制（result）

名称	返回结果	描述
oclp_proc	200	任意，不必参考返回值
oclp_play	200 | 403	200：允许播放 403：不允许播放
oclp_publish	200 | 403	200：允许推流 403：不允许推流
oclp_pull	200 | 302 | 4xx | 5xx	200：服务器什么也不做 302：服务器向location字段的地址回源 4xx | 5xx：服务器终止该条stream下的所有订阅者
oclp_push	200 | 302 | 4xx | 5xx	200：服务器什么也不做 302：服务器向location字段的地址回源 4xx | 5xx：服务器终止该流的推流操作
oclp_stream	忽略返回结果	不对响应结果做处理

16

如果 play 的时候反馈 200 就是允许播放，如果反馈 403 就是不允许播放，publish 也是如此，通过业务服务器控制客户某一次服务请求是否能被允许。前端进行 play 或者是进行 publish 时，如何把鉴权的 token 带过来？

主要通过变量：args=k=v&pargs=$pargs 在向外发送 play 查询时，如果加入 args=k=v&pargs=$pargs ，发请求时会带上这些参数，这样就可以将 rtmp 的全部自定义参数传递过来。

4. 多进程：进程间回源

多进程问题在原生的 nginx rtmp 中有很多 bug，现在的做法是通过共享内存记录下每个进程上的 stream 列表。如果 play 的进程没有流，则查询 stream 列表，并通过 unix socket 向目标进程回源拉流。除此之外，进程间的回源不会触发 ocl_playoclp_publish oclp_pull 消息。

5. 更多操作说明

PingOS：https://github.com/im-pingo/pingos

刘歧：FFmpeg Filter 深度应用

本文来自 OnVideo 视频创作云平台联合创始人刘歧在 LiveVideoStackCon 的讲师热身分享，刘歧分享了 FFmpeg 的基本原理、使用方法及开发方法。在 10 月 19-20 日的 LiveVideoStackCon 2018 上，刘歧还将分享如何通过 FFmpeg 实现视频版权保护的方法。

刘歧　LiveVideoStackCon 2018 北京　2020-01-08

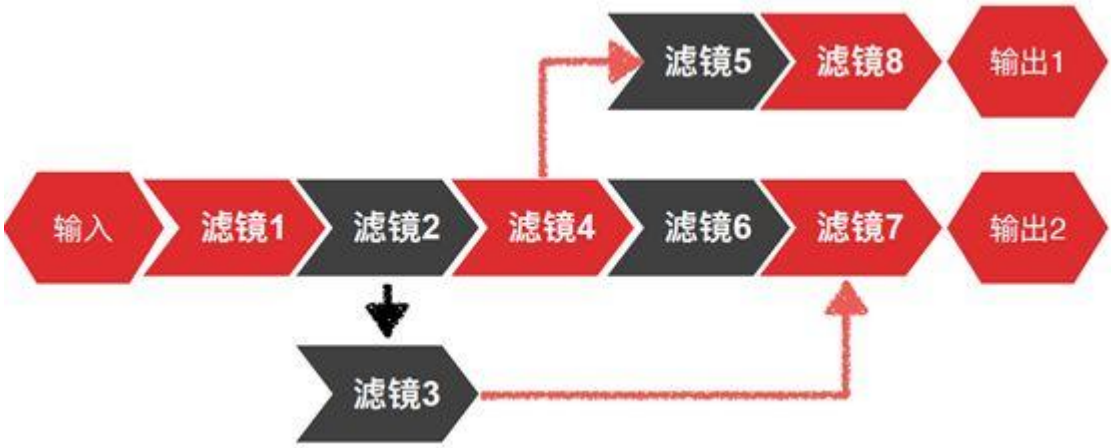
大家好我是刘歧，目前和几个朋友一起创办了视频云创作平台 OnVideo，主要负责做一些新功能预研和整体方向把控。

今天主要从四个方面来介绍一下 FFmpeg 的滤镜深度应用：
1. Filter 基本原理
2. Filter 使用方法
3. Filter 开发基础
4. Filter 开发

一. Filter 基本原理

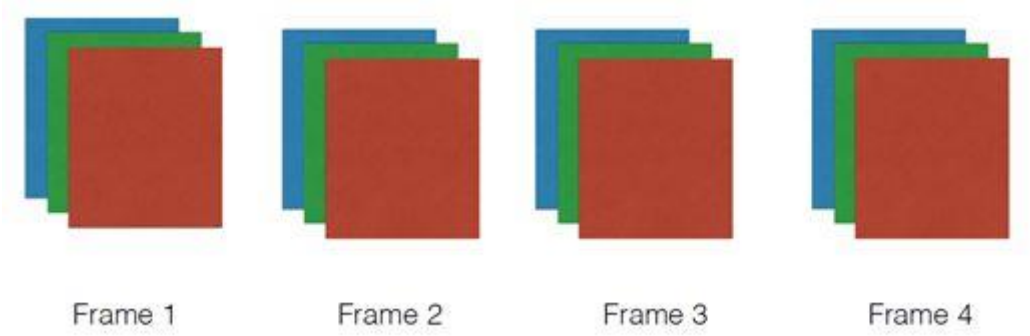
基本原理的理解可以从以下六点出发：

1. 线性属性：滤镜是一个线性的属性，是一步接着一步的往下走的，当然也可以有多线程的，但是多线程的实现不是特别好，一般的实现都是通过单线程来线性处理的。
2. 输入输出标签：滤镜都有输入和输出的标签。
3. 滤镜名
4. 滤镜参数：当我们在使用某一类滤镜时，需要知道对应的参数大小，如位置，宽高等。
5. 内置变量：在使用滤镜时，可以用到它的很多内置变量，如帧率，PTS 等，这些变量可以在命令行中使用，在调用 API 时也会用到。
6. 压缩前、解压后的 YUV/RGB/PCM 数据：对数据进行滤镜处理时使用的都是原始数据，也就是未压缩的数据。



线性处理图解

如上图所示，从滤镜线性处理图中可以看出，对每一个的滤镜处理过程必须要有输入和输出，比如在最后滤镜 8 的位置没有输出 1 来承接它的结果，则会报错。另外，如从滤镜 1 到滤镜 2 的过程，滤镜 1 的输出 Out 可以作为滤镜 2 的输入 In，从滤镜 2 再往后，滤镜 2 的输出 Out 可以分为两部分，分别作为其它滤镜处理的输入。



原始数据处理

滤镜的处理都是基于原始数据来进行的，比如视频图像的每一帧，当然也包括音频。如上图所示，每一帧可以有前后的多个层，一般情况下是两层，当有多层时，如 Frame1 是三层，则可以先处理前两层，处理之后合并成一层，再跟另一层一起去做处理。当处理完 Frame1 后送到编码器进行编码，再依次处理 Frame2、Frame3、Frame4，处理完后均送到编码器进行压缩编码为另一个格式，比如 H.264。

二. Filter 使用方法

1. 常用滤镜

常用的滤镜中重点的是：scale、trim、overlay、yadif、rotate、movie

比如常用的 scale 可以用来做缩放，trim 可以做比较精确的帧级的剪切，overlay 可以来实现视频混流，画中画或多画面等叠加处理，rotate 来做旋转，movie 可以加载第三方的视频等等，yadif 可以在摄像机上做隔行扫描，在 PC 端的播放器可以做去交错。另外，还有 delogo、crop、colorkey、chromakey、zoompan； delogo 可以用来模糊化某一区域，crop 用于图像区域剪切，colorkey 和 chromakey 是用于抠图的，chromakey 以 YUV 输入为主，colorkey 以 RGB 输入为主，zoompan 可以将图像放大。

2. 命令行

ffmpeg -filters

```
Filters:
  T.. = Timeline support
  .S. = Slice threading
  ..C = Command support
  A = Audio input/output
  V = Video input/output
  N = Dynamic number and/or type of input/output
  | = Source or sink filter
 ... abench            A->A       Benchmark part of a filtergraph.
 ... acompressor       A->A       Audio compressor.
 ... acrossfade        AA->A      Cross fade two input audio streams.
 ... acrusher          A->A       Reduce audio bit resolution.
 T.. adelay            A->A       Delay one or more audio channels.
```

通过这一命令，可以列出所有的滤镜，但需要弄清楚每一个滤镜的作用，比如上图的 adelay，它的作用是音频转音频，延缓一个或更多的音频通道。在滤镜参数里面还包括 timeline support、slice threading、command support，等在后面也会有介绍。

ffmpeg -h filters=rotate

```
Filter rotate
  Rotate the input image.
    slice threading supported
    Inputs:
       #0: default (video)
    Outputs:
       #0: default (video)
rotate AVOptions:
  angle             <string>     ..FV.... set angle (in radians) (default "0")
  a                 <string>     ..FV.... set angle (in radians) (default "0")
  out_w             <string>     ..FV.... set output width expression (default "iw")
  ow                <string>     ..FV.... set output width expression (default "iw")
  out_h             <string>     ..FV.... set output height expression (default "ih")
  oh                <string>     ..FV.... set output height expression (default "ih")
  fillcolor         <string>     ..FV.... set background fill color (default "black")
  c                 <string>     ..FV.... set background fill color (default "black")
  bilinear          <boolean>    ..FV.... use bilinear interpolation (default true)

This filter has support for timeline through the 'enable' option.
```

当某个滤镜不知道它里面都有什么参数或者参数怎么用时，那么可以通过上面这条 help 命令去查看滤镜的内容。比如在上图，查看旋转（rotate）滤镜，可以看出它支持 slice threading，而在最下面可以看到还支持 timeline 操作，并且输入和输出都是 Video 格式，再

下面就是 rotate 的一些参数，值得注意的是，在 FFmpeg 里面，尤其滤镜这部分，为了让大家在输入参数时候更简单，它会把很长的参数提供一个简化的名称。

命令行的使用开头都是：filter_complex，然后再在后面接一些字符串，如下：

下面列出几个例子的命令行使用：

1. 跑马灯

[0:v][1:v]overlay=x='if(gte(t,0), -w+(mod(n, W+w))+5, NAN)':y=0[out]

应用场景如新闻播放时画面下方的小栏预告，从右跑到左。

2. 图像旋转

[1:v]rotate=a=60*PI/180:ow=180:oh=180[v1];[0:v][v1]overlay=x=-20:y=-20[o]

3. delogo（模糊化某一区域）

delogo=x=72:y=32:w=168:h=86:band=1:show=1

下图为以上三条命令行处理后的展示效果：

在 FFmpeg 的 ffplay 里可以通过 dumpgraph 对滤镜的处理过程增加更深的了解，可以看看下面这个例子：

./ffplay -dumpgraph 1 -f lavfi "movie=a.flv[l];movie=logo.png[r];[l][r]overlay=x=0:y=0"

```
+----------------+
| Parsed_movie_0 |out0--[640x480 1:1 yuv420p]--Parsed_overlay_2:main
|    (movie)     |
+----------------+

+----------------+
| Parsed_movie_1 |out0--[129x66 1:1 rgb24]--auto_scaler_0:default
|    (movie)     |
+----------------+

                                                  +------------------+
Parsed_movie_0:out0----[640x480 1:1 yuv420p]-----main| Parsed_overlay_2 |default--[640x480 1:1 yuv420p]--out:default
auto_scaler_0:default--[129x66 1:1 yuva420p]--overlay|    (overlay)     |
                                                  +------------------+

                                                     +--------------+
Parsed_overlay_2:default--[640x480 1:1 yuv420p]--default|     out      |
                                                     | (buffersink) |
                                                     +--------------+

                                            +----------------+
Parsed_movie_1:out0--[129x66 1:1 rgb24]--default| auto_scaler_0 |default--[129x66 1:1 yuva420p]--Parsed_overlay_2:overlay
                                            |    (scale)     |
                                            +----------------+
```

在滤镜处理的过程中是先加载第一个电影，然后再加载一张 PNG 图片，为了叠加处理比较方便会先将它们转换成 YUV，再进行叠加运算，最后输出到一个 buffersink 中。

3. API

滤镜 Filter 的 API 结构包括：

- AVFilterContext
- AVFilterInOut
- AVFilterGraph
- AVFilter
- AVFilterPad

AVFilterContext：滤镜的上下文
AVFilterInOut：滤镜的输入、输出
AVFilterGraph：滤镜的图结构
AVFilter：滤镜的具体处理类
AVFilterPad：包括输入的 Pad 和输出的 Pad

调用 API 的分为两个大的步骤：
第一步，滤镜的初始化

- 初始化：
- avfilter_get_by_name 创建buffer和buffersink
- avfilter_graph_create_filter(&buffersrc_ctx | &buffersink_ctx, buffersrc | buffersink, "in" | "out", args, NULL, filter_graph);
- avfilter_graph_parse_ptr(filter_graph, filter_spec, &inputs, &outputs, NULL)) < 0)
- avfilter_graph_config(filter_graph, NULL)

初始化部分首先要确定 In 的 Buffer 和 Out 的 Buffersink，接下来就是创建一个滤镜的 Graph，再就是做一个滤镜的解析，最后就是通用配置。

第二步，滤镜处理

- 每一帧处理：
  - av_buffersrc_add_frame_flags(filter_ctx[stream_index].buffersrc_ctx, frame, 0);
    - -> request_frame
    - ->
  - av_buffersink_get_frame(filter_ctx[stream_index].buffersink_ctx, filt_frame);
    - -> get_frame_internal
      - -> ff_filter_graph_run_once
        - -> ff_filter_activate(request_frame(Output pads only.)/filter_frame(Input pads only.))

滤镜处理过程是先解码，解码之后将帧加入到 Buffer，再调用滤镜处理模块，处理之后输出到 Buffersink。

滤镜 API 的使用方法跟之前命令行的使用方法是一样的，只是略微有一点差别。如果我们不会使用多输入多输出的方式，对于 API 的使用实际上它的输入也可以用字符串处理的方式，就是命令行里边的处理方式，如以下官方 demo 的代码：

跑马灯

[0:v][1:v]overlay=x='if(gte(t,0), -w+(mod(n, W+w))+5, NAN)':y=0[out]

```
if (ifmt_ctx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_VIDEO)
    filter_spec = "movie=logo.png[scl]; [in][scl]overlay=x='if(gte(t,0), -w+(mod(n, W+w))+5, NAN)':y=0";
else
    filter_spec = "anull"; /* passthrough (dummy) filter for audio */
ret = init_filter(&filter_ctx[i], stream_ctx[i].dec_ctx,
        stream_ctx[i].enc_ctx, filter_spec);
if (ret)
    return ret;
```

图像旋转

[1:v]rotate=a=60*PI/180:ow=180:oh=180[v1];[0:v][v1]overlay=x=-20:y=-20[o]

```
if (ifmt_ctx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_VIDEO)
        filter_spec = "movie=logo.png[scl];[scl]rotate=a=60*PI/180:ow=180:oh=180[v1];[in][v1]overlay=x=0:y=0";
else
    filter_spec = "anull"; /* passthrough (dummy) filter for audio */
ret = init_filter(&filter_ctx[i], stream_ctx[i].dec_ctx,
        stream_ctx[i].enc_ctx, filter_spec);
if (ret)
    return ret;
```

Delogo（模糊化某一区域）

delogo=x=72:y=32:w=168:h=86:band=1:show=1

```
if (ifmt_ctx->streams[i]->codecpar->codec_type == AVMEDIA_TYPE_VIDEO)
    filter_spec = "delogo=x=72:y=32:w=168:h=86:show=1";
else
    filter_spec = "anull"; /* passthrough (dummy) filter for audio */
ret = init_filter(&filter_ctx[i], stream_ctx[i].dec_ctx,
        stream_ctx[i].enc_ctx, filter_spec);
if (ret)
    return ret;
```

从上面的介绍可以看出，命令行与 API 实质上并没有太大的区别。

三. Filter 开发基础

1. Module 结构

```
AVFilter ff_vf_xxx = {
    .name          = "xxx",
    .description   = NULL_IF_CONFIG_SMALL("xxx"),
    .priv_size     = sizeof(xxxContext),
    .init          = init,
    .uninit        = uninit,
    .query_formats = query_formats,
    .process_command = process_command,
    .activate      = activate,
    .inputs        = xxx_inputs,
    .outputs       = xxx_outputs,
    .priv_class    = &xxx_class,
    .flags         = AVFILTER_FLAG_SUPPORT_TIMELINE_GENERIC,
};
```

```
• static const AVFilterPad xxx_inputs[] = {
•     {
•         .name       = "default",
•         .type       = AVMEDIA_TYPE_VIDEO,
•         .filter_frame = filter_frame,
•         .config_props = config_props,
•     },
•     { NULL }
• };

• static const AVFilterPad xxx_outputs[] = {
•     {
•         .name = "default",
•         .type = AVMEDIA_TYPE_VIDEO,
•     },
•     { NULL }
• };
```

在上面的 Module 结构里面，可以很清晰的看出步骤流程，先要在滤镜的类里面注册它所对应的所有信息，如滤镜名，上下文等等；下一步就是对 Input 和 Output 的处理，对帧的滤镜处理正常都是在一个 Input 里面处理的；

```
enum {
    e_value, e_const, e_func0, e_func1, e_func2,
    e_squish, e_gauss, e_ld, e_isnan, e_isinf,
    e_mod, e_max, e_min, e_eq, e_gt, e_gte, e_lte, e_lt,
    e_pow, e_mul, e_div, e_add,
    e_last, e_st, e_while, e_taylor, e_root, e_floor, e_ceil, e_trunc, e_round,
    e_sqrt, e_not, e_random, e_hypot, e_gcd,
    e_if, e_ifnot, e_print, e_bitand, e_bitor, e_between, e_clip, e_atan2, e_lerp,
} type;
```

再就是要看滤镜是否支持 Timeline，比如做一些动态的变量，跑马灯之类的，也要有时间线的概念。在这个里面也有一些对应的内置函数，可以把他们理解为一些数字运算的接口。以及下面还有一些变量名的定义。

```
• static const char *const var_names[] = {
•     "n",   // frame count
•     "pts", // presentation timestamp expressed in AV_TIME_BASE units
•     "r",   // frame rate
•     "t",   // timestamp expressed in seconds
•     "tb",  // timebase
•     NULL
• };

• enum var_name {
•     VAR_N,
•     VAR_PTS,
•     VAR_R,
•     VAR_T,
•     VAR_TB,
•     VAR_NB
• };
```

2. 加入到 FFmpeg 中

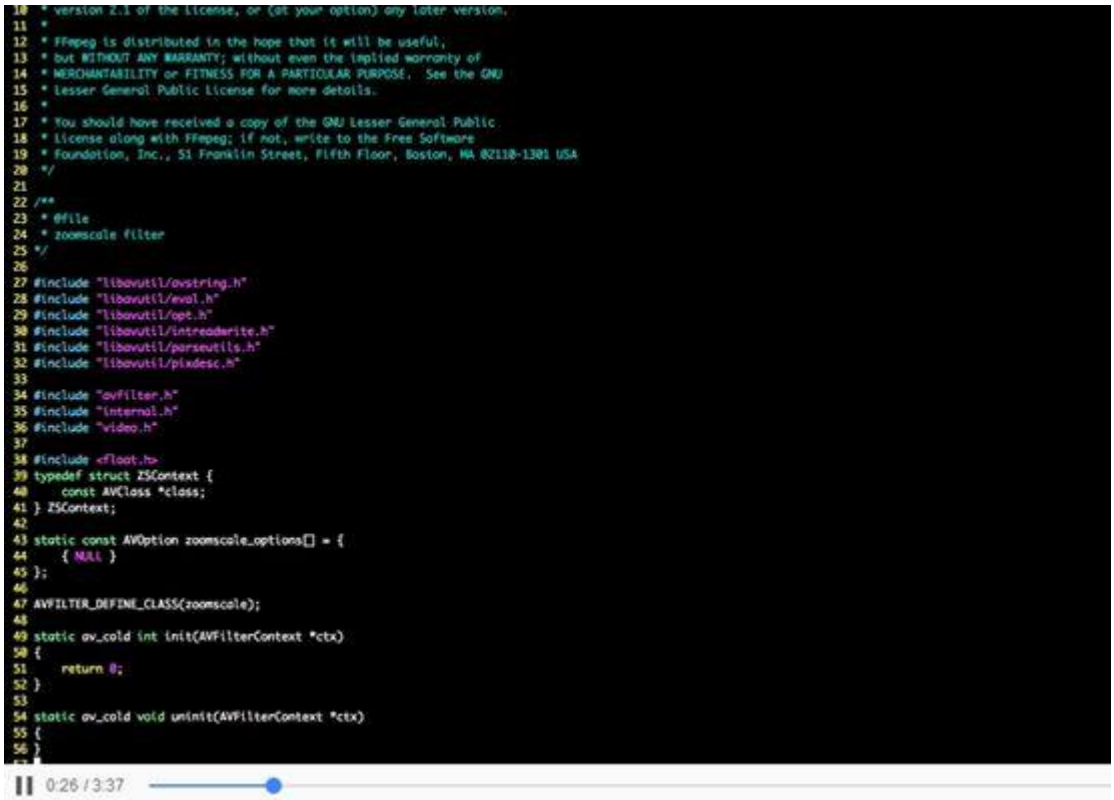
添加滤镜的时候，主要是往下面两个里面添加：

添加到 libavfilter/allfilters.c 中

添加到 libavfilter/Makefile 中

四. Filter 开发

在这一部分，为大家做了 FFMPEG 添加新滤镜的过程的录像，录像地址为 http://bbs.chinaffmpeg.com/add_filter.mp4，有兴趣的同学可以看一下



首先建立了一个 zoomscale 的滤镜文件，建立完这个文件之后，创建对应的 context，然后把刚才讲到的那些接口都写进来，定义 zoomscale 公共的类，下面我们实现了一个 filter _frame，它会把里面的 input 给释放掉，让它变成一个 NULL，此时显示的前景图像就会变成一个纯绿色的图像，如果说这里有 process commode，那么就可以在做 zmq 的时候用，如果没有的话，可以留空。关键是 filter_frame，可以当作滤镜处理的一个入口，而 get_video_buffer 可以把它理解为我们看到的很多 demo 里面写到的 get_video_buffer，即

创建一个帧的对应 Buffer，将其添加至 Makefile 之后，继续添加一个 avfilter。这是 FFmpeg4.0 之前的版本，在后面新版本里面，采用的是数组的形式，也就没有列表那么麻烦，速度会更快一些，添加完之后，可以通过 config 进行查看配置，每次 config 完之后，会有整体的一个输出。添加完成后，进行编译，之后可以在 avfilter 库里直接查看滤镜是否已经加到 FFmpeg 的应用程序里，然后调用 API 部分，直接把字符串传进来就可以使用了。

WebRTC 的拥塞控制和带宽策略

网络的波动带来的卡顿直接影响着用户的体验，在 WebRTC 中设计了一套基于延迟和丢包反馈的拥塞机制（GCC）和带宽调节策略来保证延迟、质量和网路速度之间平衡，本文中重点是介绍基于 trendline 滤波的评估模型。本文来自学霸君资深架构师袁荣喜和萍乡学院辛锋的投稿，并由 LiveVideoStack 全文发布。

袁荣喜，辛锋　2018-05-24

在视频通信的技术领域 WebRTC 已成为主流的技术标准，WebRTC 包涵了诸多优秀的技术，譬如：音频数字信号处理技术（AEC, NS, AGC）、编解码技术、实时传输技术、P2P 技术等，这些技术目的都是为了实现更好实时音视频方案。但是在高分辨率视频通信过程中，通信时延、图像质量下降和丢包卡顿是经常发生的事，甚至在 WiFi 环境下，一次视频重发的网络风暴可以引起 WiFi 网络间歇性中断，通信延迟和图像质量之间存在的排斥关系是实时视频过程中的主要矛盾。

分析 WebRTC 是如何解决这个矛盾之前，先来看看我们在在线教育互动的生产环境统计到的视频延迟和人感官的关系，大致如下：

0 ~ 400 毫秒	人感觉不到视频在通信过程中的延迟
400 ~ 800 毫秒	人能感觉到轻微延迟，但不影响通信互动
800 毫秒以上	人能感觉到延迟而且影响通信互动

也就是说，通信过程中最好将视频延迟控制在 800 毫秒以内。除了延迟，视频图像质量也是个对人感官产生差异的关键因素，我们以 640x480 分辨率每秒 24 帧的 H264 编码情况下视频码率和人感官之间的关系（这组数据是我们通过小范围线上用户投票打分的数据）：

800kbps 以上	人对视频清晰度满意,感觉不到视频图像中的信息丢失
480 ~ 800kbps	人对视频清晰度基本满意，有时能感觉到视频图像中的信息丢失
480kbps 以下	人对视频清晰度不满意，大部分时候无法辨认图像中的细节信息

从上面的描述可以知道视频质量保持在一个可让人接受的质量范围是需要比较大的带宽码率支持的，如果加上控制延迟，则更需要网络有很好速度和稳定性。但是很不幸，我们现阶段的移动网络和家用 WiFi 并不是我们想象中的那么好，很难做到在实时视频通信中一个让人非常满意的程度。为了解决以上几个问题，WebRTC 设计了一套基于延迟和丢包反馈的拥塞机制（GCC）和带宽调节策略来保证延迟、质量和网路速度之间平衡，这是一个持续循环过程，如下图：

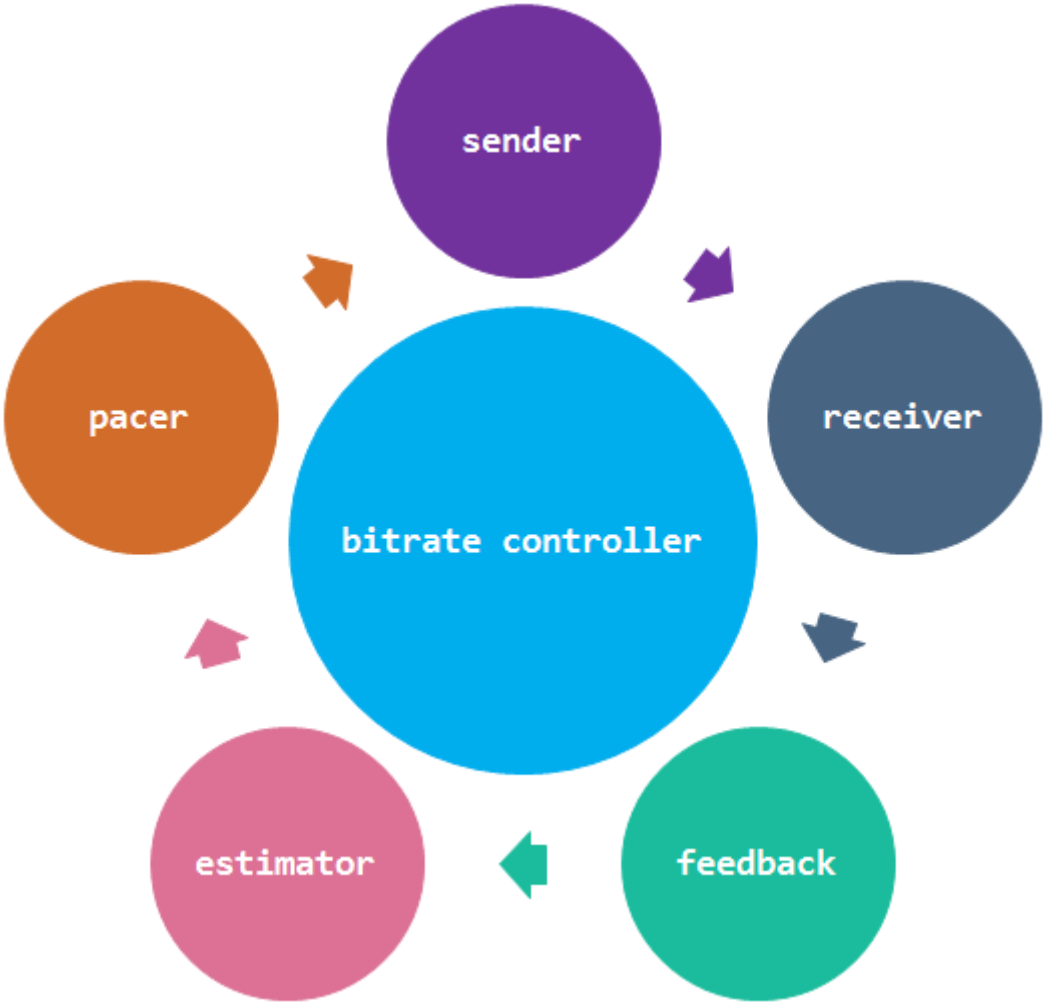


图 1：拥塞控制循环示意图

1） estimator 通过 RTCP 的 feedback 反馈过来的包到达延迟增量和丢包率信息计算出网络拥塞状态并评估出适合当前网络传输的码率，根据这个码率改变视频编码器码率，然后改变 pacer 的码率

2） pacer 会根据这个码率改变 pacer 的网络发送速度和 padding 比例，并用新的网络发送速度来定时触发发包事件。

3） sender 收到 pacer 的发送事件，进行 RTP 报文发送。

4） receiver 接收到 RTP 报文，进行 arrival time 统计和丢包统计

5） feedback 定时对 receiver 统计的信息进行 RTCP 编码，并反馈到发送端的 estimator 进行新一轮的码率评估。

以上是整个 WebRTC 拥塞控制和带宽调节过程，下面这个示意图是这个过程涉及到 WebRTC 内部模块关系。

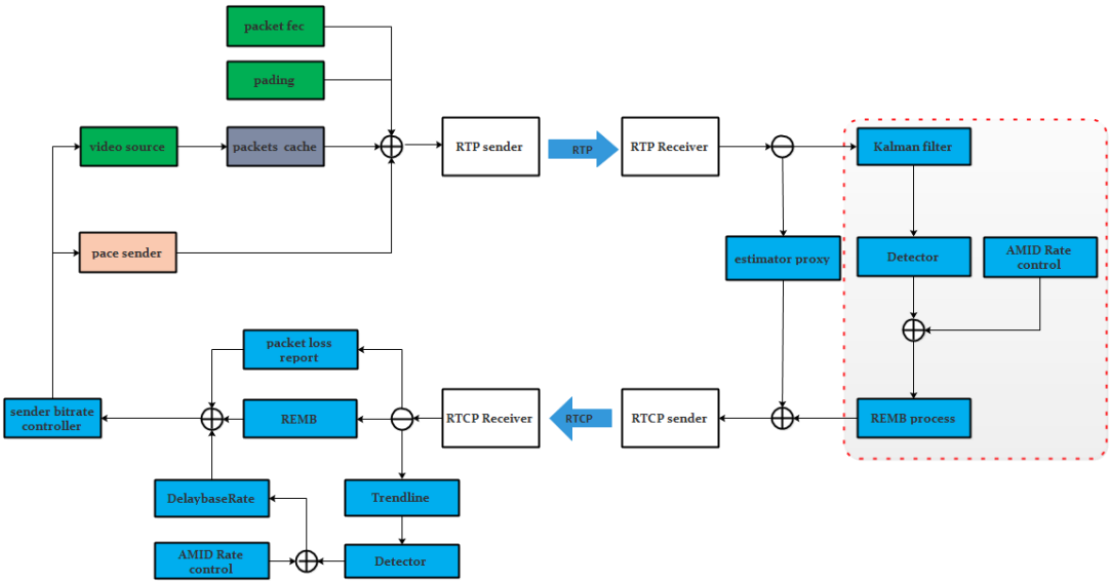


图 2：WebRTC 的拥塞控制模块关系图

需要说明的是红框中基于接收端的 kalman filter 带宽评估模型已经在新版本的 WebRTC 中不采用了，只做了向前版本兼容，新版本的 WebRTC 都是采用发送端的 trendline 滤波器来做延迟带宽评估，本文中重点是介绍基于 trendline 滤波的评估模型，下面依次来分析 WebRTC 的这五个过程。

1 estimator

estimator 的功能就是通过接收端反馈过来的包到达时刻信息、丢包信息和 REMB 信息进行当前网络状态的码率评估，WebRTC 拥塞控制有两部分：基于延迟的拥塞控制和基于丢包的拥塞控制，它是一个尽力而为的拥塞控制算法，牺牲了拥塞控制的公平性换取尽量大的吞吐量。从设计结构来描述向它输入延迟和丢包信息，它就会输出一个适应当前网络状态的码率值。示意图如下：

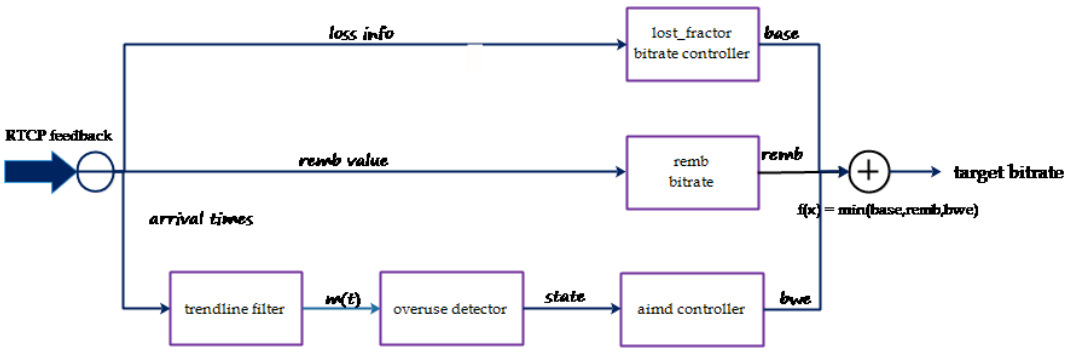


图 3：WebRTC 的 CC estimator 输入与输出

从上图可以看出，estimator 基于延迟的拥塞控制是通过 trendline 滤波再进行过载判断，最后根据过载情况进行 aimd 码率调控评估出一个 bwe bitrate 码率，这个码率会合丢包评估出来的码率和 remb 来决定最后的码率。

1.1 基于延迟的拥塞控制

基于延迟的拥塞控制是通过每组包的到达时间的延迟差（delta delay）的增长趋势来判断网络是否过载，如果过载进行码率下调，如果处于平衡范围维持当前码率，如果是网络承载不饱满进行码率上调。这里有几个关键技术:包组延迟评估、滤波器趋势判断、过载检测和码率调节。

1.1.1 包组与延迟

WebRTC 在评估延迟差的时候不是对每个包进行估算，而是采用了包组间进行延迟评估，这符合视频传输（视频帧是需要切分成多个 UDP 包）的特点，也减少了频繁计算带来的误差。那么什么是包组呢？就是距包组中第一个包的发送时刻 t0 小于 5 毫秒发送的所有的包成为一组，第一个超过 5 毫秒的包作为下一个包组第一个包。为了更好的说明包组和延迟间的关系，先来看示意图：

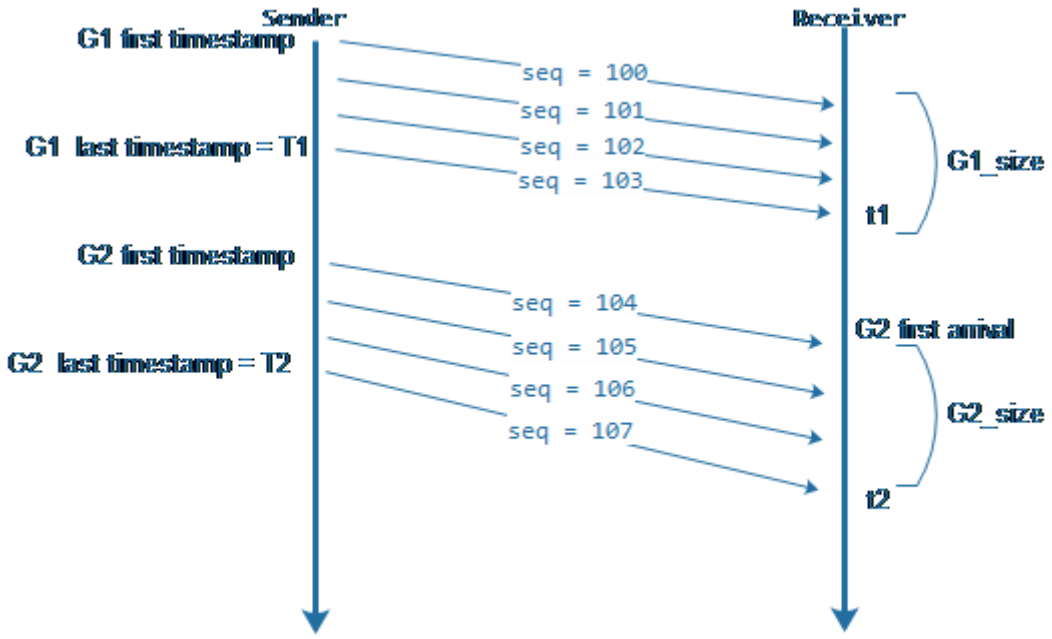


图 4：包组与延迟示意图

上图中有两个包组 G1 和 G2，其中第 100 号包与 103 号包的时间差小于 5 毫秒，那么 100 ~ 103 被划作一个包组。104 与 100 之间时间超过 5 毫秒，那么 104 就是 G2 的第一个包，它与 105、106、107 划作一个包组。知道了包组的概念，那么我们怎么通过包组的延迟信息得到滤波器要的评估参数呢？滤波器需要的三个参数：发送时刻差值（delta_timestamp）、到达时刻差值(delta_arrival)和包组数据大小差值(delta_size)。从上图可以得出：

$$\mathrm{delta}_{\mathrm{timestamp}} = T2 - T1$$
$$\mathrm{delta}_{\mathrm{arrival}} = \mathrm{t2} - \mathrm{t1}$$
$$\mathrm{delta}_{\mathrm{size}} = \mathrm{G2}_{\mathrm{size}} - G1_{size}$$

1.1.2 滤波器

我们通过包组信息计算到了 delta_timestamp、delta_arrival 和 delta_size,那么下一步就是进行数据滤波来评估延迟增长趋势。在 WebRTC 实现了两种滤波器来进行延迟增长趋势的评估，分别是：kalman filter 和 trendline filter，从图 2 中我们知道 kalman filter 是运行在接收端的，我在这里以不做介绍，有兴趣的可以参考 https://www.jianshu.com/p/bb34995c549a。

这里介绍 trendline filter，我们知道如果平稳网速下传输数据的延迟时间就是数据大小除以速度，如果这数据块很大，超过恒定网速下延迟上限，这意味着要它要占用其他后续数据块的传输时间，那么如此往复，网络就产生了延迟和拥塞。Trendline filter 通过到达时间差、发送时间差和数据大小来得到一个趋势增长值，如果这个值越大说明网络延迟越来越严重，如果这个值越小，说明延迟逐步下降。以下是计算这个值的过程。

先计算单个包组传输增长的延迟，可以记作：

$$\mathrm{delay}_{\mathrm{i}} = delta_{arrival} - delta_{timestamp}$$

然后做每个包组的叠加延迟,可以记作：

$$\mathrm{acc}_{\mathrm{delay_i}} = \sum delay_0 + delay_1 + delay_2 + \cdots + delay_i$$

在通过累积延迟计算一个均衡平滑延迟值，alpha=0.9 可以记作：

$$\mathrm{smo}_{\mathrm{delay_i}} = alpha \quad \times smo_{delay_i-1} + (1 - alpha) \times acc_{delay_i}$$

然后统一对累计延迟和均衡平滑延迟再求平均，分别记作：

$$\mathrm{x}_{\mathrm{i}} = \frac{\sum acc_{delay_0} + acc_{delay_1} + \cdots + acc_{delay_i}}{i}$$
$$\mathrm{y}_{\mathrm{i}} = \frac{\sum smo_{delay_0} + smo_{delay_1} + \cdots + smo_{delay_i}}{i}$$

我们将第 i 个包组的传输持续时间记作:

$$\text{trans}_i = t_i - first_arrival_i$$

趋势斜率分子值为:

$$\text{numerator}_i = \sum_{k=0}^{i} (trans_k - x_k) \times (smo_{delay\ k} - y_k)$$

趋势斜率分母值为:

$$\text{denoinator}_i = \sum_{k=0}^{i} (trans_k - x_k) \times (trans_k - x_k)$$

最终的趋势值为:

$$\text{trendline}_i = \frac{numerator_i}{denoinator_i}$$

1.1.3 过载检测

在计算得到 trendline 值后 WebRTC 通过动态阈值 gamma_1 进行判断拥塞程度，trendline 乘以周期包组个数就是 m_i,以下是判断拥塞程度的伪代码:

```
if |m_i| > γ then
    if m_i > 0 then
        t_OU ← t_OU + ΔT;
        if t_OU > t̄_OU then
            if m_i ≥ m_{i-1} then
                t_OU ← 0;
                s ← Overuse;
    else
        t_OU ← 0;
        s ← Underuse;
else
    t_OU ← 0;
    s ← Normal;
```

通过以上伪代码就可以判断出当前网络负载状态是否发生了过载，如果发生过载，WebRTC 是通过一个有限状态机来进行网络状态迁徙，关于状态机细节可以参看下图:

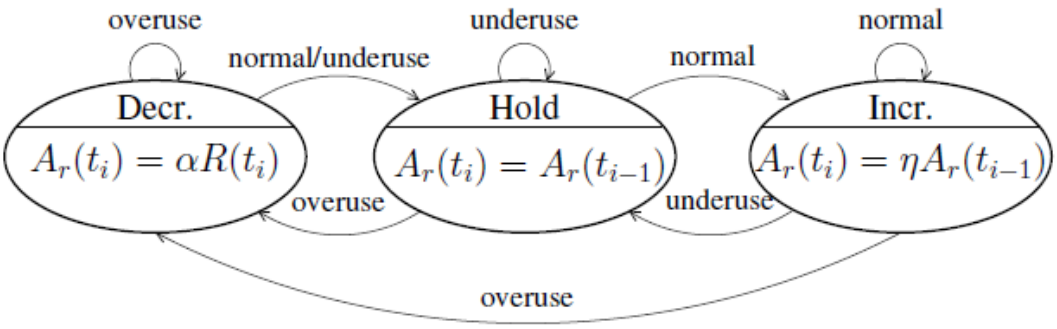


图 5:过载检测状态机

从上图可以看出，网络状态机的状态迁徙是由于网络过载状态发生了变化，所以状态迁徙作为了 aimd 带宽调节的触发事件，aimd 根据当前所处的网络状态进行带宽调节，其过程是处于 Hold 状态表示维持当前码率，处于 Decr 状态表示需要进行码率递减，处于 Incr 状态需要进行码率递增。那他们是怎么递增和递减的呢？WebRTC 引入了 aimd 算法解决这个问题。

1.1.4 AIMD 码率调节

aimd 的全称是 Additive Increase Multiplicative Decrease，意思是：和式增加，积式减少。Aimd controller 是 TCP 底层的码率调节概念，但是 WebRTC 并没有完全照搬 TCP 的机制，而是设计了套自己的算法，用公式表示为：

$$A_r(t_i) = \begin{cases} \eta A_r(t_{i-1}) & \sigma\text{=Increase} \\ \alpha R_r(t_i) & \sigma\text{=Decrease} \\ A_r(t_{i-1}) & \sigma\text{=Hold} \end{cases}$$

如果处于 Incr 状态，增加码率的方式分为两种：一种是通信会话刚刚开始，相当于 TCP 慢启动，它会进行一个倍数增加，当前使用的码率乘以系数，系数是 1.08；如果是持续在通信状态，其增加的码率值是当前码率在一个 RTT 时间周期所能传输的数据速率。

如果处于 Decrease 状态，递减原则是:过去 500ms 时间窗内的最大 acked bitrate 乘上系数 0.85,acked bitrate 通过 feedback 反馈过来的报文序号查找本地发送列表就可以得到。

Aimd 根据上面的规则最终计算到的码率就是基于延迟拥塞评估到的 bwe bitrate 码率。

1.2 基于丢包的拥塞控制

除了延迟因素外，WebRTC 还会根据网络的丢包率进行拥塞控制码率调节，描述如下：

$$A_s(t_k) = \begin{cases} A_s(t_{k-1})(1 - 0.5 f_l(t_k)) & f_l(t_k) > 0.1 \\ 1.05(A_s(t_{k-1})) & f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases}$$

解释下上面的公式：
当丢包率>2%时，这个时候会将码率（base bitrate）增长 5%,这个码率(base bitrate)并不是当前及时码率，而是单位时间窗周期内出现的最小码率,WebRTC 将这个时间窗周期设置在 1000 毫秒内。因为 loss fraction 是从接收端反馈过来的，中间会有时间差，这样做的目的是防止网络间歇性统计造成的网络码率增长过快而网络反复波动。

当 2% < 丢包率 < 10%,维持当前的码率值
当 丢包率 >= 10%, 按丢包率进行当前码率递减，等到新的码率值

丢包率决策出来的码率（base bitrate）只是一个参考值，WebRTC 实际采用的带宽是 base bitrate、remb bitrate 和 bwe bitrate 中的最小值，这个最小值作为 estimator 最终评估出来的码率

2 pacer

在 estimator 根据网络状态决策出新的通信码率（target bitrate），它会将这个码率设置到 pacer 当中，要求 pacer 按照新的码率来计算发包频率。因为在视频通信中，单帧视频可能有上百 KB,如果是当视频帧被编码器编码出来后，就立即进行 RTP 打包发送，瞬时会发送大量的数据到网络上，可能会引起网络衰减和通信恶化。WebRTC 引入 pacer，pacer 会根据 estimator 评估出来的码率，按照最小单位时间（5ms）做时间分片进行递进发送数据，避免瞬时对网络的冲击。pacer 的目的就是让视频数据按照评估码率均匀的分布在各个时间片里发送， 所以在弱网的 WiFi 环境，pacer 是个非常重要的关键步骤。以下 WebRTC 中 pacer 的模型关系：

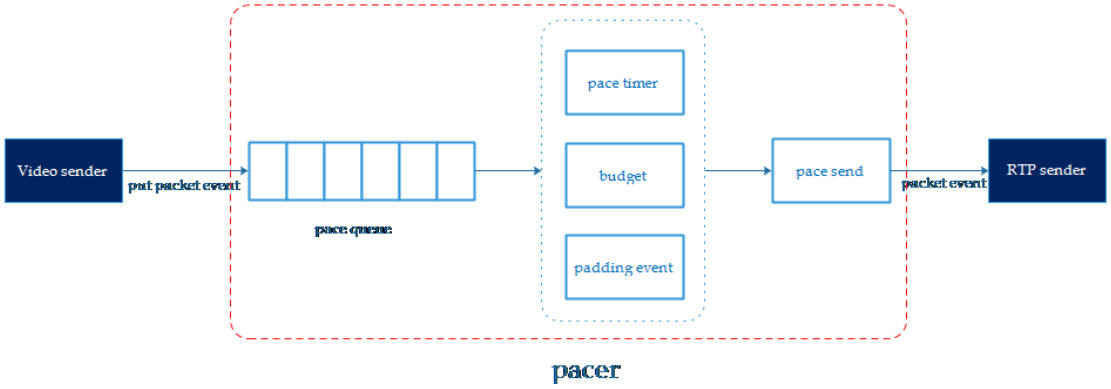


图 6：pacer 模型图

WebRTC 中 pacer 的流程比较清晰，分为三步：

1） 如果一帧图像被编码和 RTP 切分打包后，先会将 RTP 报文存在待发送的队列中，并将报文元数据(packet id, size, timestamp，重传标示)送到 pacer queue 进行排队等待发送，插入队列的元数据会进行优先级排序。
2） pacer timer 会触发一个定时任务事件来计算 budget，budget 会算出当前时间片网络可以发送多少数据，然后从 pacer queue 当中取出报文元数据进行网络发送。
3） 如果 pacer queue 没有更多待发送的报文，但 budget 却还可以发送更多的数据，这个时候 pacer 会进行 padding 报文补充。

从上面的步骤描述中可以看出 pacer 有几个关键技术：pace queue、padding、budget。

2.1 pace queue 与优先级

pace queue 是一个基于优先级排序的多维链表，它并不是一个先进先出的 fifo,而是一个按优先级排序的 list。

报文优先级规则
1） 优先级高的报文排在 fifo 的前面，低的排在后面。
2） 优先级是最先判断报文的 QoS 等级，等级越小的优先级越高
3） 其次是判断重发标示，重发的报文比普通报文的优先级更高
4） 再次是判断视频帧 timestamp，越早的视频帧优先级更高。

pacer 每次触发发送事件时是先从 queue 的最前面取出优先级最高的报文进行发送，这样做的目的是让视频在传输的过程中延迟尽量小，重传的报文尽快能到达防止等待卡顿。pace queue 还可以设置最大延迟，如果超过最大延迟，会计算 queue 中数据发送所需要的码率，并且会把这个码率替代 target bitrate 作为 budget 参考码率来加速发送。

2.2 budget

budget 是个评估单位时间内可以发送多少数据量的一个机制，因为 pacer 是会根据 pace timer 定时来触发发送检查。Budget 会根据评估出来的参考码率计算这次定时事件能发送多少字节，可以表示为：

$$\text{remain bytes} = \text{delta}_{\text{time}} \times target_{bitrate} \div 8$$

delta time 是上次检查时间点和这次检查时间点的时间差。
target bitrate 是 pacer 的参考码率，是由 estimator 根据网络状态评估出来的。
remain_bytes 每次触发发包时会减去发送报文的长度 size,如果 remain_bytes > 0，继续从 pace queue 中取下一个报文进行发送，直到 remain_bytes <=0 或者="" pace="" queue 没有更多的报文。<="" span="">

2.3 pacer 延迟

那么肯定有人会有疑问 pacer queue 和 budget 进定量计算来发送网络报文，相当于 cache 等待发送，难道不会引起延迟吗？可以肯定的说会引起延迟，但延迟不严重。pacer 产生的延迟可以表示为：

$$\text{delay} = \frac{frame_size \times 8}{bitrate}$$

假如评估出来的码率是 10mbps，一个视频关键帧的大小是 300KB,那么这个关键帧造成的 pacer delay 是 240 毫秒。从实际应用观察到的关键帧引起的 pace delay 在 200 ~ 400 毫秒之间，这个值相对于视频传输来说是比较大的，但是不严重。WebRTC 为了减少这个延迟，会评估出尽量大的 bitrate。那么怎么评估出尽量大的码率呢？从前面的 estimator 描述中我们知道要发送出尽量多的数据才能评估尽量大的码率，但是视频编码器不会发送多余的数据，所以 WebRTC 引入了 padding 机制来保障发送尽量大的数据来探测网络带宽上限。

2.4 padding

pace padding 除了保障能 pace delay 尽量小外，它可以让有限的带宽获得尽可能好的视频质量。padding 的工作原理很简单，就是在单位时间片内把 budget 还剩余的空闲用 padding 数据填满。我个人认为 padding 只是适合点对点通信，一旦涉及到多点分发，会因为 padding 占用很多服务转发带宽，这并不是一件好事情。

3 sender

WebRTC 的发送模块和拥塞控制控制相关的主要是增加了附加的 RTP 扩展来携带便宜接收端统计丢包率和延迟间隔的信息、配合 pacer 的发包策略、带宽分配和 FEC 策略的信息。

3.1 RTP 扩展

WebRTC 为了配合接收端进行延迟包序列和丢包统计做了下列扩展：

transport sequence 传输通道的只增 sequence，每次发送报文时自增长，配合接收端统计丢包、通过反馈这个 sequence 可以计算得到发包的时刻。

TransmissionOffset 发送报文的相对时刻，这个相对时刻值 t 是发送报文的绝对时刻 T1 和视频帧时间戳 T0 差值。早期的 WebRTC 是在接收端进行 estimate bitrate，所以过载判断是在接收端完成的，这个值就是为了 kalman filter 计算发包造成的延迟用的，新版本还携带这个值以便低版本的 WebRTC 能兼容。

3.2 packet cache

packet cache 是一个 key/value 结构的包缓冲池，视频帧在进行 RTP 分片打包后不会立即发送出去，而是要等待 pacer 的发送信号进行发送。所以打包后会按[id,packet]键值对插入到 packet cache 中。一般 packet cache 会保存 600 个分片报文，最大 9600 个，插入新的会将最旧的报文删除，packet cache 这样做的目的除了配合 pacer 发送外，也为了后面响应 nack 的丢包重传。

3.3 NACK 与丢包重传

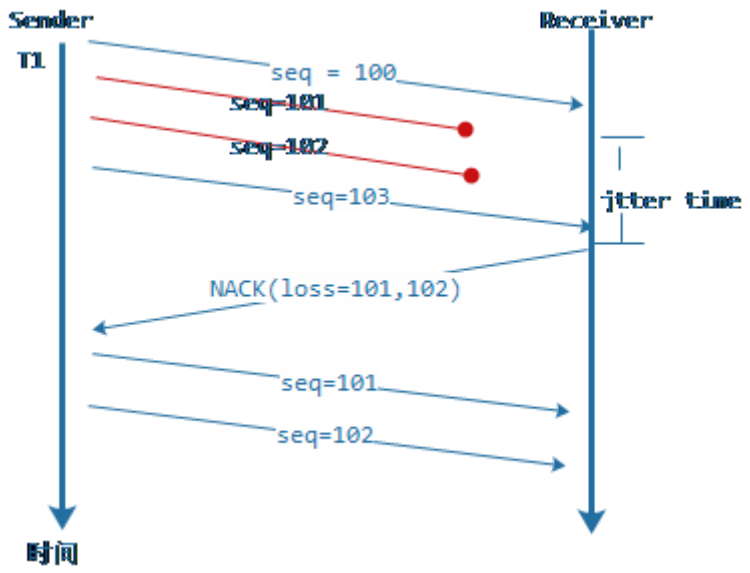


图 7:RTP NACK 过程的示意图

WebRTC 在评估到收发端之间 RTT 延迟比较小的时候会采用 NACK 来进行丢包补偿，NACK 是一个请求重发过程，其流程如上图所示。这个过程有一个问题是在网络抖动和丢包很厉害的情况下有可能造成同一时刻收到很多 NACK 的重传请求，发送端瞬间把这些重传请求放入 pacer 中进行重发，这样 pacer 的延迟会增大，而且 pace 的参考码率会随着 pace queue 的延迟控制变的很大而出现间歇性网络风暴。WebRTC 在处理 NACK 重传时设计了一个重传码率控制器,其设计原理是通过统计单位时间窗口周期中发送的字节数据来限流，如果这个时间窗内发送的数据的码率大于 estimator 评估的码率，不进行当前 NACK 请求的重传，等待下一个 NACK。

3.4 FEC 与码率分配

WebRTC 应对丢包时除了 NACK 方式，在收发端之间 RTT 很大时候会开启 FEC 来进行丢包补偿，我们在这里不介绍 FEC 具体算法，只介绍 FEC 的码率分配策略。从整个通信机制我们很容易得出这样一个共识：

$$\text{target bitrate} = \text{video bitrate} + \text{feedback bitrate} + \text{FEC bitrate} + \text{padding bitrate}$$

FEC bitrate 到底应该设置多大呢？它先根据 feedback 中反馈过来的丢包率(loss fraction)来确定使用哪一种 FEC，在根据每中 FEC 和丢包率来确定 FEC 使用的码率，但需要满足一下条件:

$$\text{FEC bitrate} < \text{target bitrate} / 2$$

feedback 的码率被设定为 target bitrate 的 5%,WebRTC 是通过控制 feekback 的频率来进行调控分配的。padding bitrate 是通过 pacer queue 和 budget 来控制的。Target bitrate 减去这些码率之和就是给视频编码器的码率。每次 estimator 评估出来码率后，会先进行这些计算得到最后的 video bitrate,并将这个值作为编码器的编码码率，以此来达到防止拥塞的目的。

4 receiver

receiver 模块的工作相对来说比较简单，它就做三件事情:记录每个报文的到达时刻(arrival timestamp)、丢包率(lost fraction)和 receiver bitrate。早期的 WebRTC 提供了图 2 红框当中 kalman filter 评估码率的评估器，因为 kalman filter 怕抖动特性且需要借助 remb 心跳进行反馈，remb 的反馈周期是 1 秒，在收发端网络间歇性断开或者大抖动下，容易失效，所以 WebRTC 采用了在发送端进行估算，整个逻辑也更加简便。

4.1 报文到达时间

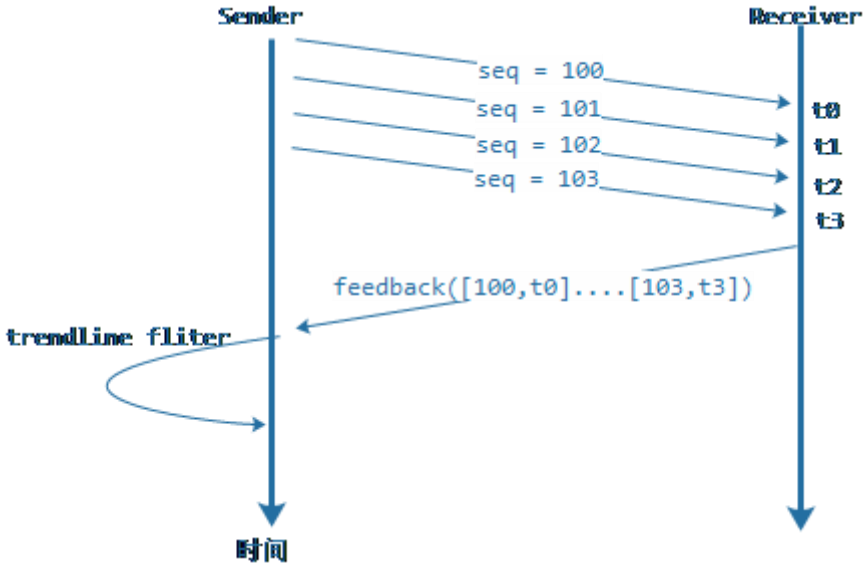


图 8：到达报文统计图

上图是一个统计 RTP 报文到达时刻的序列图，图中的 seq 是 RTP 扩展中的 transport sequence,接收端用一个 k/v（[seq,arrival timestamp]）键值对数据结构来保存最近 500 毫秒未反馈的到达时刻信息,通过时间窗口周期来进行淘汰老的到达时刻记录。

4.2 丢包率计算

丢包率计算过程是这样的，我们把上次统计丢包率时刻的最大 sequence 记着 prev_seq，把当前收到的最大 sequence 记着 cur_seq,当前统计丢失的报文记着 count，WebRTC 在 RTCP 中描述丢包率采用的是 uint8，为了保证精确度将 256 记着 100%的丢包率，那么很容易得：

$$\text{loss fraction} = \frac{count \times 256}{cur_{seq} - prev_{seq}}$$

这里需要提的是 WebRTC 在统计报文是否丢失是通过 sequence 的连续性和网络的 jitter 时间来确定的，只有落在 jitter 抖动范围之外的丢包才是算是作丢包。

4.3 接收码率统计

接收端码率统计采用的是最近单位时间窗（1000 毫秒）周期内收到的的字节数来计算，WebRTC 设计了一个 1 毫秒为最小单位的窗口数组来进行统计,每个最小单位是数字，这个数字是在这个时刻收到的网络数据大小，大致的示意图如下：

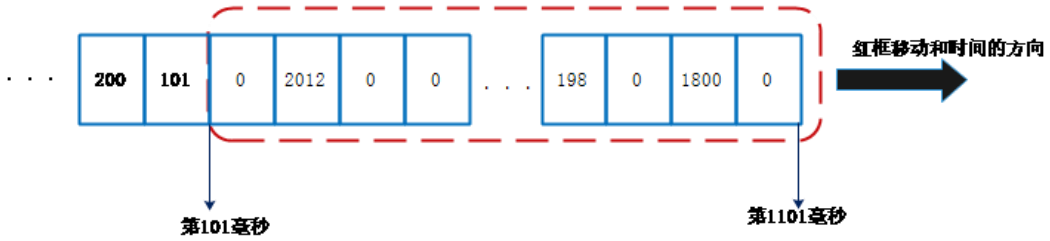


图 9：接收码率统计示意图

计算码率只需要将红框中所有的数字加起来，当时间发生改变后，就红框就向右移动并且填写新时刻接收到的数据大小，等下一个统计时刻既可。

5 feedback

前面介绍的 estimator 依赖于 feedback 反馈的报文到达时刻和丢包率来进评估码率的，也就是说 feedback 需要将这些信息及时反馈给接收端，主要是记录的报文到达时刻、通道丢包率和 remb 带宽。因为报文到达时刻和丢包率统计都是多个数据项，WebRTC 利用了 report block 来进行编码存放。为了有效的利用 RTCP 的 report block 空间，WebRTC 采用了相对时间转换和位压缩算法来对到达时间序列做编码压缩。

除了 report 编码，feedback 的周期也很重要，如果是单纯的 remb 反馈，一般是 1 秒一次反馈。但如果是需要反馈报文的到达时间，它会根据占用 5%的 target bitrate 来计算发送 feedback 的时间间隔，计算流程如下：

$$\text{feedback}_{\text{bitrate}} = \text{target}_{\text{bitrate}} \times 5\%$$
$$\text{feedback}_{\text{size}} = \text{IPv4}_{\text{header}} + \text{UDP}_{\text{header}} + \text{SRTP}_{\text{header}} + \text{report_block}_{\text{size}}$$
$$\text{feedback}_{\text{interval}} = \frac{feedback_{size}}{feedback_{bitrate}}$$

feedback interval 需要满足一个条件：50ms < interval < 250ms,这个条件中的 50ms< internal 是为了防止 interval 太小造成发送 feedback 太过频繁而消耗网络性能，而 interval < 250ms 是为了防止 feedback 频次太低造成 estimator 反应迟钝。

6 总结

以上就是 WebRTC 拥塞控制和码率调节策略的 5 个过程，里面涉及到很多传输相关的技术，我在这里也是简单介绍了下其工作原理，很多细节的并没有描述出来，也很难描述出来，有兴趣的同学可以翻看 WebRTC 的源代码。如果觉得 webRTC 代码费劲，我照虎画猫将 WebRTC 的拥塞控制用 C 重新实现了个简易版本,但是去掉了 padding，可到 https://github.com/yuanrongxi/razor 下载。

6.1 效果

WebRTC 的 GCC 在网络适应上表现还是比较良好的，既然兼顾延迟，也能兼顾丢包，网络发生拥塞时在 2 ~ 3 秒内能评估出相对的码率来适应当前的网络状态，但是会造成短时间的卡顿。对于网络发生间歇性丢包，在 2 秒左右能将传输码率适配到当前网络状态。它在网络相对稳定且延迟较大的网络进行高分辨率传输时，视频很稳定，适合长距离延迟稳定的网络环境。在弱网环境下，WebRTC 容易将码率降到很低而造成图像失真。

6.2 网络大抖动

对于乱序和抖动 WebRTC 的拥塞控制显得有点无力，如果抖动超过 rtt*2/3 时，基于 kalman filter 的带宽评估机制不起作用（不知道是不是我用错了）；基于 trendline 滤波的评估机制波动很大，敏感度不够，不能完全反应当前的网络过载状态，尤其是在终端 Wi-Fi 拥挤的情况下，比较容易造成间歇性风暴。

6.3 延迟问题

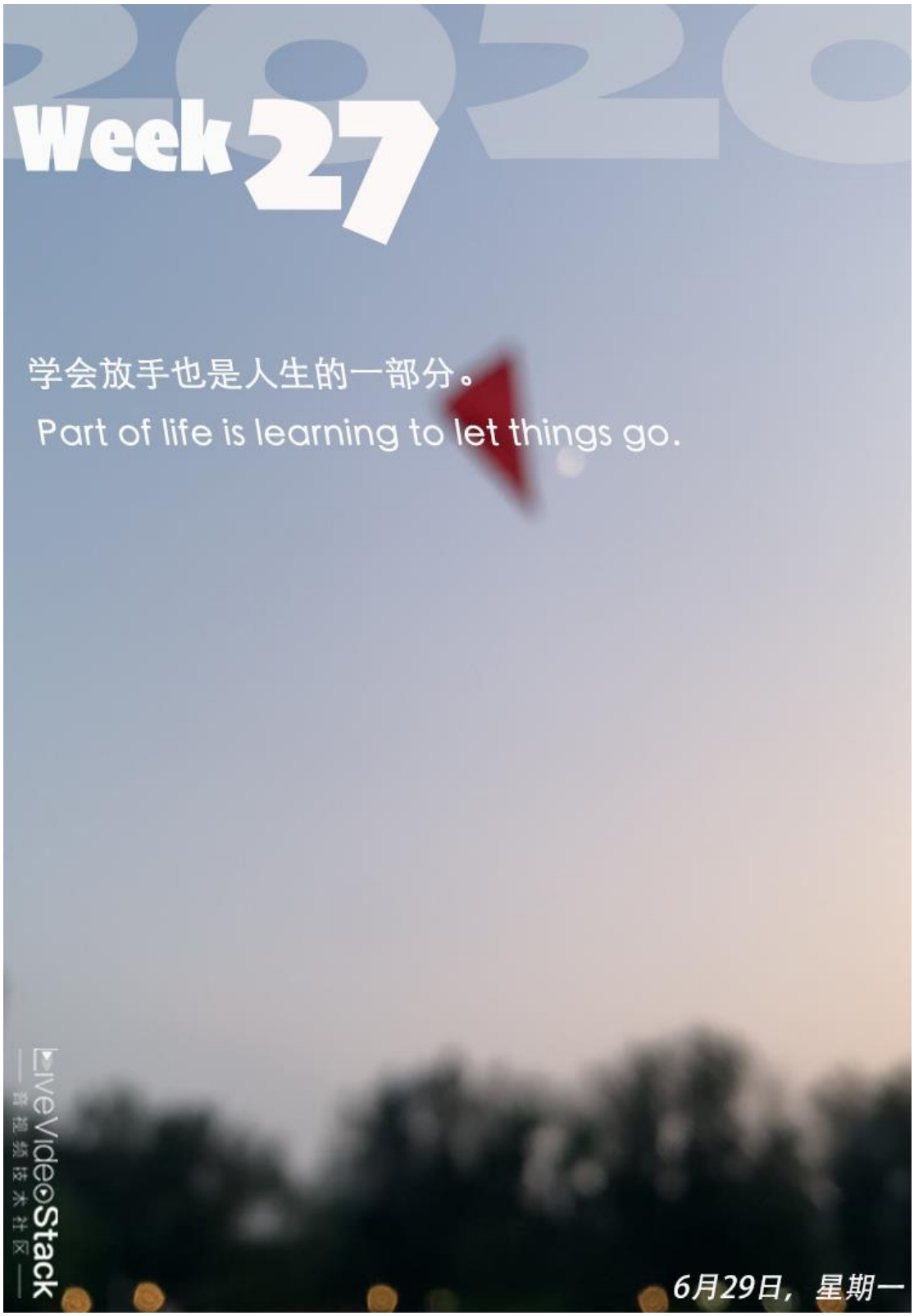
WebRTC 的 pacer 在传输大分辨率视频时，关键帧会引起大约 200 毫秒的延时，尤其是在移动 4G 网络下这个问题更加明显，海康威视工程师郑鹏提出了用 H.264 的 intre_refresh 模式来应对，在测试过程中确实比较适合 WebRTC 用来减少关键帧造成的延迟，但是 intre_refresh 是普通模式编码 CPU 的 3 倍左右，而且很多移动设备的编码器不一定支持。

总之，WebRTC 的拥塞控制存在反应慢、怕抖动的特性，但是这块也是 WebRTC 改进最为频繁的模块，几乎每个版本都有新的改进。要彻底解决这样的问题，需要从视频编码器和网络传输进行融合来解决，以后我用单独的篇幅来介绍下这样的解决方案。

仰望星空，脚踏实地

2020 年热点新闻

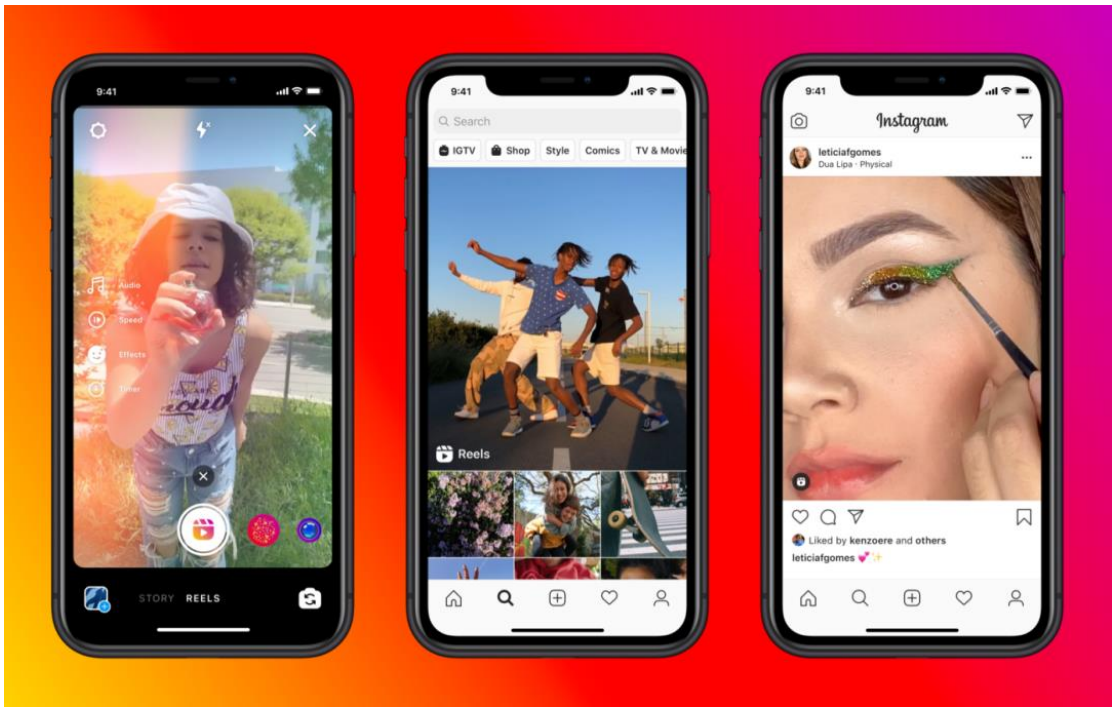
声网纳斯达克上市、TikTok 推出 TikTok For Business、亚马逊推出无代码应用| Decode the Week



世事皆如此，没人喜欢改变。
Decode the Week≠音视频技术周刊

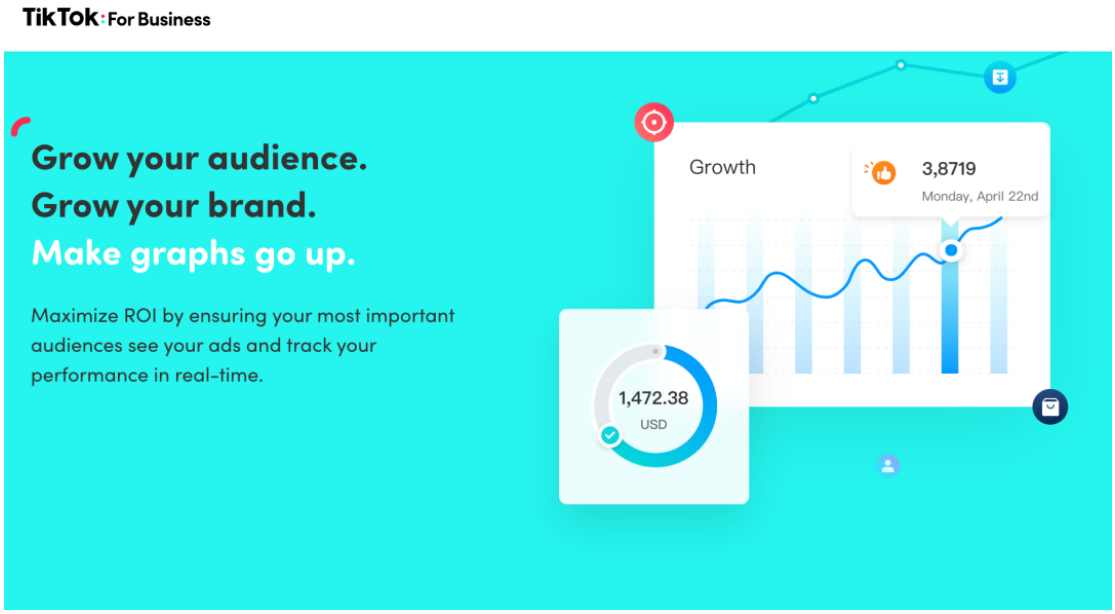
News Briefing

1. Instagram 将 TikTok 克隆版本“Reels”扩展至欧洲市场



Instagram 将自己的 TikTok 克隆版本“Reels”扩展到了新的市场——法国和德国，它允许用户录制 15 秒的视频片段，然后在平台上分享。Instagram 并不是唯一一个担心 TikTok 的平台。同样在上周，YouTube 也宣布开始测试一种新的移动端短视频应用。

2. TikTok 推出 TikTok For Business



那边Instagram和YouTube忙着复刻TikTok，这边TikTok正式推出了名为TikTok For Business（https://www.tiktok.com/business/en）的新平台，作为其目前和未来所有品牌营销解决方案的大本营。TikTok 对广告商的承诺是，不同于其他网络，广告商们将真正成为 TikTok 社区的一部分。

3. 苹果将允许开发者挑战 Appstore 规则

还记得上周的新闻“苹果称如果 Basecamp 不使用苹果商店的 IAP（应用内购买 in-app

purchase)，则不能提交任何更新或 bug 修复（Hey，Apple）”吗，苹果在 WWDC2020（全球开发者大会）的公告里回应了这一争论：

“苹果应用商店的开发者可以挑战 AppStore 的指导原则（App Store developers can challenge guidelines)。”

Basecamp 的抱怨是许多开发者都能想到的——他们觉得 App Store 的规则对他们的业务毫无意义，但同时自己在这个问题上没有发言权，也没有任何反击的方法。

而 Basecamp 最终也与苹果妥协，发布了一个带有临时电子邮件地址的免费版本。
苹果表示，自己将允许开发者对应用商店的违规行为提出上诉。此外，开发者还将获得一种机制来挑战应用商店的指导原则。苹果也将不会阻止开发人员发布提交更新或 bug 修复。而这些新规则将在今年夏天晚些时候正式推出。

4. 苹果将从中国应用商店下架数千款游戏

彭博社称，因监管许可问题，苹果公司将从 7 月份开始从其中国应用商店下架数千款游戏。游戏开发商可以在等待批准的情况下，在 App Store 上发行和销售自己的游戏。据估计，这将影响目前在中国应用商店的 6 万款游戏中的大约三分之一。

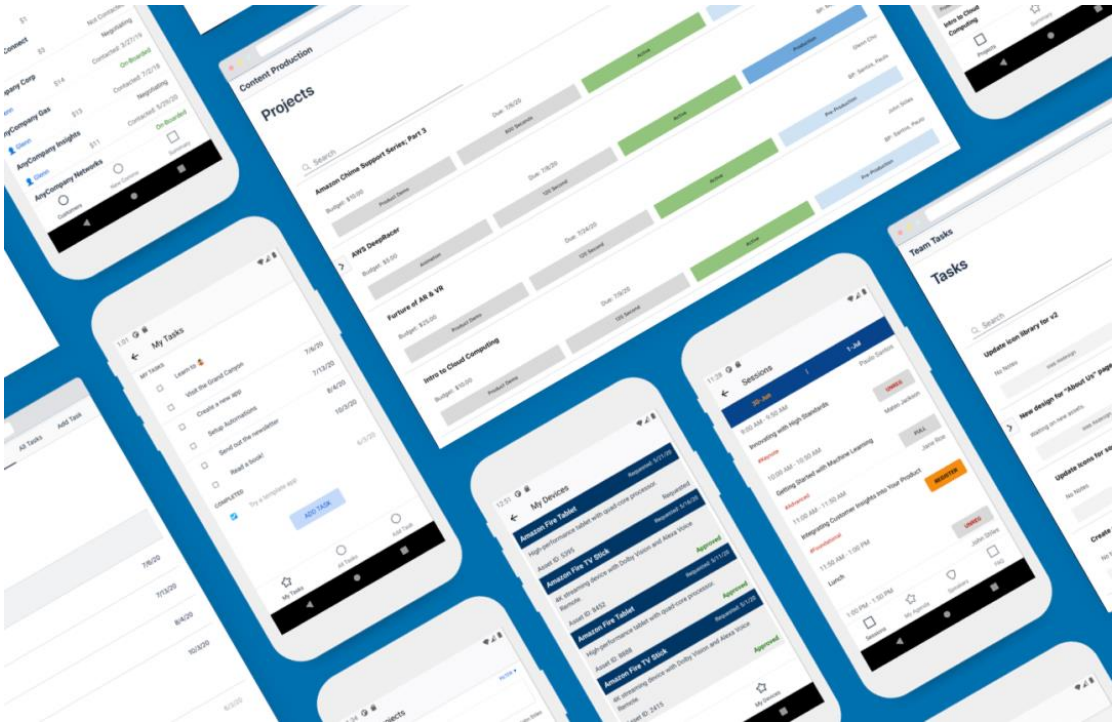
5. 微软将永久关闭所有零售店

随着其他零售商开始缓慢的重新开张，微软宣布将永久关闭其绝大多数零售店，包括在伦敦、纽约和悉尼等城市的零售店，有些将成为“微软体验中心”，而不是标准的零售店。

6. 孙正义辞去阿里巴巴董事会职务

软银创始人孙正义表示，他将离开阿里巴巴集团董事会。一个月前，马云离开了软银董事会。孙正义认为这是一次“毕业”，而阿里巴巴是他迄今为止最成功的投资之一。

7. 亚马逊推出无代码应用程序开发软件 Honeycode



Honeycode 的目的是让任何人都可以轻松地构建自己的应用程序。

AWS 副总裁 Larry Augustin 在声明中表示:“客户告诉我们，对定制应用程序的需求远远超过了开发者的能力。现在有了 Amazon Honeycode，几乎任何人都可以创建强大的定制化的应用程序，而无需编写代码。”

Honeycode 为用户提供了一组通用的模板，如待办事项列表应用程序、客户跟踪器、调查、时间表和库存管理。

Honeycode 的首批客户包括 SmugMug 和 Slack。

声网 Agora 纳斯达克上市

IPO 市场变得更加多元化且富有包容性，而公众投资者也比以往任何时候更希望看到有潜力的科技公司上市。
声网 Agora 将 IPO 发行价定在每股 20 美元，高于其 16 至 18 美元的拟议价格区间，此举突显出美国 IPO 窗口目前可能非常开放。

以每股 20 美元的价格发行 1750 万股股票，IPO 募资额达 3.5 亿美元，总融资规模达 4.6 亿美元，对于一家在 2020 年第一季度收入中仅占这一数字 10%左右的公司来说，无疑是一笔巨大的资金。

声网允许客户在其应用程序中嵌入实时视频和语音功能，该公司在美国的股票代码也是“API”。

声网在 2019 年第一季度至 2020 年第一季度的增长为 166%，但其 2020 年第一季度的毛利率低于市场对 SaaS 公司的预期，净亏损呈现上升趋势:2019 年一季度和 2020 年第二季度分别为 1340 万美元到 3640 万美元。

虽然随着公司本身的发展，声网的亏损一直在增加，但其市盈率仍远高于 10 倍。由此（同时根据数据）可以发现，声网的增长率不是由支出推动的，而是建立在适度的现金消耗和不断增长的盈利能力之上的。

但现在还不能判断的是，声网能够留住多少真正（或者说“所谓的”）“长期投资者”。科技界总是宣称深爱长期投资者——这群人往往比急于在近期新股发行中找到上行机会的动量交易者更有价格意识。

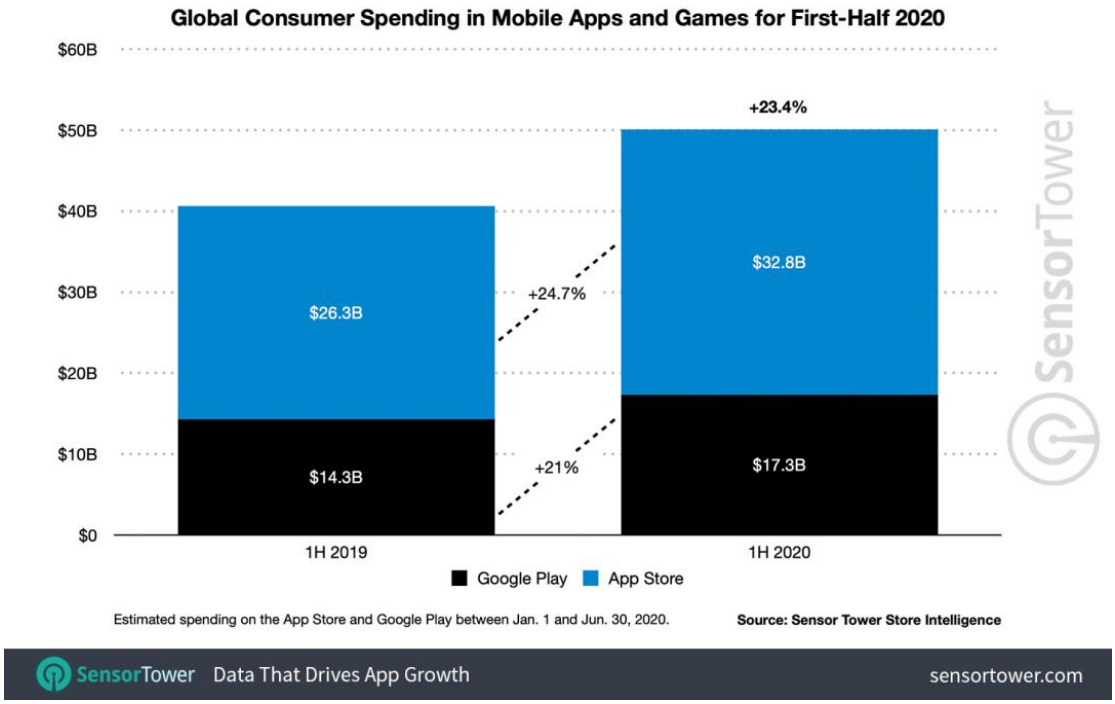
Decode the Week 选取新鲜有趣的音视频（技术/非技术）新闻与大家分享——也欢迎你通过后台留言或邮件（contribute@livevideostack.com）与我们分（爆）享（料）圈内趣闻。

TikTok 关键市遭威胁、Facebook 最薄 VR 显示器、ZOOM 下一站与视频产品 2020| Decode the Week

又或许，成功不该是人生唯一的追求/As there are so many things much more important.
Decode the Week≠音视频技术周刊

News Briefing

1. 移动端应用 2020 上半年收益超 500 亿美元



根据 Sensor Tower 的新数据，消费者在移动应用和应用安装上的支出在 2020 年上半年大幅增长，部分原因是疫情。

今年上半年，全球消费者在 App Store 和谷歌 Play 上花费了 501 亿美元，比 2019 年上半年增长了 23.4%。相比之下，在 2018 年上半年到 2019 年期间，营收增长了 20%。此外，2020 年上半年，首次应用程序安装量同比增长 26.1%，下载量达到 715 亿次。
其中苹果应用商店的下载量为 183 亿次，同比增长 22.8%，而谷歌 Play 的新应用安装量为 532 亿次，同比增长 27.3%。

2. Facebook 展示概念性最薄 VR 显示器

Facebook 现实实验室(FRL)一直在探索新的光学结构，以提高 VR 佩戴设备的外观因素、舒适度和光学性能。今年，在 virtual SIGGRAPH 会议上，FRL 首席科学家迈克尔·阿布拉什(Michael Abrash)展示了一种新的光学结构，它更加紧凑且拥有更好的视觉性能。

这一方法在一个概念验证研究设备中得到了演示，该设备仅使用薄且平展的薄膜作为光学器件，

以实现不到 9 毫米的显示厚度，同时支持的视场可与当今的消费者 VR 产品相媲美。 大多数 VR 显示器都有一个简单的折射镜片，而 Facebook 团队建议用全息光学代替这个笨重的元件。其结果是设备在厚度和重量上大幅减少:全息光学像透镜一样弯曲光线，但看起来像一个薄的透明贴纸。

虽然它指向了未来轻量化、舒适性、高性能的 AR/VR 技术的发展方向，但目前 Facebook 的工作是纯研究性质的。

4. 数十款中国应用程序在印度被禁，威胁 TikTok 关键市场增长

印度政府表示，由于担心中国公司开发的应用程序参与了威胁“印度国家安全和国防”的活动，印度上周封堵了 59 个中国公司开发的应用程序。

这项禁令本身是一项政治举措，印度政府官员声称，他们已经收到报告，称这些应用程序以未经授权的方式窃取用户数据，并将其传输到国外的服务器上。

印度禁止中国应用程序的举动对 TikTok 的影响最为显著。迄今为止，印度一直是该应用最大的海外市场，拥有约 2 亿多用户。在最近一个季度，TikTok 和其他 58 个被禁的应用程序的下载量

总计约 3.3 亿次。

据研究公司 Counterpoint 估计，这项禁令将影响印度约三分之一的智能手机用户。

上周四，谷歌和苹果开始遵守新德里的命令，阻止印度用户访问被禁的应用程序。此外，印度电信部门下令电信网络和 isp 立即屏蔽这 59 个应用程序。

TikTok 的首席执行官凯文·迈耶(Kevin Mayer)上周三表示，TikTok 符合印度的隐私和安全要求，他期待着与印度政府的各种利益相关者进行讨论。

5. 绝地求生全球营收翻番超 30 亿美元

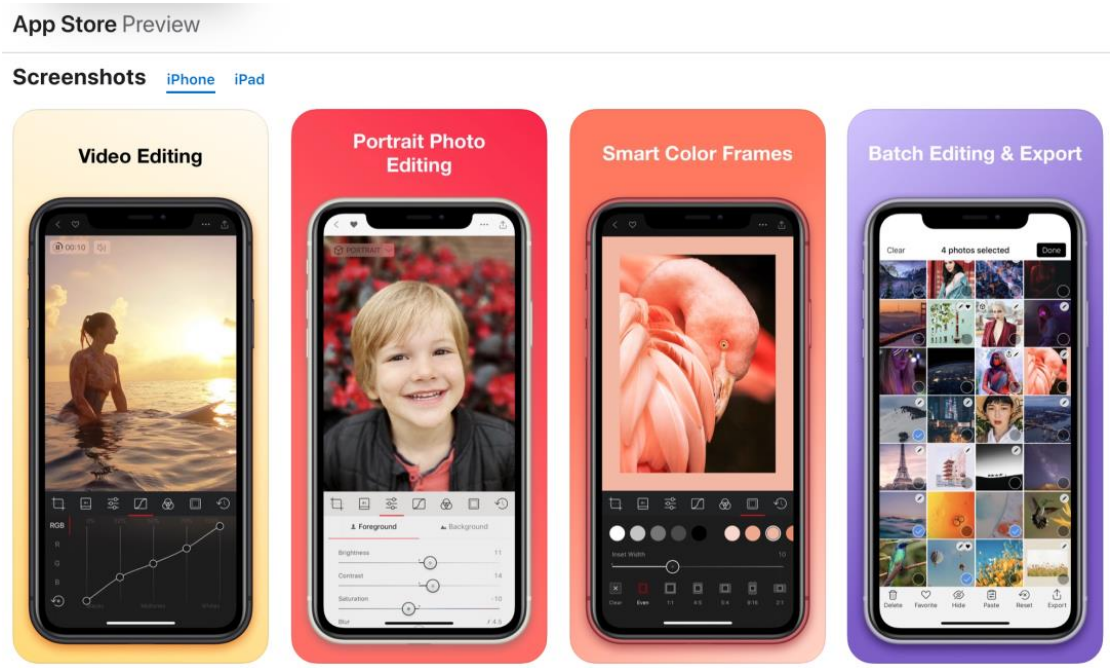
在疫情爆发之前，绝地求生就已经是手机市场上最成功的游戏之一了，同时也是消费者对跨平台游戏兴趣日益浓厚的一个例子。

根据 Sensor Tower 的一份报告，这款由腾讯支持的游戏在 7 个多月的时间里，全球营收翻了一番，达到了 30 亿美元以上，其中包括来自《和平游戏》(game For Peace)、该游戏的中国本土化以及 PUBG Mobile 的全球营收。

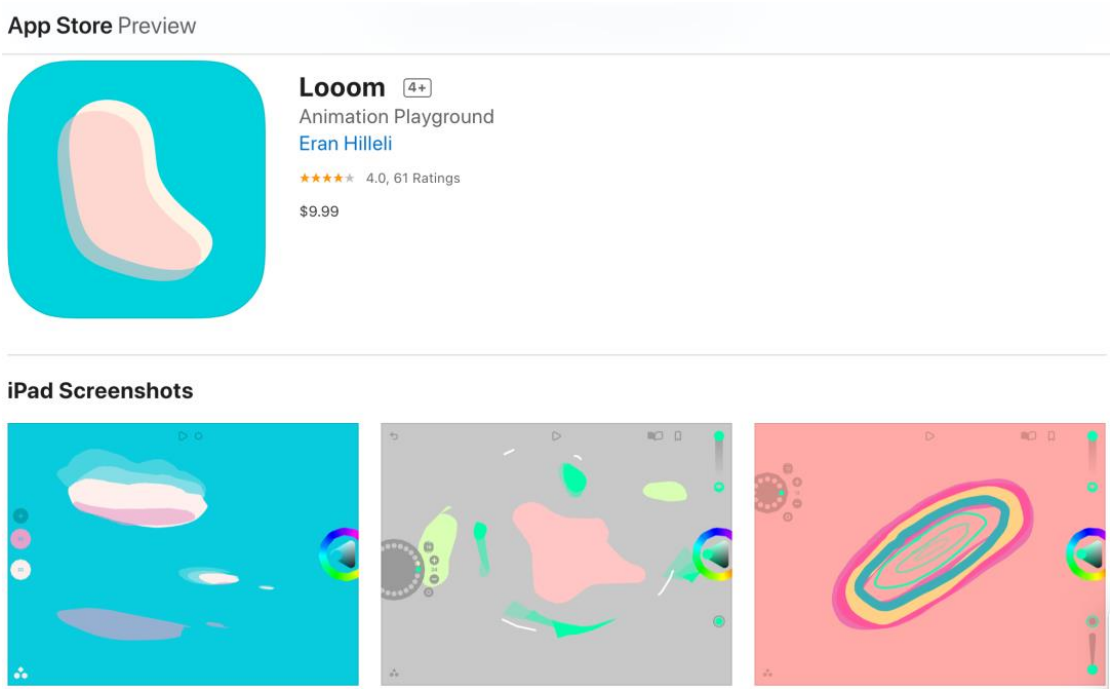
业内人士认为，疫情的冲击不会在隔离封锁期结束之后消失。相反，它将推动该行业——无论是游戏、流媒体、电子商务还是其他行业——比预期更快地达到更高的里程碑。6. 苹果设计大奖

2020 苹果设计大奖在上周宣布（往常则是在 WWDC 期间宣布）。获奖的应用程序应用到了 Apple Pencil、spatial audio、CoreML 和 Metal 2 等技术，包括：

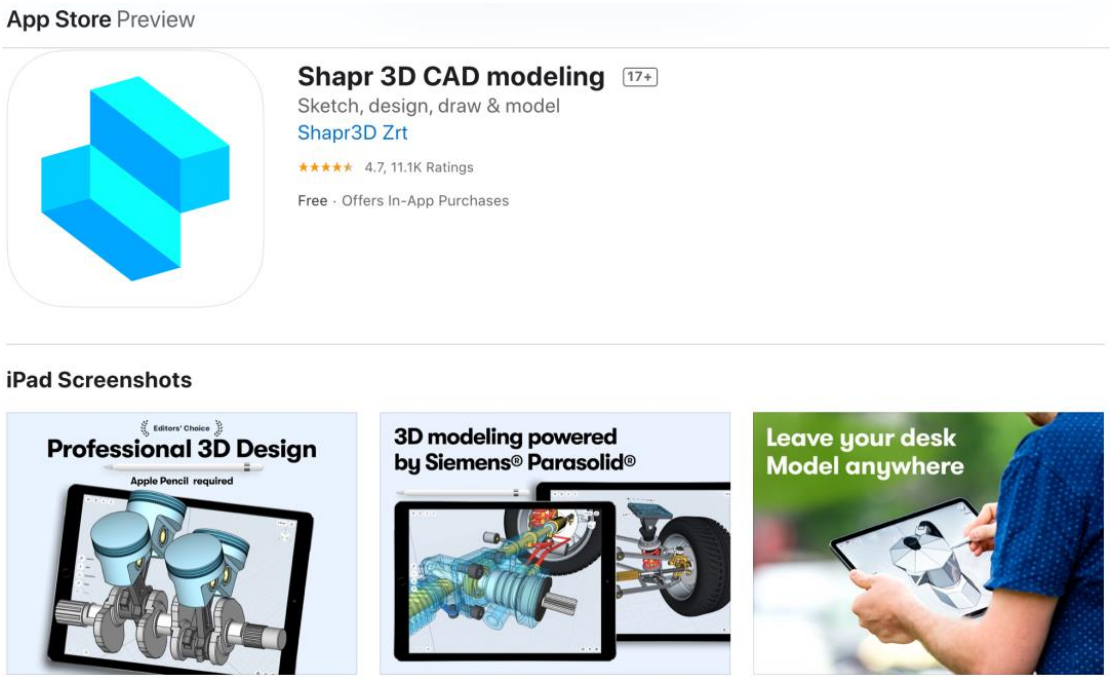
照片和视频编辑程序：Darkroom



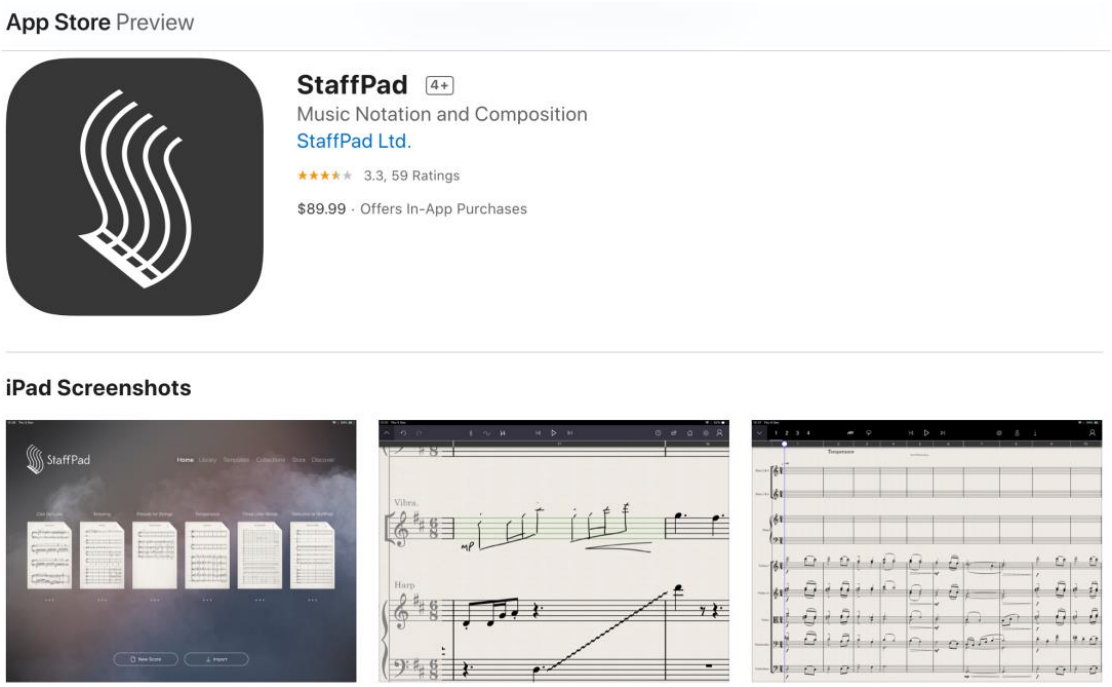
动画应用：Looom



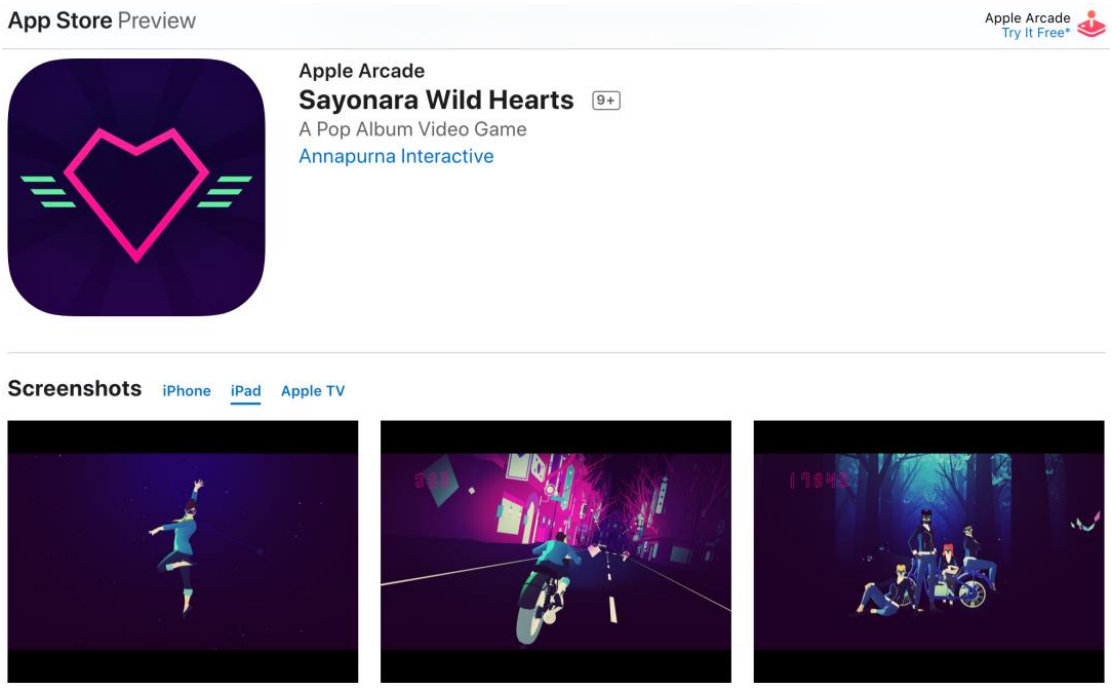
CAD 编辑：Shapr3D



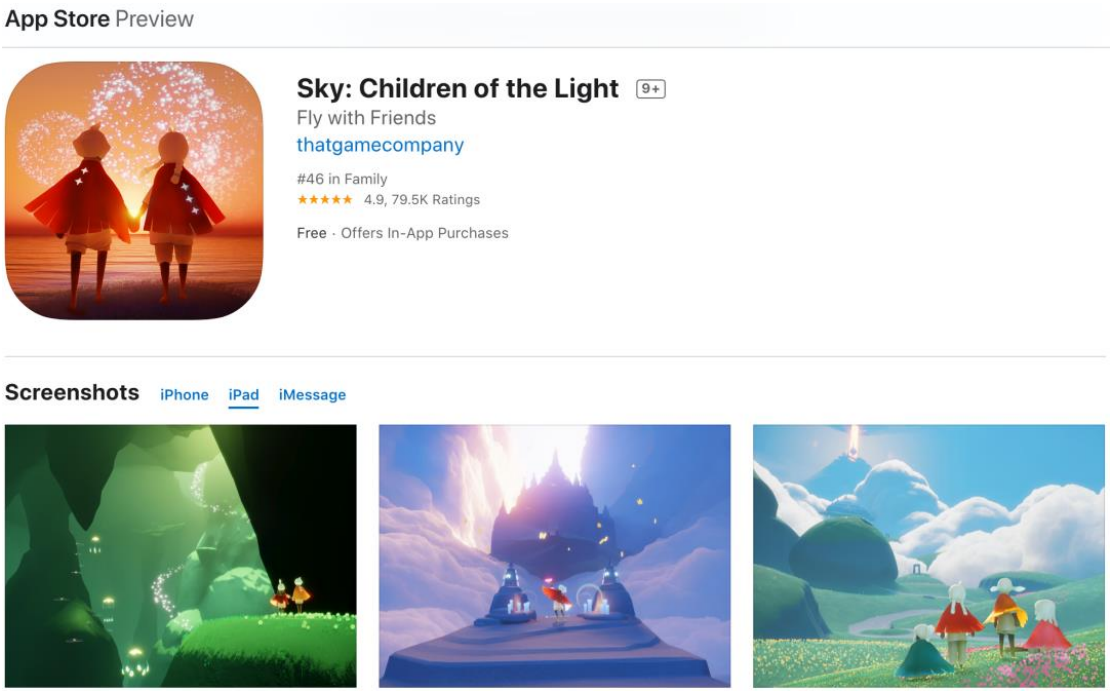
乐谱应用程序：StaffPad



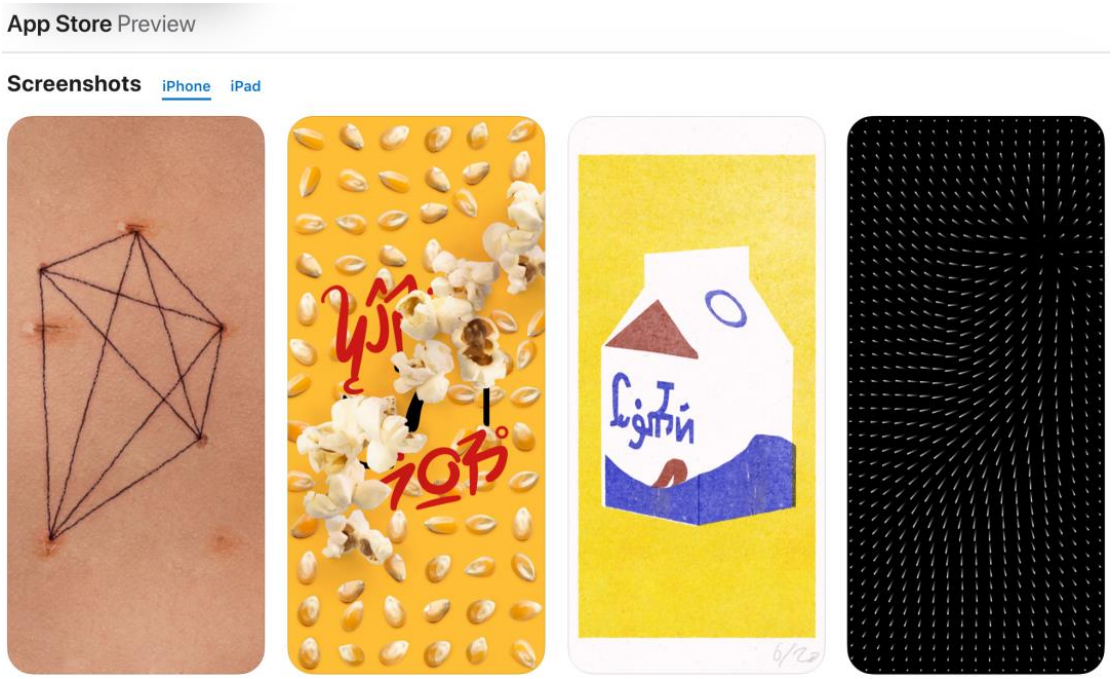
手机游戏：Sayonara Wild Hearts



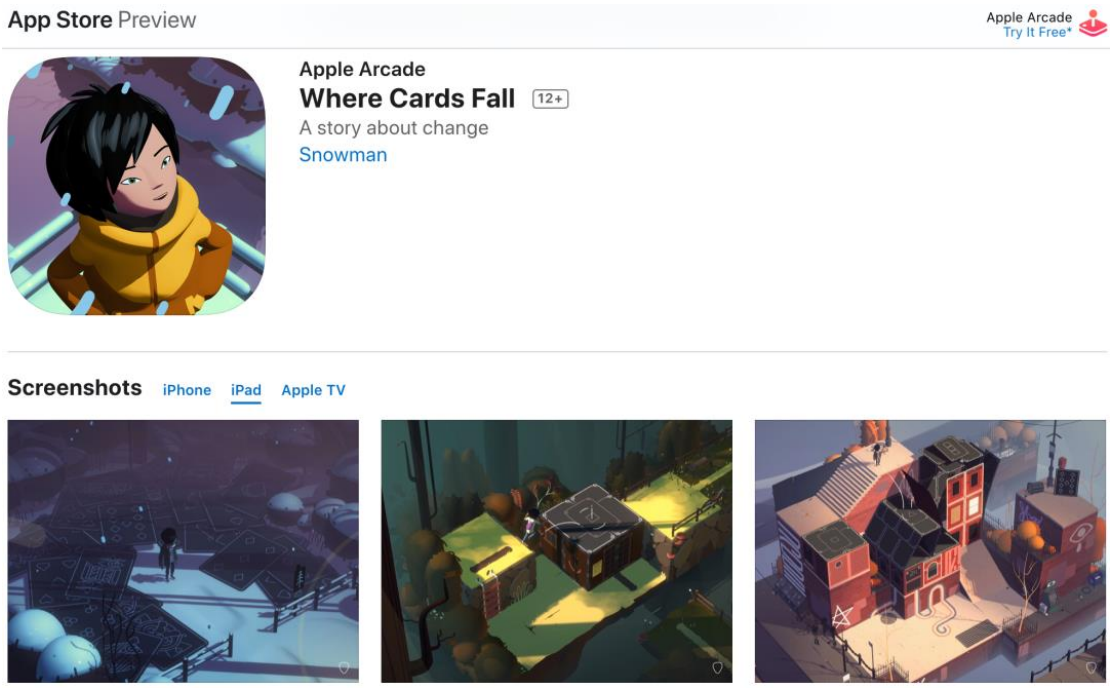
社交游戏：《天空:光明之子》



益智游戏：Song of Bloom



手机冒险游戏：Where Cards Fall



Zoom，what's the next?　　当我们在讨论视频产品时，我们在讨论什么？

Dropbox 的创立者之一——Drew Houston——告诉了人们他对 Dropbox 的愿景，但人们的回应是："这样的产品已经有成百上千个了"，Houston 的回答是："没错，但你会用哪一个呢?"

另外一个例子是 Skype。就像视频一样，VOIP 已经存在很长一段时间了，但是 Skype 解决了很多在工程和用户体验方面的问题，并最终使 VOIP 成为一种消费类型的产品。

Zoom 和以上两者所做的一样——视频通话并不是什么新玩意儿，但 Zoom 解决了许多视频会议中的小问题，从而成为大量用户的不二之选。

视频本身是一种商品，问题是服务商将如何包装它。至少在目前看来，所有的东西都将会有视频，就像所有东西都有声音一样。这就是为什么越来越多的人开始创建自己的 YouTube、哔哩哔哩、抖音、快手账号，视频账号或许将具备更多微信、Facebook、Twitter 一样的功能属性，成为人们日常生活不可或缺的一部分。而且将会有大量的垂直行业和技术堆栈的扩散。

像往常一样，更多捆绑视频的产品即将出现：Everything is video。

视频通话则是其中一个重要的部分。现在，你不一定需要一个帐户来加入一个视频会议，你甚至不需要一个应用程序——你只需要点击日历/邮件/消息中的一个链接，会议就会在浏览器/小程序中打开。

此前，Skype 既需要一个账户，又需要一个应用程序，所以才有了网络效应；WhatsApp 使用电话号码系统作为一个地址，因此可以作为用户手机联系人列表的载体；但一个群视频调用的是一个 URL 和一个日历/邮件邀请，它没有自己的社交圈层。

具体的例子是 Google Meet 仅调用了 URL，而苹果的 FaceTime 只有人没有 URL。

更进一步说，Zoom 消除的很大一部分"摩擦"是：你不需要账号、应用程序或社交媒体来使用它。相反，Zoom 使网络效应变得无关紧要并通过消除防御性来保持增长。

在视频会议之前，人们在"照片分享"上也做出了难以计数的尝试和努力：Facebook，Instagram，Snap。他们都试图超越照片，在用户体验和心理学上做足了文章。

当 Snap 推出时，已经有了无数种分享图片的方式，但 Snap 问了一大堆以前没有人真正问过的奇怪问题：为什么你必须按下相机按钮、为什么相机不能在应用程序中打开?从根本上说，Snap 的问题是："你到底为什么要发送照片?"、"潜在的社会目的是什么?"

因为最后的最后，你并不是真的在给某人发送一堆像素，而是在交流。

这是 Zoom 和它所有的竞争对手都没有真正问过的问题。Zoom 一直在问为什么很难进入会议，但并没有真正地询问你为什么要在第一时间进入会议。

为什么我们一定要在视频会议上要看到一群同事的脸部缩略图?这就是此刻的目的吗?“静音”按钮究竟是做什么用的？屏幕共享是为了什么.....这些看似简单的问题需要更为深刻却也更为基础的答案。

Decode the Week 选取新鲜有趣的音视频（技术/非技术）新闻与大家分享——也欢迎你通过后台留言或邮件（contribute@livevideostack.com）与我们分（爆）享（料）圈内趣闻。

专题内容规划

现有技术能否又如何支撑视频会议平台需求激增的突发状况？而这些平台是特殊时期的特殊解决方案，还是将成为企业长远发展的必然选择？它是否值得我们更多的期待？

专访腾讯多媒体实验室商世东：咬咬牙也要跟上新版本的发布

商世东　LiveVideoStack 专题内容　2020-04-23

离开杜比从来都不是一个困难的决定

刚刚毕业的时候，我在摩托罗拉做音频方面的工作，一做就是将近十年的时间。2010 年，我去了杜比实验室，算起来是国内入职杜比研发团队的第二个员工。当时负责整个杜比中国的工程团队，差不多有五六十个人的规模。

后来我又去了杜比澳大利亚，再后来，也就是 2019 年 1 月，我加入了腾讯多媒体实验室，在

音频技术中心负责整个音频团队的建设、产品的技术路线规划、技术的交互以及负责对多个产品，包括腾讯会议在内的端到端的音频体验，现在团队（音频）大概有 20 个人。

对我来说，首先，在杜比这样的业界领先公司工作，我学习到了很多，国际化的视野、前瞻性的技术布局、完整的思维体系、团队建设等等，这些都是让我终身获益的东西。

但离开杜比并不是一个非常困难的决定，这其中大环境、小环境的原因都有一些。我后来回顾了一下，最重要的原因还是在于杜比的主要业务是在传统领域，成长速度不尽人意。作为一家主要做技术授权业务的公司，它没有最终面对消费者的产品，相对而言公司规模也比较小，对整个生态的影响有限，很难有一个全面的把握。

换一种说法，这其中有很多的东西是个人想要但是平台无法给与的，个人成长容易碰到天花板，想要向上突破的时候，公司已经不能给自己提供这样的平台了。就这方面而言的话，在腾讯，天花板几乎不存在，在这里，只要有能力、有潜力、有愿望，个人拥有持续不断的提升空间，并且可以在这个无限上升的空间里做到最好，这对我来说是非常重要的一点。

在腾讯面临的用户数量，以前在杜比的时候是想都不敢想的。现在一天会有千万级 DAU 的产品，可以看到自己开发的产品在整个市场上遇到各种意想不到的困难、挑战、问题以及机会.

现在我们很大一部分的精力放在腾讯会议上，通过种种技术对用户使用场景进行分析并通过统计数据的上报、分析，了解用户可能遇到的问题并快速加以解决。

需要更多的思考与更确定的答案

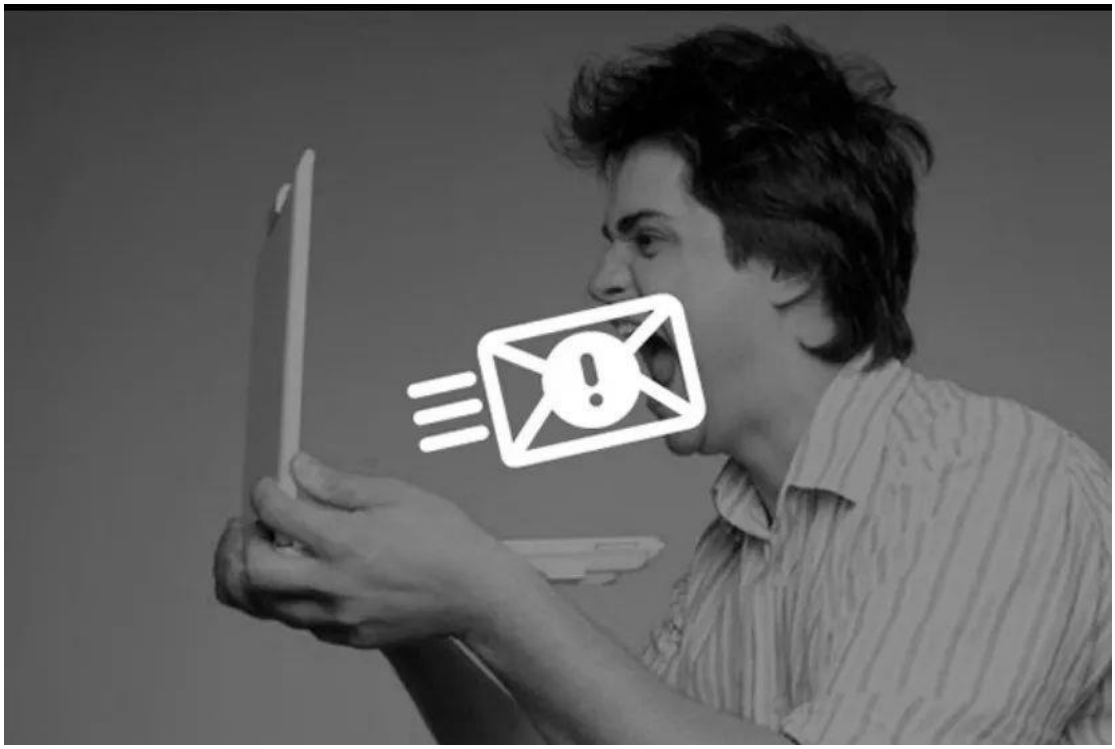


Photo by Sergey

以前我们做视频会议，考虑会议中的语音质量提升和评估的出发点都是要求对标或者超越电话（典型的比如电话话筒终端），这个是典型的双人通话场景，但是现在我们也发现视频会议大量的场景都是围绕多人的、多地的联合办公场景。以前基于电信行业终端（电话话筒为主要语音设备）定义的完整的语音质量体系需要重新审视和演进来满足现在用户的需求。

简单地说，会议软件是从模仿打电话的体验开始的，虽然从使用方式、使用场景、使用时间等等角度来看，多样性已经远远超过了电话。而且，仅仅俩人打电话是不够的，我们要一个小组打电话，光打电话还不够，还要分享视频、分享屏幕、图片和文档。从国际标准，比如 ITU-T 的演进路线图来看，制订的通话标准，也不断的从双人通话向多人、跨地区的实时、强交互场景演进。

会议当中关键的语音体验简单的来说，可以从三个维度来衡量，语音通信的可懂度，语音自然度/保真度和语音舒适度。从这三方面来说，在多人会议中面临的挑战明显要多于双人。

比如多人通话采集设备的多样性。两个人的时候当然很好协调，人多了以后采集设备如何配置，以及采集的音质、采样率、比特率；在多人场景里面音频编解码的接入配置，到底是宽带还是超宽带，最简单的办法就是所有的配置都按照最好的来，但是这样对系统资源的占用是最高的。同时因为对系统资源的占用最高，会带来总体体验的下降。

那么如何动态来在多种因素中平衡，保证多人语音通信的综合体验在三个维度上有最优的均衡，就变成一个很有挑战性的课题。

同时，不同场景的网络环境也不一样。你的网络、我的网络，可能还有第三个、第四个人的网络，有的人使用有线网络、有的则是 WiFi，有的是 4G，带宽差异巨大，网络 QoS 也参差不

齐，这时候如何有效的探测网络实际带宽，进而高效的进行流控管理，在多人场景下的复杂度远远高于双人。

所以，综上所述，一旦开始考虑多人参会的因素，你会发现，在两人通话的情形下讨论的音质体验、QoS、QoE 等等问题，现在要重新思考答案。这在国际上当前也是很热的领域，大家也都在探索答案。

对于整个视频会议体验来说，或者对于用户体验来说，这不是一个单一的因素或者指标就可以决定的。之前我们有针对网络规划的常见 QoS 三个指标，丢包，抖动和延迟，这些东西都是必要条件，但是不是充分条件。决定一个用户体验的是很多方面的因素，比如从音频角度来说，从我们做音频的角度来说，用的最多的就是上面说到的三个维度：语音可懂度、自然度、舒适度。

首先是听得懂你说的话，这是可懂度。如果音量很低，卡顿很严重，或者回声，或者很嘈杂，或者啸叫发生，那我基本就无法听了，那视频会议通话中反应比较多的卡顿来说，我们通过深度学习技术来解决网络丢包的问题。
IP 网络上面传输的时候常常会发生丢包的现象，以前处理丢包的技术比较简单粗暴，比如直接拿后面一帧填补一下或者前一帧内容直接复制。但是现在我们通过机器学习的方法，可以很准确的预测丢掉的那个音节的部分内容，在用户体验上则并没有感知。

但是光听得懂肯定不行，你肯定不可能只满足于听得懂。

比如听机器人说话，那听得懂是没有问题的，但自然度方面就不会太好，这里面就涉及很多音质的东西，不仅是延迟、卡顿，抖动，这些是网络的指标。自然度对于用户来说更多是一个感知问题，我们用音色是否明亮、是否均衡、是否丰满，等等多项指标来衡量。当我们在语音处理中不可避免需要非线性处理的时候，我们就可以通过上述的指标来保证语音的感知自然度。

在语音舒适度维度方面，我们的终极目标是视频会议语音通话的体验要尽可能的接近真实场景下的人和人间距 1 米时的通话体验，要达到这样的目标，我们还有很长的一段路要走。

目前可以想象到哪些困难我们需要克服，除了网络 QoS 三要素外，如何尽可能真实的仿真出所处的声学场景和对话形式，对于目标的达成至关重要，我们的技术演进路线图也是朝着这个方向在努力。

咖啡与信念缺一不可

Photo by Sergey

今年的需求（疫情）爆发了以后，我们整个团队一直加班加点，针对以前很多没有覆盖到的场景、没有碰到过的用户需求，陆陆续续开发了更多的技术和功能。

比如，我们发现很多老师给学生上课时需要声音共享，而且这是一个非常刚性的需求。以前我们在办公室很少有声音共享，屏幕共享就可以了，所以我们在 Windows（大部分老师都是用电脑进行教学直播互动）上提供了很多音频共享的功能。

比如我们可以在学生把麦克风禁麦后还讲话的时候提醒学生，我们通过信号处理的技术，可以把学生讲话的声音有效的从环境背景声分离处理，从而实现关麦讲话提醒这个功能。

另外，用户激增后，我们也看到了不同用户的终端设备差异性导致的产品行为的不一致。举个例子，用户开关麦克风的过程当中需要整个设备驱动进行重启。设备驱动重启的过程中，大部分设备都没有问题，但是某些设备确会导致关麦的时候扬声器无声，一秒两秒的无声用传统的质量评估手段是看不出来的，但对用户体验有很大的影响。所有这样的问题被发现后，快速迭代的能力就非常关键。

在这个过程中，对产品的思考更多时候是一个信念问题，当有这么多人使用这款产品时，我们是带有使命感在工作的。我们更多是希望早一点把需求解决、把功能做出来，早一点优化用户的体验。

所以大家常常是咬着牙，加班加点，周末也不休息。在声音共享这个功能上线的时候，我记得我们有一个同学一晚上调 bug 调到了两三点，我们当时想到底还要不要跟着这次产品更新一起发布新功能，还是说再挺一下，坚持把这最后一个问题解决掉。

因为产品的发布时间已经确定，如果这次赶不上，那意味着老师和学生们第二天上课时又没有声音，上课的效率和体验都会受到比很大的影响， 所以我们的同学那天就赶了一整个晚上，在这期间咖啡就没停过。

最后，终于在凌晨 5 点多钟的时候，问题解决了，最后还做了全面的自测，确保没有问题，第二天我们真的就硬是赶上了新版本发布的这趟车。

这个声音共享功能发布了以后，我们非常非常开心，就是那种很单纯的喜悦，希望上网课的老师们都赶紧升级到新版本，这样他们再给学生分享视频的时候，声音的分享就再也不成问题了。

声音是传递信息非常重要的媒介，所以我们的想法很简单，就是这个东西能够越早做出来越好。

VooV Meeting 的师、敌、友

Photo by Sergey

最近，Zoom 是一个非常值得关注的对象。Zoom 对我们来说，是云视频会议的先行者。因为它们在这个行业进入比较早，做得比较久，对用户需求、行为的理解也比较深刻，有很多值得我们学习的地方。

其实我也蛮喜欢微软 Teams 的产品，它也有一些自己独到的地方。它从 Skype 时代开始演

进，有比较多的积累。而且微软对底层技术有长期的投入，很多先进的思想一直在引领这个行业的进步。

因为我们做视频会议的时间相对比较短，有这样一些先行者给我们先进的思路，包括用户的支持、合作伙伴、硬件合作伙伴合作模式等等，都是值得借鉴的。

腾讯会议海外版本 VooV Meeting 也同样是我们在支持，毕竟底层技术都是一样的。

从音频角度来说，海外用户跟国内用户的诉求大体上一样，但也有一些不一样的地方。比如国内开会的时候，整个网络架构都是比较均匀的，尤其是我们看到北上广这些城市，他们本身的网络条件是不错的，延迟也处于比较好的状况。

到了海外，我们的服务器部署不像在国内这么完善，我们需要知道哪些地方的服务器部署是不足的。海外也不需要像国内这样面面俱到的部署，很多地域人口密度很低，比如美国的用户可能比较集中在东西海岸，那么就需要针对东西海岸加强部署，在后台的调度也要加以注意。

国内的话，我们的信息获得比较全面，流量高发的城市地区我们都有掌握，但海外这些需求和信息要如何获得，这就是一个问题。你怎么知道到底哪些城市在哪些时候会有一个峰值的需求？这必须要有一个对全网质量的评估和监控，这样才能近乎实时地知道用户的需求出现在什么地方，从而可以有针对性的、非常灵活快捷地进行相应的部署。

从具体使用技术的角度来说，国内外的用户使用视频会议的习惯也有些区别，国外可能习惯用 PSTN 电话的接入方式，国内则是 IP 来接入。国外很多用户倾向于使用固定电话或者移动电话接入体验，因为传统电话网络 PSTN 跟我们的 IP 网络是属于不同的运营商网络，他们的运营商网络到我们的网络里面要经过很多网关、路由，针对这种情况怎么优化体验是我们要加以考虑的，这方面的开发技术比如说频谱扩展，比如 PSTN 专门的降噪，诸如此类的需求可以有针对性的加以开发。

1+1>2

Photo by Sergey 国内有非常不错的硬件制造商，深圳就有很多了。我们和这些硬件制造商合作的时候，怎么互相之间博采众长，产生一加一大于二的效果，这里面其实有很多合作的空间。

腾讯会议在很多场景里面是要跟不同的合作伙伴联合打造一个比较好的会议体验的，而这些合作伙伴在他们的相关领域里面，像硬件领域、网络领域的未来发展是怎样的，有什么有前景的趋势，哪些变化在酝酿当中、又会什么时候发生，都是我们要持续关注的问题。

比如，我们看到麦克风不断有新技术材料出来，那相关制造商的路线规划是什么，某些技术难点突破了以后会对视频会议云通信的产品有怎样的影响，这些都需要你拥有很好的前瞻性。

另外，我们一直比较关注相关的领域。比如，5G 时代整个接入网络带宽上去了，那我们就思考当接入带宽不再成为瓶颈以后，可以在哪些用户体验上做进一步的优化。

像之前提到的教学场景，老师们可能有些高品质的音频需要传输，不仅仅是语音，还有基于音乐的特定场景。比如老师给学生放一段视频，视频里面可能不仅仅有对话，还会有高品质的音乐。那么对于音乐来讲，面临的挑战是不一样的。音乐的内容特征、频谱、码率需求，从音质体验的角度来说比语音要求要高。

这样的诉求在 3G、4G 网络视频会议下由于带宽的限制、网络的拥塞可能不得不放弃，但在 5G 的时候就会提上日程，要研究如何在新的场景下满足用户更多的需求，从而去优化他们的体验。

另外，基于 AI 的技术，我们也看到音频视频两者多模态可以更好的结合起来，能够大大往前走

一步，给用户更好的综合体验。同时，计算能力上来以后，你会发现很多事情做起来受限于当时的设备，而现在慢慢都成为了可能，我们在思考怎么把这些事情一步一步做起来。

我们现在和硬件厂商的很多合作，已经在紧锣密鼓地开展了。腾讯会议本身可以提供很好的音视频处理、传输、编解码技术，我们的硬件合作伙伴也有一些他们专有的硬件上面的特色功能。

第一步，我们之间要先相互打通。直接打通第一步以后，我们要思考怎么把自己的技术和对方的优势结合，这里面有很多工作可以做。

举个例子，用户的设备跟我们腾讯会议软件，不管是通过有线网络还是 USB 进行连接，以前在用户设备上受限于运算能力做不到的事情，现在就可以做到了。因为通常我们讲的会议电话这样的产品，它的运算能力、内存总是受限的，很多基于 AI 的算法做不到。

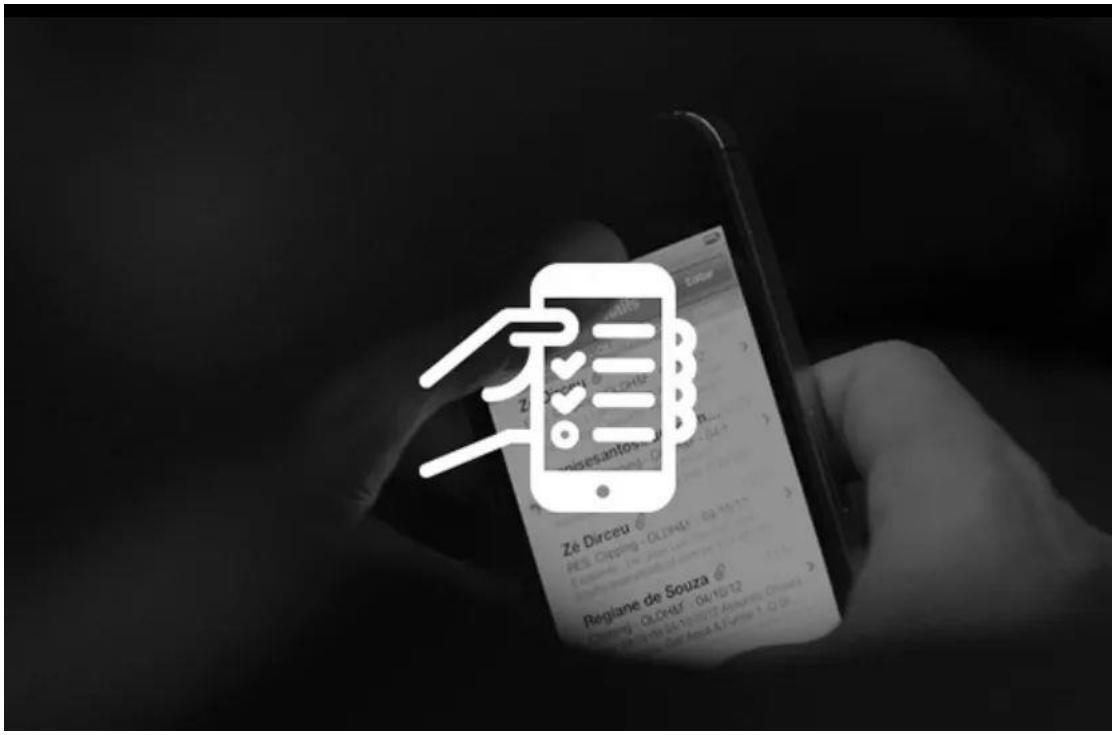
但是腾讯会议的 APP 跑在我们通用的、运算能力强大的设备里面（像电脑）就不存在这样的限制，尤其是和云结合起来后。一旦和硬件厂商打通以后，双方找到彼此之间的配合点，以前做不到的事情现在都成为可能，典型的例子比如会议内容转录、整理会议纪要等等。

像一些传统的视频会议方案提供商，它本身是一个闭环，它的解决方案里，软件硬件全都是它自己的，不存在运营多样化的挑战问题，因为每个链条它都自己把握，这样质量有保证。但是相对而言，他们的完整解决方案的价位也比较高，兼容性也会是很大的问题。

腾讯会议的话，要跟所有设备商都可以接入，面临着保证可靠性、稳定性、安全性的挑战，因为不同的厂商的产品要在它的性能范围之内互相协同，无缝连接。

在安全性方面，我不是专家，但从开发 SDK 的角度来说，我们对于用户本身的信息肯定是不做收集的。对于用户谈话，比如交流的具体内容，我们也不会做任何收集，只是用算法对这些内容的统计信息进行收集，包括通话过程当中的质量，比如是否失真、嘈杂、有卡顿等等，这些都是从技术方面统计的信息。
因为我们需要对这些技术指标进行评估，比如我发现你那边频繁出现卡顿现象，那么我希望能够知道你在什么时候、为什么发生了卡顿，如果发生卡顿的原因是技术网络架构，或者是应用本身消耗了太多的 CPU 导致的，我们就可以更有针对性地进行优化，从而给予用户更好的体验。

Everything matters



接下来我们要解决的问题，更多的会是更具挑战性的多方会议场景下面的问题。因为现在的用户场景非常多，多样化场景下，不同的场景里面用户对声音的需求是不一样的。

这就说回到用户场景的问题，当下的场景相对以前确实有很大的变化。以前腾讯会议的用户场景没有（也没有想到）现在这么多。现在你可以在家里面、在办公室里面、小型会议室里面、大型会议室里面开会，甚至各种直播场景都是以前没有想过的。在各种场景里面，你的架构、相应的技术应该如何规划、配合，在不同的场景里侧重点又是什么，不同的人都有不一样的理解。

我们这方面做得比较细，要根据用户不同的场景去考量。比如开信息传达的大会的场景，对于这种绝大部分时间都是信息单向传输的情况，延迟可能就不是那么重要了，重要的是声音的可懂度和自然度，要保证用户在长时间倾听的条件下，避免听觉疲劳和注意力分散。

而在小规模小组讨论的场景下，大家都是是熟人，交互非常激烈，你一句我一句，这时候对实时交互的需求会非常强，需要很低的延时，除了对于可懂度和语音自然度的正常要求外，对于声音的舒适度要求尽可能模仿面对面的沟通，要求很高。我们会考虑在这样的场景下，需要哪些新技术，如何在多种限定条件下权衡，来达到一个最好的综合体验和容忍度。

另外就是基于视频会议的主场景，还有哪些衍生的需求，比如会议内容转录、同声传译，以及和智能音箱的结合、如何通过语音助手来唤醒和交互……这些以前在传统的视频会议产品中见不到的功能，现在随着相关市场产品、技术的成熟，都可以进一步融合起来，在协同办公这样的一个大背景下，共同来提升效率。

诸如此类,在当下和未来，我们都会碰到不一样的挑战，你要问哪个最重要，我没有办法回答，因为用户 A 和用户 B 的需求对我来说同样重要。同时，我们也会用数据来说话，数据告诉我们用户的需求在哪里，我们就去开发哪里；如果目前还没能满足这部分的需求，那我们就把它丢到

map 里，以后的每一天都要想办法去满足它。

通过将游戏逻辑放在远程服务器上并将图像/音频流传输到客户端，它可以最大化后端控制并最小化前端工作。由此，服务器将负责处理繁重的任务，而客户端将不再受硬件限制。

云游戏之真心话：一切才刚刚开始 | 对话南京大学副教授马展

马展　LiveVideoStack 专题内容　2020-05-14

尽管互联网巨头们纷纷加入云游戏战局，但也各有各的烦恼，腾讯的问题在于云计算的发展，阿里则在于如何吸引优秀的游戏 IP，扩充内容库。

当平台和硬件不再是壁垒时，各个游戏服务的独特卖点将在竞争中占据优先地位，拥有 IP 和达成独家协议也将比以往任何时候都更为重要。

Google 为此还专门成立了 Stadia Games and Entertainment 工作室，专注于开发为用户提供独特云游戏体验的服务。

那么，“内容为王”在国内同样适用吗？Industrial Securities 的数据显示，通过 video game streaming，全球每个用户每小时平均花费 5 元（人民币），而中国游戏玩家平均每小时仅愿意花费 2.2 元（人民币）。

我们的问题在于，玩家们（甚至是所有互联网用户）并没有意识到：好的内容是需要付费的。

在此前的报道中，腾讯云高管也曾表示，腾讯将更多地依靠游戏内广告以及向游戏设计方出售数据等其他收入来源，而不会采用国外的订阅制模式来获利。腾讯当然可以选择这样一条发展路径，那么那些拥有有限 IP 的游戏制造商和小公司呢？

以下内容由 LiveVideoStack 与马展的采访整理而成

下一个视频十年会是什么样？

我从 2003 年开始做视频，当时从参与国内视频编码 AVS 标准化开始学习压缩，后续拓展至图像视频处理、采集，还有传输呈现等等。2003 年我还在读本科，参与了国内 AVS 第 1 代、第 2 代和第 3 代的一些技术工作。目前我主要是还是围绕视频压缩进行研究。

现在国内有很多面向标准，如 AVS、HEVC，AV1 研发的大团队，但我和他们做的有些差异，

我一直在探索一种新的、基于人类生物视觉的全新视频编码方式。据我所知，清华、北大、科大也有这方面的工作小组，也做的非常好。

目前像 HEVC 这种通用的视频压缩，估计有 1000 倍左右的压缩率，但我希望能够做到 100 万倍的压缩率，因为我们生物视觉里面的压缩率是在 1000 万倍这个量级。这个量级其实是在上世纪五、六十年代左右就已经在脑视觉研究中提出来了，只不过一直都没有实质性的突破。

视频技术是有明显的阶段支撑的。大概 30 年前左右，视频的分发的都是通过 DVD 或者 VCD，最典型的就是美国的 red box 和 blockbuster。如果你想看电影，要么去电影院，要么就是去超市租一个 DVD，然后带回家看。

再然后，应该是 2000 年前左右，互联网普及。我们开始用 FTP 下载电影，这个叫渐进式下载，现在可能已经没有了，BT 也是在后来才出现的。基本上我们每天上完晚自习回到寝室，就把电脑开着，下载一晚上的视频，然后第 2 天再去看下好的电影，其实挺好玩的。我记得当时比较火的一些大片，都是用这种法子下载下来看的，比如说《我的野蛮女友》之类的。

Photo from Movie 《我的野蛮女友》

2010 年左右，Netfix 出现。刚开始的时候大家都觉得 Netflix 不可能，觉得用户不可能专门通过网络点播去看所有的电影电视。但现在，Netflix 及类似的网络视频直播点播服务基本上一统天下了。

于是我们可以明显的看到，分享和传输的通道从原来的线下转移到了线上。线上则是从非实时制到现在的实时。Netflix 的实时并不是纯粹的实时，它是通过了 CDN 缓存的方式，让用户就近体验到最高质量的视频服务。

但 Netflix 的成功造就了互联网一大批数百亿美金级的 CDN 公司的崛起，像 Akamai，以及国内许多做 CDN 的企业。

在我的学生准备博士课题的时候，我就和他们讨论，一定要去思考下一个 10 年视频技术是什么样的。

2013、2014 年左右，我还在美国三星研究院，当时我们就在想下一代的视频会是怎么样。如果依然按照传统的那条路走下去，很可能会遇到瓶颈。那个时候三星就已经在研发一些 AR、VR 甚至 holographics 相关的虚拟现实产品。虚拟现实其实也就是要实现人与视频的互动。

下一个视频十年（你可能也看到了这个趋势，毕竟 live streaming 的核心该做的也都做完了），视频肯定不仅仅只是一个消费类的工具，它可能成为消费类的内容，也可能会成为一种与人交互的形式（互动视频）。到时我们不仅仅要考虑视频的质量，还要考虑互动情况下互联网的延迟（也因为云越来越火爆）。

解决问题的最终途径

在云游戏起来之前，我们就已经开始研发互动视频的相关技术了。以前仅仅是着眼于解决带宽和质量的问题，现在我们则希望能够解决带宽、质量和延迟这三个因素之间的关系问题，以及在之后怎样去做一个联合的调制。对此，我们提出了三个很重要的技术。

第一点就是如何选择传输视频的分辨率、帧率、码率等等。要做到让视频呈现在用户面前时，它的质量是最佳的，延迟和带宽消耗也是最低的。

现在大家都觉得 5G 可以解决一切，但也有可能又回到当年 4G 应用前夜，那个时候大家都觉得 4G 出来就什么问题都解决了，到最后发现，有些东西反而越用越慢，微信视频也常常是“当前通话对方信号不佳”。

所以我们就提出，能不能打破现在所有的这些视频通讯的规则式的传输。所谓“规则式”就是

我们必须要按照一些特定的人工或者经验型的协议。所以，我们希望通过数据驱动来做这件事，因为每个人看到的视频、用到的视频，以及消费的视频都不一样。

我们通过一年多以来搜集的数据发现，用户在玩游戏或者在看视频的时候，也是有倾向性的。有不同的倾向性以后，网络的表现也是不一样的，用游戏来比喻的话，就是有的人喜欢玩英雄联盟，有的人喜欢吃鸡，这两个网络分布完全不一样。

也就是说，数据驱动有点类似于个性化推荐，比如说谷歌搜索。在谷歌，用户搜索的东西越多，它对用户的定位就越精准，这其实就是根据用户习惯来去 search 的。

即使我们都在看爱奇艺，可能我们看的内容也不一样；我们都刷抖音，我们刷的内容也不一样。这就要联系到第二点，就是要考虑到每个人不同的行为习惯及网络习惯（有的人喜欢用 WiFi，有的人喜欢用 LTE，还有的人喜欢用固网），以及在不同的网络接入状态下，要如何保证最佳的用户体验（因为当用户拿着手机进行物理移动时，网络带宽都会有变化）。

现在我们研究一个新的功能，叫做“Networked Video Learning for individuals”。我们认为每个人的网络视频消费都是特征化的（每个人都不一样），所以我们必须要为每个人都用数据驱动的方式，选择一个最佳的策略。

同时，5G 可以提高人与基站之间的带宽没错，但是基站与基站之间都是通过的光纤（或者说是主干网），主干网的带宽如果没有增加的话，5G 就算增加了 1000 倍，对用户来说也没有任何作用，因为主干网的数据还是在基站之间跑的。

通过春节之后视频会议出现的一些状况也不难发现，当并发上来的时候，我们的互联网是没有办法承受的。比如说服务器在北京，用户在上海甚至是深圳，在这种情况下，如何在考虑用户体验最佳的同时选择一个最短路径，这个是要解决的第三个问题。

从这三个方面，我们基本上把云游戏端到端的一些核心技术都摸了一遍。

订阅制真的适合我们吗？

从谷歌的 Stadia 的模式来看，订阅制肯定是云游戏以后的主流变现方式，但是国内的情况和美国不一样，或者说是和欧美不一样，因为国内的游戏公司是一家独大的状态。

很多时候国内用户的行为会受到某家（大）公司的影响（并不能完全复刻国外的订阅制，同时存在版权问题），在这方面也要结合国内市场的实际情况来考虑。

但是从整个网络视频的发展来看的话，订阅制的趋势还是在的。刚开始爱奇艺的付费内容也比较少，现在付费内容也是越来越多，用户的接受度也越来越高，所以这个趋势是不可挡的，只不过在国内的普及可能会稍微慢一点。

所以不管是从头部企业，还是从现金流运营商方面，我们看到大家都在摩拳擦掌，希望能做点什么东西出来。现在也有了 5G 的新基建，当新基建都到位之后，你不可能还只满足于每天看看爱奇艺吧?

这时候肯定要有一些新的东西，这“新的东西”当中就包括云游戏。而且我觉得，云游戏或者是云互动可能也是最直观、最直接可以落地的“新东西”，因为它不需要对原有的系统做很大的改造。

像 Cloud VR、Cloud XR，可能还会涉及到让用户再去买一个新的设备。但用户对新设备的付费意愿能有多大呢？新设备都是侵入式的存在，大家会不会买单也是一个问题。

鱼和熊掌可以兼得

有人说手机云游戏大部分是单人的策略性游戏，这个其实不一定，从我的角度来看的话，我觉得手机是赋予了游戏一种新的连结。

所谓新的连结，就是以前大家玩一个 PC 游戏，比如说特别火爆的英雄联盟，大家可能要去用电脑，但是现在的 00 后或 90 往后一点的孩子们用电脑的频次越来越少了，大部分都是以平板为主。

但手机云游戏存在一个关于如何吸引沉浸式的电竞玩家的问题，电竞玩家又涉及到一个行业区分的问题。电竞玩家追求的就是极致的体验（他们称之为手感），比如一定要是机械键盘，鼠

标也不能无线以及一大堆在普通玩家眼里稀奇古怪的要求，这就有点像院线和 Netflix 的区别。

我可以在家用 Netflix 看电影，但也不排除我想去体验一下高档的院线。
在美国的话，一般家庭空间比较宽敞，所以他会比较愿意买 XBox 或者 ps4。但在国内，咱们说现在还是“蜗居时代“，五六十平米的空间，你买个 XBbox 放在家里，活动起来也不方便。

隐私怎样都是个问题
有关玩家的个人信息，陈浩做的胖鱼炫道平台（云游戏之大冒险：5G，等还是不等 | 专访云格致力陈浩）是将云端信息存在游戏公司。比如说你玩英雄联盟，那么你必须自己要有英雄联盟的账号。

你可以把胖鱼炫道想成是一个快速的 gateway，所以也不会做任何用户信息的存储。
但不可否认的是，有些公司还带着一点走灰色地带的想法。当然，我认为人们的版权意识只会越来越强。

云游戏的平台只是一个交通工具，你这个交通工具怎么还能把乘客的身份证给抢来呢。

虽然一切才刚刚开始
只说云游戏的话，我觉得现在没有说谁做的好谁做的不好，因为真正意义的云游戏还没开始呢。

单纯从游戏玩家的角度来说的话，那我觉得现在大家都做得都不是太好，虽然有的平台号称自己每天有多少多少，但在全球的游戏玩家里面，这个数字实在不值得一提。即使是谷歌的的 stadia，就目前来说，估计也还在烧钱，我觉得不算是很成功的范例。同样一份游戏拷贝，线下卖 100 万份，在 Stadia 可能只卖 2000 份。

Photo from Google Stadia Website

国内的问题就在于，从互联网刚开始普及的时候，几个巨头就让用户产生了一种错觉，那就是互联网的一切都是免费的，但是其实所谓“免费“的东西才是最贵的，因为很可能你所有的信息都被他们搞去了。

这就造成现在有一些游戏不得不完全免费，这就会出现很多盗版，当然他们不会说自己盗版。相反，欧美那边的版权约束和用户的隐私意识都比较强，所以那边的云游戏供应商不得不讲清楚：我们不会给你免费，但同时我也不会存储你的很多数据。

如果还是不改变现在这种免费的心态，那就不知道国内的云游戏该怎么往下发展了。
在内容方面，同样是因为云游戏还在起步阶段，所以其实云游戏的内容库里都还是以前早就已经过审的东西，没有什么新的游戏内容出现。后面是不是会有一些新的审核方式，中间如果出现了一些敏感或者全新的内容，又要怎么去把控，这个是应该去考虑的。

当下，游戏在技术方面的功能与服务基本上能够支持的都支持了，只不过质量高低的差别。比如，腾讯能做到端到端 100~120 毫秒的延迟，陈浩他们的胖鱼端到端是 30~40 毫秒。

国内的游戏市场，很多时候不是技术说了算的，大部分时候是看谁的地盘大。云游戏想要发展，是需要像腾讯或者网易这样的一些大厂去做点什么的。

就拿说腾讯来说吧，如果腾讯能够站出来说，我们愿意做些事情来支持、推动这个行业的发展，像谷歌一样，这样这个行业才会更好，才会有可能往好的方向发展下去，否则的话有可能就会变得更加乱象丛生。

Quick Question

1000 平方公里内的 1000 位玩家同时在线玩同一款云游戏，可以做到让他们彼此之间感受不到任何延迟吗？

我们有做过一个测试，400 公里以内的玩家，我们能够把他的延迟控制在（端到端）30~40 毫秒之内，在这个范围里，用户基本上不会感觉到延迟的存在。

但是 1000 公里的话就会比较麻烦，因为 1000 公里可能就跨省了，特别是在国内，国内的运营商分省级区域管辖，所以在跨省的时候它会涉及到主干网之间的切换。

另外一个比较麻烦的事情，就是国内的电信运营商（联通、移动和电信）还存在着所谓的物理隔绝，但这个问题很可能会在未来被解决掉。

在当下要实现 1000 公里以内 1000 人在线玩同一款游戏，而且每个人都感受不到延迟，我觉得基本上不太可能。其实大部分还是运营商的问题，从我们平台来说的话，像服务器或者是用户终端的延迟其实都是可控的，同时也能做到很好的优化。

现在不管是边缘计算，还是主干网高带宽的交换机，都在逐步往下推动。目前大部分还是 100 个 g 的交换，明后年的话，基本上就是 400 个 g 的交换。所以，在一两年之后要实现这样一个场景应该是不成问题的。

从开发者到决策者，你的眼界足够了吗？

码代码，到白头|专访 SRS 创始作者&阿里云 RTC 服务器团队负责人杨成立

杨成立　LiveVideoStack 人物专访　2020-05-20

Code never lies, comments sometimes do; Code never lies, people sometimes do.

人会说谎，或多或少，或有意或无意，但好的代码不会说谎。屏幕上的一行行字符给人兴奋感，给人贴心感，给人世俗意义的快乐，也给人世俗之外的领悟。

```
int main（）
/* Program starts here */
```

2009 年，我刚刚进入视频行业，从 Flash 播放器开始做起，也做过 FFmpeg 转码服务器。2012 年，ChinaCache 如日中天，要做 CDN 自研的视频服务器，我也就开始做视频服务器。

到 2013 年底，我开始做开源视频服务器 SRS。

从进入视频行业开始，我就一直很幸运，在每个公司都有被信任和委以重任。特别感谢 ChinaCache 的付亮，不仅给我从 0 到 1 做视频 CDN 的机会，还不时在管理上启蒙我；感谢观止创想的杨默涵、雷健和于冰给予我的信任，让我负责技术团队，经历过的人才知道，创业维艰，信任无间；感谢叔度引荐我进入高手如云的阿里，开启了我从直播到 WebRTC 的技术新旅程。

我最初也是看着 Nginx 的代码写的视频服务器，但做到级联时遇到了复杂状态机的问题，由于 epoll 异步回调导致服务器逻辑变得特别复杂。

当时我就去请教了 ChinaCache 的技术大神 Michael Talyansky，他告诉我状态机应该用 coroutine 解决，coroutine 不是进程也不是线程……大神看我仍然一脸懵逼，又补充道：可以认为 coroutine 就是线程，不过没有锁切换开销也没有竞争条件。

回去之后我补了很久的功课，了解了 coroutine 的详细实现，并且用 coroutine 改写了视频服务器，才发现这招果然简单有效、直达痛点。这让我认识到人外有人，天外有天，只有踏实努力，不断进步，才能体会到技术的乐趣。

/ in a galaxy far far away */*

我从 2013 年底开始做 SRS，本来只想做个编码器收流的服务器，后来随着直播需求的爆发，关注的人越来越多，SRS 的功能自然也就越加越多。

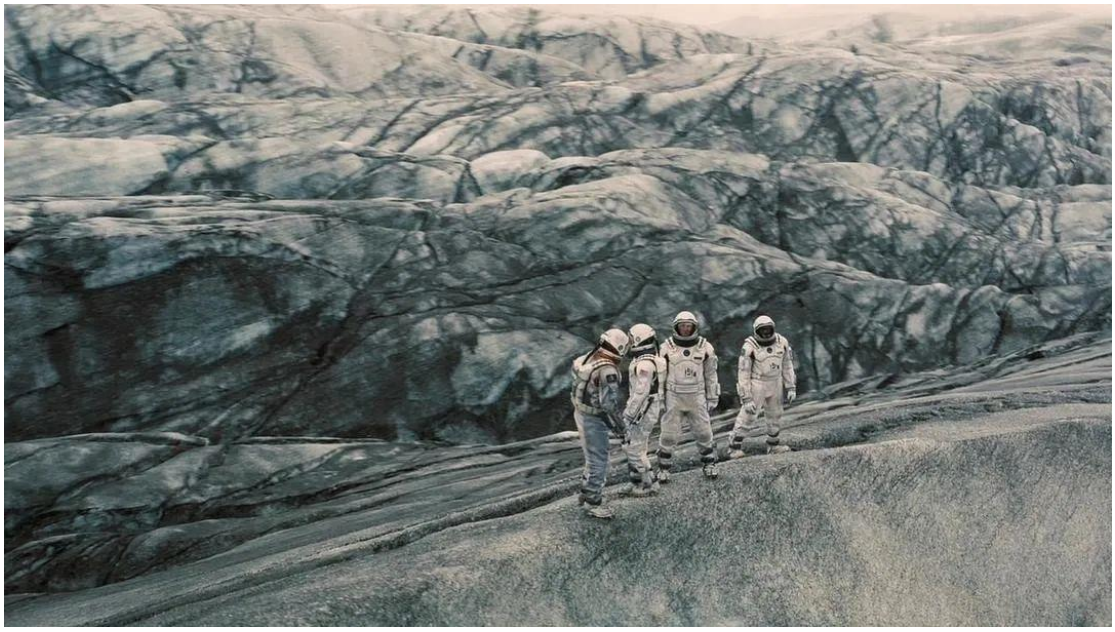
2015 年底，SRS1.0 发布。不过，关注度比较高的还是 2017 年左右发布的 SRS2.0，实际上 SRS2.0 在 2016 年就基本成型了，只是一直在完善稳定性，所以就迟迟没有发布。

2019 年底，SRS3.0 发布。这一次，SRS 基本具备了直播领域视频服务器应该有的核心功能。直播本质上是互联网的在线娱乐视频，随着互联网基础设施的完善（CDN、云计算和浏览器）而大放异彩，进而在各行各业都得到了广泛的应用，比如秀场、电商、教育、广电等等。

2020 年初，SRS4.0 有了质的飞跃，开始支持 K8S 部署和核心协议覆盖，协议支持了 SRT、GB28181 和 WebRTC。这在很大程度上，是因为互联网基础设施的进一步完善，是互联网进一步向其他领域渗透的结果。比如 SRT 是应用在广电和远距离传输领域，GB28181 是监控和智能家居领域，WebRTC 则是最近热度很高的视频会议领域。
这 7 年来，我所经历的最大的困境，是一直在想：SRS 是否应该存在？直播的发展、SRS 的架构是否已经走到了尽头？

但经过了这几年，我觉悟到，没有走到尽头的开源项目和技术，只有走到尽头的意志和想法——战斗以战斗意志丧失为终止，开源以初心不灭为重生。这才有了 SRS4.0 的诞生，以及另外 4 位新加入的 Maintainers。现在，SRS 的更新速度已经超过了 2013 年的初创期。

//
// Dear Maintainers:
//



最初的 Maintainer 只有我和文杰大神。文杰是我在 ChinaCache 的同事，我离职后，他接替我继续维护和研发 CDN 视频服务器。

文杰对技术的敏感度高，又有很大的开发热情。有一次公司 Outing 去海边玩，闲聊时，文杰就问我视频服务器有哪些技术难点（我当时在从 0 到 1 研发视频服务器）。

后来文杰去了学而思，每隔一阵儿就听他说又搞定了一个啥，想要在业内做到最好。每次见面，文杰都说要回来搞搞 SRS 了，再不搞就老了。

SRT 的 Maintainer 是施维大神，施维也是我在观止的同事。当时我创业已经有几个年头了，开始心生退意，觉得有些技术难点自己搞不动了。但施维大神来了之后，一个人解决了很多难题，不仅技术好，人也踏实。

几年后，闲聊时，他提起自己在搞 SRT，我们就开始讨论 SRS 集成 libsrt 可能会遇到的问题，因为 libsrt 开线程了。后来他解决了这个问题，把 libsrt 集成到了 SRS，而且改了 FFmpeg，支持了动态编码。因为 SRT 在 SRS 上只是一个环节，关键还是要在编码器上降低码率。

GB28181 的 Maintainer 是夏立新大神。立新关注 SRS 比较久，写了不少总结资料，画了不少图，这些被分享到群里后，当时也轰动了一把。一直以来，SRS 都是使用文档多，程序结构文档少，所以大家对于 SRS 的实现分析和总结文档还是很喜闻乐见的。

监控一直是视频行业很重要的一部分，之前和田野大神聊如何支持监控，田野说要支持 GB28181，还做了详细的分析。后来，我就在群里鼓励大家在 SRS 上支持 GB28181，立新就默默的把代码给码出来了。

其实立新现在并没有在做监控，而是转做 WebRTC 了，不过他之前做监控比较久，技术扎实而且热爱技术，尤其代码码得好。

WebRTC 的 Maintainer 有三位：肖志宏大神、B 神和刘连响大神。服务器主要是志宏在做，B 神是 Opus 转 AAC，连响是 Native 客户端和信令协议。

志宏我很早就认识了，当时有人要在 ARM 上跑 SRS，需要写汇编支持 setjmp 和 longjmp，志宏在群里说这个应该可以搞，大师兄接着说代码得码出来才算数。结果过了没多久，志宏就把汇编搞定了。WebRTC 的难度也是最高的，但宏神 2 周搞定了播放，3 周就搞定了推流，封神之路就是这么简单。

Maintainer 有一个钉钉群，我们会详细交流每次提交，比较复杂的问题还会拉视频会议一起交流，这个群的名字叫做"SRS 造风者"。盘古开天辟地时并没有风，大神说要有风，所以大地就有了风。我们也是希望能给视频行业做出一点积极的贡献。

我以前觉得开源项目能展示自己的技术能力，按照自己对技术的理解和技术理想来构建一个可以长期发展的产品，最近和众大神一起码代码、交流技术，让我觉得开源项目最大的乐趣其实在于能和一群牛逼的人一起做一点有价值有意义的事。

#define TRUE LIFE//Tech is so much fun

LiveVideoStack 的采访大纲里有问我的 daily routine，那么我的作息算是比较规律的，平常起得也比较早，正常情况 6 点半就起床锻炼身体了。年纪大了，不锻炼的效果立竿见影现世报。

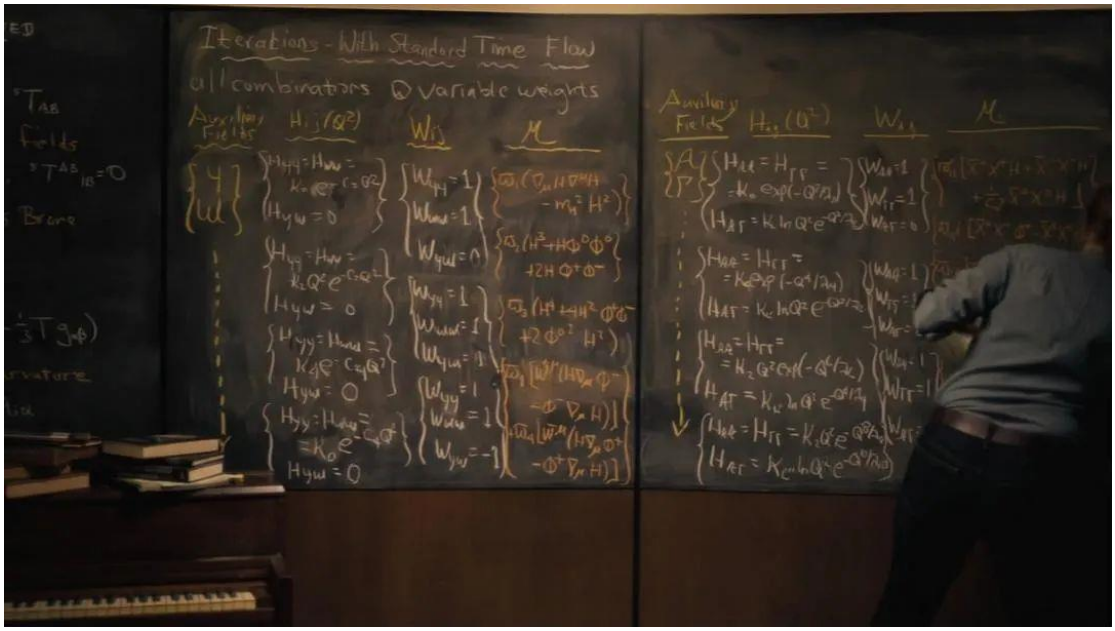
一般的上班时间都是 9 点多，但也因为我起得早，8 点左右我就能到公司，也就拥有了 1 小时左右完全属于自己的时间，这段时间我会做一些长期以来想要完成的事情，比如以前就是学口语，还有补补课、码 SRS 代码，或者了解一些别的技术。

无论是在工作还是生活中，我都尽力保持乐观。我理解的乐观，是知道无路可走还在努力并且能够乐在其中，而不是知道熬一熬就有个大饼，所以才暂时委屈下自己。知道委屈一下就有张大饼，傻子都能够保持乐观；而真正的乐观，是自己真的已经知道无路可走了，还是要一如既往的努力和乐观。

我所理解的技术有一种乐趣，是一种专业的乐趣。商业产品谁家好，有时候很难说得清，而开源项目好不好，拉出来看看就知道了，一种茹毛饮血的原始力量，不加修饰的直白，按在地上摩擦的野蛮，不然呢？

LivevideoStack 的总编包研说得很对，还是应该多讲讲自己的想法，比如为什么要做 SRS 这个项目，未来怎么发展等等。特别是现在，大风起兮，全球应疫而生的视频业务都在爆发，相比之下，2015 年的直播爆发只是国内的事情。这时候，SRS 不出来刷刷存在感，那就属于逆势而为了——一个优秀的开源项目一定蕴含光芒，风起云涌时就要光芒四射，照亮人心。

/** TODO：Remove this function*/



我希望自己能码代码到头发全白，呃……至少白一半吧，给大家证明下老头子也能写代码。所以现在每天发愁，为什么头发还不快点白。自从开始搞 WebRTC，看 RFC 协议，视力下降得非常快，我怕我头发没白眼睛先瞎了，那样就没法儿自证预言了。

SRS 接下来最难的，就是一个“取舍”的问题。到今天，已经加了不少东西进 SRS 了，有些东

西肯定是要舍弃的，没有舍就没有得，不会舍就不能得。加法比减法好做，减法也更容易做错，之前放弃过交叉编译脚本，结果被社区提 issue 提怕了，又给加了回来了。
其实没有人能预见未来，开源项目应当码好代码，不避锋芒，不媚逢迎，不辞风雨，不患得失，不畏将来，不念过往。

《C 程序设计语言（The C Programming Language）》的共同作者之一 Brian Kernighan 那一句“Don't comment bad code — rewrite it”至今仍被奉为金科玉律，这大概是基于“comments do not make up for bad code”的逻辑。同样，简洁干净的代码 speak for themselves，懂的自然懂，不懂的运行之后也一样能懂。

代码如此，写代码的人却不同。人生如同一场电影，当我们回首往事时，常常忘了剧中人物都有哪些，所以，每一个场景都需要做许多注脚。今天，LiveVideoStack 有机会为 SRS 做一个小小的注脚，感到既惶恐又幸运。

编辑：Coco Liang 图片来源：《星际穿越》

李郁韬：短期爆发还是未来趋势？腾讯云海量音视频通信服务背后的技术发展

面临特殊时期，无论远程办公、还是学校“停课不停学”都成为时下的必然选择，众多视频会议平台也在特殊环境下开启了短期免费开放及功能服务。但与此同时不可避免会带来突发的大规模在线视频与协作需求与流量冲击，面对高并发、高可用、高性能的挑战，平台背后技术如何支持？未来趋势会在哪里？我们邀请到腾讯云视频通信业务总经理李郁韬分享腾讯云在编解码、视频网络传输等方面技术优化。

Tommy　LiveVideoStack 人物专访　2020-03-03

LiveVideoStack：Tommy 你好，距离上次采访你过去了一年多的时间，腾讯视频云产品发生了哪些变化？

李郁韬：视频云这一年变化还真挺大的，首先从业务上，我们基本对接了市场上绝大部分客户的直、点播的客户，大盘的业务量增长比较明显，目前在整体带宽、音视频通话时长基本上翻了一倍。

在产品矩阵方面，我们在持续夯实我们流量型 PaaS 平台的同时，也推出了不少新的业务产品，包括更低延迟表现的快直播产品、这次雷神山云监工上用到的慢直播产品、面向制作域的腾讯云剪及云导播台产品、针对企业培训及商业直播场景的直播 SaaS 产品，另外，我们也将我们视频 AI 的产品能力升级为了“视频智能”产品家族。

在腾讯内部，视频云已经成为了腾讯视频业务事实上的中台，微信、企业微信、腾讯会议、腾讯教育等绝大部分视频产品已经跑在了我们视频中台上，在内部，我们也积极推动公司内部各种视频开源协同项目，推动效率提升及技术共享。另外，在下一代音视频编解码、音视频传输协议、音视频前后处理算法等方面，我们也在推动各技术开源社区的建设。

融合直播技术与微信生态，打通全平台 PaaS+SaaS 服务

LiveVideoStack：我们观察到小程序上支持课程开播，大大简化了教学直播的流程，也可以快速加入腾讯会议。除此之外，小程序的多媒体服务上还有哪些期待呢？

李郁韬：小程序云直播解决方案是我们融合腾讯云直播技术与微信生态优势，提供的云端直播能力及全平台打通的直播 PaaS 服务，将直播能力注入各业务场景，助力企业快捷实现小程序直播。提供经微信官方认证的小程序直播插件，引入插件就能快速便捷地将直播能力植入自有

的微信小程序中。在这次疫情期间，直播除了之前泛娱乐场景外，也更多进入了人们的生产、生活场景中。

另外，我们也联合合作伙伴，上线了针对企业内训等场景的 SaaS 解决方案：腾讯云欢句直播，这是为企业提供更轻量的小程序一站式 SaaS 直播服务，助力企业获取观众多方位画像，提升线上培训及销售跟进效果；在直播中支持将 PPT 等文档同步展示给观众，形式上也支持红包等功能，互动性更强。直播结束后自动生成回放视频，观看回放时 PPT 演示与视频同步播放。

LiveVideoStack：是否向海外用户开放腾讯云视频云的能力，比如小程序直播、H5 直播等？

李郁韬：目前已经在向海外用户开放腾讯视频云的相关能力，以【多中心点部署】在中国香港，泰国，新加坡，德国，多伦多，硅谷，俄罗斯，南非，韩国等地都已完成多个中心机房建设，并在逐渐扩大中心机房建设覆盖国家区域。
为了应对海外国家内部网络杂，海外国家跨国网络质量不一，减少卡顿，提供稳定可靠的服务，对于海外直播场景，腾讯云从架构、网络、安全、资源等方面针对性地进行了优化。

依托于腾讯云的出海战略和长期在海外的投入，在全球 50 多个国家和地区建设超过 1300 个传输节点，总带宽储备超 100T，与超过 50 家全球运营商合作，海外加速点 200+。
想体验或使用的朋友，现在可以在云直播控制台直接开启。

腾讯自研网络传输协议+V265 编码技术，全面提升用户体验

LiveVideoStack：去年 12 月看到腾讯会议的发布会，2 月初就遇到“全国人民都在远程工作”的挑战，如何在短时间内给予腾讯会议更好的支持的？与此同时，如何保障其他服务质量不降低，有哪些经验可以和大家分享？

李郁韬：为了保证复杂网络环境下多端接入方式中的音视频传输质量，腾讯会议采用了我们在音视频领域长期积累的云流控引擎技术，通过融合经典信号处理、心理听觉和深度学习理论，针对端到端音视频通信链路上的复杂特性，继承了在微信、QQ、王者荣耀等大型业务中多年沉淀的探测、选路、调度、传输等关键技术，对音视频在云端和用户终端的网络传输环境进行实时探测计算，选择最优网络路径进行传输。

同时，腾讯自研的网络传输协议和 V265 编码技术，在降低音视频传输过程中丢包率的同时，还能保证在屏幕分享场景和在不同终端上的视频清晰度，保证用户的会议使用体验。
在质量评估方面，腾讯会议还借助腾讯多媒体实验室上百个符合 ITU/3GPP/AVS 等国内外标准的指标进行评判。为了更好的衡量 QoE，腾讯多媒体实验室构建了大规模音视频主观质量数据库，并基于这些数据来研发客观质量评估算法，然后部署到业务线中，闭环监控全网的用户体验质量。

另外，用户在使用腾讯会议时，普遍感觉环境杂音较少。这个体验同样得益于腾讯云音视频在降噪方面的技术积累。腾讯会议考虑到远程会议的环境多样性，针对多人、多场景实时通信系统中的 3A 问题（降噪、回声抑制、增益控制），进行智能化降噪处理，让与会者克服噪音干扰，获得专注的会议环境。

LiveVideoStack：腾讯云视频云团队是如何应对用户使用量暴增的情况的？比如通过“降级服务”保证用户的基本服务，比如使用延迟更高的 RTMP 替代 WebRTC 低延迟线路？

李郁韬：腾讯视频云直播为用户提供 TRTC (Tencent Real-Time Communication) + WebRTC 快直播（上行走 RTMP 推流或 FLV、HLS、RTMP 回源，下行支持标准 WebRTC 协议输出）+ CDN (FLV/HLS/DASH) 大融合解决方案，像教育、会议、互动对实时性要求比较高的场景优先走 TRTC 接入(全球<300ms 延时），同时 trtc 接入会实时混流并 rtmp 推流到直播平台，用户可以根据在线并发量配置超过一定并发阈值(比如 100w）以后自动切一部分流量到 WebRTC 快直播(延时 500ms 左右）或普通 CDN 接入(延时跟用户 GOP 及 CDN buffer 配置有关，普遍延时在 2-5 秒左右)。视频云 CDN 在全球 50 多个国家和地区建设超过 1300 个传输节点，总带宽储备超 100T，与超过 50 家全球运营商合作，海外加速点 200+。

LiveVideoStack:我知道有些客户已经在测试腾讯视频云的 SRT 方案和 WebRTC CDN 产品，相较于 QUIC、RTMP 和低延迟 HLS，SRT 和 WebRTC CDN 的优势是什么？
李郁韬：SRT 相比 QUIC，有专门针对直播场景的优化，传输控制中更好的结合了实时码率的评估和 pacing rate 发送间隔的计算；另外，SRT 在传输时是可配置的允许丢包的，腾讯视频云根据音视频的编码特点，支持有选择的丢包，在降低帧率但不影响画面质量的情况下，尽可能的保障了播放的流畅性。SRT 基于 UDP 比基于 TCP 的 RTMP 和 HLS，有效解决了在长距离链路传输场景中延迟高、抗抖动性差的问题；经过实际测试，SRT 相比 RTMP 具有明显的低延迟和低卡顿的特性我们在 SRT 的扩展性上，支持所有基于 TCP 的协议，包括 RTMP/FLV/HLS，QUIC 主要用于在 FLV、HLS。

LiveVideoStack：关于视频 Codec 有哪些新消息可以透露的？

李郁韬：2019 年 9 月，腾讯正式宣布加入开放媒体联盟 AOMedia，推进视频 AV1 标准商业化，并成为董事会员之一，同时也是董事会员中唯一的中国企业。日前，腾讯视频云直播（FLV/HLS/DASH)、点播均已支持 AV1/AVS2 标准，据悉，腾讯云也是国内首家直播+点播同时支持 AV1/AVS2 视频处理业务的公有云厂商。VVC/AVS3 腾讯视频云也正在开发和工程验证阶段，2020 年也会陆续对客户开放。

LiveVideoStack：关于深度学习，从 AI 增强的 Codec，内容理解和自动化剪辑，腾讯视频云正在做哪些相关工作？

李郁韬：腾讯视频云明眸·极速高清-智能动态编码技术，通过智能场景识别，业界领先的音视

频编码、图像深度学习与画质增强等技术积累，对视频进行动态感知编码和画质增强处理，提升视频观看体验，为直播、点播等行业以更低的码率提供更高清的流媒体服务。极速高清可以在点播和直播业务下，相比开源软件在相同画质下平均分别节省超 30%+和 40%+的带宽成本；或在同等带宽条件下，提供更加高清优质的视频画质体验，感受“质”的变化。极速高清 AI 增强这块在 4K 超分、HDR、色彩增强、智能弹幕、插帧等这几方面技术在游戏赛事直播、广电高清频道以及 4K/8K 体育赛事直播上现在都有比较成功的客户应用。

腾讯视频云智眸·采识编审基于腾讯各实验室最新研究成果，为广电新媒体、教育、直播及在线视频场景提供视频内容理解剪辑、智能识别、智能编辑、智能审核等功能，70 周年 CCTV 国庆阅兵自动剪辑，游戏直播赛事精彩片段实时自动生成，直播实时语音/OCR 识别、涉黄、涉暴、涉政实时审核等。结合智眸·采识编审的视频内容理解和剪辑、编辑、识别、审核等原子能力，腾讯视频云最近推出的腾讯云剪和导播台产品在这次新冠疫情阶段在教育、会议和体育赛事直播上为媒体平台、PGC/UPGC、MCN、直播平台以及电竞内容等内容制作者和机构提供在线视频创作工具、编辑、推流等一整套 pipeline 流程解决方案。

音视频下一个风口：线下到线上的全面转型？

LiveVideoStack:从直播、短视频、直播答题、在线教育、视频会议、企业协作，多媒体应用经历了从互联网到行业，从 2C 到 2B 的渗透，你认为下一个风口在哪里？

李郁韬：短期来看，就如前面我提到的，受这次疫情的推动，线下业务、线下内容、线下商业会加速往线上迁移，包括大家看到的办公协同、音视频通信、视频营销、娱乐体验等等；有个数据，近一个月期间，腾讯云在音视频通话的分钟数达到了日均 30 亿分钟的规模，这个挺大量级的一个数字；另外，之前 VR 一直不温不火，那如果线下商业往线上迁移，类似 VR 看房、VR 看车等场景，真的是可以落地了；另外基于音视频 AI 的智能内容生产、虚拟主播、智能客服等场景也将会越来越多。

中长期看，比较看好在线办公协同和在线教育的发展，经过这段时间在家复工复课，大家感受到了科技对于紧急事件的巨大帮助，体验到了越来越丰富和稳定的音视频通信服务可以一定程度替代传统的线下沟通方式，相信未来大家也更能接受这种在线教育和办公的生活方式。

LiveVideoStackCon 技术专家

查看更多专家信息

LiveVideoStackCon 2017 北京

优秀出品人

陆其明
爱奇艺　技术总监

夏鹏
搜狐千帆直播　联合创始人

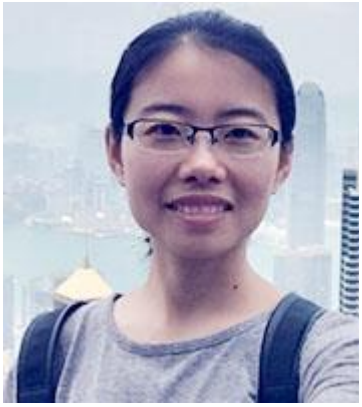
殷宇辉
360 高级总经理

董海冰
前 iTutorGroup　研发总监

查看更多专家信息

优秀讲师

刘炜
上海云熵网络科技有限公司联合创始人&CTO



赵丽丽
美图 数据部门北京技术总监

和君
TutorABC 大前端负责人

郭亮
快手 直播技术负责人

李炼
TutorABC 架构师

袁荣喜
学霸君 高级技术总监

赵加雨
拍乐云 创始人&CEO

李大龙
腾讯视频移动端播放内核技术负责人

朱政
微帧科技 联合创始人，CEO

查看更多专家信息

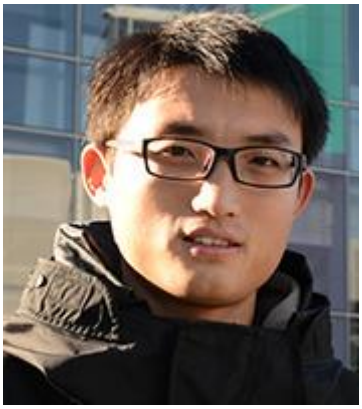
傅德良

Hulu 高级软件开发经理

陈强

360

人工智能研究院，视频大数据组技术负责人



展晓凯

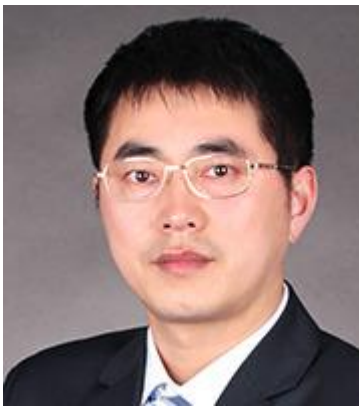
全民快乐 研发高级总监

查看更多专家信息

WebRTCon 2018

优秀出品人

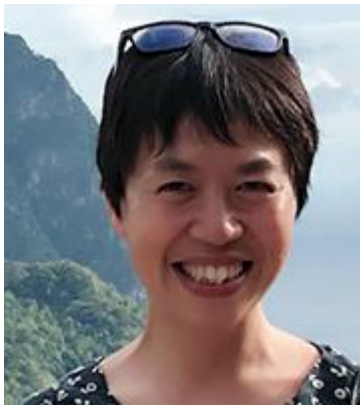
董海冰
前 iTutorGroup 研发总监



宋利
上海交通大学 教授，博士生导师

赵加雨
拍乐云 创始人&CEO

优秀讲师



Zoe Liu
微帧科技 联合创始人/首席科学家

陈虓将
阿里巴巴 高级技术专家

张琰彬
英特尔亚太研发有限公司 Open WebRTC Toolkit（OWT）测试负责人

查看更多专家信息

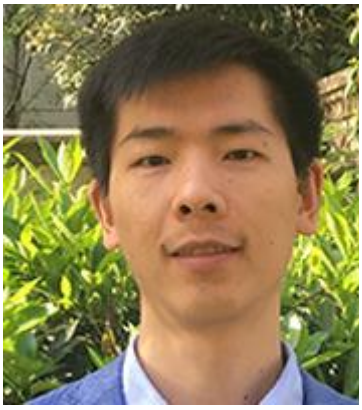
周正宁
Aupera 傲睿智存 CTO

王荣刚
北京大学 教授

吴桐
网易云信 多媒体资深技术架构师

冯巍
爱奇艺 技术产品中心研究员

王视鎏
爱奇艺 技术产品中心研究员



石硕
前战旗直播 流媒体研发工程师

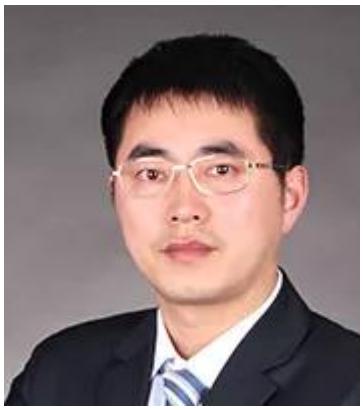
张轲
腾讯音视频实验室 高级工程师

冯昶
Zealcomm 创始人兼首席执行官

查看更多专家信息

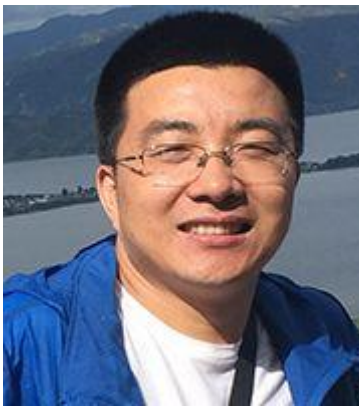
LiveVideoStackCon 2018 北京

优秀出品人



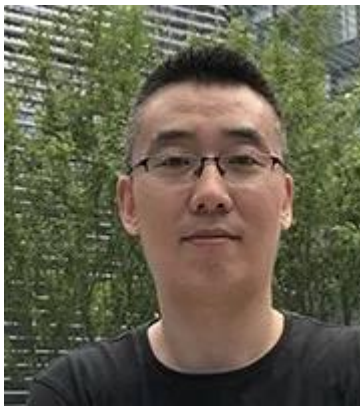
宋利
上海交通大学 教授，博士生导师

王立众
中国三星研究院 音频研发负责人



李庆丰
新浪微博 研发中心研发总监

优秀讲师



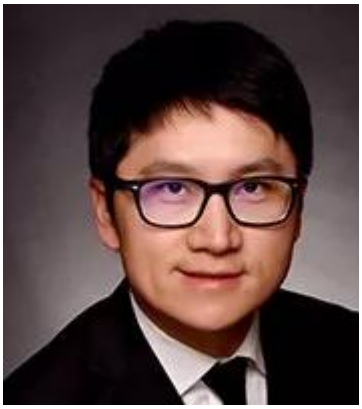
刘志勇
新浪微博 视频平台资深架构师

于冰
快手 音视频技术负责人

William Robert Law
Akamai 首席架构师

查看更多专家信息

李大龙
腾讯视频移动端播放内核技术负责人



沈悦时
Twitch Principal Research Engineer

查看更多专家信息

LiveVideoStackCon 2019 上海

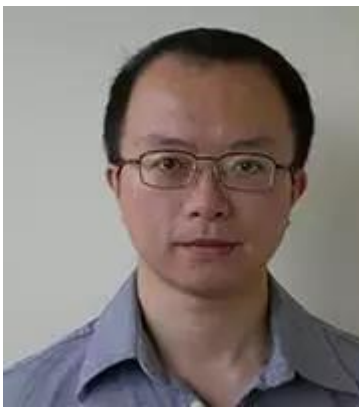
优秀出品人

袁荣喜
学霸君 高级技术总监

张昊
中南大学 教授

优秀讲师

王盛
哔哩哔哩 高级工程师



范醒哲
Cascade Range Networks, Inc.
CTO/联合创始人

虞露
浙江大学 教授

查看更多专家信息

高孟平

腾讯 音视频实验室专家

杨攀

融云 联合创始人，CTO

查看更多专家信息

LiveVideoStackCon 2019 北京

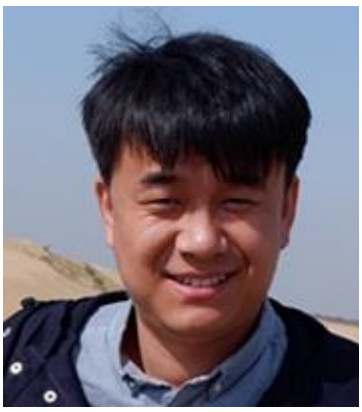
优秀出品人

姜雨晴
MediaTrack 音视频研发负责人

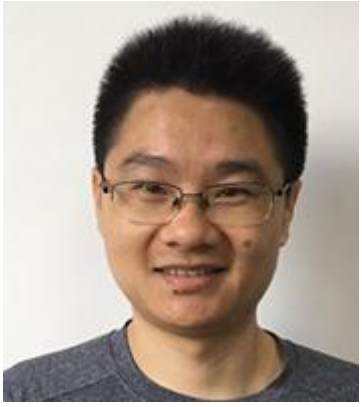
刘东
中国科学技术大学 副教授

优秀讲师

刘歧（悟空）
OnVideo 视频云创作平台
联合创始人



霍东海
新浪微博 视频平台架构师



邱国钦
即构科技 互联网业务开发技术总监

查看更多专家信息

许耀武

谷歌 首席软件工程师，技术总监

李智

Netflix 高级软件工程师

查看更多专家信息

LiveVideoStackCon 2019 深圳

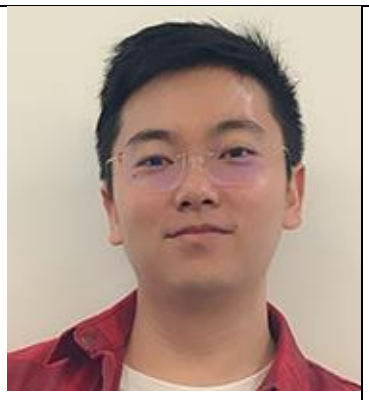
优秀出品人

徐嵩
中国移动咪咕公司
总监

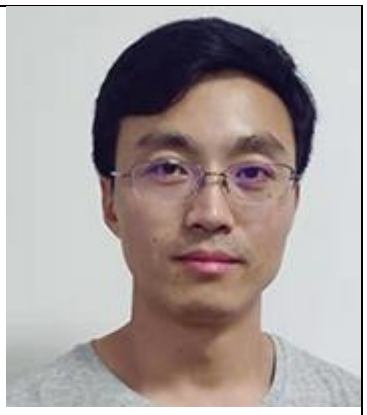
蒋宁波
即构科技
联合创始人兼技术负责人

优秀讲师

沈悦时
Twitch Principal Research Engineer

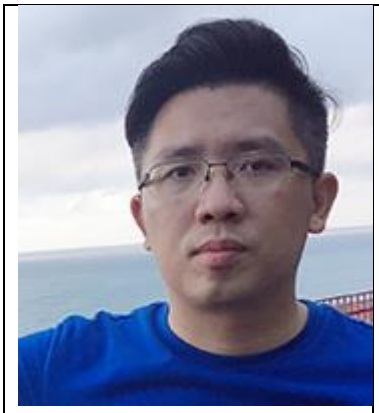
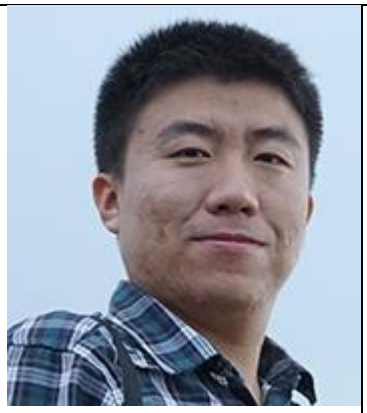
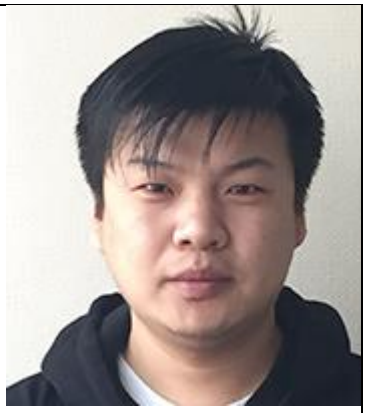


邓建勋
虎牙直播
音视频后台高级架构师



常高伟
阿里巴巴
淘系技术部技术专家

查看更多专家信息

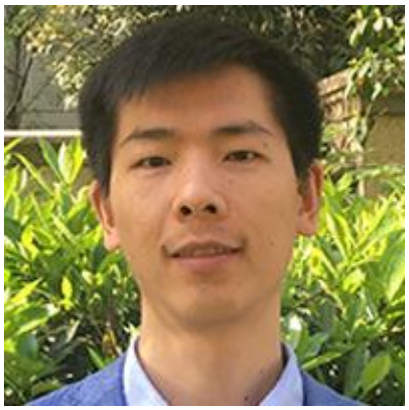
		
符乐安 字节跳动 IES 多媒体音视频团队 视频算法方向技术负责人	赵珺 腾讯 多媒体实验室高级工程师	李淼 融云 联合创始人，首席架构师

查看更多专家信息

LiveVideoStackCon 2020 首届音视频线上峰会

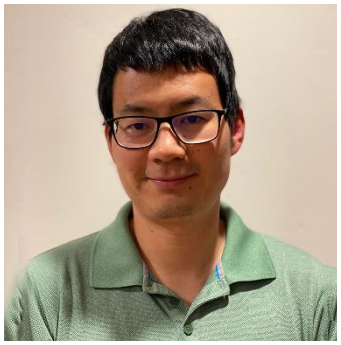
优秀出品人

刘华平
网易云音乐
音视频实验室负责人



石硕
前战旗直播
高级流媒体工程师

优秀讲师



韩敬宁
Google
Senior Staff Engineer

李佳俐
声网 Agora
高级视频算法工程师

张博力
安徽广播电视台
直播技术工程师

LiveVideoStackCon 合作伙伴

LiveVideoStackCon 2017 北京

Akamai
Experience the Edge

百家云
BAIJIAYUN
一站式视频技术服务提供商

CODING
CLOUD DEVELOPMENT

个推
GeTui

HGST
a Western Digital brand

金山云

青云 QING CLOUD

声网 agora.io

TUPUTECH
图普科技

涂图
TUTUTECH

网宿科技

SpeedyCloud
迅达云

云帆加速
yfcloud.com

intel

又拍云

ZEGO 即构

WebRTCon 2018

LiveVideoStackCon 2018 北京

Aupera
Making Video Alive

Akamai
Experience the Edge

百度智能云

百家云
BAIJIAYUN
一站式视频技术服务提供商

环信
easemob.com

金山云

ZEGO 即构

京东智联云

美摄
MEISHE

七牛云

三体云

声网 agora.io

腾讯云

拓课云
Talk-Cloud

网易云信

相芯科技
Faceunity technology

XILINX

intel

LiveVideoStackCon 2019 上海

Akamai Experience the Edge

百度智能云

BIGTERA

CTACCEL Accelerated Computing

ELEVOC 听得见的智能

环信 easemob.com

美摄 MEISHE

七牛云

融云 RONGCLOUD

容联·云通讯

三体云

声网 agora.io

腾讯云

网易云信

相芯科技
Faceunity technology

intel

ZEGO 即构

LiveVideoStackCon 2019 北京

Akamai
Experience the Edge

Aupera
Making Video Alive

BITMOVIN
Software to Solve Complex Video Problems

CTACCEL
Accelerated Computing

当虹科技
Arcvideo Tech

DEEPOLY

Fraunhofer
IIS

好视通

京东智联云

金山云

美摄
MEISHE

NGCODEC
NOW PART OF XILINX

融 云
RONGCLOUD

镕铭半导体
NETINT TECHNOLOGIES

realnetworks

三体云

声网 agora.io

腾讯云

V-NOVA

网易云信

网易易盾

相芯科技
Faceunity technology

XILINX

YOUKU
优酷技术

ZEGO 即构

LiveVideoStackCon 2019 深圳

Akamai Experience the Edge	Aupera
BROADCOM®	白山云科技 BAISHANCLOUD
CTACCEL Accelerated Computing	DEEPOLY
Fraunhofer IIS	HUAWEI
金山云	美摄 MEISHE
Microsoft	融云 RONGCLOUD
镕铭半导体 NETINT TECHNOLOGIES	三体云

声网 agora.io	腾讯云
网易易盾	网易云信
相芯科技 Faceunity technology	XILINX
小溪流 Cloudstream	intel
ZEGO 即构	

LiveVideoStackCon 2020 首届音视频线上峰会

百度智能云

HUAWEI

金山云

京东智联云

快手
记录世界 记录你

美摄
MEISHE

声网 agora.io

腾讯云

网易云信

XILINX

intel

结束

联系我们

市场与票务合作　王宇豪　13439544160
rpandora.wang@livevideostack.com

商务合作与会议赞助　刀刀　13601184114
kathy@livevideostack.com

会议内容与讲师推荐　包研　13810191484

ant.bao@livevideostack.com

日常内容与线上分享　高阳　17603229125
doyle.gao@livevideostack.com

志愿者推荐　李佩恒　18535769931

更多精彩内容

LiveVideoStack 公众号　　**LiveVideoStack 新闻站**