WebRTCon

七牛 WebRTC 连麦服务端架构实践

七牛云直播流媒体负责人 谢然

主办方：LiveVideoStack 音视频技术社区

LiveVideo StackCon 音视频技术大会

技术开启新“视”界

Technology Bring New Vision

2018.10.19-20　北京丽亭华苑酒店

LiveVideoStackCon 2018正在招募出品人、讲师

自荐或推荐：speaker@livevideostack.com

大会购票通道

LiveVideoStack 音视频技术社区

CSDN

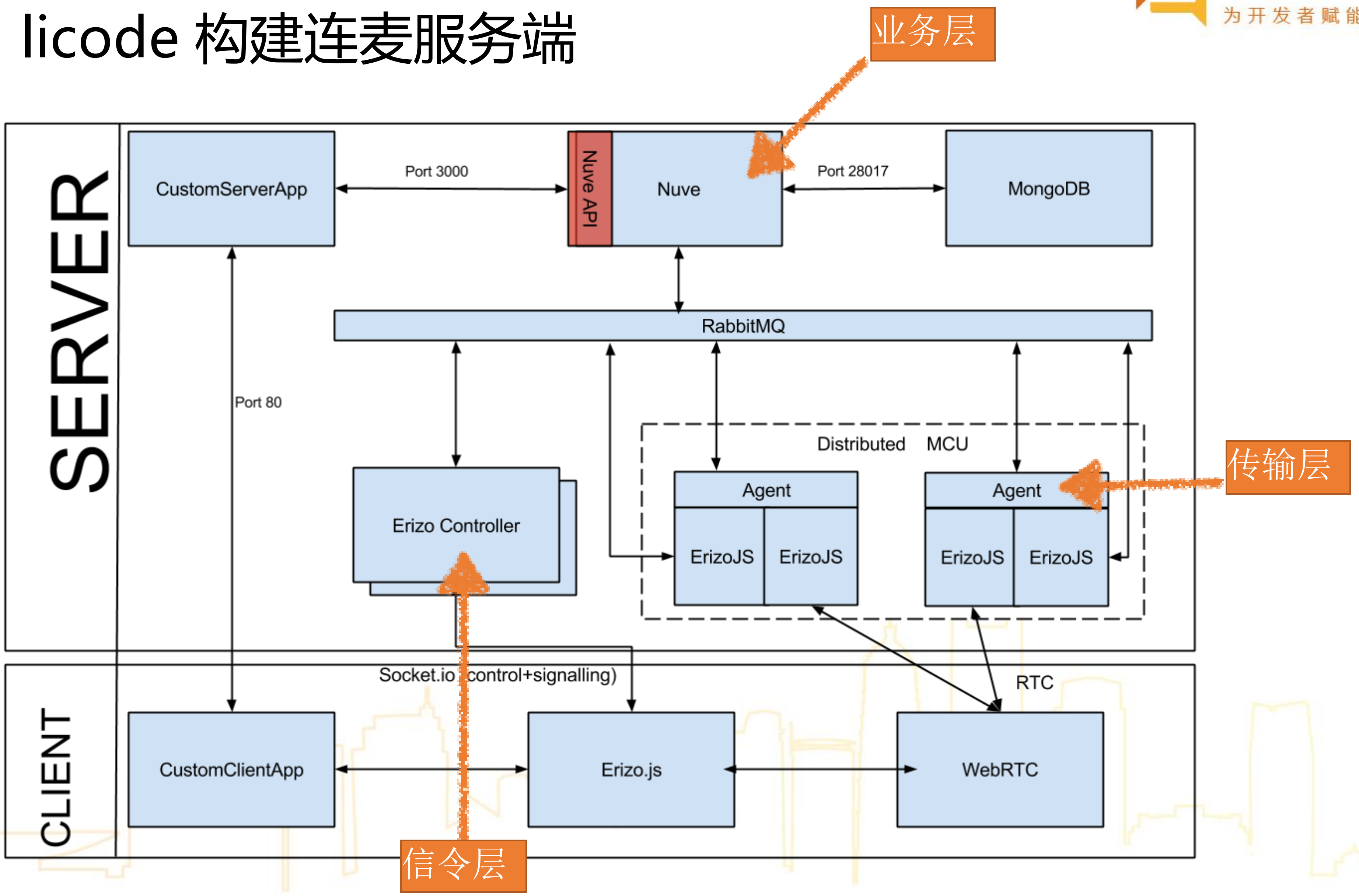
目录

- 使用 licode 的传输层实现连麦 SFU
  - licode 架构层
  - 剥离传输层
- 使用 WebRTC/ffmpeg 实现服务端合流 MCU
  - WebRTC 相关代码修改
  - GPU 加速
- 自研边缘透明加速
  - ICE 概述
  - 实现原理

使用 licode 构建连麦服务端

- 连麦架构
  - 业务层（用户管理，房间管理）
  - 信令层（房间内部消息，ICE）
  - 传输层（Rtp/Rtcp）

使用 licode 构建连麦服务端



使用 licode 构建连麦服务端

- licode 传输层架构
  - libnice
  - libsrtp
  - Pipeline
  - OneToManyProcessor

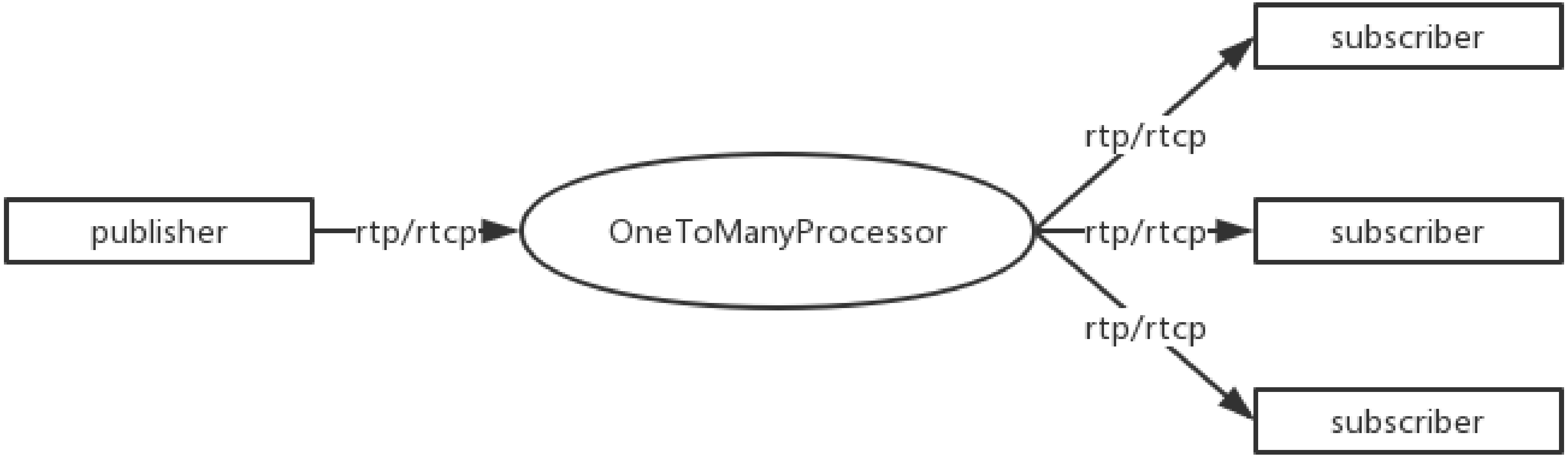
使用 licode 构建连麦服务端

- Pipeline
  - 按一定顺序处理 rtp 包

```
pipeline_->addFront(PacketReader(this));
pipeline_->addFront(RtpDumpHandler(this, false));
pipeline_->addFront(LayerDetectorHandler());
pipeline_->addFront(RtcpProcessorHandler());
pipeline_->addFront(FecReceiverHandler());
pipeline_->addFront(LayerBitrateCalculationHandler());
pipeline_->addFront(QualityFilterHandler());
pipeline_->addFront(IncomingStatsHandler());
pipeline_->addFront(RtpTrackMuteHandler());
pipeline_->addFront(RtpSlideShowHandler());
pipeline_->addFront(RtpPaddingGeneratorHandler());
pipeline_->addFront(PliPacerHandler());
pipeline_->addFront(BandwidthEstimationHandler());
pipeline_->addFront(RtpPaddingRemovalHandler());
pipeline_->addFront(RtcpFeedbackGenerationHandler());
pipeline_->addFront(RtpRetransmissionHandler());
pipeline_->addFront(SRPacketHandler());
pipeline_->addFront(SenderBandwidthEstimationHandler());
pipeline_->addFront(OutgoingStatsHandler());
pipeline_->addFront(RtpDumpHandler(this, true));
pipeline_->addFront(PacketWriter(this));
```

使用 licode 构建连麦服务端

- OneToManyProcessor



WebRTCon 为开发者赋能 为行业加速

使用 licode 构建连麦服务端

- 剥离传输层
    - erizo_controller/erizoJS（去掉）
    - erizoAPI（保留 or 去掉）
    - erizo（保留）

使用 licode 构建连麦服务端

- 定制微服务接口
  - new-conn={ "ice_servers": [{ "urls": ["stun:..."] },] }
  - set-remote-desc-create-answer={ "id": "xx", "sdp": "xx" }
  - on-ice-candidate={ "id": "xx", "candidate": "xx" }
  - add-ice-candidate={ "id":"xx", "candidate": "xx" }
  - licode-new-one-to-many-processor={ }
  - licode-one-to-many-processor-set-pub={ "id": "xx", "pub": "xx" }
  - licode-one-to-many-processor-add-sub={ "id": "xx", "sub": "xx" }

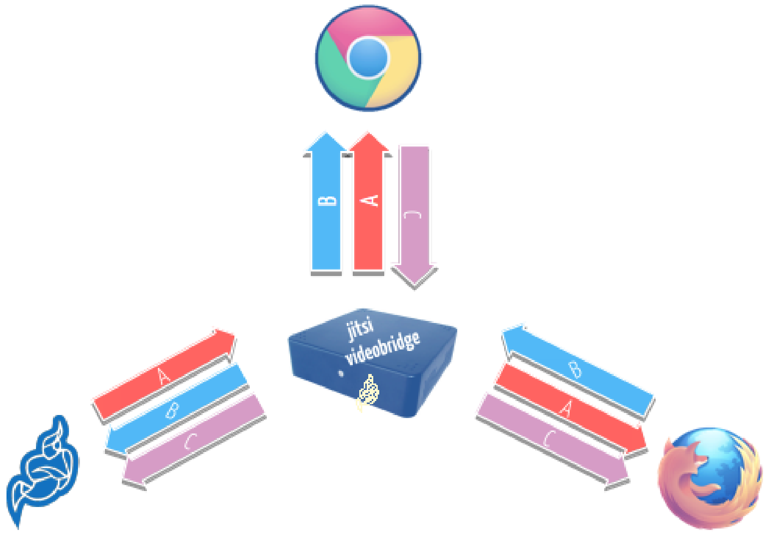
使用 licode 构建连麦服务端

- ExternalOutput
  - 可以实现 WebRTC to RTMP 的传输层转换
  - 参数是 url
  - 将 RTP 包拼成 ffmpeg 的 AVPacket 然后用 avformat 发到 url
  - 和 client 一样作为 subscriber 加在 OneToManyProcessor 下

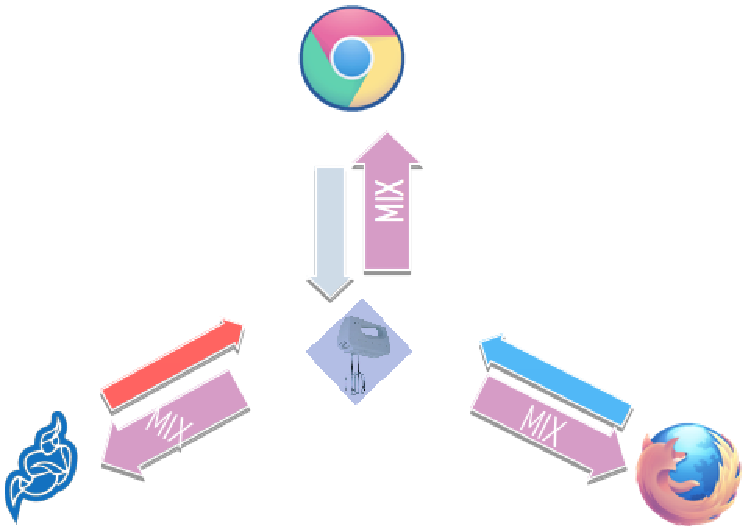
使用 WebRTC/ffmpeg 实现服务端合流

WebRTCon 为开发者赋能 为行业加速

THE SFU

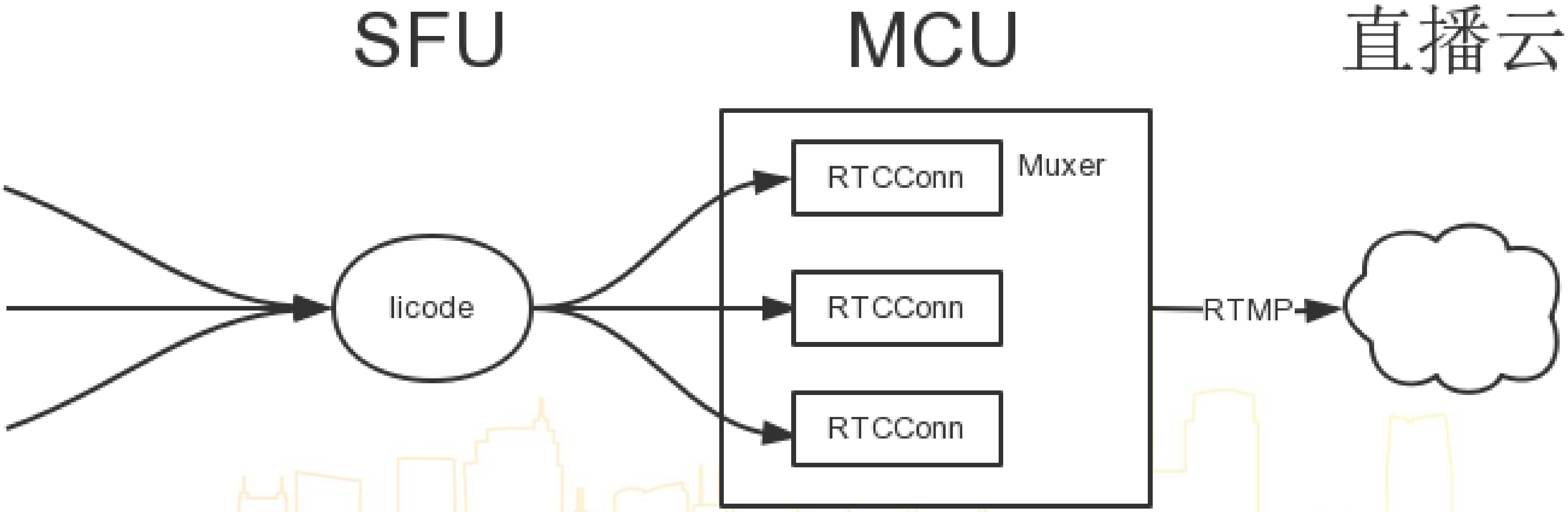


THE MCU



使用 WebRTC/ffmpeg 实现服务端合流

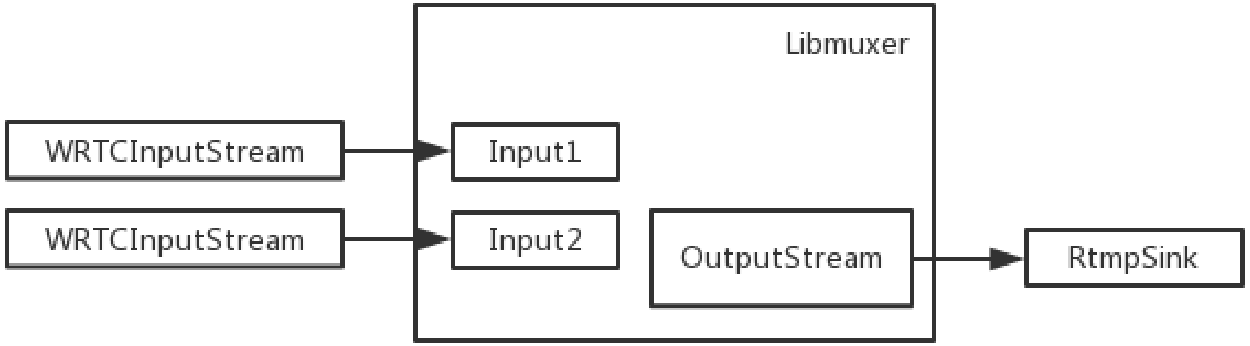
- 七牛服务端合流架构



使用 WebRTC/ffmpeg 实现服务端合流

WebRTCon 为开发者赋能 为行业加速

- 合流流媒体层设计
  - Stream
    - WRTCInputStream
    - WRTCOutputStream
    - FileInputStream
  - Sink
    - RtmpSink
    - FileSink
  - Libmuxer
    - OutputStream
    - AddInputStream



使用 WebRTC/ffmpeg 实现服务端合流

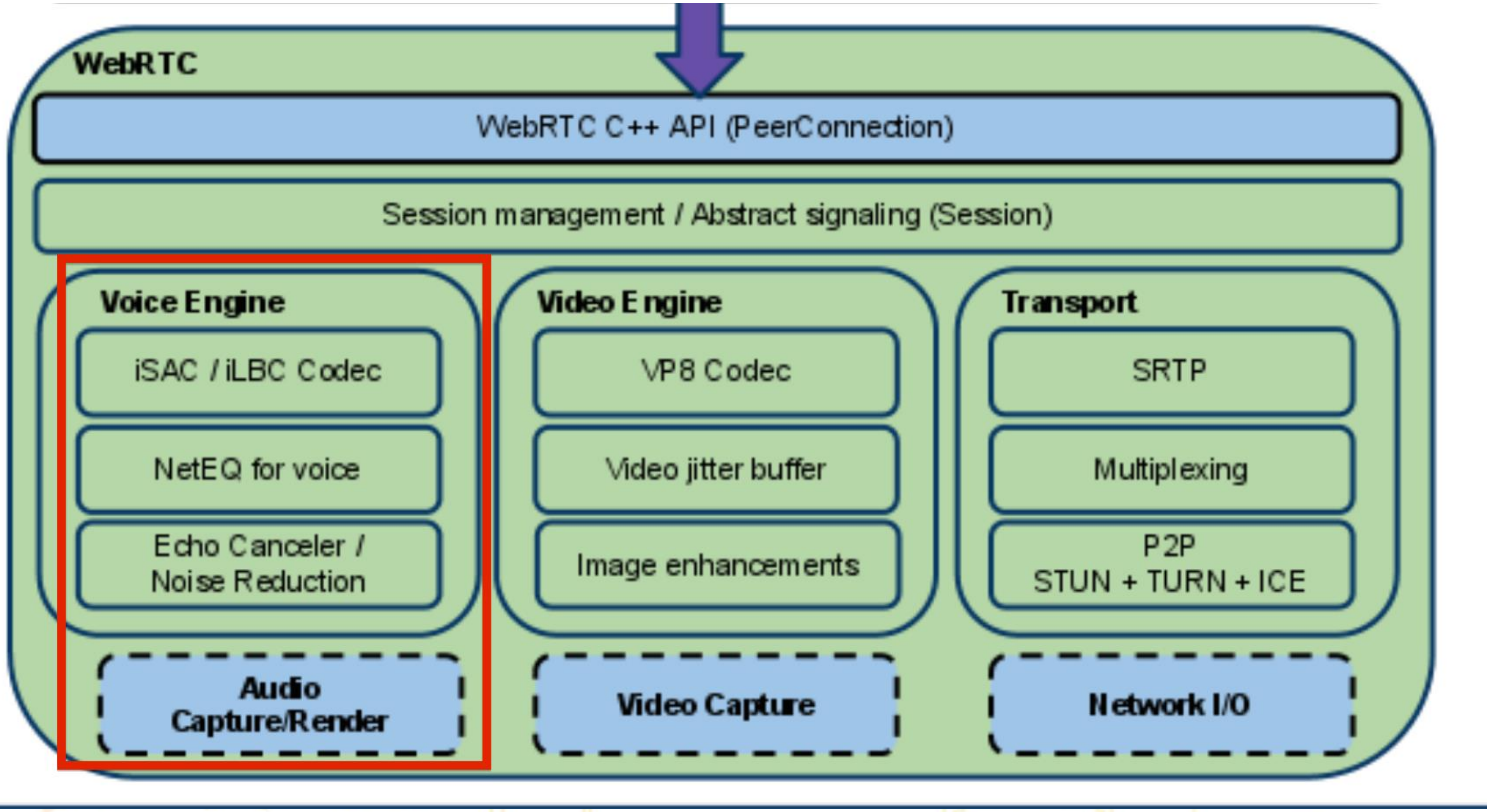
- 技术选型
  - WebRTC 官方 repo VS licode client
    - licode 支持的传输层特性不全
  - ffmpeg

使用 WebRTC/ffmpeg 实现服务端合流

- WebRTC 相关代码修改
  - 对音频处理部分的修改
  - ffmpeg 编译选项的修改

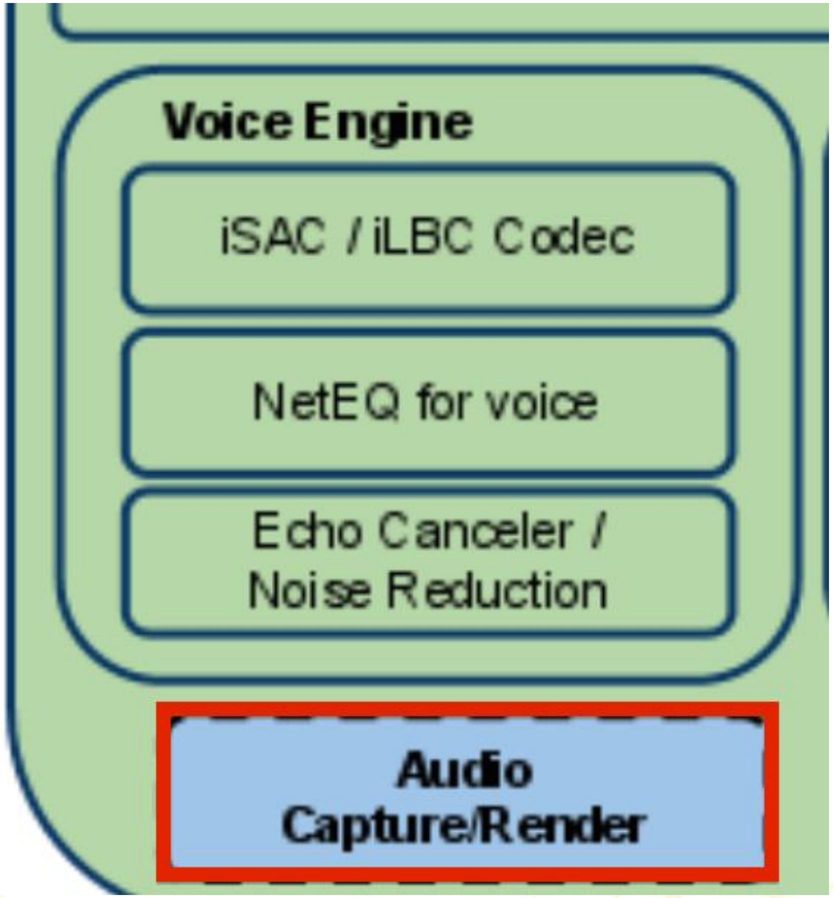
使用 WebRTC/ffmpeg 实现服务端合流

- 对音频处理部分的修改
  - 仅针对客户端
  - 非服务端需要



使用 WebRTC/ffmpeg 实现服务端合流

- 对音频处理部分的修改
  - Audio Capture/Render
    - 被 Voice Engine 依赖
    - 依赖平台的音频驱动
    - 服务器无法使用音频驱动
    - 需要实现 FakeAudioDevice
  - Voice Engine
    - 单输入
    - 较难剥离



使用 WebRTC/ffmpeg 实现服务端合流

- 对音频处理部分的修改
  - 实现 FakeAudioDevice
    - modules/audio_device/include/audio_device.h
      - RegisterAudioCallback()
      - StartPlayout() / Playing()
      - StartRecording() / Recording()
    - 参考现有的 AudioDevice 来实现 FakeAudioDevice

使用 WebRTC/ffmpeg 实现服务端合流

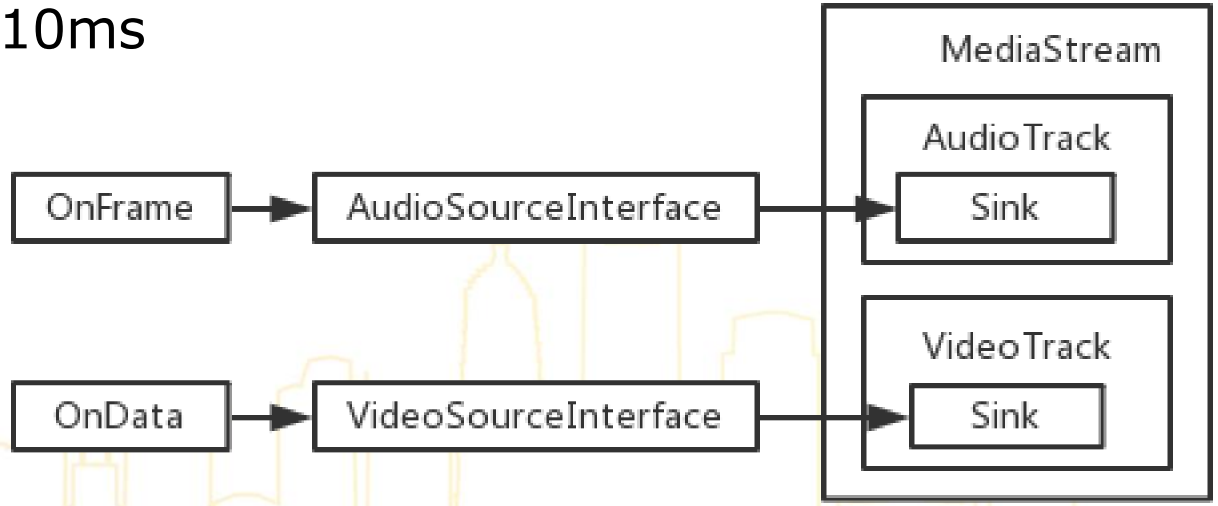
- ffmpeg 编译选项的修改
  - 原生仅支持 openh264 decoder（LICENSE 原因）
  - 换成 ffmpeg 自带的 h264 decoder
  - 换成硬件加速的 h264 decoder
  - 增加 format/decoder/encoder 用于调试

使用 WebRTC/ffmpeg 实现服务端合流

- ffmpeg 编译选项的修改
  - 特殊的编译方式
  - 生成 ffmpeg_generated.gni
  - chromium/scripts/build_ffmpeg.py
  - chromium/scripts/generate_gn.py
  - chromium/scripts/copy_config.sh

使用 WebRTC/ffmpeg 实现服务端合流

- 成为推流端
  - 创建 AudioSourceInterface / VideoSourceInterface
  - 创建 AudioTrackInterface / VideoTrackInterface
  - 创建 CreateLocalMediaStream
  - 调用 Source 的 OnFrame / OnData 填数据
  - OnData 音频 frame size 必须是 10ms



使用 WebRTC/ffmpeg 实现服务端合流

- 使用 GPU 加速
  - Nvidia Tesla 平台介绍

使用 WebRTC/ffmpeg 实现服务端合流

- 在 ffmpeg 中使用 Nvidia Tesla 方案
  - https://developer.nvidia.com/ffmpeg

FFmpeg GPU HW-Acceleration Support Table

	Fermi	Kepler	Maxwell (1st Gen)	Maxwell (2nd Gen)	Maxwell (GM206)	Pascal
H.264 encoding	N/A	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3
HEVC encoding	N/A	N/A	N/A	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3
MPEG2, MPEG-4, H.264 decoding	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3
HEVC decoding	N/A	N/A	N/A	N/A	FFmpeg v3.3	FFmpeg v3.3
VP9 decoding	N/A	N/A	N/A	FFmpeg v3.3	FFmpeg v3.3	FFmpeg v3.3

使用 WebRTC/ffmpeg 实现服务端合流

- 在 ffmpeg 中使用 Nvidia Tesla 方案
  - ./configure --enable-cuda --enable-cuvid --enable-nvenc --enable-nonfree --enable-libnpp —extra-cflags=-I/usr/local/cuda/include —extra-ldflags=-L/usr/local/cuda/lib64
  - 以闭源库方式提供

使用 WebRTC/ffmpeg 实现服务端合流

- CPU vs GPU 结果对比（CPU）
  - 高分辨率高画质占用 CPU 较多
  - CPU 占用率波动大
  - 内存占用率稳定

规格/输出	CPU	MEM
libx264_faster_480P	95%	106Mb
libx264_faster_720P	160%	122Mb
libx264_faster_1080P	250%	150Mb
libx264_veryfast_480P	80%	104Mb
libx264_veryfast_720P	115%	118Mb
libx264_veryfast_1080P	160%	142Mb
libx264_ultrafast_480P	58%	102Mb
libx264_ultrafast_720P	77%	112Mb
libx264_ultrafast_1080P	110%	128Mb

使用 WebRTC/ffmpeg 实现服务端合流

- CPU vs GPU 结果对比（GPU）
  - GPU 占用率在 20%~35%
  - CPU 占用率稳定在 35%~55%
  - 内存 & 显存占用较多
  - 总显存为 8G

规格/输出	CPU	MEM	DISP MEM
nvenc_medium_1080P	55%	435Mb	279Mb
nvenc_medium_720P	42%	405Mb	230Mb
nvenc_medium_480P	36%	256Mb	170Mb
nvenc_fast_1080P	55%	430Mb	279Mb
nvenc_fast_720P	42%	375Mb	230Mb
nvenc_fast_480P	36%	256Mb	170Mb

使用 WebRTC/ffmpeg 实现服务端合流

- CPU vs GPU 结果对比（结论）
  - 越高分辨率高画质的编码 GPU 优势越明显

实现边缘透明加速

- 保障各地区边缘用户正常接入连麦
  - 加速传输层（udp）
  - 加速信令层（tcp）

实现边缘透明加速

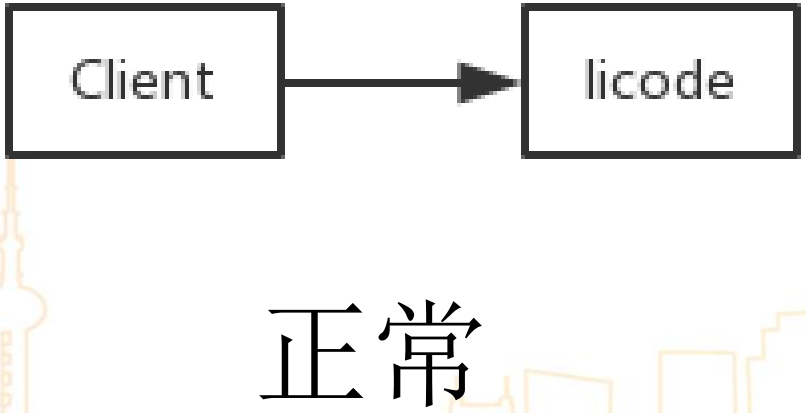
- ICE 简介
  - https://tools.ietf.org/html/rfc5245
  - candidate 的作用

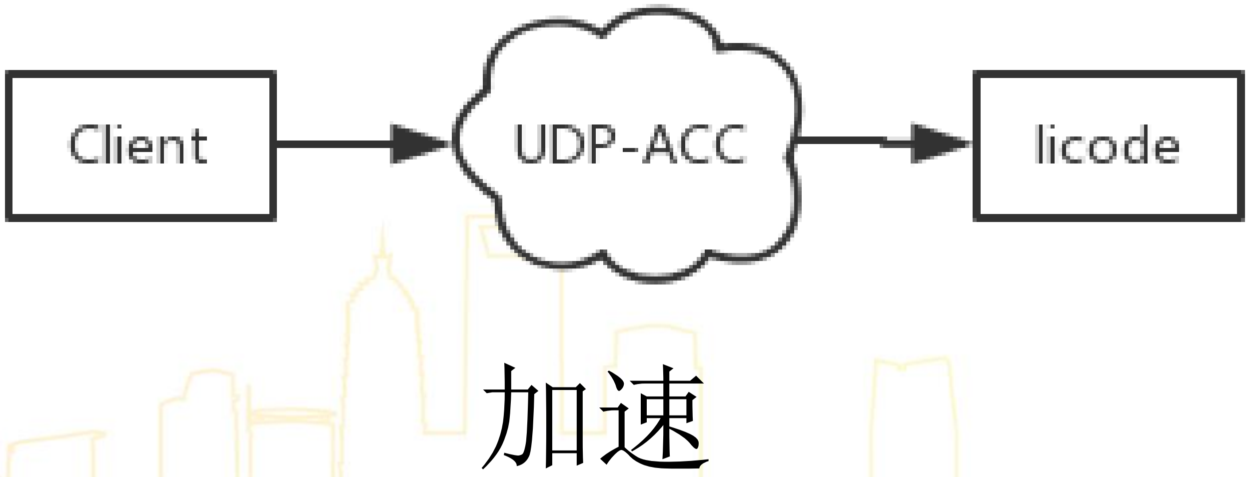
a=candidate:1467250027 1 udp 2122260223 192.168.0.196 46243 typ host generation 0
a=candidate:1467250027 2 udp 2122260222 192.168.0.196 56280 typ host generation 0

  - 控制 candidate 下发机制以实现加速
    - 提前分配路径
    - 更改 candidate 端口

实现边缘透明加速

- 正常路径 vs 边缘加速路径

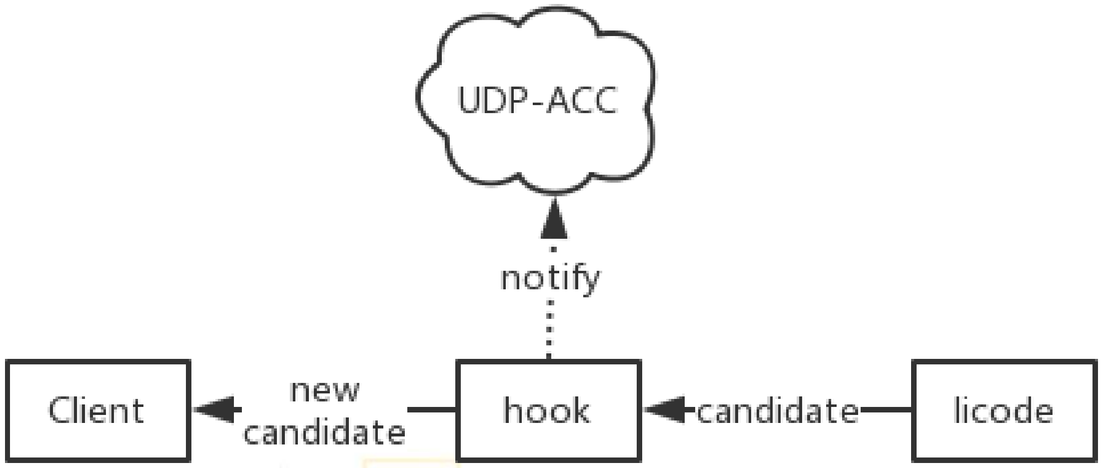




实现边缘透明加速

- candidate 下发：正常 vs 加速

正常



加速

Thank You！

主办方：LiveVideoStack