WebRTC 音视频同步机制实现分析

音视频同步事关多媒体产品的最直观用户体验，是音视频媒体数据传输和渲染播放的最基本质量保证。音视频如果不同步，有可能造成延迟、卡顿等非常影响用户体验的现象。因此，它非常重要。一般说来，音视频同步维护媒体数据的时间线顺序，即发送端在某一时刻采集的音视频数据，接收端在另一时刻同时播放和渲染。

本文在深入研究 WebRTC 源代码的基础上，分析其音视频同步的实现细节，包括 RTP 时间戳的产生，RTCP SR 报文的构造、发送和接收，音视频同步的初始化和同步过程。RTP 时间戳是 RTP 数据包的基石，而 RTCP SR 报文是时间戳和 NTP 时间之间进行转换的基准。下面详细描述之。

1 RTP 时间戳的产生

个人认为，RTP 时间戳和序列号是 RTP 协议的精华所在：前者定义媒体负载数据的采样时刻，描述负载数据的帧间顺序；后者定义 RTP 数据包的先后顺序，描述媒体数据的帧内顺序。关于 RTP 时间戳，摘录 RFC3550 定义如下[1]：

“The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations. The resolution of the clock must be sufficient for the desired synchronization accuracy and for measuring packet arrival jitter (one tick per video frame is typically not sufficient). ”

由以上定义可知，RTP时间戳反映RTP负载数据的采样时刻，从单调线性递增的时钟中获取。时钟的精度由RTP负载数据的采样频率决定，比如视频的采样频率一般是90khz，那么时间戳增加1，则实际时间增加1/90000秒。

下面回到WebRTC源代码内部，以视频采集为例分析RTP时间戳的产生过程，如图1所示。

视频采集线程以帧为基本单位采集视频数据，视频帧从系统 API 被采集出来，经过初步加工之后到达 VideoCaptureImpl::IncomingFrame()函数，设置 render_time_ms_为当前时间(其实就是采样时刻)。

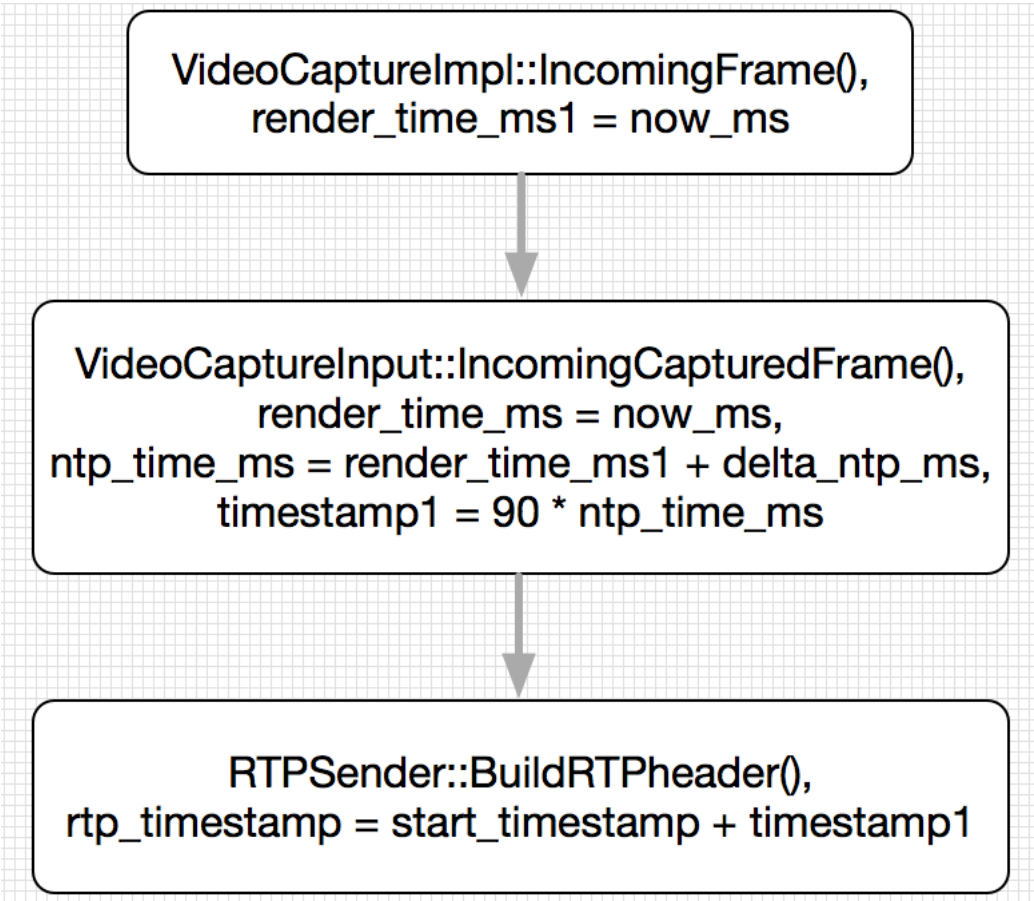

图 1 RTP 时间戳构造过程

执行流程到达 VideoCaptureInput::IncomingCapturedFrame()函数后，在该函数设置视频帧的 timestamp，ntp_time_ms 和 render_time_ms。其中 render_time_ms 为当前时间，以毫秒为单位；ntp_time_ms 为采样时刻的绝对时间表示，以毫秒为单位；timestamp 则是采样时间的时间戳表示，是 ntp_time_ms 和采样频率

frequency 的乘积，以 1/videoFrequency 秒为单位。由此可知，timestamp 和 ntp_time_ms 是同一采样时刻的不同表示。

接下来视频帧经过编码器编码之后，发送到 RTP 模块进行 RTP 打包和发送。构造 RTP 数据包头部时调用 RtpSender::BuildRTPheader()函数，确定时间戳的最终值为 rtphdr->timestamp = start_timestamp + timestamp，其中 start_timestamp 是 RtpSender 在初始化时设置的初始时间戳。RTP 报文构造完毕之后，经由网络发送到对端。

2 SR 报文构造和收发

由上一节论述可知，NTP 时间和 RTP 时间戳是同一时刻的不同表示，区别在于精度不同。NTP 时间是绝对时间，以毫秒为精度，而 RTP 时间戳则和媒体的采样频率有关。因此，我们需要维护一个 NTP 时间和 RTP 时间戳的对应关系，该用以对两种时间的进行转换。RTCP 协议定义的 SR 报文维护了这种对应关系，下面详细描述。

2.1 时间戳初始化

在初始化阶段，ModuleRtpRtcpImpl::SetSendingStatus()函数会获取当前 NTP 时间的时间戳表示(ntp_time * frequency)，作为时间戳初始值分别设置 RTPSender 和 RTCPSender 的 start_timestamp 参数(即上节在确定 RTP 数据包头不时间戳时的初始值)。

视频数据在编码完之后发往 RTP 模块构造 RTP 报文时，视频帧的时间戳

timestamp 和本地时间 capture_time_ms 通过 RTCPSender::SetLastRtpTime()函数记录到 RTCPSender 对象的 last_rtp_timestamp 和 last_frame_capture_time_ms 参数中，以将来将来构造 RTCP SR 报文使用。

2.2 SR 报文构造及发送

WebRTC 内部通过 ModuleProcessThread 线程周期性发送 RTCP 报文，其中 SR 报文通过 RTCPSender::BuildSR(ctx)构造。其中 ctx 中包含当前时刻的 NTP 时间，作为 SR 报文[1]中的 NTP 时间。接下来需要计算出此刻对应的 RTP 时间戳，即假设此刻有一帧数据刚好被采样，则其时间戳为：

```
rtp_timestamp = start_timestamp_ + last_rtp_timestamp_ +
    (clock_->TimeInMilliseconds() - last_frame_capture_time_ms_) *
    (ctx.feedback_state_.frequency_hz / 1000);
```

至此，NTP 时间和 RTP 时间戳全部齐活儿，就可以构造 SR 报文进行发送了。

2.3 SR 接收

接收端在收到 SR 报文后，把其中包含的 NTP 时间和 RTP 时间戳记录在 RTCPSenderInfo 对象中，供其他模块获取使用。比如通过 RTCPReceiver::NTP()或者 SenderInfoReceived()函数获取。

3. 音视频同步

前面两节做必要的铺垫后，本节详细分析 WebRTC 内部的音视频同步过程。

3.1 初始化配置

音视频同步的核心就是根据媒体负载所携带 RTP 时间戳进行同步。在 WebRTC 内部，同步的基本对象是 AudioReceiveStream/VideoReceiveStream，根据 sync_group 进行相互配对。同步的初始化设置过程如图 2 所示。

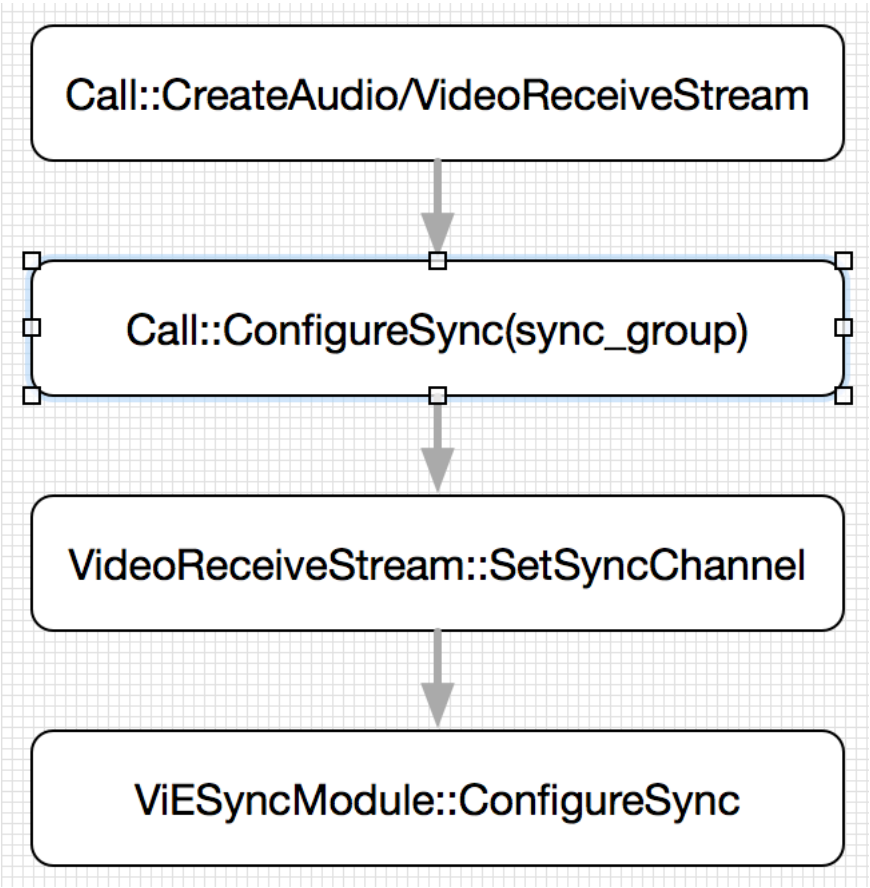

图 2 音视频同步初始化配置

Call 对象在创建 Audio/VideoReceiveStream 时，调用 ConfigureSync()进行音视频同步的配置，配置参数为 sync_group，该参数在 PeerConnectionFactory 在创建 MediaStream 时指定。在 ConfigureSync()函数内部，通过 sync_group 查找 sync_stream_mapping 得到 AudioReceiveStream，再在 video_receive_streams 中查

找得到 VideoReceiveStream。然后调用 VideoReceiveStream::SetSyncChannel 进行同步。最后在 ViESyncModule::ConfigureSync()函数中，把和音视频相关的参数进行保存，包括音频的 voe_channel_id、voe_sync_interface, 和视频的 video_rtp_receiver、video_rtp_rtcp。

3.2 同步过程

音视频的同步过程在 ModuleProcessThread 线程中执行。ViESyncModule 作为一个模块注册到 ModuleProcessThread 线程中，其 Process()函数被该线程周期性调用，实现音视频同步操作。

音视频同步的核心思想就是以 RTCP SR 报文中携带的 NTP 时间和 RTP 时间戳作为时间基准，以 AudioReceiveStream 和 VideoReceiveStream 各自收到最新 RTP 时间戳 timestamp 和对应的本地时间 receive_time_ms 作为参数，计算音视频流的相对延迟，然后结合音视频的当前延迟计算最终的目标延迟，最后把目标延迟发送到音视频模块进行设置。目标延迟作为音视频渲染时的延迟下限值。整个过程如图 3 所示。

首先，从 VideoReceiver 获得当前视频延迟 current_video_delay，即 video_jitter_delay,decode_delay 和 render_delay 的总和。然后从 VoEVideoSyncImpl 获得当前音频延迟 current_audio_delay，即 audio_jitter_delay 和 playout_delay 的总和。

然后，音视频分别以各自的 rtp_rtcp 和 rtp_receiver 更新各自的 measure。其基本操作包括：从 rtp_receiver 获取最新接收的 RTP 时间戳 latest_timestamp 和对应的

本地接收时刻 latest_receive_time_ms，从 rtp_rtcp 获取最新接收的 RTCP SR 报文中的 NTP 时间 ntp 和 RTP 时间戳 rtcp_timestamp。然后这些数据都存储到 measure 中。注意 measure 中保存最新两对 RTCP SR 报文中的 NTP 时间和 RTP 时间戳，用来在下一步计算媒体流的采样频率。

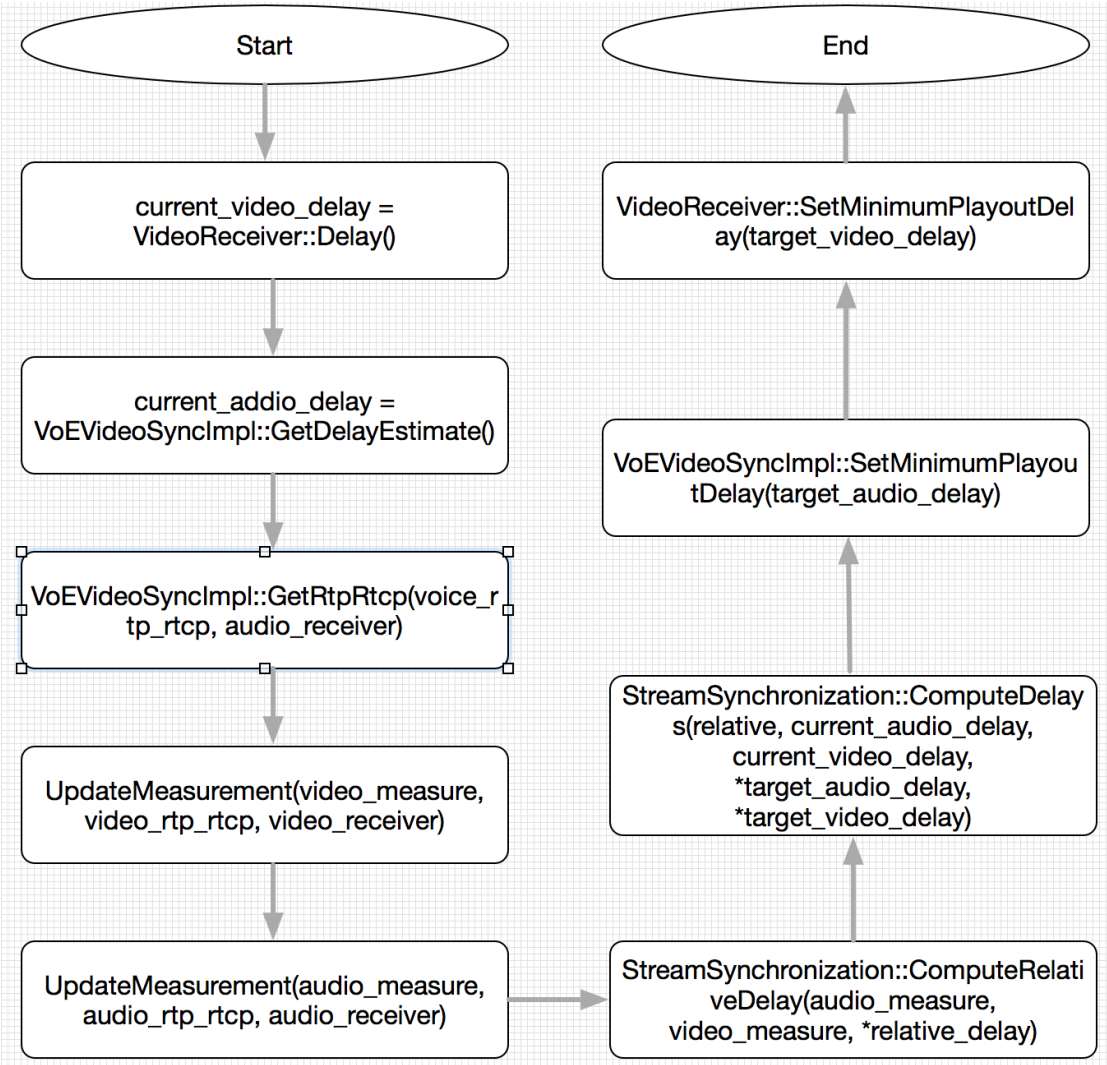

图 3 音视频同步过程

接下来，计算最新收到的音视频数据的相对延迟。其基本流程如下：首先得到最新收到 RTP 时间戳 latest_timestamp 对应的 NTP 时间 latest_capture_time，这里用到 measure 中存储的 latest_timestamp 和 RTCP SR 的 NTP 时间和 RTP 时间戳

timestamp，利用两对数值计算得到采样频率 freqency，然后有 latest_capture_time = latest_timestamp / frequency，得到单位为毫秒的采样时间。最后得到音视频的相对延迟：

```
relative_delay = video_measure.latest_receive_time_ms -
                 audio_measure.latest_receive_time_ms -
                 (video_last_capture_time - audio_last_capture_time)
```

至此，我们得到三个重要参数：current_video_delay, current_audio_delay 和 relative_delay。接下来用这三个参数计算音视频的目标延迟：首先计算总相对延迟 current_diff = current_video_delay – current_audio_delay + relative_delay，根据历史值对其求加权平均值。如果 current_diff ＞ 0，表明当前视频延迟比音频延迟长，需要减小视频延迟或者增大音频延迟；反之如果 current < 0，则需要增大视频延迟或者减小视频延迟。经过此番调整之后，我们得到音视频的目标延迟 audio_target_delay 和 video_target_delay。

最后一步，我们把得到的目标延迟 audio_target_delay 和 video_target_delay 分别设置到音视频模块中，作为将来渲染延迟的下限值。至此，一次音视频同步操作完成。该操作在 ModuleProcessThread 线程中会周期性执行。

4 总结

本文详细分析了 WebRTC 内部音视频同步的实现细节，，包括 RTP 时间戳的产生，RTCP SR 报文的构造、发送和接收，音视频同步的初始化和同步过程。通过本文，对 RTP 协议、流媒体通信和音视频同步有更深入的认识。

参考文献

[1] RFC3550 - RTP: A Transport Protocol for Real-Time Applications