WebRTC 基于 GCC 的拥塞控制(下) - 实现分析

本文在文章[1]的基础上，从源代码实现角度对 WebRTC 的 GCC 算法进行分析。主要内容包括： RTCP RR 的数据源、报文构造和接收，接收端基于数据包到达延迟的码率估计，发送端码率的计算以及生效于目标模块。

拥塞控制是实时流媒体应用的重要服务质量保证。通过本文和文章[1][2]，从数学基础、算法步骤到实现细节，对 WebRTC 的拥塞控制 GCC 算法有一个全面深入的理解，为进一步学习 WebRTC 奠定良好基础。

1 GCC 算法框架再学习

本节内容基本上是文章[1]第 1 节的复习，目的是再次复习 GCC 算法的主要框架，梳理其算法流程中的数据流和控制流，以此作为后续章节的行文提纲。GCC 算法的数据流和控制流如图 1 所示。

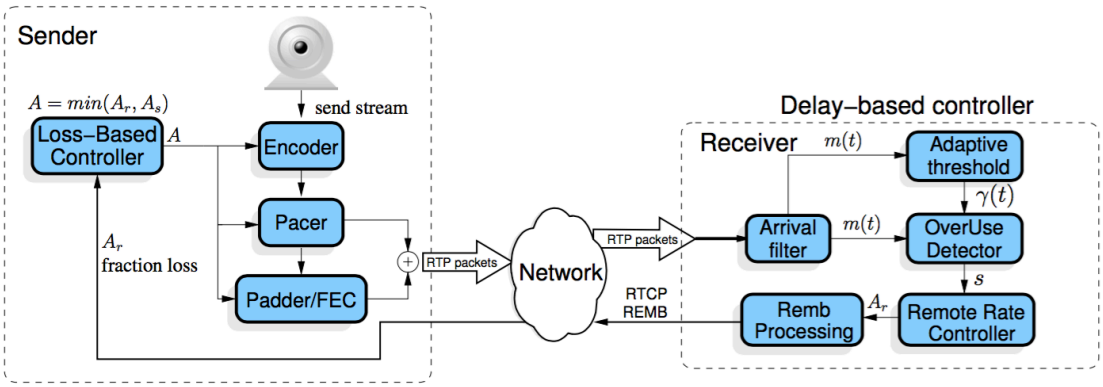

图 1 GCC 算法数据流和控制流

对发送端来讲，GCC 算法主要负责两件事：1)接收来自接收端的数据包信息反馈，包括来自 RTCP RR 报文的丢包率和来自 RTCP REMB 报文的接收端估计码率，综合

本地的码率配置信息，计算得到目标码率 A。2)把目标码率 A 生效于目标模块，包括 PacedSender、RTPSender 和 ViEEncoder 等。

对于接收端来讲，GCC 算法主要负责两件事：1）统计 RTP 数据包的接收信息，包括丢包数、接收 RTP 数据包的最高序列号等，构造 RTCP RR 报文，发送回发送端。2)针对每一个到达的 RTP 数据包，执行基于到达时间延迟的码率估计算法，得到接收端估计码率，构造 RTCP REMB 报文，发送回发送端。

由此可见，GCC 算法由发送端和接收端配合共同实现，接收端负责码率反馈数据的生成，发送端负责根据码率反馈数据计算目标码率，并生效于目标模块。本文接下来基于本节所述的 GCC 算法的四项子任务，分别详细分析之。

2 RTCP RR 报文构造及收发

关于 WebRTC 上的 RTP/RTCP 协议的具体实现细节，可参考文章[3]。本节主要从 RR 报文的数据流角度，对其数据源、报文构造和收发进行分析。其数据源和报文构造如图 2 所示，报文接收和作用于码率控制模块如图 3 所示。

在数据接收端，RTP 报文从 Network 线程到达 Worker 线程，经过 Call、VideoReceiveStream 到达 RtpStreamReceiver 对象。在该对象中，主要执行三项任务：1)接收端码率估计；2) 转发 RTP 数据包到 VCM 模块；3)接收端数据统计。其中 1)是下一节的重点，2)是 RTP 数据包进一步组帧和解码的地方；3)是统计 RTP 数据包接收信息，作为 RTCP RR 报文和其他数据统计模块的数据来源，是我们本节重点分析的部分。

在 RtpStreamReceiver 对象中，RTP 数据包经过解析得到头部信息，作为输入参

数调用 ReceiveStatistianImpl::IncomingPacket()。该函数中分别调用 UpdateCounters()和 NotifyRtpCallback()，前者用来更新对象内部的统计信息，如接收数据包计数等，后者用来更新 RTP 回调对象的统计信息，该信息用来作为 getStats 调用的数据源。

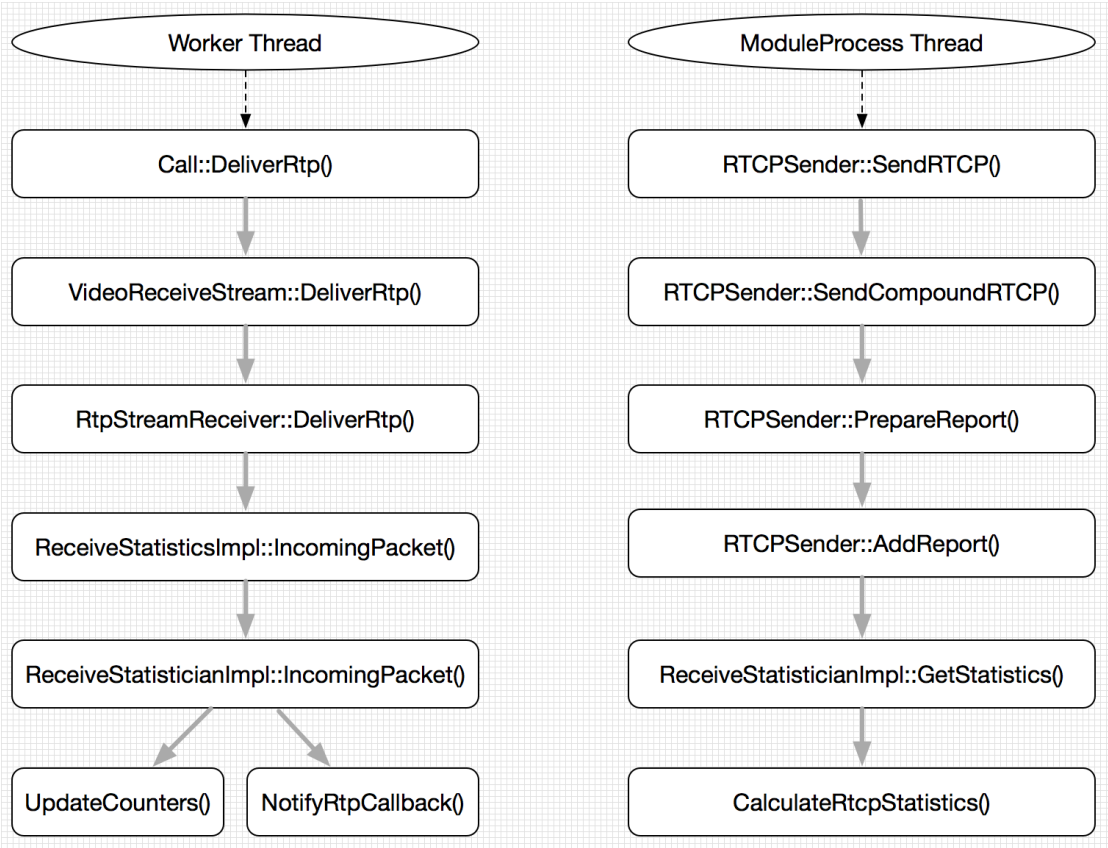

图 2 RTCP RR 报文数据源及报文构造

RTCP 发送模块在 ModuleProcess 线程中工作，RTCP 报文周期性发送。当线程判断需要发送 RTCP 报文时，调用 SendRTCP()进行发送。接下来调用 PrepareReport()准备各类型 RTCP 报文的数据。对于我们关心的 RR 报文，会调用 AddReportBlock()获取数据源并构造 ReportBlock 对象：该函数首先通过 ReceiveStatistianImpl::GetStatistics()拿到类型为 RtcpStatistics 的数据

源，然后以此填充 ReportBlock 对象。GetStatistics()会调用 CalculateRtcpStatistics()计算 ReportBlock 的每一项数据，包括丢包数、接收数据包最高序列号等。ReportBlock 对象会在接下来的报文构造环节通过 BuildRR()进行序列化。RTCP 报文进行序列化之后，交给 Network 线程进行网络层发送。

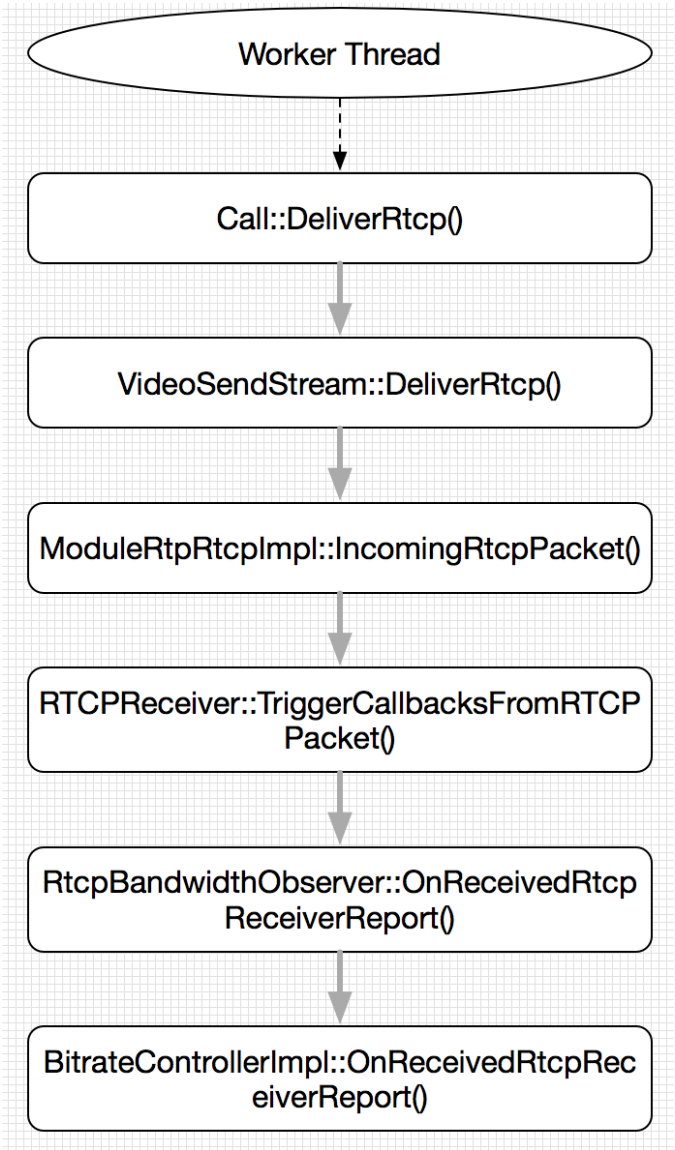

图 3 RTCP RR 报文接收及反馈

在发送端(即 RTCP 报文接收端),RTCP 报文经过 Network 线程到达 Worker 线程，

最后到达 ModuleRtpRtcpImpl::IncomingRtcpPacket()进行报文解析工作。解析完成以后，调用 TriggerCallbacksFromRTCPPackets()反馈到回调模块。在码率估计方面，会反馈到 BitrateController 模块。ReportBlock 消息最终会到达 BitrateControllerImpl 对象，进行下一步的目标码率确定。

至此，关于 RTCP RR 报文在拥塞控制中的执行流程分析完毕。

3 接收端基于延迟的码率估计

接收端基于数据包到达延迟的码率估计是整个 GCC 算法最复杂的部分，本节在分析 WebRTC 代码的基础上，阐述该部分的实现细节。

接收端基于延迟码率估计的基本思想是：RTP 数据包的到达时间延迟 m(i)反映网络拥塞状况。当延迟很小时，说明网络拥塞不严重，可以适当增大目标码率；当延迟变大时，说明网络拥塞变严重，需要减小目标码率；当延迟维持在一个低水平时，目标码率维持不变。其主要由三个模块组成：到达时间滤波器，过载检查器和速率控制器。

在实现上，WebRTC 定义该模块为远端码率估计模块 RemoteBitrateEstimator，整个模块的工作流程如图 4 所示。需要注意的是，该模块需要 RTP 报文扩展头部 abs-send-time 的支持，用以记录 RTP 数据包在发送端的绝对发送时间，详细请参考文献[4]。

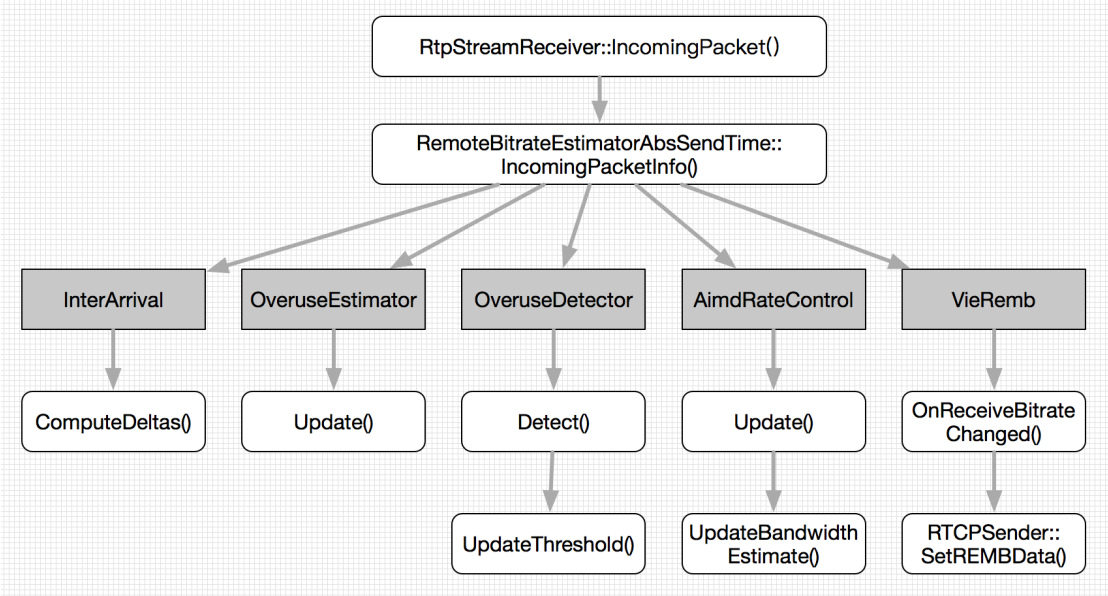

图 4 GCC 算法基于延迟的码率估计

接收端收到 RTP 数据包后，经过一些列调用到 RtpStreamReceiver 对象，由该对象调用远端码率估计模块的总控对象 RemoteBitrateEstimatorAbsSendTime，由该对象的 IncomingPacketInfo()函数负责整个码率估计流程，如图 4 所示，算法从左到右依次调用子对象的功能函数。

总控函数首先调用 InterArrival::ComputeDeltas()函数，用以计算相邻数据包组的到达时间相对延迟，该部分对应文章[1]的 3.1 节内容。在计算到达时间相对延迟时，用到了 RTP 报文头部扩展 abs-send-time。另外，实现细节上要注意数据包组的划分，以及对乱序和突发时间的处理。

接下来算法调用 OveruseEstimator::Update()函数，用以估计数据包的网络延迟，该部分对应文章[1]的 3.2 节内容。对网络延迟的估计用到了 Kalman 滤波，算法的具体细节请参考文章[2]。Kalman 滤波的结果为网络延迟 m(i)，作为下一阶段网络状态检测的输入参数。

算法接着调用 OveruseDetector::Detect()，用来检测当前网络的拥塞状况，该部分对应文章[1]的 3.2 节内容。网络状态检测用当前网络延迟 m(i)和阈值 gamma_1 进行比较，判断出 overuse，underuse 和 normal 三种网络状态之一。在算法细节上，要注意 overuse 的判定相对复杂一些：当 m(i) > gamma_1 时，计算处于当前状态的持续时间 t(ou)，如果 t(ou) > gamma_2，并且 m(i) > m(i-1)，则发出网络过载信号 Overuse。如果 m(i)小于 m(i-1)，即使高于阀值 gamma_1 也不需要发出过载信号。在判定网络拥塞状态之后，还要调用 UpdateThreshold()更新阈值 gamma_1。

算法接着调用 AimdRateControl::Update()和 UpdateBandwidthEstimate()函数，用以估计当前网络状态下的目标码率 Ar，该部分对应文章[1]的 3.3 节。算法基于当前网络状态和码率变化趋势有限状态机，采用 AIMD(Additive Increase Multiplicative Decrease)方法计算目标码率，具体计算公式请参考文章[1]。需要注意的是，当算法处于开始阶段时，会采用 Multiplicative Increase 方法快速增加码率，以加快码率估计速度。

此时，我们已经拿到接收端估计的目标码率 Ar。接下来以 Ar 为参数调用 VieRemb::OnReceiveBitrateChange()函数，发送 REMB 报文到发送端。REMB 报文会推送到 RTCP 模块，并设置 REMB 报文发送时间为立即发送。关于 REMB 报文接下来的发送和接收流程，和第 1 节描述的 RTCP 报文一般处理流程是一样的，即经过序列化发送到网络，然后发送端收到以后，反序列化出描述结构，最后通过回调函数到达发送端码率控制模块 BitrateControllerImpl。

至此，接收端基于延迟的码率估计过程描述完毕。

4 发送端码率计算及生效

在发送端，目标码率计算和生效是异步进行的，即 Worker 线程从 RTCP 接收模块经回调函数拿到丢包率和 REMB 码率之后，计算得到目标码率 A；然后 ModuleProcess 线程异步把目标码率 A 生效到目标模块如 PacedSender 和 ViEEncoder 等。下面分别描述码率计算和生效过程。

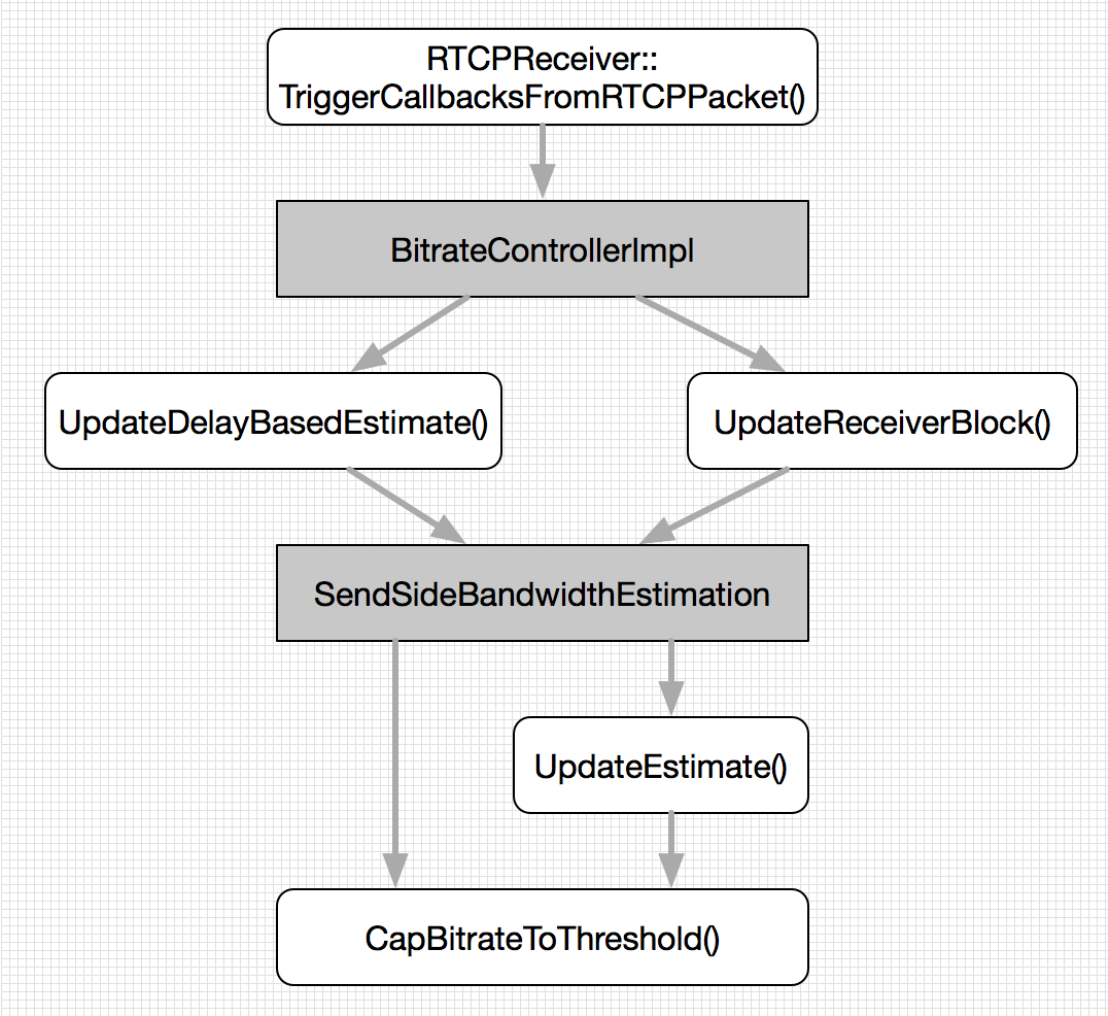

图 5 发送端码率计算过程

码率计算过程如图 5 所示：Worker 线程从 RTCPReceiver 模块经过回调函数拿到 RTCP RR 报文和 REMB 报文的数据，到达 BitrateController 模块。RR 报文中的

丢包率会进入 SendSideBandwidthEstimation::Update()函数中计算码率，然后进入 CapBitrateToThreshold()进行最终目标码率的确定，码率计算公式如文章[1]第 2 节所述。而 REMB 报文的接收端估计码率 Ar 则直接进入 CapBitrateToThreshold()函数参与目标码率的确定。目标码率由文章[1]的 3.4 节所示公式进行确定。需要注意的是，RR 报文和 REMB 报文一般不在同一个 RTCP 报文里。

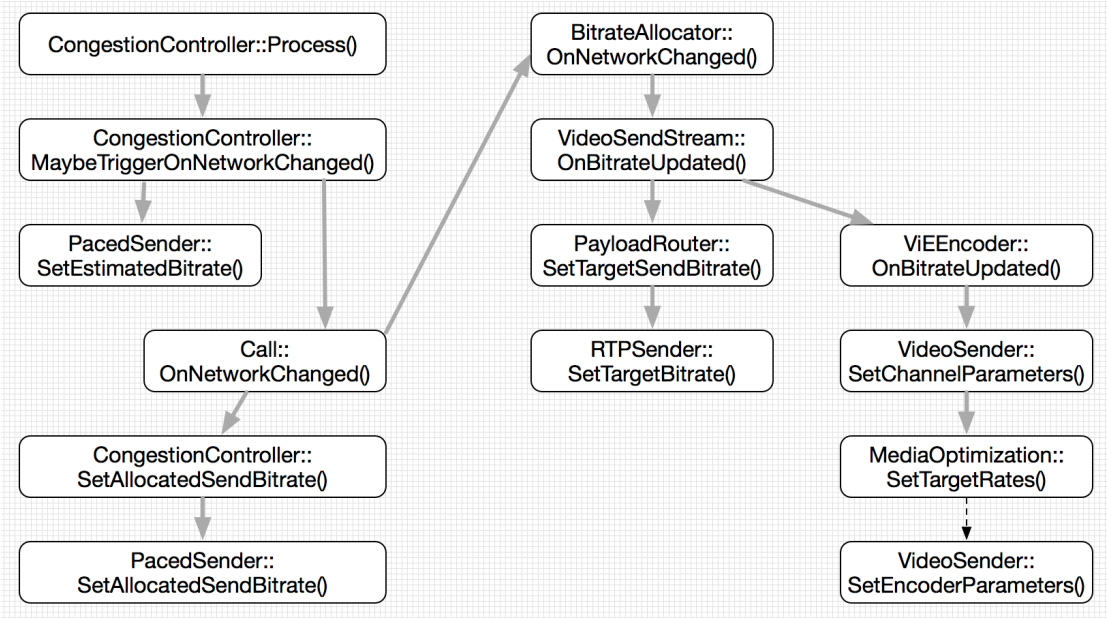

图 6 发送端码率生效过程

发送端码率生效过程如图 6 所示：ModuleProcess 线程调用拥塞控制总控对象 CongestionController 周期性从码率控制模块 BitrateControllerImpl 中获取当前最新目标码率 A，然后判断目标码率和之前相比是否发生变化。如果变化，则把这一变化设置到相关模块中，主要包括 PacedSender 模块，RTPSender 模块和 ViEEncoder 模块。

对于 PacedSender 模块，设置码率主要是为了平滑 RTP 数据包的发送速率，尽量

避免数据包 Burst 造成码率波动。对于 RTPSender 模块，设置码率是为了给 NACK 模块预留码率，如果预留码率过小，则在某些情况下对于 NACK 报文请求选择不响应。对于 ViEEncoder 模块，设置码率有两个用途：1)控制发送端丢帧策略，根据设定码率和漏桶算法决定是否丢弃当前帧。2)控制解码器内部码率控制，设定码率作为参数传输到解码器内部，参与内部码率控制过程。

至此，发送端码率计算和生效过程分析完毕。

5 总结

本文结合文章[1]，深入 WebRTC 代码内部，详细分析了 WebRTC 的 GCC 算法的实现细节。通过本文，对 WebRTC 的代码结构和拥塞控制实现细节有了更深层次的理解，为进一步学习 WebRTC 奠定良好基础。

参考文献

[1] WebRTC 基于 GCC 的拥塞控制(上) - 算法分析

http://www.jianshu.com/p/0f7ee0e0b3be

[2] WebRTC 视频接收缓冲区基于 KalmanFilter 的延迟模型.

http://www.jianshu.com/p/bb34995c549a

[3] WebRTC 中 RTP/RTCP 协议实现分析

http://www.jianshu.com/p/c84be6f3ddf3

[4] abs-send-time. https://webrtc.org/experiments/rtp-hdrext/abs-send-time/