WebRTC 中 QoS 实现初探

一 理论基础

算法：Google Congestion Control

RFC draft: A Google Congestion Control Algorithm for Real-Time Communication

Paper: Experimental Investigation of the Google Congestion Control for Real-Time Flows

Paper: Understanding the Dynamic Behavior of Google Congestion Control for WebRTC

RFCs：

rfc3550: RTP - A Transport Protocol for Real-Time Applications

rtc3611: RTP Control Protocol Extended Reports (RTCP XR)

rfc4585: Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)

Rfc5104: Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)

二 GCC 算法流程

2.1 GCC 总体流程

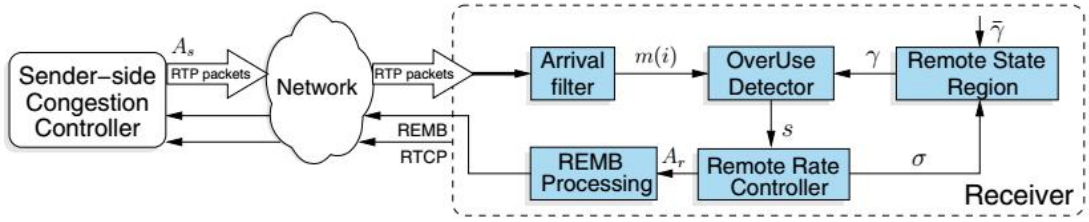


图 1 GCC 算法总体流程

2.2 GCC 算法细节

文档：矛与盾-互联网音视频通信的抗丢包与带宽自适应

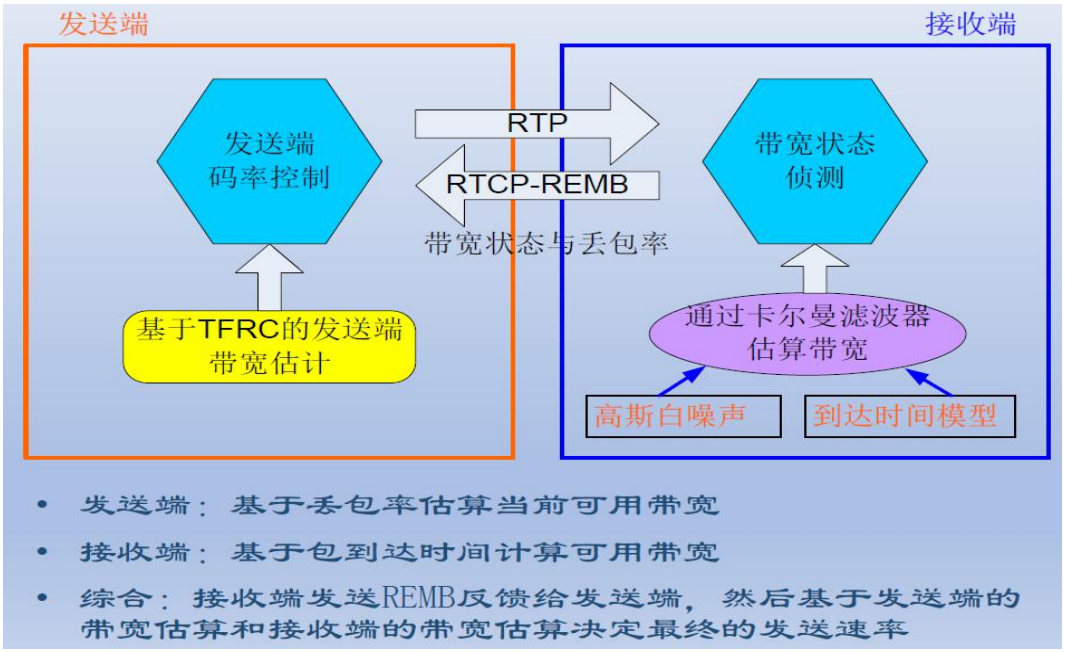


图 2 GCC 的带宽自适应模型

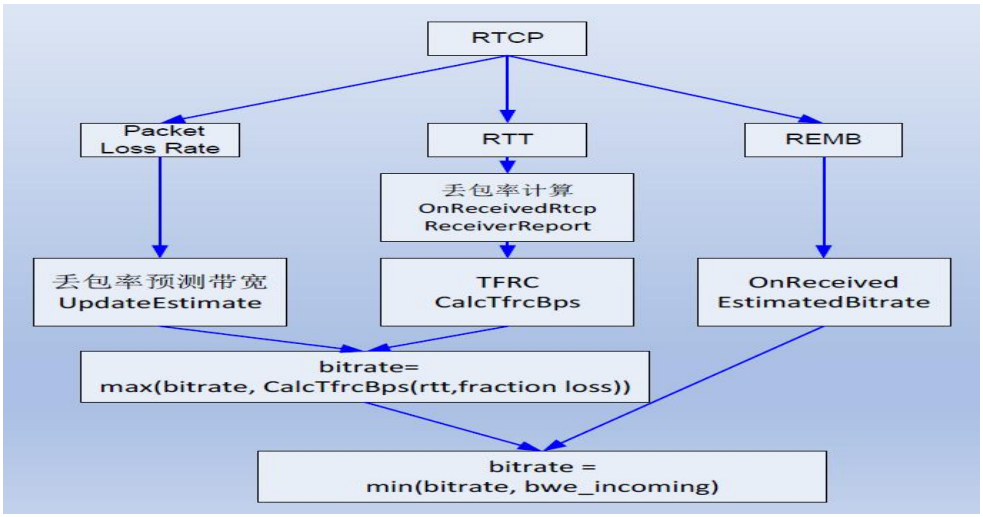


图 3 发送端流程

发送端：

1. 基于丢包带宽估算，丢包信息来自RTCP RR(Receiver Report)

$$A_s(t_k)=\begin{cases}A_s(t_{k-1})(1-0.5f_l(t_k)) & f_l(t_k)>0.1\\ 1.05(A_s(t_{k-1})+1\text{kbps}) & f_l(t_k)<0.02\\ A_s(t_{k-1}) & \text{otherwise}\end{cases}$$

2. 计算TCP-Friendly Send Rate

$$A_{TF}=\frac{8\times s}{R\times\sqrt{2bp/3}+RTO\times 3\times\sqrt{3bp/8}\times p\times(1+32p^2)}$$

3. 根据接收端发送的REMB包确定接收端的接收速率

$$BW_{in}=BitrateFromREMB$$

4. 确定发送速率As

$$A_s\le\min(BW_{in},Max\_Bitrate\_Configured)$$

$$A_s\ge\max(Min\_Bitrate\_Configured,A_{TF})$$

图 4 发送端带宽计算

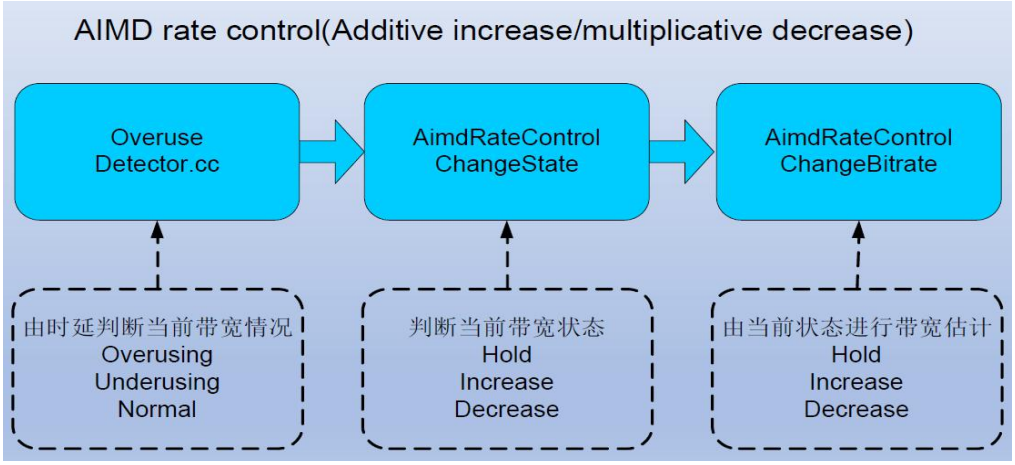
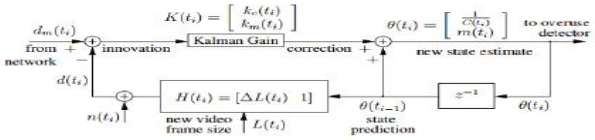


图 5 接收端带宽估计

接收端：

1. Arrival Filter

$$d_m(t_i)=t_i-t_{i-1}-(T_i-T_{i-1})$$

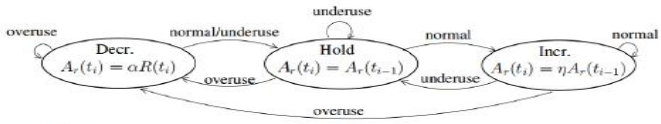


2. Overuse Detector

产生驱动信号控制Rate Controller模块的状态机转移

$$s=\begin{cases}overuse, m(t_i)>\gamma\\ underuse, m(t_i)<\gamma\end{cases}$$

3. Rate Controller



4. REMB Processing

- 将估算的Ar通过REMB发送至发送端，间隔为1s
- 当Ar下降超过0.03，立刻发送REMB

图 6 接收端带宽计算

三 WebRTC 中 GCC 算法的实现(以 videodata 数据流为例)

3.1 功能模块

WebRTC 中关于 QoS 实现的代码分布在若干模块中，各模块功能及相互关系记录如下：

call：表示一个会话实例，包含多个 A/V stream，开启两个线程 module_process、pacer_thread，集成 BitrateAllocator、CongestionController 对象，继承实现 PackerReceiver 和 BitrateObserver 接口。功能上：1）接收 RTP 数据包，交给底层 recv stream 处理；2）接受来 RTCP 数据包，交给底层 recvstream 处理; 3）处理和 QoS 相关的 REMB 报文和 ReceiverReport 报文，则在底层解析之后通过 observer 回调到 RtcpBandwidthObserver 对象，该对象通过发送端带宽控制算法计算出最新带宽，把结果反馈到 VideoCodecModule 模块和 PacerSend 模块，进而控制 video codec 码率和 smooth send 发送速率。

BitrateController/BitrateAllocator：实现 GCC 基于丢包率的发送端带宽控制算法。call 对象把接收到的 Rtcp 报文向下交给 RtcpReceiver，RtcpReceiver 首先对报文内容进行解析，然后触发关心 rtcp 报文内容的回调函数，对于 remb/tmmbr/rr 报文，其回调对象是 RtcpBandwidthObserver，该对象的 OnReceivedEstimatedBitrate()接口用于处理 remb/tmmbr 报文内容，OnReceivedRtcpReceiverReport()接口用于处理 rr 报文内容。BitrateController 模块根据丢包率重新计算出当前带宽，并通过调用 MaybeTriggerOnNetworkChanged()接口向其观察者(也就是 call 对象)发送 OnNetworkChanged()消息。Call 对象在收到 OnNetworkChanged()消息之后，分别通知 VCM 和 PacedSender 执行码率控制策略。

Pacing：pacedsender 线程实现，起到平滑网络发送速率的作用。Media data 在经过 codec 编码之后，并在发送到 network 之前，经过该 pacedsender 线程控制速率。Pacedsender 线程接收来自 Call 对象的最新估计带宽控制，然后据此控制 RTP 数据包发送到网络的速率。

RemoteBitrateEstimator：接收端带宽控制算法实现。基于 AIMD 算法控制带宽。Call 接收到 RTP 数据包后，在 ViEReceiver 中首先会将 RTP 数据包信息交给 RemoteBitrateEstimator 进行带宽估计，然后把 RTP 数据包转到 VCM 进行解码操作，最后更新接收端统计信息，为发送 ReceiverReport 报文更新统计数据（如 fractionLoss 等）。RemoteBitrateEstimator 基于数据包的延时进行码率估计，判断当前带宽的使用状况。整个工作过程在一个有限状态机中进行，如图 7 所示。

经过 RemoteBitrateEstimator 估计后会发送 REMB 报文（如果达到发送条件），而经过接收端数据统计之后，则由 RTPRTCP 模块定期发送 ReceiverReport 报文。

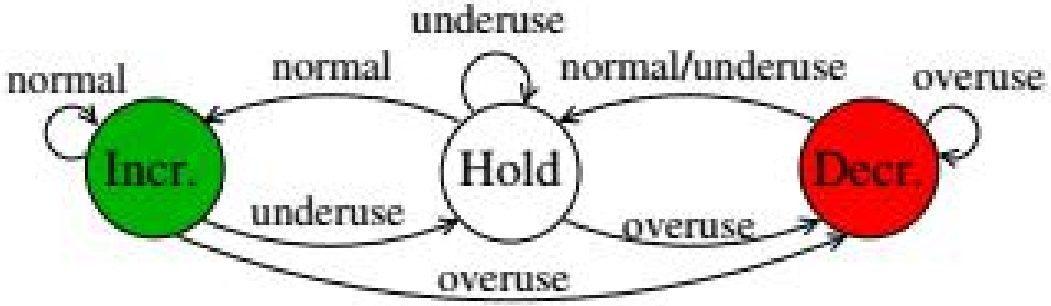


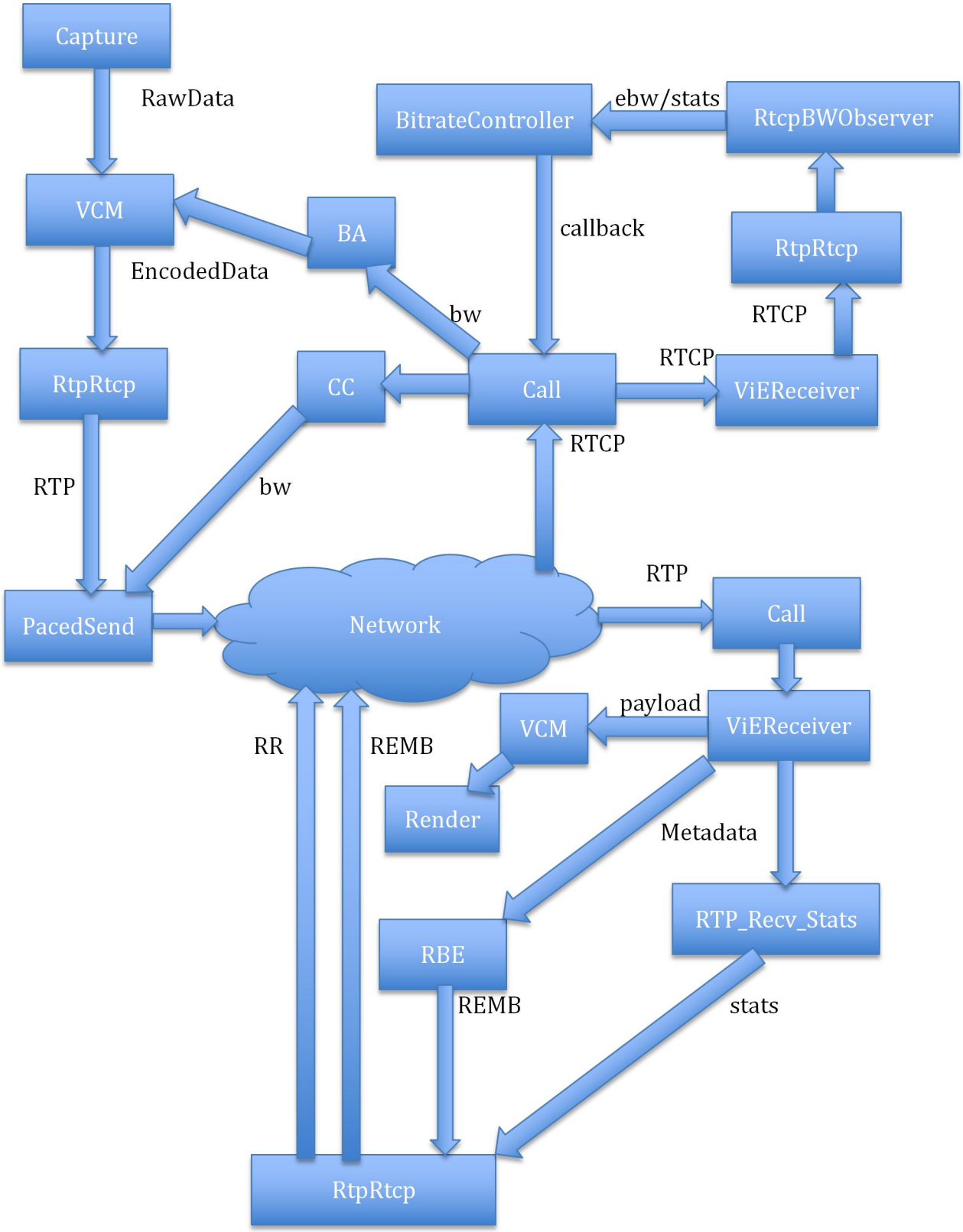
图 7 接收端带宽估计有限状态机

CongestionController：拥塞控制算法的总控模块，同时兼顾发送端的码率控制 BitrateController 和接收端的码率估计 RemoteBitrateEstimator。在发送端，更新后的码率会经过 CongestionController 反馈到 PacedSender 线程。作为模块，CongestionController 在函数 Process()中驱动 BitrateController 和 RemoteBitrateEsitimator 模块的 Process()函数的执行。

RTP_RTCP：该模块 RTP/RTCP 报文的构建、发送、接收、解析以及上层回调等功能。Media data 在 codec 编码之后，会经过该模块打包成 RTP 模块发送到网络；接收端在收到 RTP 数据包之后，会经过该模块解包后发送到 codec 进行解码工作；并且同事进行 QoS 的带宽估计和数据统计等工作。RTRTCP 模块的 Process()函数会定期（每秒）发送 RTCP 报文到接收端。接收端在收到 RTCP 报文后会由 RTPRTCP 模块解包获得 rtcp 信息，并由回调函数交由上层处理。

3.2 总体流程图

如下页流程图所示：


Capture
RawData
VCM
EncodedData
RtpRtcp
RTP
PacedSend
BitrateController
ebw/stats
RtcpBWObserver
BA
callback
RtpRtcp
bw
RTCP
CC
Call
RTCP
ViEReceiver
RTCP
bw
RTP
Network
RTP
Call
payload
VCM
ViEReceiver
RR
REMB
Render
Metadata
RTP_Recv_Stats
RBE
REMB
stats
RtpRtcp

四 总结

本文结合 Google Congestion Control 算法和相关 RFC 文档，对 WebRTC 中 QoS 的实现代码进行了分析：初步理清了发送端基于丢包率和对端估计码率的带宽控制策略和接收端基于数据包到达时间和丢包率的带宽估计策略；对整个基于 RTCP 报文的控制回路有一个较为清晰的认识。但是目前对于 GCC 算法细节的代码实现还需要进一步深入研究。