音视频通话QoS(服务质量)剖析-系列文章

原文链接：https://blog.csdn.net/CrystalShaw/article/details/80432267

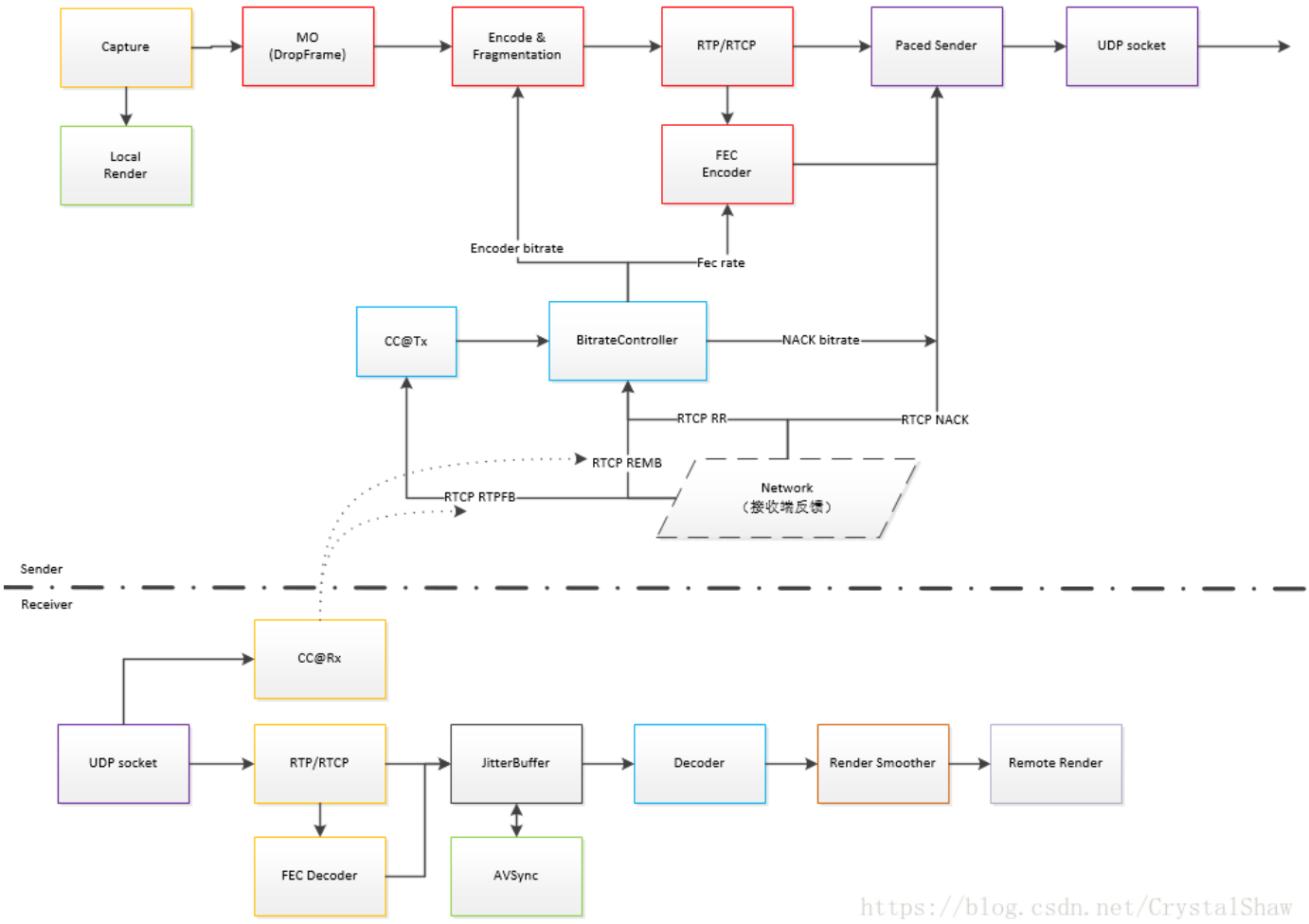
评注：该文档系列对应的源码不是最新版本的，仅供原理参考用。

系列文章见每小节对应的链接。

目前总结出webrtc用于提升QOS的方法有：

NACK、FEC、SVC、JitterBuffer、IDR Request、PACER、Sender Side BWE、VFR（动态帧率调整策略）。

这几种方法在webrtc架构分布如下：



具体实现原理如下：

一、NACK

与NACK对应的是ACK，ACK是到达通知技术。以TCP为例，他可靠因为接收方在收到数据后会给发送方返回一个“已收到数据”的消息（ACK），告诉发送方“我已经收到了”，确保消息的可靠。NACK也是一种通知技术，只是触发通知的条件刚好的ACK相反，在未收到消息时，通知发送方“我未收到消息”，即通知未达。

NACK是在接收端检测到数据丢包后，发送NACK报文到发送端；发送端根据NACK报文中的序列号，在发送缓冲区找到对应的数据包，重新发送到接收端。NACK需要发送端发送缓冲区的支持，RFC5104定义NACK数据包的格式。若在JB缓冲时间内接收端收到发送端重传的报文，就可以解决丢包问题。对应上图发送端的RTCP RTPFB

具体请参考https://www.yuque.com/docs/share/175eb5a6-c3e9-4fc9-896f-5881695f16ee?#《webrtc QOS方法一（NACK实现）》

二、FEC

FEC是发送端在发送报文的时候，将之前的旧包也打包到新包里面，若接收端有丢包，就用新包里面冗余的旧包恢复数据。

webrtc实现该冗余功能，有三种方式：

1、RED就是RFC2198冗余。将前面的报文直接打入到新包里面，在接收端解析主包和冗余包。

2、ULPFEC，目前webrtc仅将SVC编码的Level 0视频帧打包成FEC。其余层有丢包，就逐步将帧率，保证视频相对流畅。用到的协议是：RFC5109。

3、FLEXFEC较ULPFEC，增加纵向OXR运算。增加网络抗丢包能力。

具体请参考https://www.yuque.com/docs/share/66083040-2753-4c95-bdee-e4547371bcf3?#《webrtc QOS方法二.1（FEC原理）》

三、SVC

webrtc的VPX用到的是时间可适性（Temporal Scalability）算法。通过改变一个GOP内帧的线性参考关系。防止网络丢包对视频传输造成的影响。

具体请参考：https://www.yuque.com/docs/share/4e646f6d-355b-4977-8c25-d292d82b811c?#《webrtc QOS方法三（SVC实现）》

webrtc使用NACK+FEC+SVC作为QOS的解决方案。参考链接：
https://ieeexplore.ieee.org/document/6738383/

四、JitterBuffer

JitterBuffer实现原理是，在收到网络上的RTP报文后，不直接进行解码，需要缓存一定个数的RTP报文，按照时间戳或者seq的顺序进行重排，消除报文的乱序和抖动问题。JitterBuffer分动态JitterBuffer和静态JitterBuffer两种模式。静态JitterBuffer缓存报文个数固定。动态JitterBuffer是根据网络环路延时的情况，动态调整缓存报文个数。

具体请参考https://www.yuque.com/docs/share/116cf6e0-4c91-4477-98f6-383470ce6adb?#《webrtc QOS方法四（JitterBuffer）》

五、IDR Request

关键帧也叫做即时刷新帧，简称IDR帧。对视频来说，IDR帧的解码无需参考之前的帧，因此在丢包严重时可以通过发送关键帧请求进行画面的恢复。关键帧的请求方式分为三种：RTCP FIR反馈（Full intra frame request）、RTCP PLI 反馈（Picture Loss Indictor）或SIP Info消息，具体使用哪种可通过协商确定。

具体请参考：https://www.yuque.com/docs/share/82dd5bfa-8ee7-4232-95ac-08359e5d4a77?# 《webrtc QOS方法五（IDR request）》

六、PACER

PACER，是网络报文平滑策略。一个视频帧有可能分别封装在几个RTP报文，若这个视频帧的RTP报文一起发送到网络上，必然会导致网络瞬间拥塞。以25fps为例，若这帧视频的RTP报文，能够在40ms之内发送给接收端，接收端既可以正常工作，也缓冲了网络拥塞的压力。PACER就是实现把RTP同一时刻生产的若干包，周期性的发送，防止上行流量激增导致拥塞。

具体请参考https://www.yuque.com/docs/share/950304b5-08de-4bbb-bc6f-0fae48d23444?# 《webrtc QOS方法六（pacer实现）》

七、Sender Side BWE或REMB（Receiver Estimated Maximum Bitrate）

这个算法的思路是根据接收端的丢包率或延时情况维护一个状态机。以根据丢包率为例，在判断为overuse时，就根据一定的系数减少当前发送端的码率值，当判断为underuse时又根据增加系数来增加发送端的码率值；然后将这个值通过rtcp包发送给发送端，发送端根据该值来动态的调整码率。

具体请参考https://www.yuque.com/docs/share/6e02d5da-7ea9-47ea-b876-b064d0c569c0?# 《webrtc QOS方法七（Sender Side BWE）》

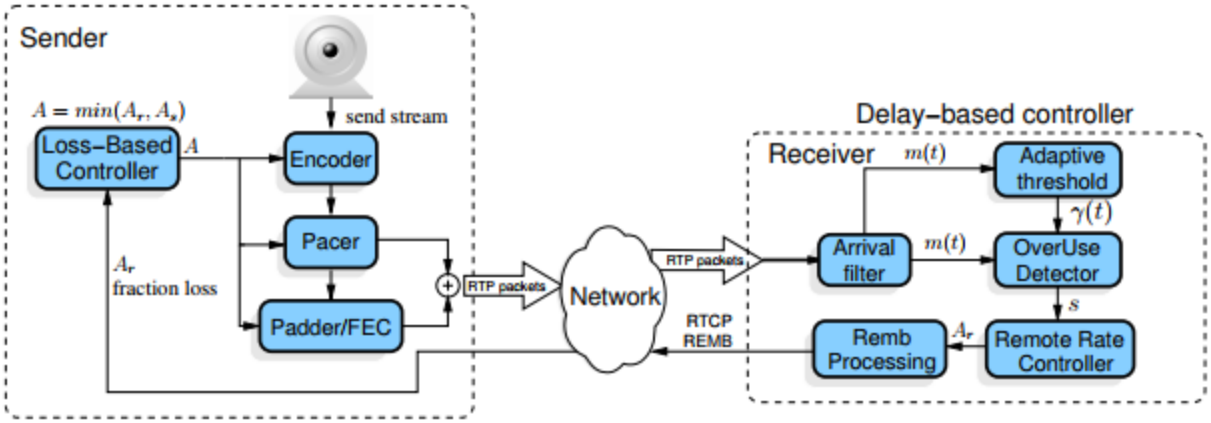


Figure 1: Google Congestion Control architecture

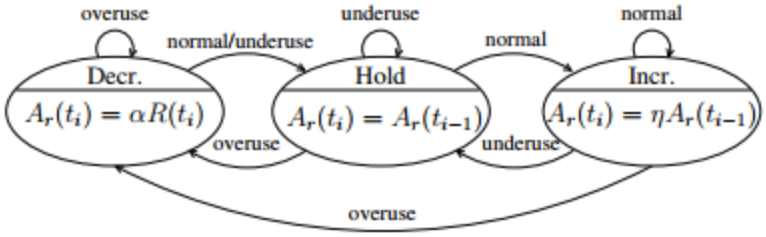


Figure 4: Remote rate controller finite state machine

八、动态帧率调整策略

视频发送端根据Sender Side BWE或REMB等参数调整出一组比较合适的码率值，当网络条件好的时候，码率值会比较大，当网络条件比较差的时候，码率值会比较低。但是若是发送端仅调整码率，不调整帧率，当网络条件比较好的时候，仅仅提升了视频质量，没有充分利用网络条件，提升实时性。当网络条件比较差的时候，码率降的比较低，若不降低帧率，视频质量会大幅度下降。所以需要增加一种机制，根据发送端的码率值，动态调整发送端的帧率值。

具体请参考https://www.yuque.com/docs/share/c43083a3-80bb-4a6a-a097-0a2b4042e409?# 《webrtc QOS方法八.1（帧率调整）》
https://www.yuque.com/docs/share/fbc528ce-2669-4fd3-9d35-9cd04bd50ca0?# 《webrtc QOS方法八.2（发送端帧率调整原理及实现流程）》

九、花屏问题

https://www.yuque.com/docs/share/6dae5ce5-68b8-4961-953b-92e844fac3f6?# 《webrtc QOS方法九（花屏问题解决方法）》

————————————————————