盾与矛：互联网音视频通信的抗丢包与带宽自适应

中国电信北京研究院 社交通信产品线

丁鹏

2015.12

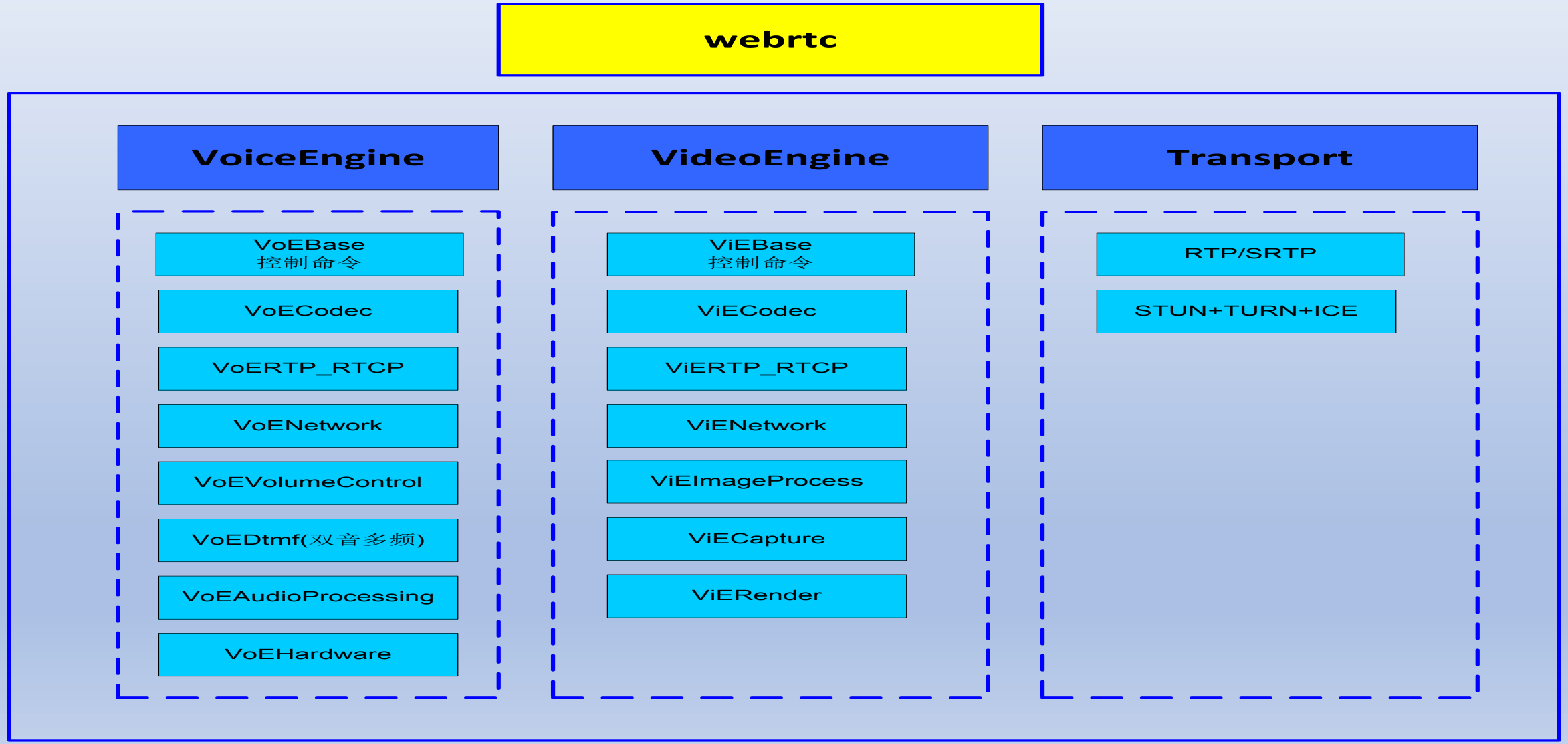
音视频通信：功能VS品质

- 功能不等于品质：在复杂网络环境中，突发流量、错误传输、超时等都会引起丢包，严重影响音视频通话品质.
- 品质如何保证？

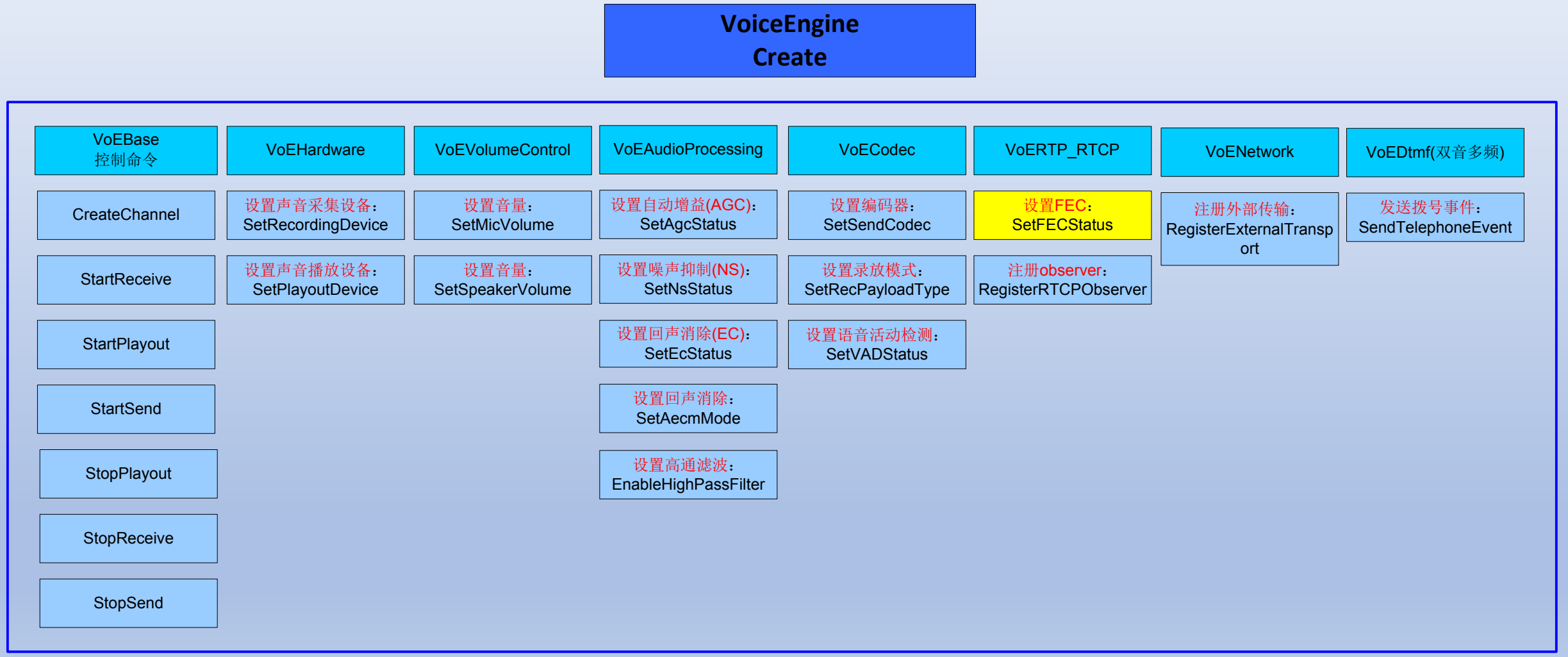
盾与矛：抗丢包与带宽自适应

- 矛寻求突破
- 盾给予保护
- 带宽自适应是矛
- FEC，NACK是盾

webrtc架构图


webrtc
VoiceEngine
VoEBase
控制命令
VoECodec
VoERTP_RTCP
VoENetwork
VoEVolumeControl
VoEDtmf(双音多频)
VoEAudioProcessing
VoEHardware
VideoEngine
ViEBase
控制命令
ViECodec
ViERTP_RTCP
ViENetwork
ViEImageProcess
ViECapture
ViERender
Transport
RTP/SRTP
STUN+TURN+ICE

VoiceEngine



VideoEngine

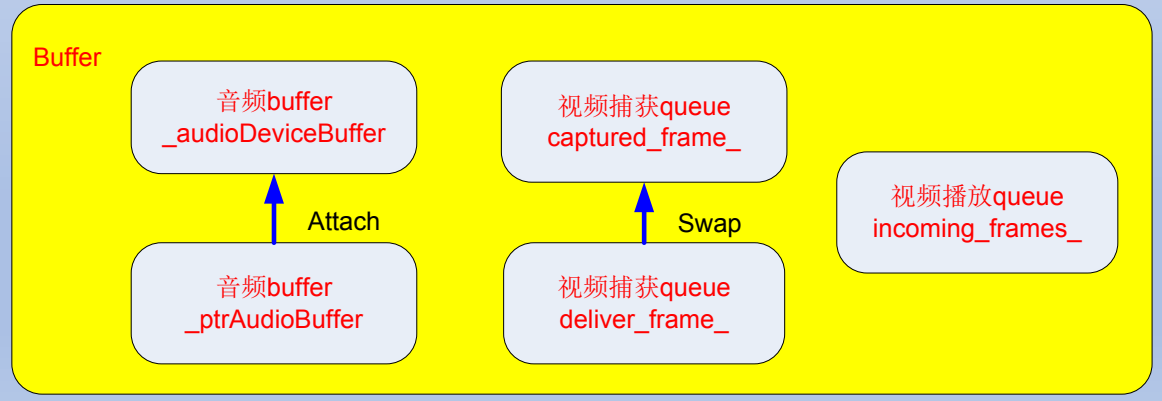
VideoEngine Create

ViEBase 控制命令	ViECapture	ViEImageProcess	ViECodec	ViERTP_RTCP	ViENetwork	ViERender
CreateChannel	分配视频设备：AllocateCaptureDevice	去闪烁：EnableDeflickering	设置编码器：SetSendCodec	设置REMB：SetRembStatus	注册外部传输：RegisterSendTransport	设置显示窗口：AddRenderer
StartReceive	连接视频设备：ConnectCaptureDevice	去噪：EnableDenoising	设置解码器：SetReceiveCodec	设置NACK：SetNACKStatus		开启渲染：StartRender
StartSend	开始捕获视频：StartCapture	色彩增强：EnableColorEnhancement	设置解码观测：RegisterDecoderObserver	设置NACK/FEC混合：SetHybridNACKFECStatus		停止渲染：StopRender
StopReceive				设置关键帧申请：SetKeyFrameRequestMethod		
StopSend						
SetVoiceEngine						
disableContentAnalysis						
SendKeyFramePerReq						

webrtc线程与数据传输

系统调度的threads

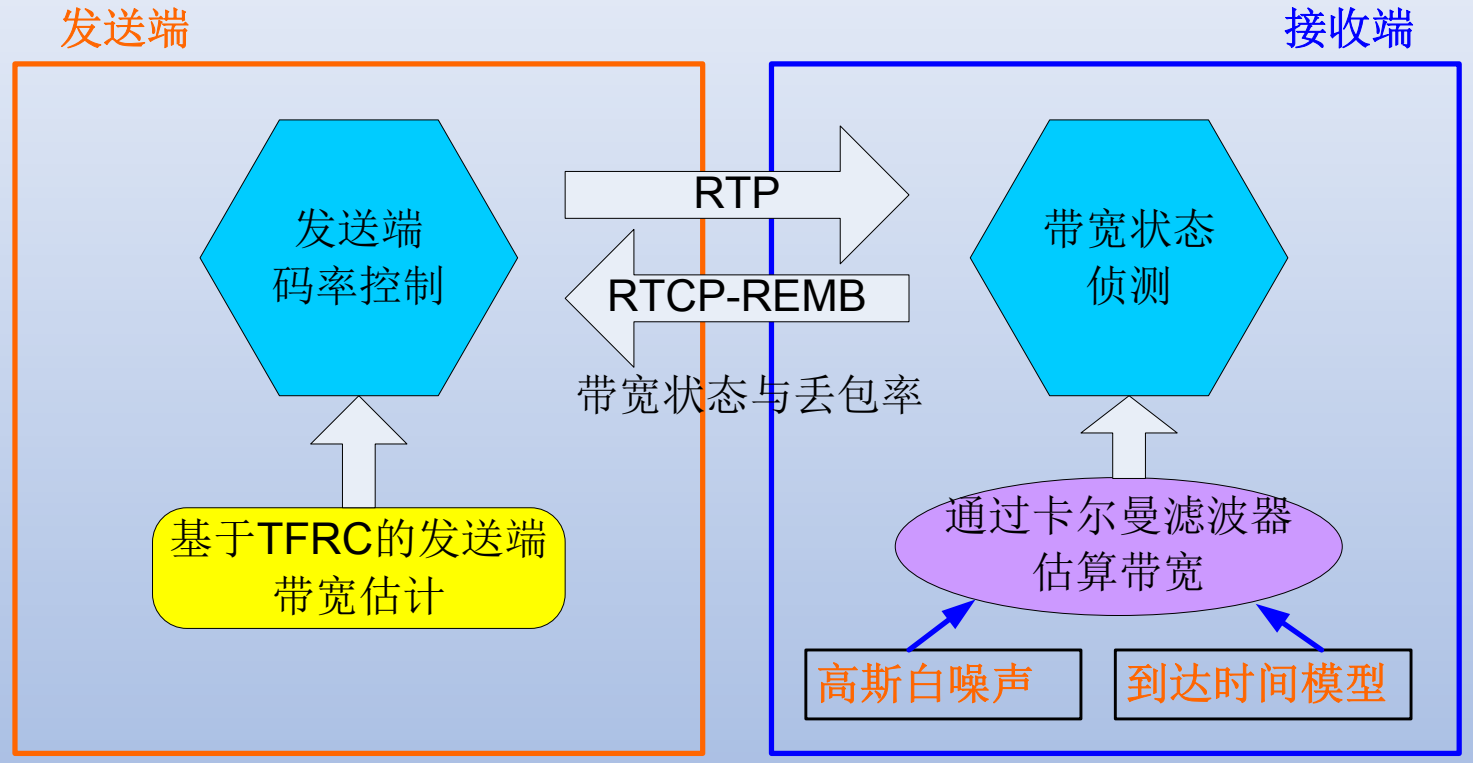
AudioDevice RecThreadProcess kRealtimePriority 音频捕获 编码一个线程实现
AudioDevice PlayThreadProcess kRealtimePriority 音频解码播放
VideoRender IncomingVideoStreamProcess kRealtimePriority 视频渲染
VideoRender JavaRenderThreadProcess kRealtimePriority 视频渲染
VideoEngine ViECaptureProcess kHighPriority 视频捕获 编码一个线程实现
VideoEngine ChannelDecodeProcess kHighestPriority 视频解码



ProcessThreadImpl的_modules记录了一些其它运行的threads，都是kNormalPriority，目的是调度这些threads不要block超过100ms

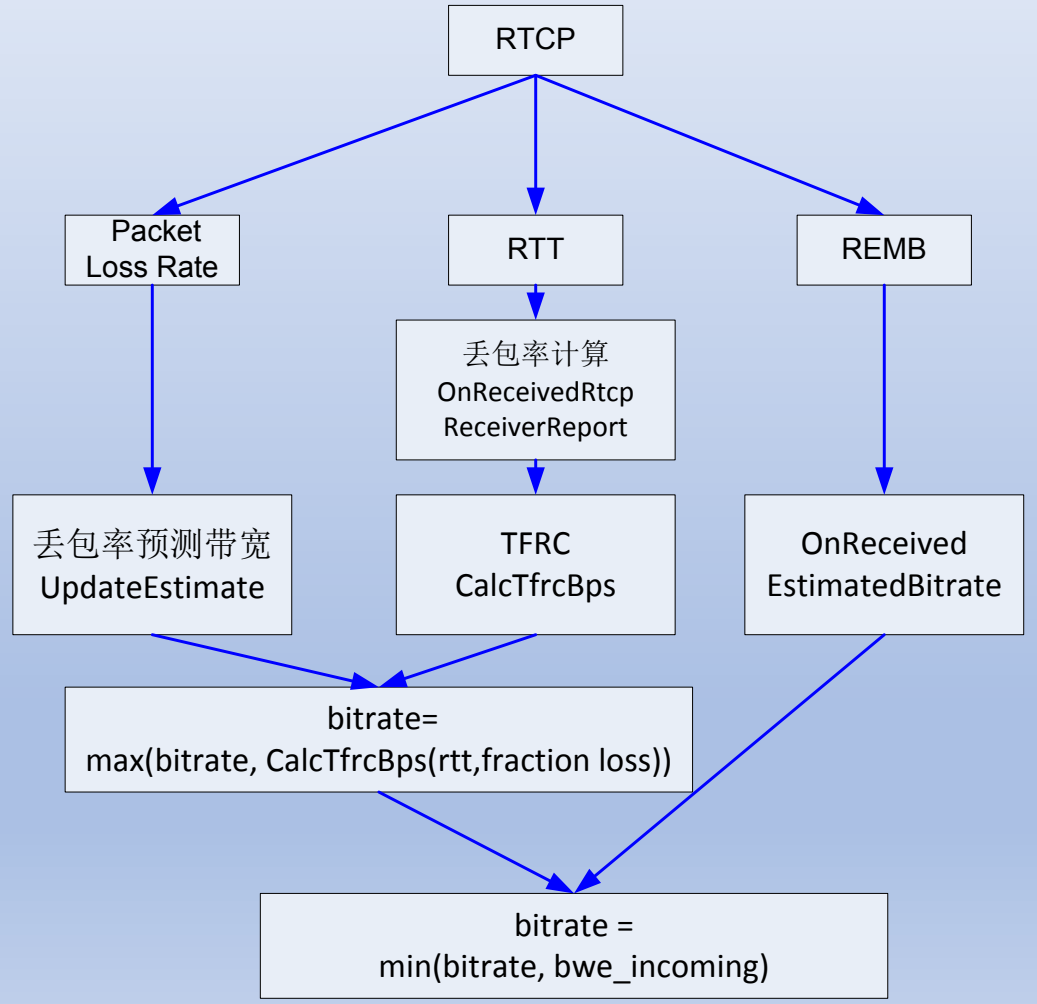
module_process_thread_.RegisterModule(default_rtp_rtcp_.get())
module_process_thread_.RegisterModule(paced_sender_.get())
module_process_thread_.RegisterModule(&vcm_) VideoCodingModule 视频编码
module_process_thread_.RegisterModule(overuse_detector_.get())
module_process_thread_.RegisterModule(capture_module_)
module_process_thread_.RegisterModule(rtp_rtcp_.get())
module_process_thread_.RegisterModule(rtp_rtcp) removed_rtp_rtcp_
module_process_thread_.RegisterModule(&vie_sync_)
_moduleProcessThreadPtr->RegisterModule(_rtpRtcpModule.get())
_processThreadPtr->RegisterModule(&_monitorModule)
process_thread_->RegisterModule(rbe_.get()) RemoteBitrateEstimator
process_thread->RegisterModule(call_stats_.get())
_shared->process_thread()->RegisterModule(_shared->audio_device())

带宽自适应模型



- 发送端：基于丢包率估算当前可用带宽
- 接收端：基于包到达时间计算可用带宽
- 综合：接收端发送REMB反馈给发送端，然后基于发送端的带宽估算和接收端的带宽估算决定最终的发送速率

带宽自适应流程图：发送端



带宽自适应数学模型：发送端

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

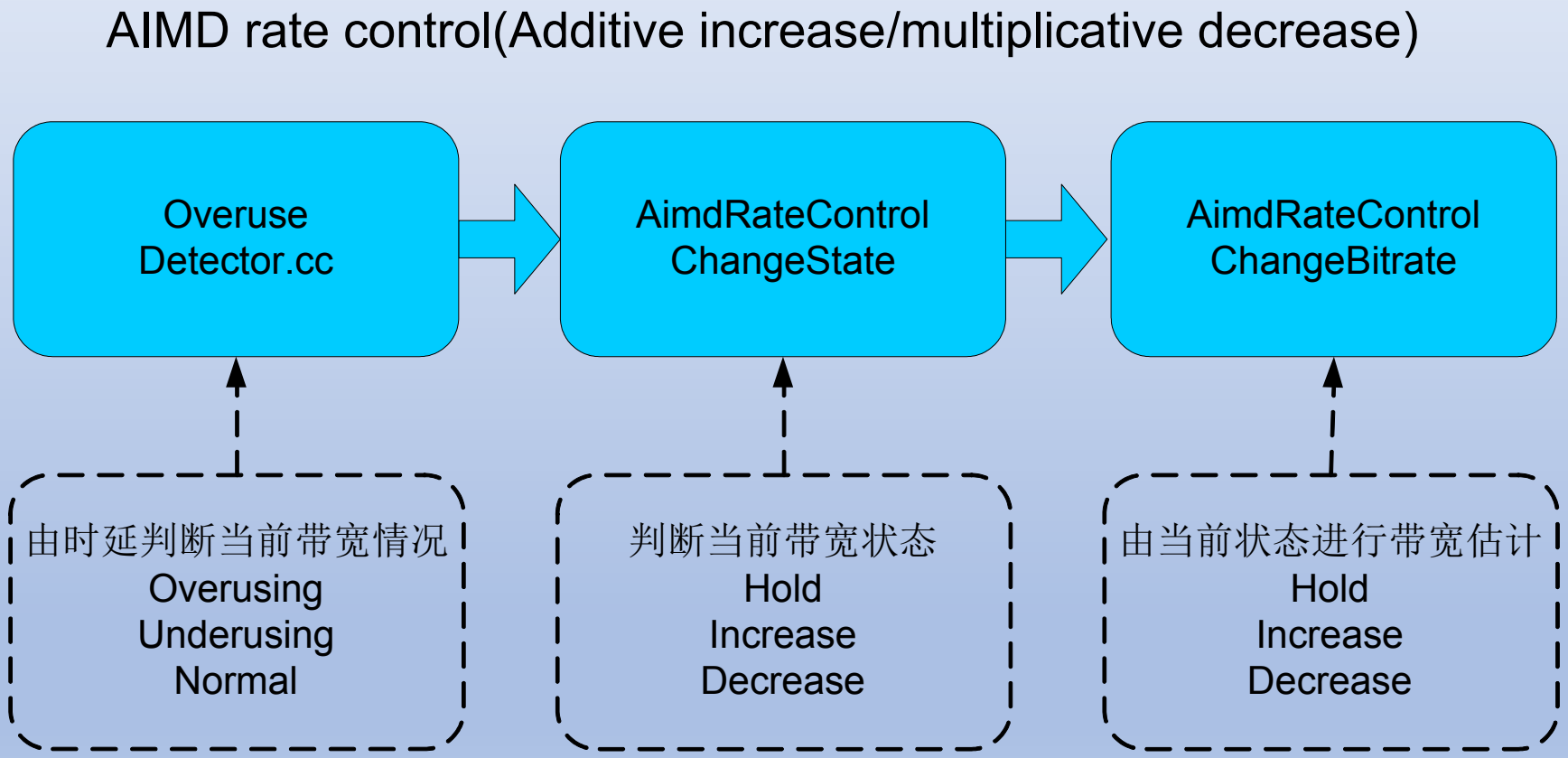
$$BW_{in}=BitrateFromREMB$$

4. 确定发送速率As

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

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

带宽自适应流程图：接收端

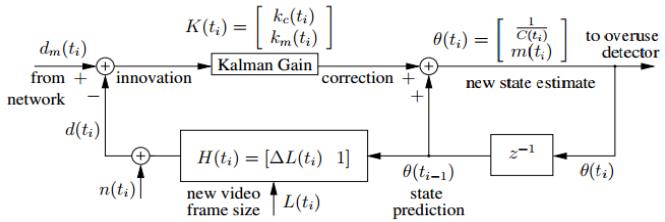


带宽自适应数学模型：接收端

接收端：

1. Arrival Filter

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

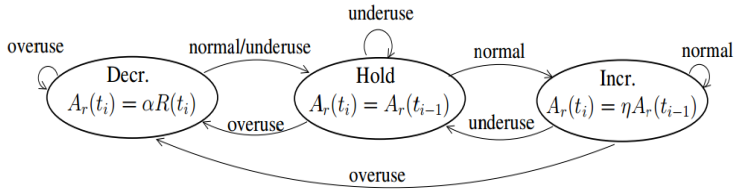


2. Overuse Detector

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

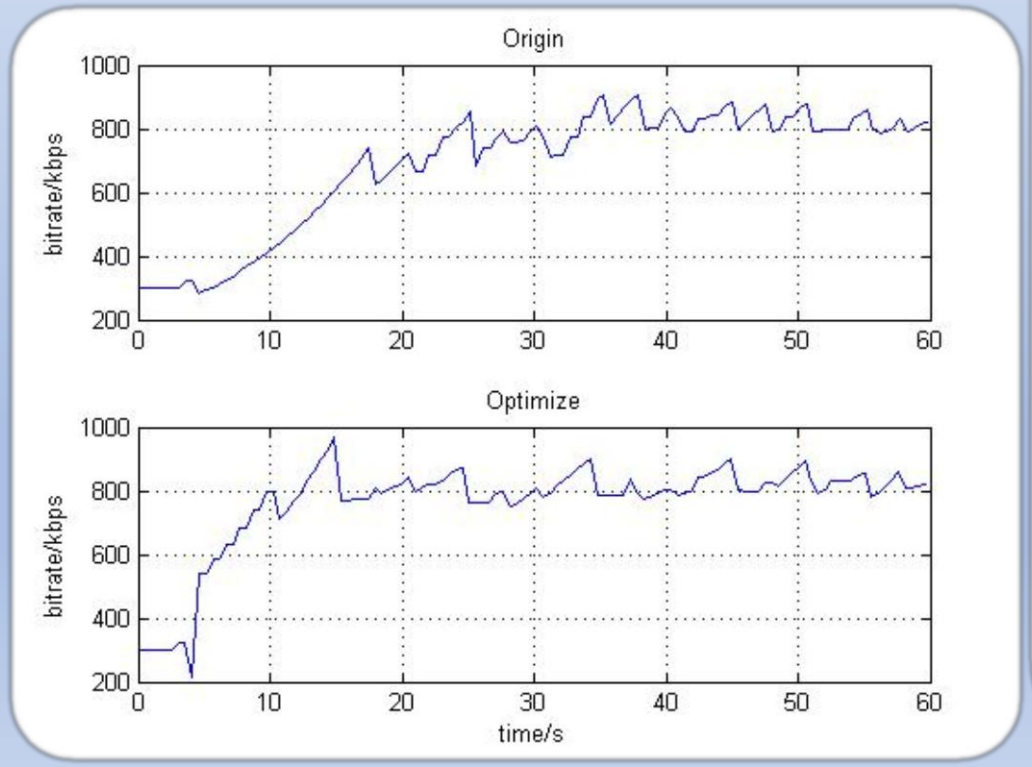


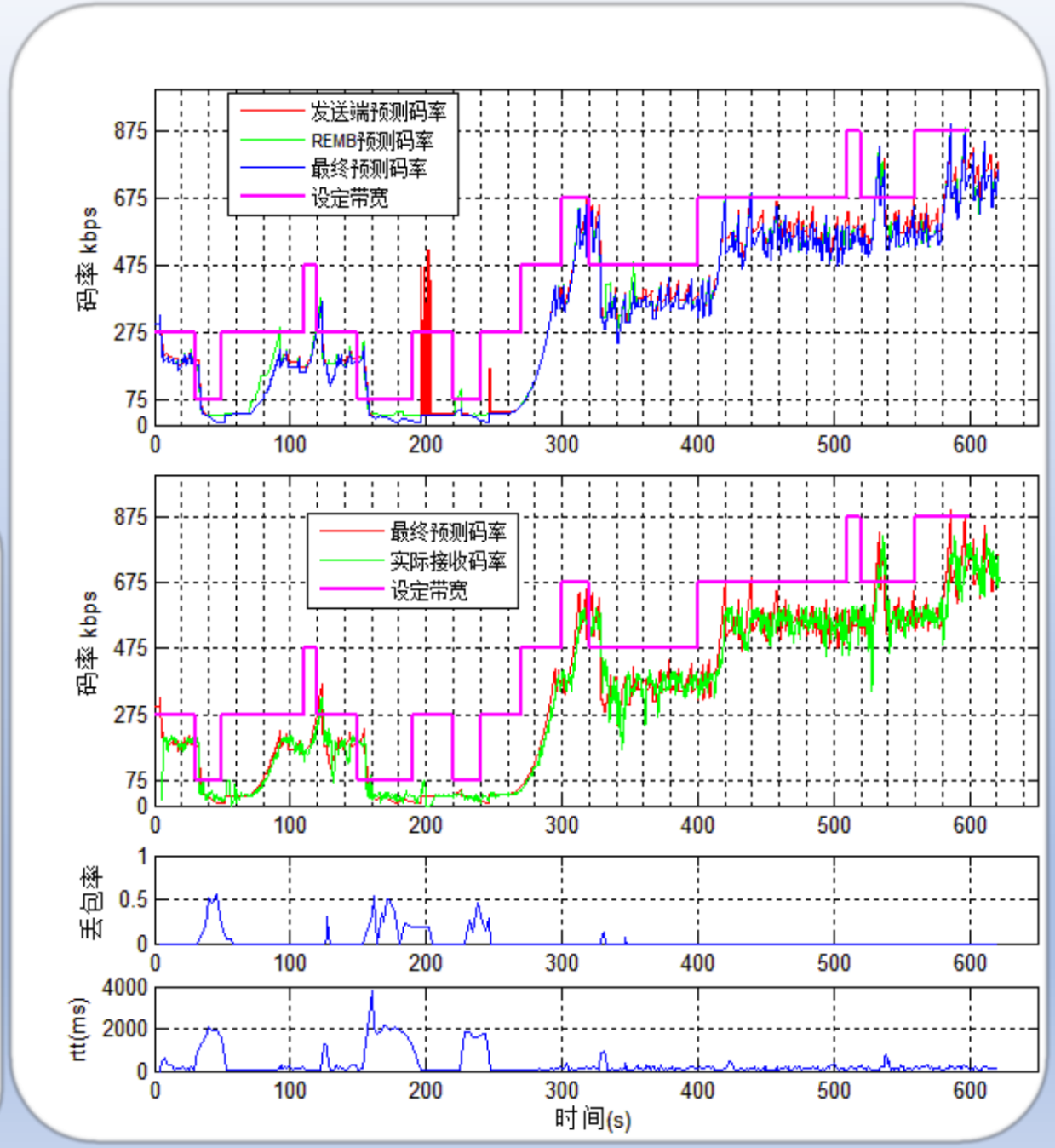
4. REMB Processing

- 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

带宽自适应优化

- 优化启动阶段码率估计
- 优化REMB状态判定门限





NACK与FEC

- NACK(Negative-Acknowledgement)：重传机制，发送端重新发送丢失的包，RFC 5104 定义了NACK流程。

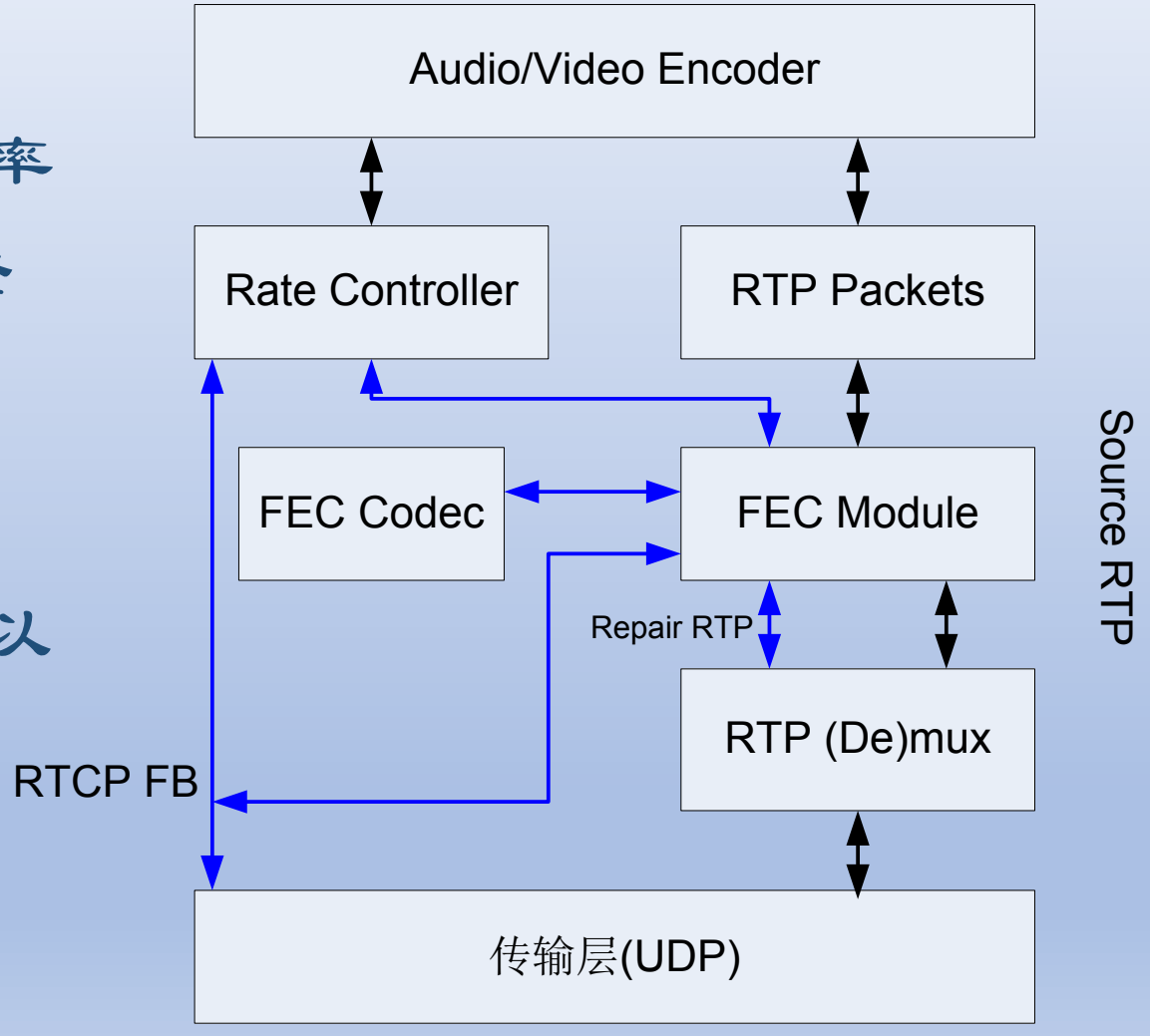
SDP:
a=rtcp-fb:100 ccm fir(full INTRA-frame request)
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli(Picture Loss Indication)

- FEC(Forward Error Correction)：前向纠错，将发送数据附加冗余纠错码，根据纠错码对数据进行纠正，RFC5109定义了FEC流程。

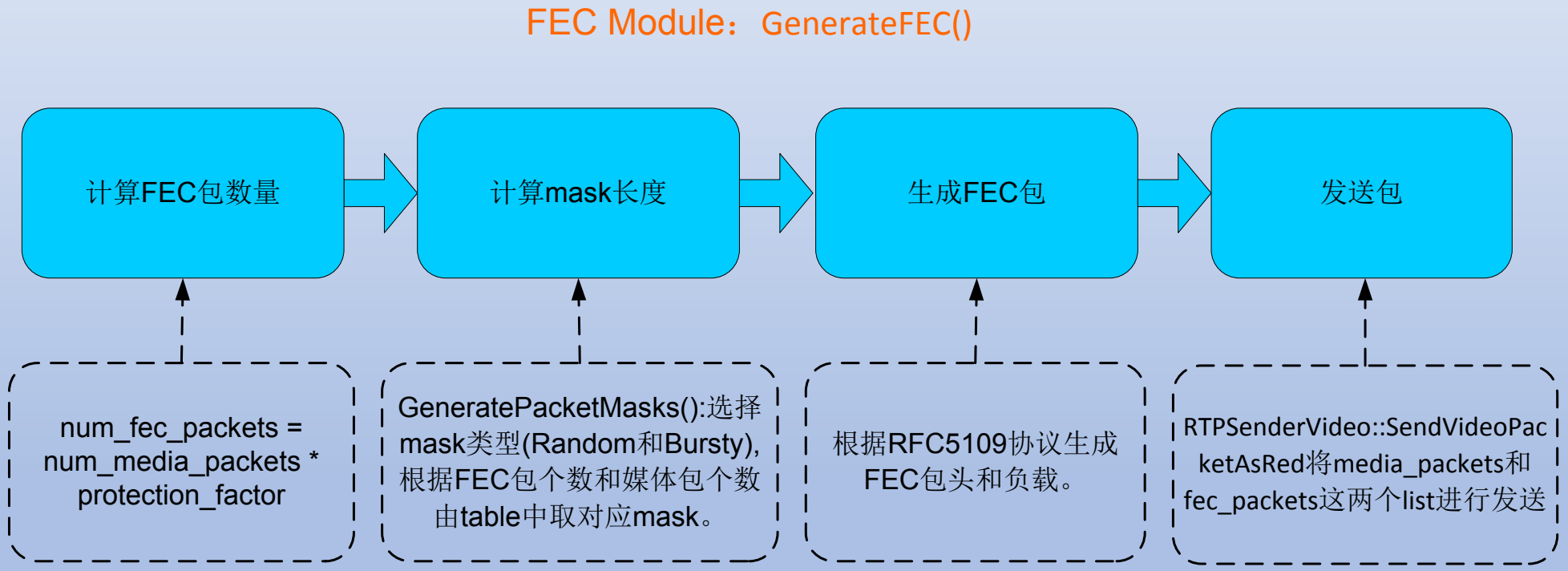
SDP:
a=rtpmap:116 red/90000(Redundant Audio Data)
a=rtpmap:117 ulpfec/90000

FEC编解码流程图

1. FEC参数由视频分辨率，比特率以及丢包率计算出的索引值查表获得。
2. 针对不同的视频帧（I帧或P帧），FEC的参数略有不同，以实现非对等保护。

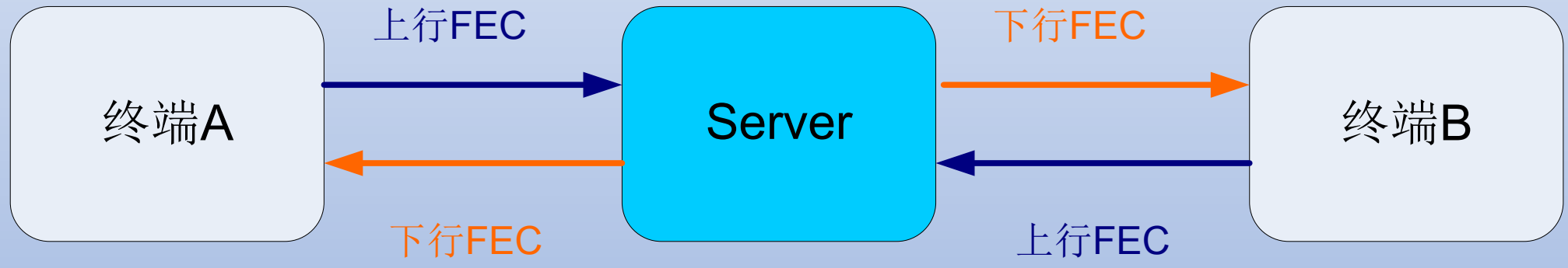


FEC生成模块


FEC Module：GenerateFEC()
计算FEC包数量
计算mask长度
生成FEC包
发送包
num_fec_packets = num_media_packets * protection_factor
GeneratePacketMasks():选择mask类型(Random和Bursty),根据FEC包个数和媒体包个数由table中取对应mask。
根据RFC5109协议生成FEC包头和负载。
RTPSenderVideo::SendVideoPacketAsRed将media_packets和fec_packets这两个list进行发送

FEC优化

- 将上行通道和下行通道分别进行FEC保护



带宽自适应与抗丢包综合策略

1. 当丢包率超过阈值，大幅度降低发送速率
2. 通过服务器数据跟踪可用网络带宽，提高带宽的利用率
3. 减少延时对于带宽估计的影响
4. 控制FEC冗余率为实际丢包率的倍数

技术？思想？哲学？

- 技术是枯燥的
- 思想是鲜活的
- 哲学熠熠生辉

从一个技术点到矛与盾的哲学

天翼RTC
www.ChinaRtc.com

感谢您的聆听，期待合作！