**SRT传输库评估报告(V1.0.0)**

[www.mediapro.cc](http://www.mediapro.cc)

1. **SRT传输库简介**

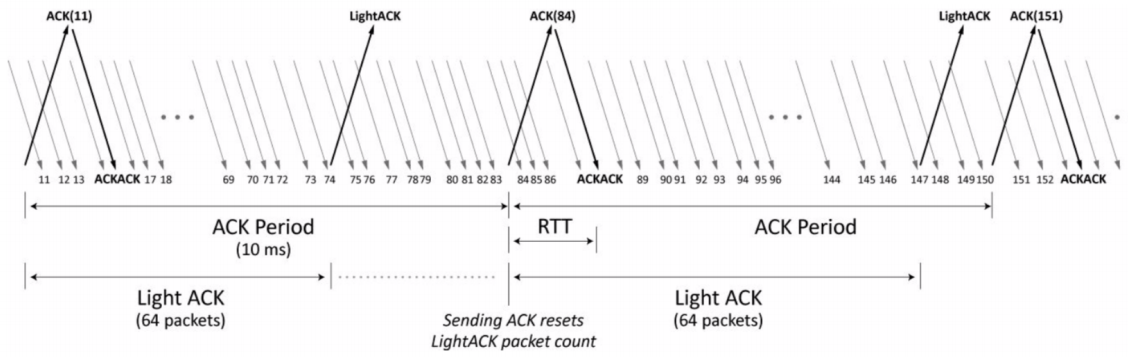
SRT是Haivision开源的一套集FEC前向纠错、ACK、NAK选择性重传、JitterBuff、拥塞控制、传输安全保障等技术于一体的实时传输解决方案。方案基于UDP协议进行扩展，目前SRT在广电领域获得了不错的应用，因其弱网抵抗以及较好的实时性，必将取代基于TCP的RTMP伪直播方案。SRT方案使用C++开发，提供C风格接口，依赖OpenSSL\Pthreads库，官方编译脚本支持Windows、Linux、Mac\IOS系统，可自行参考移植到Android。

本文对SRT封装库SD-SRT进行测试，该封装库主要针对音视频领域需求，进行了帧码流拆分合并、丢包冻结、自动重连、EPoll收发、状态回调等功能的扩展，并对外提供简洁易用的接口，可用于内网点对点或者公网CS架构，其接口说明见附录。

希望通过本测试，加深对于SRT的传输特征、参数配置、适用场合的认识。

从官方声明可见，SRT具有以下特点：

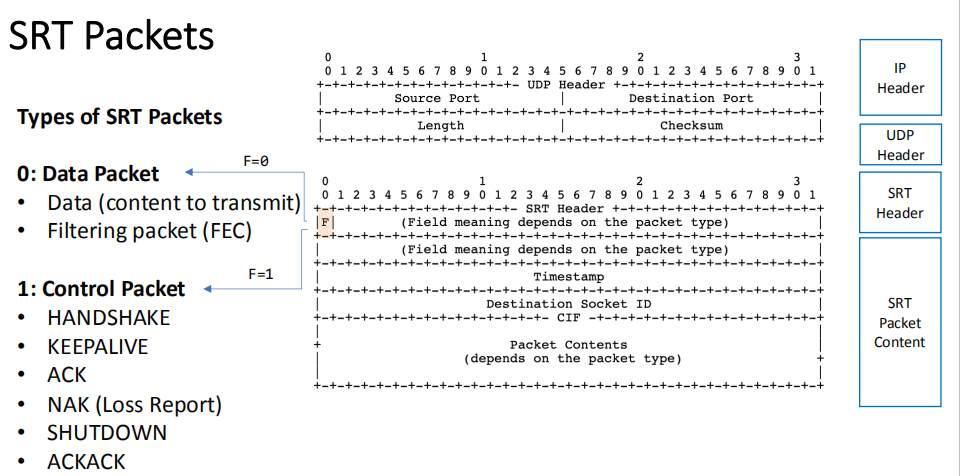
1. 基于ARQ（ACK+NAK），ACK机制相比TCP ACK做了一定改进，采用Full ACK和Light ACK结合的模式，其中Full ACK每10ms发送一次，对端收到ACK后会返回ACKACK信令。Light ACK则无需ACKACK，每64个包发送一次。



SRT利用Full ACK完成RTT的计算、反馈接收端的接收缓存情况等。

1. SRT中的NAK机制可与FEC相结合，可以选择三种模式：仅对FEC恢复失败的情况下发起NAK重传请求、丢包均发起NAK、丢包不发起NAK。另外SRT会在预估超时的情况下，提前结束无意义的重传动作，在延时和码率允许的情况下支持反复重传直至成功。
2. 在1.4版本之后支持2D 异或FEC（默认关闭，需自行以Filter方式启用）。注：经过测试，收发双方均设置FEC成功，码流中确实产生冗余包，但未见显著的丢包恢复能力，当开启FEC时，SRT库长时间拷机存在异常奔溃的情况。正如Nimble Streamer描述的那样（https://blog.wmspanel.com/search/label/srt），SRT中的FEC功能正处于开发完善中，存在较大风险，用户根据自身情况决策是否在当前版本开启。在本次测试中，我们使用SRT的默认配置即关闭FEC功能。
3. 发端Smooth处理和收端JitterBuff处理。
4. 仿socket，支持双向通讯，支持Epoll异步处理。
5. 支持AES 128/192/256加密
6. 内容无关，即可以传输音视频也可以传输其他类型数据。
7. 提供较为详尽的统计信息（上下行丢包率、码率、RTT、重传成功率等），支持带宽估计参考（定时插入探针包，通过包间隔评估码率）。
8. 支持Message消息模式、文件Buff模式和Live实时模式（默认），前两者可保证传输不丢包但不保证实时性，后者以实时性为目标允许丢包。本文仅考虑Live实时模式。

SRT基于UDP，在UDP基础上增加16字节头部，其包结构如下：



默认情况，SRT按MTU 1500字节进行配置，去除20字节IP头、8字节UDP头、16字节SRT头后，支持负载可达到1456字节，又因其保持对TS容器的友好性，默认限定负载长度为188\*7=1316字节。（Windows平台为了网络性能达到最优，建议用户设置UDP负载小于1024，此时用户可设置SRT最大长度不超过1024 - 16）。

1. **实验环境**

**SRT版本说明**

本次测试使用SRT版本：1.4.1-152-a4ff6ab

**网络环境说明**

为了保证测试环境一致性和可重现性，我们将在较好的网络环境下借助第三方弱网模拟工具Clumsy，模拟各类网络情况。

Clumsy是一款小巧而功能强大的开源弱网模拟工具，支持windows平台，可用于模拟：丢包(Drop)、延时(Lag)、重复(Duplicate)、乱序(Out of order)、篡改(Tamper)、抖动(Throttle)等。其项目地址：<http://jagt.github.io/clumsy/cn/index.html>

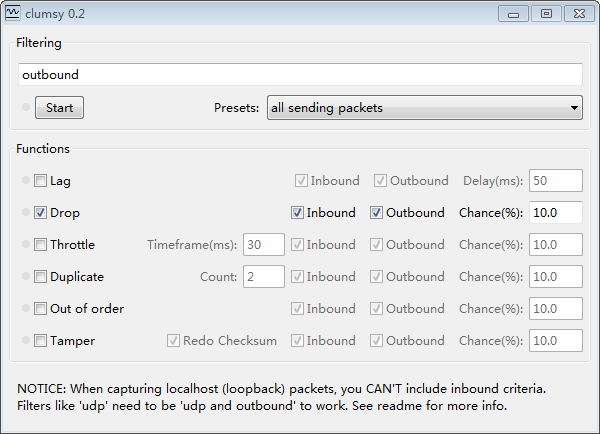


图3 Clumsy对所有发送的包按10%进行丢包处理示意

如果要对发往指定某IP的UDP包进行丢包，可将Filtering条件设置为：

udp and (ip.DstAddr == 192.168.31.33)

本次测试我们使用两台PC机器，二者接入到同一个WIFI网络，信号强度充足，其中一台作为发送端并在其上执行Clumsy，另一台作为接收端。

**测试DEMO说明**

本次测试使用windows平台下的桌面投屏DEMO，DEMO分为发送端和接收端，发送端采集自身桌面和扬声器音频，压缩后通过SD-SRT 点对点SDK发往接收端，后者解码并渲染输出，从而实现屏幕共享功能。在此场景下，DEMO接收端将充SRT服务端，DEMO发送端充当SRT客户端。

**A、接收端**

该组DEMO的功能与投屏类应用类似，我们首先启动接收端DEMO。进入UDP-AVClient-ScreenPlay文件夹，双击启动AVClient.exe即可。



图4 接收端DEMO启动界面

接收端启动后，将显示其投屏码（IP:PORT），发送端可以使用该投屏码进行投屏。

当发送端码流到来时，**接收端将使用一个新的窗口“Remote Video”显示远端画面**，如下图所示：



图5 接收端独立的窗口展示远端画面

**注意**：“Remote Video”窗口是一个全屏窗口，用户可以自行在底部任务栏切换。当远端停止音视频传输时，该窗口内容无更新。该窗口不会响应鼠标事件，只能底部切换。

接收端文件夹下的AVClient.ini文件为其配置文件，对配置文件的修改需要重启客户端方能生效。配置文件包括如下几项：

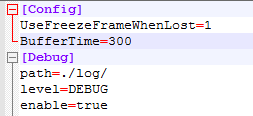


图6 接收端配置文件

**UseFreezeFrameWhenLost** 表示当出现视频丢包无法恢复时，为了不展现出花屏而将画面冻结，直至完整的关键帧到来再继续送显。该值一般设置为1开启。

**BufferTime**表示接收端Jitter buff缓存毫秒数，对应SRT的SRTO\_RCVLATENCY参数。为了抵抗网络传输、丢包重传等行为带来的抖动，SRT需要设置接收端缓存以保障输出的流畅性（可以在发送端和接收端中任意一方或双方设置，实际缓存时间取二者中的较大值），SRT的默认缓存时间为120ms，官方建议设置为RTT\*4，最小值不低于120ms。因为流畅性和实时性（时延）是一对矛盾的指标，Jitter buff必然将引入一定延时。在后面的测试过程中，将会对BuffTime进行调整，查看调整前后的效果对比。

**B、发送端**

发送端为UDP-AVClient-ScreenCap目录下的AVClient.exe，在启动前我们需要先对其进行配置，配置文件为AVClient.ini

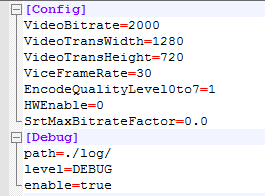


图7 发送端配置文件

**VideoBitrate**表示发送端使用的视频编码码率，单位kbps，设置为2000即表示2Mbps

**VideoTransWidth**表示发送端使用的视频编码宽度，**VideoTransHeight**表示视频编码高度，**ViceFrameRate**表示视频编码帧率（本程序使用Direct桌面采集，在性能较低的机器上可能采集帧率无法达到30fps，编码帧率仍然会按30fps配置编码器）

**EncodeQualityLevel0to7**表示当采用X264软编码时，使用的preset级别，0表示ultrafast，1表示superfast，2表示veryfast，3表示faster，4表示fast，5表示medium，6表示slow，7表示slower。等级越高同等码率下的图像质量越好，但CPU占用也越高，请根据自身机器配置而定，建议设置为1。当使用X264软编码时，使用5秒一个IDR帧。当使用硬编码时，使用3秒一个IDR帧。

**HWEnable**表示是否启用硬编码，程序支持Intel QSV硬编码和Nvidia硬编码，相比X264能获得更低的CPU占用。不过硬编码的缺点是灵活性不足，无法支持传输层IDR帧请求机制。

**SrtMaxBitrateFactor**表示设置SRT最大码率MAXBW为编码码率VideoBitrate的几倍，我们称之为峰值码率容忍度。比如该值为2.0，VideoBitrate为2000时，则将设置SRT的SRTO\_MAXBW为2.0\*2000 kbps（实际内部会转换为SRT所要求的BytePerSecond）。当设置为0.0时，SRT将不限码率。

SRT的 MAXBW对于传输性能影响较大，主要体现在两个方面，发送端将依据MAXBW调整发送数据间隔，实现Smooth发送平滑处理。另一方面，允许的MAXBW越大，短时间内允许的重传次数更多，越能提高弱网重传成功率，但也会给网络带来更大的码率波动压力。实际过程中应该根据自身编码器码率及其波动范围、信道带宽来综合权衡，设置合理的MAXBW。

启动发送端后进入如下界面，输入接收端展示的投屏码（IP:PORT）即可开始SRT连接。



图8 发送端启动界面

连接后，客户端将进入下图所示的待共享屏幕状态。可以点击主界面启动按钮或者使用悬浮球来启动桌面共享。启动后，接收端就能看到发送端的桌面并能听到发送端扬声器播放的音乐了。



图9 发送端开始共享桌面

1. **测试项说明**

**说明：**同市面上各大实时视频服务商一样，DEMO也提供丢帧冻结机制，这样用户无法察觉到丢帧带来的花屏，从而获得更好的用户体验。因此本次测试中，丢包最终将体现为画面卡顿。

**评测项：流畅度**

关于流畅度，我们将分为以下几个级别：

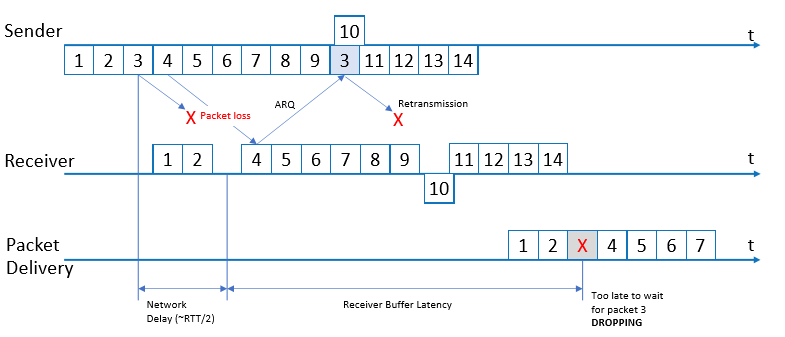
1. 画面流畅
2. 偶尔微弱卡顿（附加：卡顿时长+频率描述）
3. 明显卡顿（附加：卡顿时长+频率描述）
4. 较长时间卡顿

**评测项：延时**

延时计算方式：在发送端打开毫秒精度秒表，接收端将看到秒表值，使用手机对二者屏幕拍照，计算二者差值得到总延时。整个系统中，延时主要有非传输层延时和传输层延时两部分组成。非传输层延时包括：采集、编码、解码、渲染引入的延时，本DEMO实际采集帧率无法达到恒定30fps，对整体延时稍有影响。

传输层延时主要由接收端JitterBuff引入，后者用于消除网络因丢包重传、网络本身带来的抖动。JitterBuff越大，播发端缓存的数据越多。

需要说明的是延时指标和流畅性指标往往是一对矛盾，播发端缓存的数据越多，流畅性越好，延时也越大，反之若缓存的数据较少或者不缓存，则延时更低，但与此同时它的弱网抵抗力越差，重传恢复成功率越低进而影响流畅性。



上图是SRT的Too-Late Packet Drop机制描述，虽然为3号包的丢失发起了NAK请求，但发送端在收到NAK请求后判断3号包即便发出也已经超出了其接收端的dead line，已经错过了它的输出时间，而放弃重传。同样即便3号包被重传，接收端也会因其错过输出时间而直接丢弃之。

**评测项：清晰度**

DEMO图像质量与传输层无紧密关系，主要由用户指定的编码分辨率、码率、桌面画面内容决定。注：帧率降低时，帧间相关性降低，运动估计残差更大，同等码率下编码质量会稍弱。

1. **测试结果**
2. **丢包测试**

为了研究接收端缓存时间、发送端峰值码率容忍度对于丢包抵抗力的影响，我们设计以下测试（发送端编码码率为2Mbps，720P分辨率，X264软编码，30fps，发送端全屏播放影片《美女与野兽》，画面中等复杂度）。

A、丢包率5%，发送端峰值码率容忍度设置为2.0，接收端缓存时间依次设置为200ms、300ms、400ms、500ms，观察卡顿频率、峰值码率、延时三个指标。

|  |  |  |  |
| --- | --- | --- | --- |
| 接收缓存时间 | 卡顿频率 | 实际峰值/平均码率 | 实际延时 |
| 120ms（SRT默认值） | 约20秒卡顿一次 | 2.7Mbps/2.1Mbps | 300ms |
| 200ms | 约30秒卡顿一次 | 2.7Mbps/2.1Mbps | 420ms |
| 300ms | 约40秒卡顿一次 | 2.9Mbps/2.1Mbps | 530ms |
| 400ms | 约40秒卡顿一次 | 2.9Mbps/2.1Mbps | 630ms |
| 500ms | 约40秒卡顿一次 | 2.9Mbps/2.1Mbps | 720ms |

通过本项实验，我们发现接收端缓存时间对丢包抵抗有一定正向作用，缓存时间过短，容易在发送端放弃重传（预判超出接收端包输出时间）或者接收端收到后因超时而主动丢弃。当缓存时间达到要求后，继续增大缓存时间对丢包抵抗力无明显作用。

1. 丢包率5%，接收端缓存时间设置为500ms，发送端峰值码率容忍度依次设置为3.0、4.0、0.0（无限制），观察卡顿频率、峰值码率、延时三个指标。

|  |  |  |  |
| --- | --- | --- | --- |
| 峰值码率容忍度 | 卡顿频率 | 实际峰值/平均码率 | 实际延时 |
| 3.0 | 约220秒卡顿一次 | 3.2Mbps/2.2Mbps | 720ms |
| 4.0 | 约240秒卡顿一次 | 3.5Mbps/2.3Mbps | 720ms |
| 无限制（SRT默认值） | 长时间无卡顿 | 3.6Mbps/2.3Mbps | 720ms |

通过本项实验，我们确定发送端码率峰值放得越宽，丢包抵抗力越强，这是因为短时间内可以更高频率的进行丢包重传，增加了重传次数也就提升了成功率。当我们的网络允许较大的码率波动时，非受限的MAXBW设置可以获得显著的质量提升。

1. 接收端缓存时间设置为500ms，发送端峰值码率容忍度设置为0.0（无限制），依次设置丢包率为10%、20%、30%、50%，观察卡顿频率、峰值码率三个指标。

|  |  |  |
| --- | --- | --- |
| 丢包率 | 卡顿频率 | 实际峰值/平均码率 |
| 10% | 长时间无卡顿 | 3.5Mbps/2.4Mbps |
| 20% | 约300秒卡顿一次 | 3.5Mbps/2.5Mbps |
| 30% | 约30秒卡顿一次 | 4.5Mbps/2.9Mbps |
| 50% | 约10秒卡顿一次 | 8Mbps/4Mbps |

随着丢包率的继续上升，即便不限制码率峰值，由于缓存时间的限制，在有限的时间内重传仍然存在失败的可能，导致最终丢包卡顿。此时需要增大接收缓存时间来进一步提高丢包抵抗力。

1. **重复测试**

发送端使用Clumsy 设置Duplicate发送重复率**5%、12%、20%、30%，每次重复1包（Count设置为2）。**发送端使用峰值码率容忍度设置为0.0（无限制），接收端使用缓存时间120ms。

5%重复包时，连续观察20分钟，画面流畅，延时稳定在300ms左右。

12%重复包时，连续观察20分钟，画面流畅，延时稳定在300ms左右。

20%重复包时，连续观察20分钟，画面流畅，延时稳定在300ms左右。

30%重复包时，连续观察20分钟，画面流畅，延时稳定在300ms左右。

**可见单纯的重复包对SRT影响很小。**

1. **乱序测试**

发送端使用Clumsy 设置Out of order发送乱序率**30%。**发送端使用峰值码率容忍度设置为0.0（无限制），接收端使用缓存时间依次为120ms、300ms、500ms。

|  |  |  |
| --- | --- | --- |
| 缓存时间 | 卡顿频率 | 实际峰值/平均码率 |
| 120ms | 约100秒卡顿一次 | 3.5Mbps/2.5Mbps |
| 300ms | 约120秒卡顿一次 | 3.5Mbps/2.5Mbps |
| 500ms | 约155秒卡顿一次 | 3.5Mbps/2.5Mbps |

可见乱序包对系统影响很大，接近丢包的影响，可能系统内部的乱序容忍窗口上限较小，很多乱序当做丢包处理（SRT会根据超时迟到包的偏离间隔来更新乱序容忍窗口）。但缓存时间增大时，对乱序的恢复能力明显增加，这可能是乱序容忍窗口扩大以及重传成功率提升两方面因素导致。

1. **延时测试**

发送端使用Clumsy 设置Lag发送延时**50、100、200、400、600ms。**

**经过测试，线路延时最终会叠加到总体延时之上，测试结果符合预期。**

1. **抖动测试**

发送端峰值码率容忍度设置为0.0（无限制），接收端使用缓存时间为120ms，进行如下实验：

发送端使用Clumsy 设置Throttle分别5%、12%、20%、30%概率抖动30ms。

测试结果：5%~30%概率30ms抖动对流畅性、延时无可感知的影响。

发送端使用Clumsy 设置Throttle 30%概率抖动100ms**。**

测试结果：小概率卡顿，延时增长到500ms，关闭抖动后延时仍为500ms未回归。说明SRT根据抖动情况自动增大缓存时间，避免因缓存不足而持续卡顿。

1. **极速测试**

发送端峰值码率容忍度设置为0.0（无限制），接收端使用缓存时间为5ms，不开启丢包等其他弱网测试，视频卡顿频繁。也证实了官网的说法，即便网络RTT非常小，也不要修改接收端缓存时间小于默认值120ms。

1. **断网测试**

本DEMO基于SD-SRT库，内部实现了自动重连机制，当物理网络断开或者丢包率过高导致SRT断开后，SD-SRT将不断尝试重连，网络恢复后将快速重连上并恢复业务。实际验证网线插拔，突发丢包率达到80%以上等场景，业务均可恢复。

1. **SRT中的FEC方案**

SRT采用滤镜的方式引入FEC并且默认情况下关闭了FEC功能，可通过SRTO\_PACKETFILTER选项设置FEC描述字符串来启用。描述字符串格式为：

"fec,cols:%d,rows:%d,layout:%s,arq:%s"

其中col用于描述2D 异或FEC的列数，rows描述行数，layout描述FEC的布局（even、staircase），arq描述FEC与NAK的结合方式。下图是一种3行6列even布局的FEC示意图，图中1~18号灰色包为媒体包，R1~R9号橙色包为冗余包，它们共同组成一个FEC Group。其中1号冗余包由1~6号媒体包异或得到，4号冗余包由1、7、13号媒体包异或得到。



2D FEC rows=3, cols=6

以上布局，冗余率为9/18=50%。

假设丢包1、2、3、4、5、6、R1、7共8个，可以由8~12 + R2恢复7号媒体包，7、13、R4恢复1号媒体包，8、14、R5恢复2号媒体包并按此规则依次恢复3、4、5、6。

假设丢包13、14、15、16、17、18、R3、R4，将无法恢复。

假设超出8个丢包，比如丢包1、2、3、4、5、6、R1、7、8，只能恢复3、4、5、6号媒体包。这种两头丢失、中间恢复的情况对于视频流用途不太大，因为视频流往往采用丢包冻结机制，一帧中任意丢包均丢弃整帧码流避免花屏。对于音频包，任何包的恢复都是有利的。两头丢失、中间恢复对于重传来说可以减少部分包的重传，但相比全部重传是否一定产生优势比较难说。（对比2D异或FEC，若采用RS FEC 50%的冗余（18 + 9），可以抵抗任意9个丢包）

以上为even布局，包的发送顺序为：1、2、3、4、5、6、R1、7、8、9、10、11、12、R2、13、R4、14、R5、15、R6、16、R7、17、R8、18、R9、R3，可见在Group的尾部形成了较为密集的数据发送（带宽增长了一倍），对网络并不友好。Staircase阶梯布局正是为了解决这一问题，通过阶梯排列可以将冗余包的发送错开，避免集中发送带来的码率波动。

值得注意的是2D FEC的冗余度是由行和列决定的，列越大抵抗连续丢包的能力越强。如果上图改成6行3列，其最多抵抗连续5个丢包。

开启FEC时，若发生丢包，FEC恢复处理将引入抖动，比如收到的媒体包和冗余包：1、3、4、5、6、7、8、R1、R2、R3，其中2号媒体包的丢失需要暂停输出，并等到R1号冗余包到来才能恢复并输出。对于2D异或FEC，为消除FEC带来的抖动，至少需要准备的延时LatencyFec为N个包的发送时长：

N = (R + (C - 1)) + 2

其中R为row行大小，C为cols列大小，N个包的发送时长与码率、帧率强相关。极端情况下假设码率比较低，1帧视频仅1个包，帧率30fps，N为30时，最小需要准备的缓存时间为1秒。

当FEC的ARQ选项配置为ALWAYS时，即只要发现丢包即刻发起NAK重传请求，则推荐的延时为max(RTT\*4, LatencyFec)。当FEC的ARQ选项配置为ONREQ时，推荐的延时为RTT\*4 + LatencyFec即二者之和，这是因为ONREQ模式下，需要等到确认FEC失败才发起NAK重传，确认失败的条件是当前FEC GROUP已经接收完成了仍旧无法恢复GROUP内丢失的包。

QOS-FEC-NACK方案的重传机制与SRT有所不同，QOS-FEC-NACK会根据GROUP内已检测到的丢包数目、包类型结合GROUP的冗余包数量提前预判是否需要发起重传，不需要等到下一个GROUP到来才发起重传请求，这样做的目的是尽量减少重传等待时间进而减小延时，对于已经具备媒体包恢复条件或者媒体包未丢失的情况，FEC冗余包的丢失不会触发重传，因为它们已经没有意义。

1. **SRT与QOS-FEC-NACK方案的区别**

|  |  |  |
| --- | --- | --- |
|  | **SRT** | **QOS-FEC-NACK** |
| **机制** | ARQ（ACK+NACK）+ FEC | NACK + FEC |
| **模式** | Live /File/Message | Live |
| **主要技术** | 发端Smooth  JitterBuff  2D XOR FEC，固定冗余度  FEC作为NACK重传参考  超时时间内反复重传  NACK信令冗余防丢失  NACK主动放弃机制  带宽估计  内部传输模式（内部维护socket）  OpenSSL安全 | 发端Smooth  JitterBuff  1D RS FEC 自适应冗余度  FEC 视频帧边界形成大GROUP抗连续丢包抵抗力强  FEC作为NACK重传参考  仅单次重传机会  NACK预判提前发起  NACK信令冗余防丢失  NACK主动放弃机制  内部传输模式  外部传输模式  码率自适应参考信息提供 |
| **推荐场景** | 推荐单链路上4\*RTT延时（服务器转发模式下总延时8\*RTT）。适合实时性要求1秒左右的单向抗弱网直播或者RTT较小、带宽不受限但需要抵抗突发丢包的内网场合。  有望成为一些领域的标准协议。 | 推荐在实时性要求较高的互动场合使用，允许一定的丢包（体现为画面卡顿）。适合要求带宽波动比较稳定可控的场合。  对外无依赖性、轻量级、跨平台性能佳。 |
| **缺点** | 默认未开启FEC的情况下，弱网抵抗力重点依赖ARQ机制，总体延时偏大。  FEC为新增功能稳定性还有所欠缺，FEC未与视频帧信息结合，同样冗余度的情况下抗丢包能力弱。 | 即便放宽延时指标，也无法保证100%的接收成功率。（FEC+单次NACK重传，若重传仍然丢失或者重传超时，都将体现为最终丢包）。  为私有协议，收发双方均需集成才能互通。 |

1. **总结**

我们测试了SRT在默认关闭FEC的场景下的弱网抵抗力，对其传输性能影响较大的两个参数（接收缓存时间、最大码率）进行了重点评估。当前SRT版本的推荐应用场景为：实时性要求1秒左右的单向抗弱网直播或者RTT较小、带宽不受限但需要抵抗突发丢包的内网场合。另外SRT提供的可靠传输文件模式和消息模式可用于改进现有TCP传输方案，提高弱网下的吞吐率。我们将进一步跟踪SRT项目的进展，待其FEC功能稳定性提升后另行评估。

1. **参考文献**

### Efficient usage of SRT latency and maxbw parameters， Nimble Streamer的SRT推荐配置

<https://blog.wmspanel.com/2019/06/srt-latency-maxbw-efficient-usage.html>

### SRT FEC (forward error correction) support in Nimble Streamer，Nimble Streamer的SRT FEC风险说明

<https://blog.wmspanel.com/2020/04/srt-fec-forward-error-correction.html>

1. SRT\_Protocol\_TechnicalOverview\_DRAFT\_2018-10-17.pdf
2. <https://datatracker.ietf.org/meeting/107/materials/slides-107-dispatch-srt-overview-01>
3. Interoperable Retransmission Protocols with Low Latency and Constrained Delay

<http://kth.diva-portal.org/smash/get/diva2:1335907/FULLTEXT01.pdf>

**附录**

**SD-SRT封装库接口**

1. **系统初始化接口**

/\*\*\*

\* 环境初始化，系统只需调用一次，主要用于SRT环境以及日志模块的初始化

\* @param: outputPath 表示日志存放路径，支持相对路径和绝对路径，若目录不存在将自动创建

\* @param: outputLevel表示日志输出的级别，只有等于或者高于该级别的日志输出到文件，取值范围参考LOG\_OUTPUT\_LEVEL

\* @return:

\*/

void SDSrtAvCom\_Enviroment\_Init(const char\* outputPath, int outputLevel);

void SDSrtAvCom\_Enviroment\_Free();

1. **创建和删除SD-SRT对象**

/\*\*\*

\* 创建SrtAvCom

\* @param unLogId: 日志ID，仅用于日志输出时的对象标识。

\* @return: 返回模块指针，为NULL则失败

\*/

void\* SDSrtAvCom\_Create(UINT unLogId);

/\*\*\*

\* 销毁SrtAvCom，使用者应该做好与其他API之间的互斥保护

\* @param pRtp\_avcom: 模块指针

\* @return:

\*/

void SDSrtAvCom\_Delete(void\* pRtp\_avcom);

1. **启动**

/\*\*\*

\* 开始工作

\* @param strLocalIP: 本地IP地址，允许为NULL，为非NULL时将绑定到该IP（网卡）。

\* @param shLocalPort: 本地通信端口（该端口用于音频，视频端口号将在此基础上加1），对于客户端模式时，允许设置本地端口号为0，此时将由系统自动选择可用的端口。

\* @param strRemoteIP: 对方IP地址，当为服务端模式时设置为NULL

\* @param shRemotePort: 对方收发端口（该端口用于音频，视频端口号将在此基础上加1），当为服务端模式时设置为0

\* @param pfVideoRecvCallBack: 接收到视频数据后的对外输出回调函数

\* @param pfAudioRecvCallBack: 接收到音频数据后的对外输出回调函数

\* @param pObject: 调用上述两个回调函数时的附带透传形参，模块内部不会解析本参数仅做透传处理

\* @return: TRUE FALSE

\*/

BOOL SDSrtAvCom\_Start(

void\* pRtp\_avcom,

const char \*strLocalIP,

USHORT shLocalPort,

const char \*strRemoteIP,

USHORT shRemotePort,

CallBackFuncRecvVideoData pfVideoRecvCallBack,

CallBackFuncRecvAudioData pfAudioRecvCallBack,

void \*pObject);

1. **结束**

/\*\*\*

\* 停止SrtAvCom工作

\* @param pRtp\_avcom: 模块指针

\* @return:

\*/

void SDSrtAvCom\_Stop(void\* pRtp\_avcom);

1. **发送视频**

/\*\*\*

\* 发送视频数据

\* @param pRtp\_avcom: 模块指针

\* @param byBuf: 传入一帧带起始码的裸码流，内部自行拆分拼接。

\* @param nLen: 数据长度

\* @return:

\*/

BOOL SDSrtAvCom\_SendVideoData(void\* pRtp\_avcom, unsigned char \*byBuf, int nLen);

1. **发送音频**

/\*\*\*

\* 发送音频数据

\* @param pRtp\_avcom: 模块指针

\* @param byBuf: 传入一帧音频裸码流，可以是ADTS，内部无拆包透传

\* @param nLen: 数据长度

\* @return:

\*/

BOOL SDSrtAvCom\_SendAudioData(void\* pRtp\_avcom, unsigned char \*byBuf, int nLen);

1. **设置通用传输参数**

/\*\*\*

\* 设置基础传输参数，请在Start接口之前调用

\* @param pRtp\_avcom: 模块指针

\* @param nRecvDelayMs: 接收缓存时间，建议4\*RTT，单位ms。可在发送端或接收端设置，将取其中较大的值

\* @param nMaxBitrateKbps：最大传输码率，建议3\*VideoEncBitrate，单位kbps。需要在发送端设置，当设置为0时表示不受限。若码率本身比较平稳，可设置为2\*VideoEncBitrate。

\* @return:

\*/

BOOL SDSrtAvCom\_SetBaseTransParams(void\* pRtp\_avcom, int nRecvDelayMs, int nMaxBitrateKbps);

1. **设置视频通道传输参数**

/\*\*\*

\* 设置视频通道FEC传输参数，请在Start接口之前调用

\* @param pRtp\_avcom: 模块指针

\* @param bEnable: 是否启用FEC，收发双方需保持一致

\* @param nCols: FEC Group列数

\* @param nRows: FEC Group行数

\* @param eLayoutMode：2D FEC布局模式

\* @param eArqMode：FEC-ARQ配合模式

\* @return:

\*/

BOOL SDSrtAvCom\_SetVideoFecParams(void\* pRtp\_avcom, BOOL bEnable, int nCols, int nRows, E\_SRT\_FEC\_LAYOUT eLayoutMode, E\_SRT\_FEC\_ARQ eArqMode);

1. **设置音频通道传输参数**

/\*\*\*

\* 设置音频通道FEC传输参数，请在Start接口之前调用

\* @param pRtp\_avcom: 模块指针

\* @param bEnable: 是否启用FEC，收发双方需保持一致

\* @param nCols: FEC Group列数

\* @param nRows: FEC Group行数

\* @param eLayoutMode：2D FEC布局模式

\* @param eArqMode：FEC-ARQ配合模式

\* @return:

\*/

BOOL SDSrtAvCom\_SetAudioFecParams(void\* pRtp\_avcom, BOOL bEnable, int nCols, int nRows, E\_SRT\_FEC\_LAYOUT eLayoutMode, E\_SRT\_FEC\_ARQ eArqMode);

1. **获得视频通道统计数据**

/\*\*\*

\* 获取视频通道统计信息

\* @param pRtp\_avcom: 模块指针

\* @param pfRttMs: RTT，单位毫秒

\* @param pfUpLossRate: 上行丢包率.内部已经乘100

\* @param pfDownLossRate: 下行丢包率.内部已经乘100

\* @param pfEstimatedUpBitrate：上行带宽估算.Kbps

\* @param pfUpBitrate：上行码率.Kbps

\* @param pfDownBitrate：下行码率.Kbps

\* @return:

\*/

BOOL SDSrtAvCom\_GetVideoTransStatis(void\* pRtp\_avcom, double \*pfRttMs, double \*pfUpLossRate, double \*pfDownLossRate, double \*pfEstimatedUpBitrate, double \*pfUpBitrate, double \*pfDownBitrate);

1. **获得音频通道统计数据**

/\*\*\*

\* 获取音频通道统计信息

\* @param pRtp\_avcom: 模块指针

\* @param pfRttMs: RTT，单位毫秒

\* @param pfUpLossRate: 上行丢包率.内部已经乘100

\* @param pfDownLossRate: 下行丢包率.内部已经乘100

\* @param pfEstimatedUpBitrate：上行带宽估算.Kbps

\* @param pfUpBitrate：上行码率.Kbps

\* @param pfDownBitrate：下行码率.Kbps

\* @return:

\*/

BOOL SDSrtAvCom\_GetAudioTransStatis(void\* pRtp\_avcom, double \*pfRttMs, double \*pfUpLossRate, double \*pfDownLossRate, double \*pfEstimatedUpBitrate, double \*pfUpBitrate, double \*pfDownBitrate);

**附录：结构体说明**

//2D FEC布局模式

Typedef enum E\_SRT\_FEC\_LAYOUT

{

//普通连续模式

e\_SRT\_FEC\_LAYOUT\_EVEN = 0,

//阶梯模式，可以降低一定码率波动（冗余包分散发送）

e\_SRT\_FEC\_LAYOUT\_STAIR

} E\_SRT\_FEC\_LAYOUT;

//FEC-ARQ配合模式

typedef enum E\_SRT\_FEC\_ARQ

{

//只要丢包均会发起NAK，不管FEC能否恢复

e\_SRT\_FEC\_ARQ\_ALWAYS = 0,

//仅在FEC失败时发起NAK

e\_SRT\_FEC\_ARQ\_ONREQ,

//关闭NAK

e\_SRT\_FEC\_ARQ\_NEVER,

} E\_SRT\_FEC\_ARQ;

/\*输出接收到的视频数据 回调函数

\* @param bComplete用来表示当前帧数据是否完整（无局部丢包）

\* @param bPrevTotalFrameLost用来表示当前帧与上一次输出帧之间无整帧丢失的情况，即本帧序号与上一帧序号是否连续。

\* 通过以上两个标志，结合关键帧判定标志，外层可以很方便的实现丢帧冻结机制

\*/

typedef void (\*CallBackFuncRecvVideoData)(void\* pObj, int nLen, unsigned char \*byBuf, unsigned int unPTS, BOOL bComplete, BOOL bPrevTotalFrameLost);

/\*输出接收到的视频数据 回调函数\*/

typedef void (\*CallBackFuncRecvAudioData)(void\* pObj, int nLen, unsigned char \*byBuf, unsigned int unPTS);