Transport-cc指的是Transport-wide Congestion Control。WebRTC最新的拥塞控制算法（Sendside BWE）基于Transport-cc，接收端记录数据包到达时间，构造相关RTCP包，然后反馈给发送端，在发送端做带宽估计，从而进行拥塞控制。之所以基于Transport-cc，放到发送端进行带宽估计，除了方便维护，也增加了相关算法的灵活性，因为大多数处理逻辑都放到了发送端。WebRTC中为了使用Transport-cc，需要用到RTP报头扩展以及增加新的RTCP类型。这里我们介绍下Transport-cc中的RTP以及RTCP。

RTP Header扩展

Transport sequence number

首先我们先来复习下RTP固定报头结构：

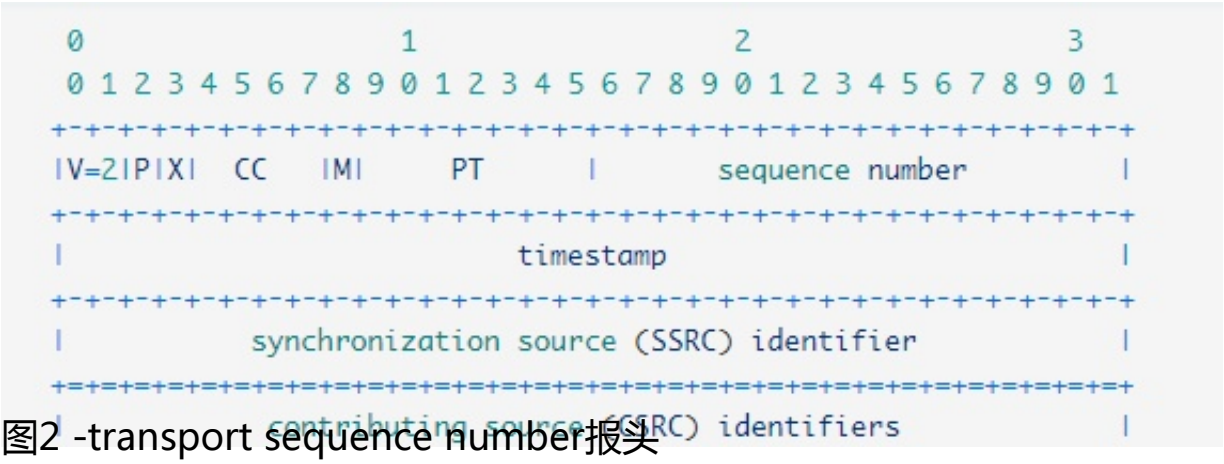

图2 -transport sequence number报头

由于属于RTP报头扩展，所以可以看到以0xBEDE固定字段开头，表示One-Byte Header类型的扩展。One-Byte Header相关知识请参考：[实时传输协议RTP(一)]

transport sequence number占两个字节，存储在One-Byte Header的Extension data字段。由于按4字节对齐，所以还有值为0的填充数据。

对于同一个PeerConnection下的每个包，这个transport sequence number是从1开始递增的。这里我们看下Wireshark中对带transport sequence numberRTP报头扩展的解析：

```
void PacketRouter::SendPacket(std::unique_ptr<RtpPacketToSend> packet,
                              const PacedPacketInfo& cluster_info) {
  rtc::CritScope cs(&modules_crit_);
  // With the new pacer code path, transport sequence numbers are only set here,
  // on the pacer thread. Therefore we don't need atomics/synchronization.
  // 如果当前RTP包注册了TransportSequenceNumber扩展
  if (packet->HasExtension<TransportSequenceNumber>()) {
    packet->SetExtension<TransportSequenceNumber>(++transport_seq_ & 0xFFFF);
```

图4 - 代码导读

TransportFeedback RTCP

报文格式

Transport-cc中，接收端通过TransportFeedback RTCP向发送端反馈收到的各个RTP包的到达时间信息。首先我们看下TransportFeedback包格式定义：

```
      0                   1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
```

图6 - packet chunk

chunk type (T):1 bit，值为0。
packet status symbol (S):2 bits，标识包状态。
run length (L):13 bits，行程长度，标识有多少个连续包为相同状态。

```
      0                   1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
```

图8 - Status Vector Chunk

1. chunk type (T):1 bit，值为1。
2. symbol size(S):1 bit，为0表示只包含"packet not received" (0)以及"packet received"（1）状态，每个状态使用1bit表示，这样后面14bits的symbol list能标识14个包的状态。为1表示使用2bits来标识包状态，这样symbol list中我们只能标识7个包的状态。
3. symbol list:14 bits，标识一系列包的状态, 总共能标识7或14个包的状态。

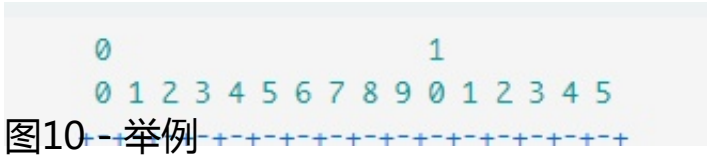

图10 - 举例

symbol size为1，这样只能标识7个包的状态。第一个包为"packet not received"（00）状态，第二个包为 "packet received, w/o timestamp"（11）状态，再接着三个包为"packet received"（01）状态，最后两个包为"packet not received"（00）状态。

Receive Delta

以250us(0.25ms)
为单位，表示RT
P包到达时间与前面一个RTP包到
达时间的间隔，对于记录的第一个RTP包，该包的时间间隔是相对reference time的。

如果在packet chunk记录了一个"Packet received, small delta"状态的包，那么就会在receive
delta列表中添加一个无符号1字节长度receive
delta，无符号1字节取值范围[0,255]，由于Receive
Delta以0.25ms为单位，故此时Receive Delta取值范围[0, 63.75]ms。

如果在packet chunk记录了一个"Packet received, large or negative delta"状态的包，那么就会在receive
delta列表中添加一个有符号2字节长度的receive delta，范围[-8192.0, 8191.75] ms。如果时间间隔超过了最大限制，那么就会构建一个新的TransportFeedback RTCP包，由于reference
time长度为3字节，所以目前的包中3字节长度能够覆盖很大范围了。

总结起来就是：对于收到的RTP包在TransportFeedback RTCP receive delta列表中通过时间间隔记录到达时间，如果与前面包时间间隔小，那么使用1字节表示，否则2字节，超过最大取值范围，就另起新RTCP包了。

对于"Packet received, small delta"状态的包来说，receive
delta
最大

值63.75ms，那么一秒时间跨度最少能标识1000/63.75~=16个包。由于receive delta为250us的倍数，所以一秒时间跨度最多能标识4000个包。

packet chunk以及receive delta的使用是为了尽可能减小RTCP包大小。packet chunk
用到了不同编码方式，对于收到的RTP包才添加到达时间信息，而且是通过时间间隔的方式记录到达时间。

代码导读

在RemoteEstimatorProxy中处理RTP包的到达时间，构造Transport-cc报文，反馈给发送端。大概函数调用流程如下：

RemoteEstimatorProxy::IncomingPacket↓

RemoteEstimatorProxy::Process↓

RemoteEstimatorProxy::SendPeriodicFeedbacks↓

RemoteEstimatorProxy::BuildFeedbackPacket

RemoteEstimatorProxy::IncomingPa
cket
中，如果RTP包带有TransportSequenceNumber扩展，会记录该RTP包的到达时间，然后添加到构造的Transport-cc报文中。我们看下主要处理的代码：