

第五章 RTP控制协议

- RTCP的组成
- RTCP包如何传输
- RTCP包格式
- 安全和隐私
- 包校验
- 参与者数据库
- 时序规则

RTP中主要有两部分，分别是在第四章中描述的数据传输协议和本章描述的控制协议。

控制协议RTCP，用于定期报告接收到数据的质量、参与者标识、其他源的描述信息、会话成员改变的通知以及同步媒体流所需要的信息。

RTCP的组成

RTCP的实现分为三部分，数据包格式、时序规则以及参与者数据库。

有几种类型的RTCP包。本章后面标题为《RTCP数据包格式》的小节中介绍了这五种标准数据包类型，以及必须将它们汇总到复合数据包中以方便进行传输的规则。在标题《数据包校验》部分中介绍了可以检查RTCP数据包正确性的算法。

根据本章后面的标题为《RTCP时序规则》的部分中所述的规则，定期发送复合数据包。数据包之间的间隔我们称之为传输间隔。所有RTCP活动都发生在整数倍的间隔时间内。除了作为包间隔的时间外，它还是计算接收质量统计信息的时间，以及更新源描述和音视频同步的时间。时间间隔会根据所使用的媒体格式和会话的大小而有所不同。通常，小型会话连接的间隔时间大约在5秒，但是大型的会话会增加到几分钟。在计算报告发送间隔时优先从发送方考虑，因此发送方的发送源信息和音视频同步信息发送得比较频繁。而接受者的上报通常较少。

每一个实现都应该基于从RTCP包接收到信息维护一个参与者数据库（Participant database），此数据库除了用于定期填写发送的接收报告的数据包外，还用于在接收到音频和视频流之间进行音视频同步，并且维护源描述信息。本章后面的标题为《安全和隐私》的部分中，会提到参与者数据库中的隐私问题。同样，在本章中的《参与者数据库》部分描述了如何维护参与者数据库。

RTCP数据包传输

每一个RTP的会话都是由一个网络地址和一对端口号做为标识：一个端口用于RTP数据，另一个端口用于RTCP数据。其中RTP数据端口应为偶数，RTCP端口应该是RTP端口加1，例如，如果媒体数据正在使用UDP5004端口，那么控制信道将在与RTP相同的IP地址上的端口5005上被发送

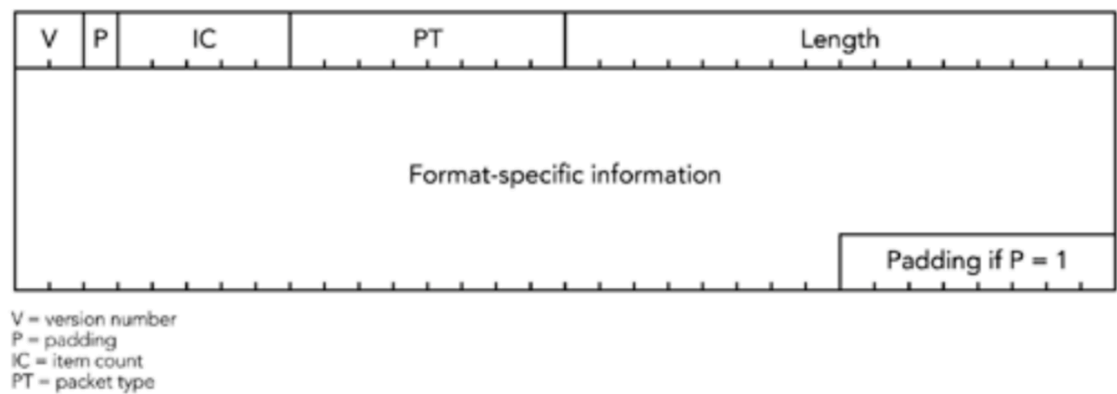
所有的会话参与者都必须发送复合的RTCP包，然后接收其他参与者传输过来的复合数据包。注意，接收后的应答信息（feedback）是需要发送给多方会话中的所有参与者的。反馈信息可以单播（unicast）的形式发送给转换器（translator）再通过转换器分发，也可以通过多播发送。RTCP的P2P特性让会话中的每个参与者了解其他参与者的信息、它们的存在、接收质量以及可选的个人信息如姓名、邮箱、地址和手机号。

RTCP包格式

RTCP协议规范中定义了五种类型的RTCP包：接收方报告（RR）、发送方报告（SR）、源描述（SDES）、成员管理（BYE）和应用程序定义（APP）。它们遵循统一的格式（图5.1），其中Format-specific信息，每种类型都不同。

图5.1 基本的RTCP包格式

Figure 5.1. The Basic RTCP Packet Format



五种包类型的包头都是4个八位字节组成的，包含五个字段。

- 版本号（V）：对于当前版本的RTP协议，版本号为2（截止到本书编纂为止），目前还没有推出新版本的计划，并且之前的版本并没有广泛的被使用

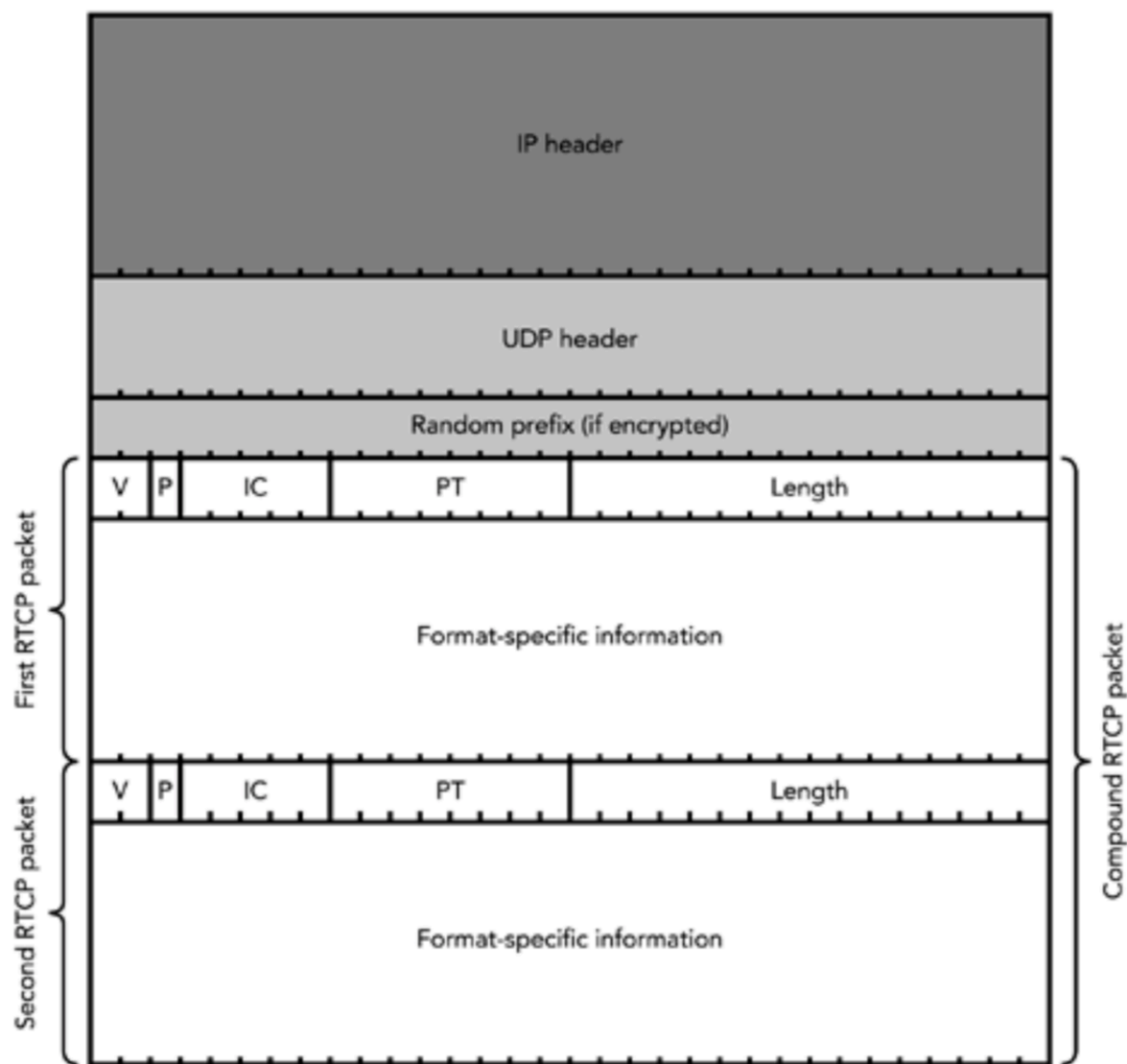
- 填充 (P)：填充位表示，所要填充的数据已经超出了目前所能容纳的位数。如果此位被设置为1，那么意味着包尾已经被一个或多个八位字节填充，最后一位八位所填充的内容表示此包的总数大小。它的用法和RTP数据包中的填充为非常类似，在第四章《RTP数据传输协议》的填充位一节中有关于此位的介绍。错误的填写此位是RTCP协议使用过程中常见的问题，我们会在本章后面的《打包问题和包校验》中描述正确的用法。
- 条目计数 (IC)：某些包类型中包含了一个list的条目，可能作为固定的、用于特定类型的信息的补充。这些条目字段需要标示出包中包含的条目的总数（这个字段在不同的包中有不同的命名方法，这取决于具体如何使用此字段）。每个RTCP包最多包含31个条目，同时也受到MTU (maximum transmission unit) 的限制。如果需要传输超过31个条目的场景，那么应用程序必须生成多个RTCP包。Item Count字段为0的时候表示此包中的条目为空（但是并不意味着包中内容为空）。如果不需要Item count字段那么此字段可以用于其他的目的。
- 包类型 (PT)：此字段标识了传输的包中所携带的信息的类型。在RTP的规范中定义了五种标准数据包类型，将来可能还会定义其他的类型（例如，报告额外统计信息或者传递其他特定源的信息）。
- 长度：此字段标识包头之后的内容总长度。因为所有的RTCP的数据包的长度必须为32位的整数倍，所以这个字段放的是32位字的个数，因为如果按照八位字节计算会出现此字段和总长度不一致的情况。0是一个有效长度，表示这个包只包含4个8位字节的包头（包头字段IC在这种情况下也是0）。

RTCP的包头之后存储的是包具体数据（其具体格式要根据包类型来决定）和可选填的填充字段。包头和后面的具体数据组成了一个完整的RTCP包。下面几个小节将详细描述五种标准类型的RTCP包。

RTCP包不会单独的被传输。它需要打包在一起形成复合包 (compound packets) 进行传输。每一个复合包都会被一个底层的包封装（通常是UDP/IP包）用来传输。如果要对复合包进行加密，那么RTCP的包组的前缀通常是一个32位的随机数。复合包的结构如图5.2所示。

5.2 RTCP包中复合包的结构

Figure 5.2. Format of a Compound RTCP Packet



RTCP的复合包的组成由一组规则来控制。在对五种RTCP包进行更详细的描述之后，这些规则会在本章的“打包问题”中进行描述。

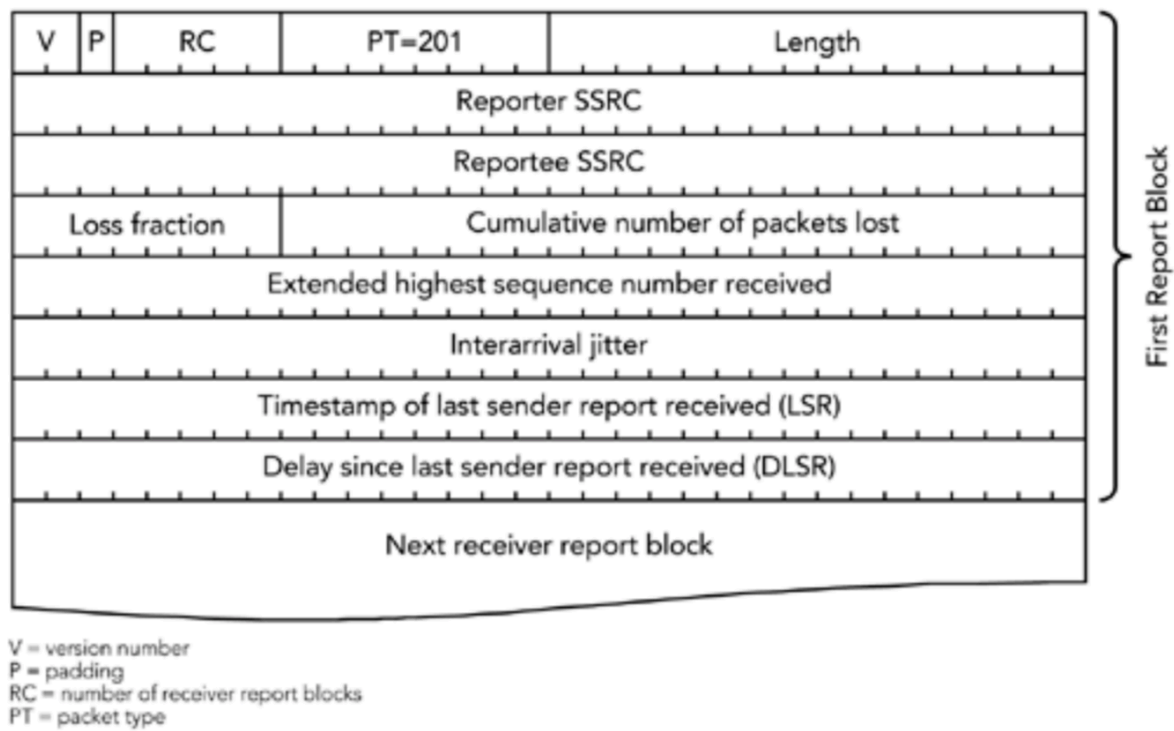
RTCP中的RR：接收报告（receiver reports）

RTCP的主要用途之一就是接收质量报告，它通过接收数据的所有参与者发送的RTCP的接收报告（RR）包来完成。

RTCP协议中RR数据包格式

接收报告由类型为201的包来标识，起格式如图5.3所示。接收方报告中包含发送报告的参与者（报告者SSRC）的SSRC（同步源），后面跟着0个或者多个报告块，用RC字段来表示。

Figure 5.3. Format of an RTCP RR Packet



许多RTCP的包类型在固定部分的后面有一个条目列表（Item List），这个结构类似于接收方报告。所以此处需要注意，即使条目列表是空的，数据包的固定部分仍然保持不变，这也就是说，如果接收方报告中没有报告块（report block），那么要将数据包中的RC设置为0，长度设置为1，对应4个8位字节的固定RTCP包头，外加4个8位的报告者SSRC。

每一个报告块（report block）都是描述单个同步源的接收质量，而报告者（reporter）从当前报告的间隔期间，接收从该同步源发过来的RTP包。每一个RTCP的RR包总共有31个报告块。如果有超过31个激活的发送者，那么接收者应该在一个复合数据包中发送多个RR数据包，每个报告块有7个字段，总共24个字节。

Reportee（被报告者）SSRC标识此报告块相关的参与者。报告块中的统计数据，表示的是在生成RR数据包的参与者处，被报告方接收到的同步源的数据包的接收质量。

累计丢包数是一个24位带符号的整数，它表示预期应该到达的包的数量，减去实际接收到的包的数量。预期的包数的定义是，最后接收到的扩展序列号，减去接收到的初始序列号。接收到的包的总数包括任何延迟到达或者重传过来的包，因此可能会大于预期的数量，因此累计丢包数有可能是负值。累计丢包数的计算区间是统计的整个会话期间的，而不是在每个间隔期间。如果在会话期间丢包的总数大于0x7FFFFFFF，那么此字段会在0x7FFFFFFF处于最大饱和值。

许多RTCP的统计信息是基于整个会话期间，而不是报告间隔的区间。但是，如果发生了SSRC冲突，或者序列号空间中存在一个非常大的间隙，使得接收方无法判断此字段是否已经被回绕，那么统计信息重置为0。

在同步源的RTP数据包中接收到的扩展最高序列号（extended highest sequence number）的计算，是在第四章《RTP数据传输协议》的序列号一节中讨论的方法。由于可能存在包重新排序的情况，所以并不一定是接收到的最后一个RTP包的扩展序列号。扩张序列号是基于会话计算的，而不是基于包间隔计算的。

丢包率（loss fraction）的定义是在这个报告间隔中所丢失包的数量，除以预期到达的数量。丢包率表示为一个定点数，该定点数的二进制小数点位于字段的左边缘。即丢包率乘以256后的整数部分（即如果传输中有1/4的包丢掉，那么丢包率应该是 $1/4 * 256 = 64$ ）。如果接收到的包的数量大于预期（由于存在重复包的情况），使得丢包数为负值，那么丢包的部分设置为0。

到达间隔抖动（Interarrival jitter）是对被报告者（Reportee）同步源发送的数据包的网络传输时间统计方差的估计。它是以时间戳单位衡量的，因此它像RTP时间戳一样用32位无符号整数表示。

为了测量网络传输时间的方差，有必要对传输时间进行测量。由于发送方和接收方通常都没有设置同步时钟，所以无法监测绝对的传输时间。相对传输时间是在相同的单位度量内，计算包到达时数据包的RTP时间戳和接收方的RTP时钟之间的差值。这个计算需要接收方为每一个源维护一个时钟，并与该源的媒体时钟以相同的速率运行，从该时钟获取时间戳。（这个时钟可能是接收方本地播放时钟，如果它与源的时钟运行速率相同的话）由于发送方和接收方之间的时钟缺少同步机制，相对传输时间包含一个未知的恒定偏移。但是这不

是问题，因为我们仅仅对传输时间的方差感兴趣：即接收方两个数据包的时间戳与离开发送方时的时间戳间距之差。在后面的计算中，由于不同步时钟引起的恒定偏移，会被减去。

如果 S_i 是来自包 i 的RTP时间戳， R_i 是包 i 到达的时间（以RTP时间戳单位），那么相对传输时间就是 $(R_i - S_i)$ ，对于两个包 i 和 j ，相对传输时间的差异可以表示为：

$$D(i, j) = (R_j - S_j) - (R_i - S_i)$$

注意，时间戳 R_x 和 S_x 是32位无符号整数，而 $D(i, j)$ 是有符号的整数。使用模运算来计算（在C语言中，这意味时间戳类型为unsigned int, 假设 `sizeof (unsigned int) == 4`）。

当每个数据包被接收后，使用该数据包与前一个数据包（不一定是按照序列号顺序的前一个数据包）的相对传输时间 $D(i, j)$ 之差来计算到达间隔抖动（interarrival jitter）。根据以下公式，抖动是以滑动平均的形式存在。

$$J_i = J_{i-1} + \frac{(|x_{i-1}, i| - J_{i-1})}{16}$$

无论何时生成接收报告，被报告对象SSRC的 J_i 的当前值就作为到达间隔抖动（interarrival jitter）。

最后一个发送者报告（last sender report, LSR）时间戳是64位NTP（网络时间协议（Network Time Protocol））格式的时间戳中间的32位，包含在最近从被报告者的SSRC接收到的RTCP的SR包中。如果SR没有收到，那么此字段可以设置为0。

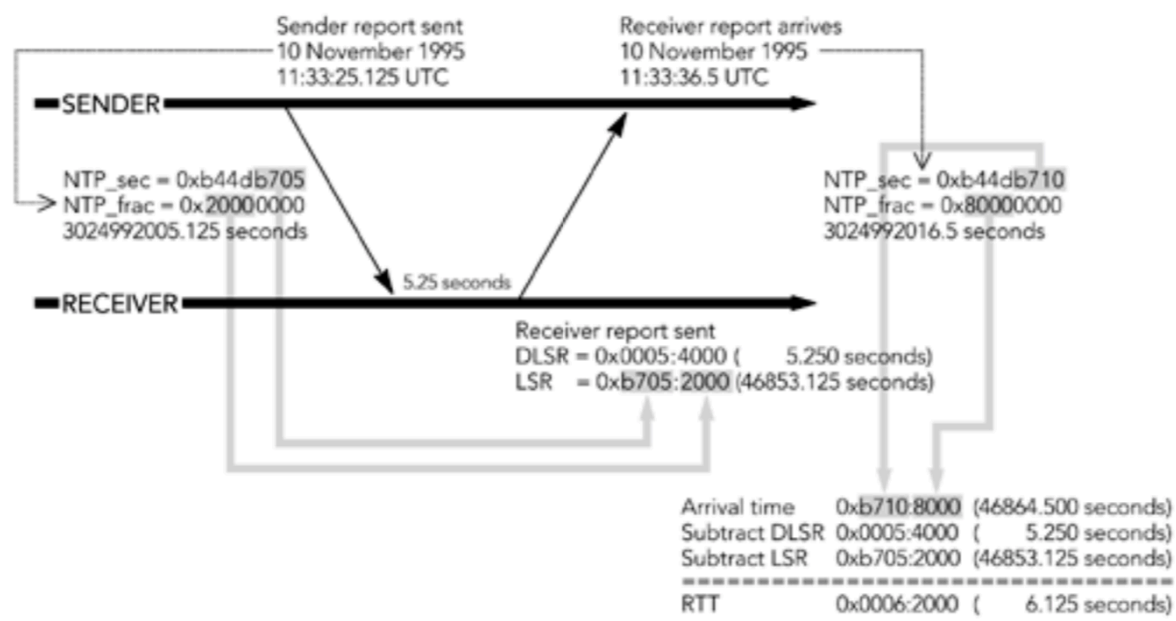
自上次发送者报告起的延迟（delay since last sender report, DSLR）是从被报告者SSRC接收到最后一个SR数据包到发送此接收报告块之间的延时，以1/65,536秒为单位。如果没从该被报告者收到SR，则DSLR字段设置为0。

RR数据解析

RR包中的接收质量反馈不仅对发送方有用，对其他参与者和第三方监控工具也有用。RR包中提供的反馈允许发送方根据反馈调整其传输。此外，其他的参与者可以确定当前产生的问题，是自己的还是多个接收方共有的，网络管理人员可以使用仅接收RTCP包的监控器来评估其网络的性能。

发送方可以使用LSR和DLSR字段来计算它与每个接收方之间的往返时间（rtt）。当接收到一个与之相关的RR包时，发送方用当前的时间减去LSR字段，以得到发送SR到接收此RR之间的延迟。发送方然后再减去DLSR字段以消除接收方延迟带来的偏移，从而获得网络往返时间。该流程如图5.4所示，这是取自RTP规范的一个示例。（注意，RFC1889标准中有一个错误，此错误已经在RTP新版本中得到更正）

Figure 5.4. Sample Round-Trip Time (RTT) Computation.



注意，计算的结果是网络往返时间（RTT），它不包括在端点上的任何处理时间。例如，接收方必须缓冲数据以用来消除抖动所带来的影响，然后才能去播放媒体（参见第六章《媒体采集、播放和时序》）。

由于延迟会阻碍交流，所以RTT在应用程序的交互中是相当重要的。研究表明，当RTT超过300毫秒（这个时间是大概的，这取决于通话双方以及正在做的事情）时，很难进行对话。发送方可以利用RTT来优化媒体编码，例如通过生成包含较少数据的包来减少打包延迟，或者推动使用纠错码的方式（参见第九章《错误恢复》）。

丢包率（loss fraction）指示的是接收方短期的丢包，通过观察报告统计中的趋势，发送者可以判断丢包是短暂的还是长期的。RR数据包中的许多统计数据是累加值，以便进行长期均值的计算。通过对比两个RR数据包之间的差异，可以进行长期和短期的评估，从而让丢包报告更灵活。

例如，我们可以通过统计数据累加值得到的RR数据包间隔的丢包率，也可以直接上报丢包率。丢包的累加值的差值表示该时间间隔内的丢包数量，扩展后的最后序列号的差值，表示该时间间隔期望收到的包数。这两个值的比例就是丢包率。如果使用连续的RR包进行计算，这个数字应该等于RR包中的Loss Fraction字段，但是这个比率也给出了对一个或者多个RR包丢失时的丢包率的预估，当有重复的包时，它可能显示负的丢失率。使用Loss Fraction字段的优点是它可以用单个RR包提供丢包信息。这一点在大型会话中非常有用，在这些会话中，如果报告的间隔过大，可能会导致暂时没法收到两个RR数据包。

可以根据丢包率来设置适当的媒体格式和纠错方式（参见第九章《错误恢复》）。特别是，更高的丢包率意味着，我们应该选择更能容错的格式，并且如果可能的话，应该降低数据传输速率（因为大多数丢包是由于拥塞引起的；参见第二章《分组网络中的音视频通信》以及第十章《拥塞控制》）。

抖动（jitter）字段可以用来检测网络中拥塞的开始，抖动的突然增加常发生在丢包开始之前。抖动的影响取决于网络拓扑结构和流的数量，高程度的统计多路复用降低了增加的抖动和数据包拥塞发生之间的相关性。

发送方应该注意，对抖动的预估应该取决于发送的数据包的间隔和他们的时间戳的匹配。如果发送方延迟发送某些数据包，则该延迟应该被视为网络抖动的一部分。这可能是视频的一个问题，在视频领域，多个包通常产生相同的时间戳，但这些包是间隔传输而不是突然一起发送。当然，这也不一定是一个问题，因为对抖动的测量还会给出关于接收方所需要的缓冲空间的估计（因为缓冲空间需要适应抖动和发送延迟）。

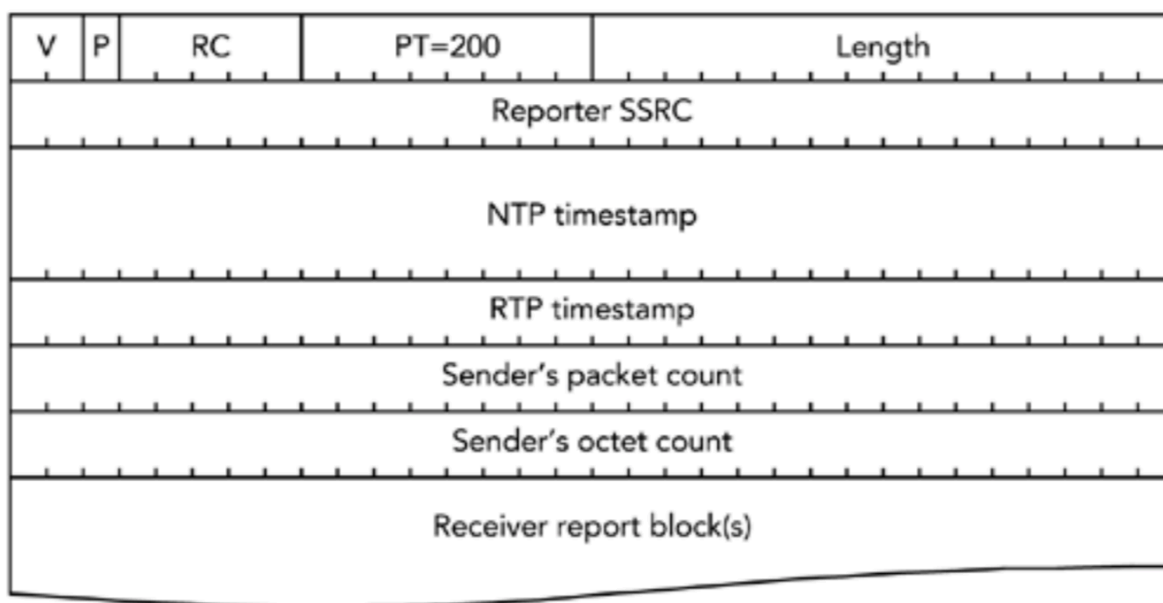
RTCP SR：发送报告

除了来自于接收者的接收报告之外，RTCP还负责传输最近发送数据的参与者发送的发送者报告（SR）包。他们提供正在发送的媒体信息，主要是为了使接收方能够同步多个媒体流（例如，音视频同步）。

RTCP中SR包的格式

发送者报告的包类型为200，其格式如图5.5所示。有效负载包含一个24字节的发送者信息块，后面跟着0个或多个接收方报告块，由RC字段标识，类似于接收方报告报。当发送者也是接受方的时候，接收方报告块就出现了。

Figure 5.5. Format of an RTCP SR Packet

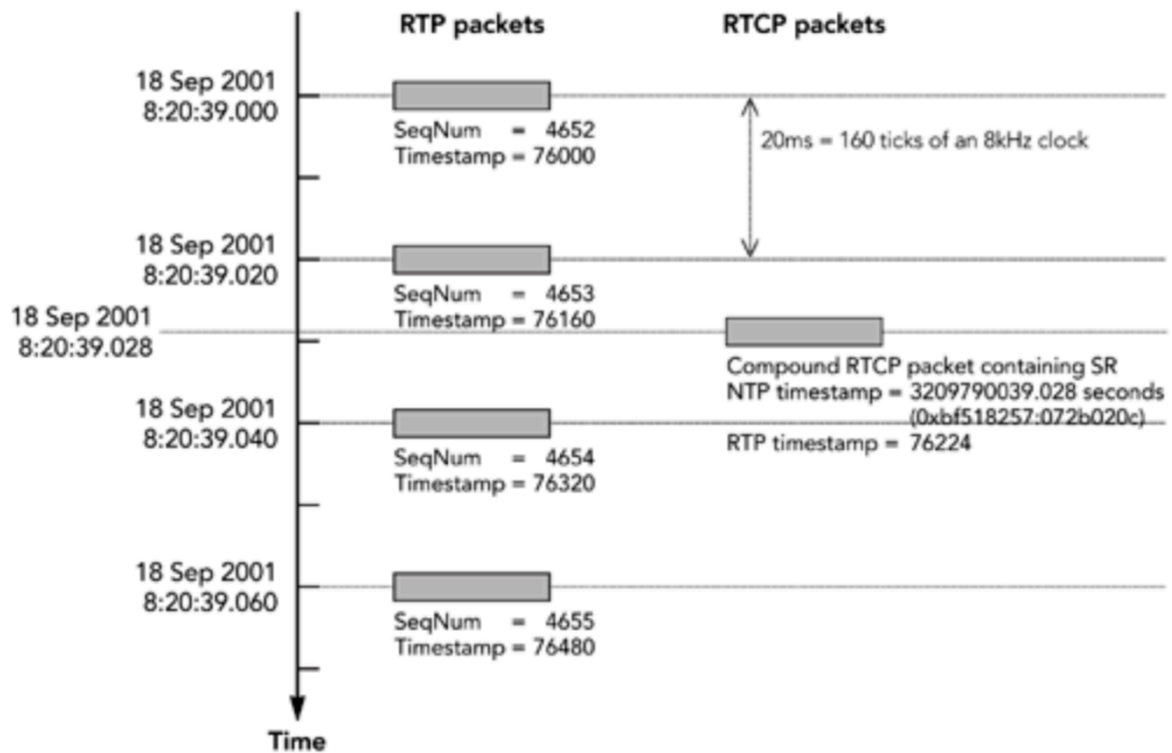


NTP时间戳是一个64位的无符号值，表示发送这个RTCP SR包的时间。它的格式是NTP时间戳，时间从1900年1月1日开始计算秒，低32位代表秒的小数部分（fractions of second）（也就是64位定点值，二进制小数点位于32位之后）。如果要将UNIX的时间戳（从1970年1月1日开始的秒数）转化为NTP时间，那么需要添加2,208,988,800秒。

虽然RTCP的SR包的NTP字段使用了NTP时间戳格式，但是，时钟并不见得必须要与网络时间协议（Network Time Protocol）同步，也不必具有任何特定的精度、分辨率及稳定性。但是，对于要同步的两个媒体流的接收方，这些流必须要有相同的时钟。NTP协议有时对同步发送时钟很有用，但是只有当要同步的媒体流由不同的系统生成时才需要它。这些问题将在第七章《音视频同步》中进一步讨论。

RTP时间戳与NTP时间戳的对应的时间是相同的，但是，它是以RTP媒体时钟的基准单位表示的。这个值，通常与前一个数据包的RTP时间戳不同，因为自该数据包中的数据被采样已经经过了一段时间了。图5.6显示了SR包时间戳的一个示例。SR包具有与发送它的时间相对应的RTP时间戳，这与前后的RTP数据包都不对应。

Figure 5.6. Use of Timestamps in RTCP SR Packets



发送方的包计数，是这个同步源自会话开始以来，生成的数据包的总数。发送方的字节计数是这些数据包的有效负载（payload）中包含的字节数（不包括包头或者填充）。

如果发送方改变其SSRC（例如，由于产生冲突），则会重置发送方的包计数以及字节计数字段。如果发送方传输很长的时间，最终会产生回绕，但是这通常不会造成问题。如果使用32位模运算，且两者之间的计数不超过2的32次方，则即使有回绕，从新的值中减去老的值也将得到正确的结果（在C语言中，计数器的类型为无符号整型sizeof(unsigned int) == 4）。对发送方的数据包计数和字节计数，可以使得接收方能够计算发送方的平均数据速率。

SR数据的解析

应用程序可以在不接收数据的情况下，通过SR信息，计算间隔中的平均有效负载数据速率和平均包速。两者的比值就是平均有效负载大小。可以假设数据包的丢失与数据包大小无关，特定的接收方接收的数据包数量乘以平均有效负载大小（或相应的数据包大小），就可以得出该接收方可用的表现吞吐率（apparent thoughtput）。

时间戳是用来生成一个媒体时钟和已知的外部参考（NTP格式时钟）的对应。因此这才使得音视频同步成为可能，如第七章所述。

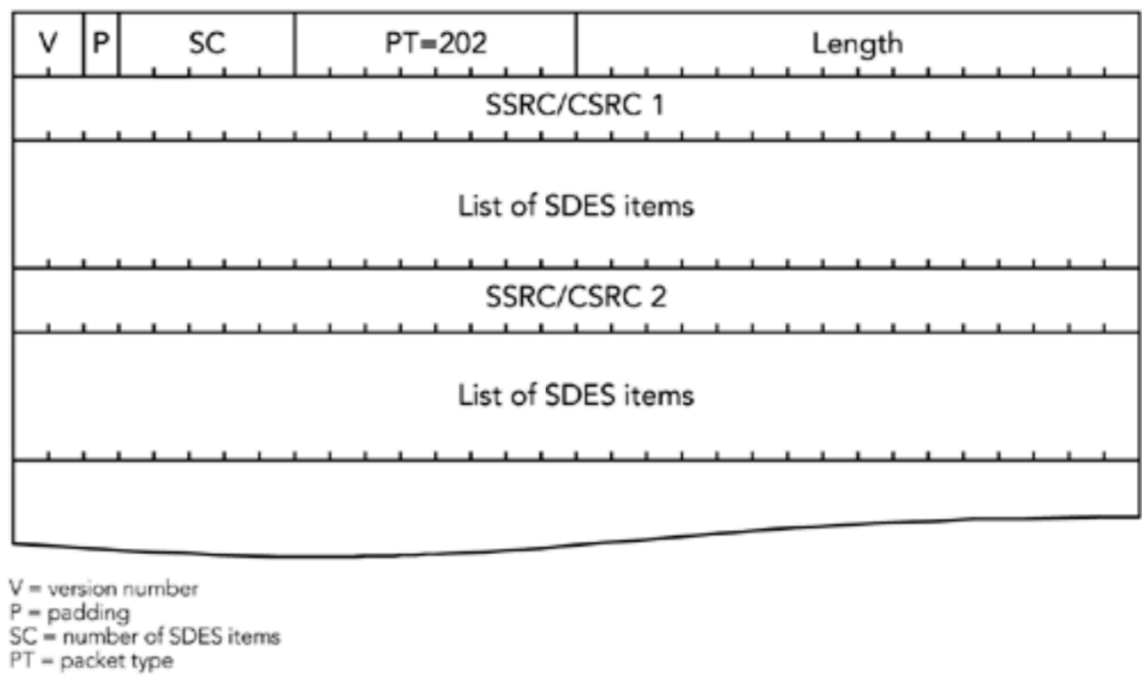
RTCP SDES：源描述（Source Description）

RTCP也可以用来传递源描述（SDES）数据包，提供参与者认证和补充性细节，如位置、电子邮件地址和电话号码。SDES包中的信息通常由用户输入，通常显示在应用程序的图形用户界面中，尽管这取决于应用程序的性质（例如，一个系统提供从电话系统到RTP的网关，它可能使用SDES包来传递呼叫者ID）。

RTCP中SDES包格式

每个源描述包在RTCP包中类型定义为202，具体如图5.7所示。SDES包包含0个或者多个SDES项列表，具体个数由包头SC字段表示，每个SDES项包含一个源的信息。

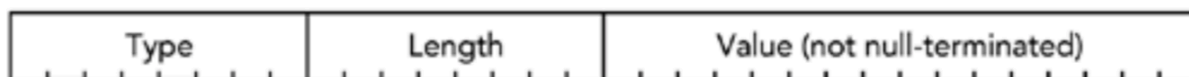
Figure 5.7. Format of an RTCP SDES Packet



应用程序有可能生成SDES项为空列表的包，在这种场景下，RTCP公共包头中的SC和length字段都为0.在正常情况下，SC应该为1（混流器（mixers）和转换器（translators）所聚集的转发信息产生的包会有较大的SDES项的列表）。

每个SDES项目列表从所描述的源的SSRC开始，然后是一个或者多个条目，具体格式参见图5.8所示。每个条目以类型和长度字段开始，然后是UTF-8格式的项目文本本身。length字段标识文本中有多少字节；文本不是以NULL结尾的。

Figure 5.8. Format of an SDES Item



每个SDES项中的条目都是以连续的方式打包到包中，没有分隔或者填充。条目列表（list of item）以一个或者多个空的字节结束，当解析到第一个字节为0类型的时候，意味着这个列表结束。0类型字节后面不会跟长度字节，但是如果需要填充，则包括其他的空字节，直到达到32-bit边界为止。这个填充（padding）与RTCP包头中的P位表示的填充是分开的。带有零项的列表（四个空字节）是有效的，但是没有意义。

在RTP规范中定义了几种类型的SDES条目，其他的条目可能在未来被定义。条目类型为0的是预留项，标识了条目列表的结束。其他标准条目类型有CNAME、NAME、EMAIL、电话、LOC、工具、注释和PRIV。

SDES的标准条目

CNAME项（type = 1）为每个参与者提供了一个规范名称（CNAME）。它提供了一个独立于同步源的稳定且持久的标识符（因为如果应用程序重启或发生SSRC冲突，SSRC将改变）。CNAME可以用于关联来自不同RTP会话的参与者的多个媒体流（例如，关联需要同步的语音和视频），以及在媒体工具重启时命名参与者。这是唯一的强制性的SDES条目，所有实现都需要发送SDES CNAME项。

CNAME是根据参与者的用户名和主机ID地址并通过算法进行分配的。例如，如果作者是使用基于IPv4的应用程序，那么CNAME可能是csp@10.7.42.16。如果是IPv6应用，那么使用冒号分隔的数字形式的地址。如果应用程序运行在一个没有用户名概念的系统上，则只使用主机IP地址（无用户名或@符号）。

只要每个参与者只加入一个RTP会话（或一组要同步的相关会话），使用用户名和主机的IP地址，就可以生成一致的唯一标识符。如果要同步来自多个主机或多个用户的媒体流，则这些流的发送方必须联合起来生成一致的CNAME（通常是由一个参与者通过算法选择的名称）。

使用私有地址和网络地址转换（NAT）服务意味着，IP地址会存在全局不唯一的情况。为了使音视频同步和相关的RTP会话的其他用途能正确的被NAT操作，转换器（translator）还必须在穿过域边界（domain boundaries）时将RTCP CNAME转换为唯一的形式。这个转换在跨越多个RTP流时必须是一致的。

NAME项（type = 2）表示参与者的名称，主要用于在参与者列表中显示，做为用户界面的一部分。这个值通常由用户输入，因此应用程序一般不假定它是任何值；特别的它不应该被认为是唯一的。

EMAIL项（type = 3）负责传递参与者的电子邮件地址，格式符合rfc822 - 例如,jode@example.com。应用程序在发送前，应该验证电子邮件的值是否符合电子邮件的语法，然后再将其包含在SDES中。因为接收方不会假定这个地址的拼写是有效的。

PHONE项（type = 4）表示参与者的电话号码。RTP规范建议用一个完整的国际电话号码，它带有一个加号来代替国际访问码（例如，+1 918 555 1212是一个美国的号码），但是许多实现允许用户输入这个值而不需要去校验格式。

LOC项（type = 5）表示参与者的位置。许多实现允许用户直接输入值，但是可以用各种格式来表示位置。例如，有一个实现方案与全球定位系统相连，并将GPS坐标做为位置信息。

TOOL项（type = 6）表示参与者使用RTP实现 - 工具（implementation - the tool）。此字段主要用来调试和营销目的（marketing purposes）。它应该包含实现的名称和版本号。通常，用户无法编辑此字段的内容。

NOTE项（type = 7）允许参与者对任何事做一个简短的陈述。对于“非即时回复的消息（back in five minutes）”它可以很好的工作。但是它不能作用于即时通讯的场景，因为RTCP包之间存在很大的延迟。

PRIV项（type = 8）是一种私有的扩展机制，用于定义实验的或特定于应用程序的SDES扩展。此项的文本，从一个额外的一个字节长度的字段和前缀字符串开始，后面紧跟着值字符串来填充其余部分。其目的是初始化扩展前缀名和后面的扩展字段值。PRIV这个项很少被使用，如果定义了新的SDES类型，那么可以更有效的管理扩展。

CNAME是应用程序必须要传输的唯一的SDES项（其他项可选）。在实现中应该准备接收任何SDES项，即使忽略了它们。SDES存在各种隐私问题（请参阅本章后面的《安全和隐私》一节），这意味着除非得到用户明确的授权，否则在实现中不应该发送除了CNAME之外的任何信息。

图5.9显示了一个包含CNAME和NAME项的完整RTCP源描述包的示例。请注意在SDES项list的末尾使用了填充（padding），以确保数据包是32bits的倍数。

Figure 5.9. A Sample SDES Packet

V	P	SC=1	PT=202	Length=10			
SSRC							
Type=1 (CNAME)		Length=15		c		s	
p		@		1		0	
.		7		.		4	
2		.		1		6	
9		Type=2 (NAME)		Length=13		C	
o		l		i		n	
		P		e		r	
k		i		n		s	
Type=0		0		0		0	

解析器（parser）问题

在解析SDES包的时候，有三点需要注意：

- 1. SDES条目的文本最后不是以null作为结尾的，这也就是说在以null为结尾的字符串的语言中操作SDES条目需要特别小心。例如，在C语言中，应该使用strncpy()操作，因为它允许拷贝指定长度的字符串（使用strcpy是危险的，因为文本不是以null结尾的）。如果不小心使用，非常容易造成缓冲区溢出，这是非常严重的问题
- 2. SDES条目的文本使用UTF-8格式，本地字符集在使用前需要转换。在使用前应该先查询本地系统使用的语言环境，之后再在系统字符集和UTF-8之间进行转换。很多应用程序会在不经意间生成错误的字符并输入到SDES包中。实现中要对这种错误做容错，并使

其更加健壮（例如，如果使用不正确的字符集，会导致UTF-8解析出无效的Unicode字符）。

- 3. SDES项的文本由于可能被用户操作，所以不能相信其具有安全性。特别是它还可能包含一些有意向不到的副作用的通配符（metacharacters）。例如，一些用户使用脚本语言允许触发通配符并进行指令替换，这种方法可以使攻击者有办法执行任意的代码。所以，实现中应该采取措施确保其环境可以安全的处理SDES数据。

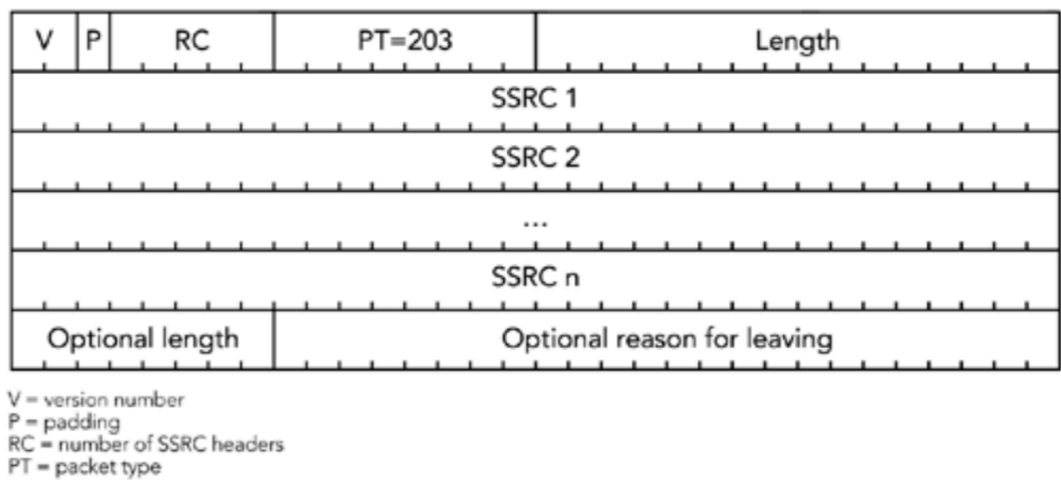
RTCP 释放连接(bye)：成员控制

RTCP通过RTCP的Bye包为成员提供松散的控制，如果收到Bye表明某些参与者已经离开了会话。Bye包是在参与者离开会话的时候，或者当其他参与者因为冲突而改变SSRC的时候生成的。Bye包有可能会在传输过程中丢失，并且有些应用程序也不能生成此包。所以，即便没有收到Bye包，针对一段时间没有活跃的参与者，接收方应该有超时机制。

Bye数据包的重要性在一定程度上取决于应用程序。它总是表明一个参与者正在离开RTP会话，但是参与者之间可能存在着另外的一种信号连接关系（例如，SIP、RTSP或者H.323）。RTCP BYE包不会终止参与者之间的任何其他关联关系。

Bye包的标识类型为203，其格式如图5.10所示。公共的RTCP包头中的RC字段表示SSRC标识符的数量。其存在为0的可能性，标识为0时无用。在接收到Bye包时，实现时应该假设此源已经离开了会话，并忽略来自该源的任何后续RTP和RTCP包。最重要的是，当收到Bye包之后，需要为此参与者保留一段时间的连接状态，因为要允许延迟到达的数据包被接收到。

Figure 5.10. Format of an RTCP BYE Packet



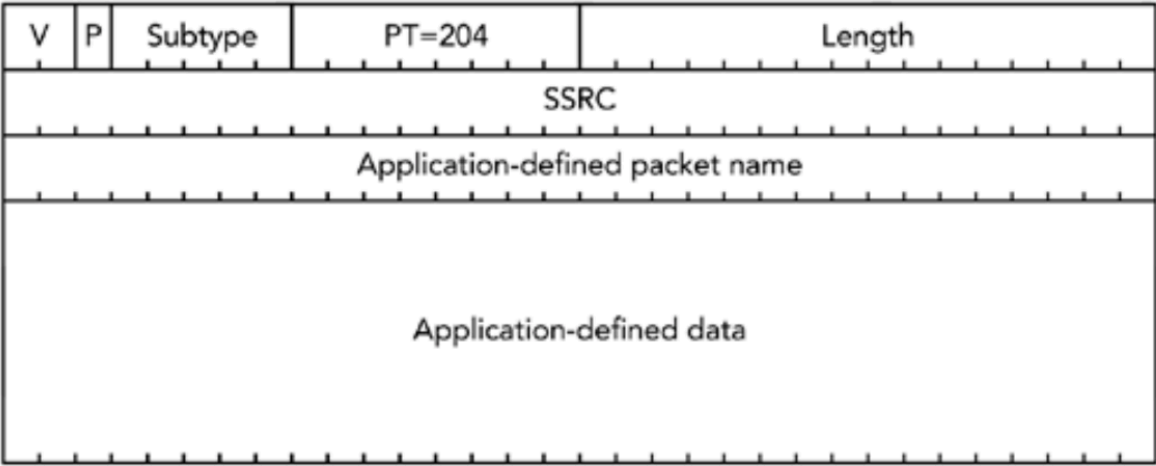
本章后面的《参与者数据库》一节，专门对参与者超时和RTCP Bye包相关的状态维护的问题进行了说明。

Bye包还可能包含了表示离开会话原因的文本，适合在用户界面中显示。但是，这个文本是可以选填的，在实现过程中我们需要去接收它（即使文本可能会被忽略）。

RTCP APP：应用程序定义的RTCP包

最后一类RTCP包（APP）允许应用程序来自己定义扩展。它的包类型为204，格式如图5.11所示。应用程序定义的包名（application-defined packet name）由4个字符组成唯一的标识，每个字符都得从ASCII字符集中选择，并区分大小写。建议选择包名称来匹配它所代表的应用程序，并由应用程序来协调子类型值的选择。包其余部分被用于应用程序的特定需求。

Figure 5.11. Format of an RTCP APP Packet



应用程序自定义的包用于RTCP的非标准扩展和验证新特性。目的是，验证者首先使用APP来验证新特性，然后如果新特性有广泛的用途，那么就注册为新的包类型。一些应用程序生成的包或实现方案，应该忽略识别不出来的应用程序包。

组包（Packing）问题

如前所述，RTCP包不会单独发送，而是组包成一个复合数据包进行传输，下面详细介绍下复合包的组包规则和结构。

如果生成复合RTCP包的参与者是活跃的数据发送方，那么该复合包必须以RTCP SR包开始。否则必须从RTCP RR包开始。即使还没有发送或接收数据，这也是正确的，在这种情况下，SR/RR包不会包含接收方的报告块（包头字段RC为0）。另一方面，如果从多个源接收数据，并且报告太多，导致无法放入一个SR/RR包，则复合后的数据应以一个SR/RR包开始，后面在跟着多个RR包。

跟在SR/RR包后面的是一个SDES包。这个包必须包含一个CNAME条目，它可能包含其他的条目。包含其他（非CNAME）SDES条目的频度由使用中的RTP配置文件决定。例如，音视频配置文件指定在发送的每三个复合RTCP包中可以包含其他条目，其中NAME条目在八次该slot中发送七次，其余的SDES循环占用第八个slot。其他配置文件可以指定不同的选择。

当准备好进行传输时，Bye包必须做为最后一个数据包发送。要发送的其他RTCP包可以按任何顺序。这些严格的排序规则，旨在使数据包的校验更容易，因为错误定向的数据包，大概率不会满足这些约束。

在生成复合RTCP包时，一个潜在的问题就是如何处理大量活跃发送者的会话。如果存在超过31个活跃的发送者，那么有必要在复合包中增加额外的RR包。这可以根据需要重复此过程，直到达到MTU的上限。如果发送方太多，以致于接收方报告不能被MTU容纳，则必须忽略某些发送方的接收报告。如果出现这种情况，那么被忽略的报告，应该在生成的下一个复合包中被包含（要求接收方跟踪每个间隔中报告的源）。

当包中包含的SDES条目超过最大包大小时，也会出现类似的问题。在包含额外接收报告还是包含源的描述信息之间的权衡，需要在实现中考虑。因为没有绝对正确的选择。

有时候需要将一个复合RTCP包填充并超出其原始大小。在这种场景下，填充只是添加到复合包中的最后一个RTCP包中，P位（P bit）在最后一个包中被设置。填充（padding）是某些实现中的不正确的部分。本章会在后面的《包校验》一节讨论常见的问题。

安全与隐私

在使用rtcp过程中，有很多涉及到隐私的问题，尤其是在源描述包中。尽管这些信息包是可选的，但是他们的使用可能会暴露个人重要的信息，因此，应用程序不应该在没有通知用户其信息被使用的情况下，发送SDES信息。

使用SDES CNAME包是一个例外，因为这些包是强制性的。在CNAME包中包含IP地址会存在一个潜在的问题。不过包的IP头也包含相同的信息。由于RTP包会通过NAT转发，所以IP协议头中的地址转换，也应该在CNAME中的地址上进行。但在实践的过程中，许多NAT的实现时，并不知道RTP，因此会存在内部IP地址泄露的问题。

用户名的暴露可能是一个更大的问题，在这种场景下，只要使用CNAME进行关联的应用程序之间可以保持一致，那么应用程序就可以重写或者忽略用户名。

一些接收方可能不希望他们的存在被其他方看到。如果这些接收方完全不发送RTCP那是可以的，但这样做就阻止了发送方使用质量信息来调整它们的传输以匹配接收方。

为了实现媒体流的保密性，可以对RTCP包进行加密，当加密时，每个复合包包含一个额外的32位随机前缀，如图5.12所示，以帮助避免明文攻击。

Figure 5.12. Example of an Encrypted RTCP Packet, Showing the Correct Use of Padding

Random prefix				
V	P=0	RC=1	PT=201 (RR)	Length=7
SSRC				
Reportee SSRC				
Loss fraction		Cumulative number of packets lost		
Extended highest sequence number received				
Interarrival jitter				
Timestamp of last sender report received (LSR)				
Delay since last sender report received (DLSR)				
V	P=1	SC=1	PT=202 (SDES)	Length=9
SSRC				
Type=1 (CNAME)		Length=15	d	o
e		@	1	0
.		5	1	.
2		.	2	2
3		Type=2 (NAME)	Length=9	J
o		n	n	y
		D	o	e
Type=0		0	0	0
0 (padding)		0 (padding)	0 (padding)	4 (padding count)

隐私和安全会在第13章《安全注意事项》中进行详细的讨论。

包校验

确认接收到的数据包是RTP还是RTCP这非常重要。前面提到的封包规则，允许严格校验RTCP包。RTCP流的成功验证可以高度的保证其所对应的RTP流也是有效的，但这并不是说RTP包的校验就不需要了。

List 5.1 展示了校验过程的伪代码，以下是要点：

- 所有的包必须是复合的RTCP包。
- 所有包的版本号必须等于2
- 复合包中的第一个RTCP包的包类型必须等于SR或者RR
- 如果需要填充，只添加到复合包中的最后一个包。复合RTCP包中所有的其他包的填充标志位必须为0.
- 各个RTCP数据包的长度字段，必须总计接收到的复合RTCP数据包的总长度。

因为未来可能会定义新的RTCP包类型，所以验证过程不应该要求每个包类型必须是RTP规范定义中的五种包类型之一。

List5.1 包校验的伪代码

```
1  validate_rtcp(rtcp_t *packet, int length)
2  {
3      rtcp_t *end = (rtcp_t *) (((char *) packet) + length); rtcp_t *r = packet;
4      int l=0;
5      int p=0;
6      // All RTCP packets must be compound packets
7      if ((packet->length+ 1) * 4) == length)
8      {
9          ... error: not a compound packet
10     }
11     // Check the RTCP version, packet type, and padding of the first
12     // in the compound RTCP packet...
13     if (packet->version != 2)
14     {
15         ...error: version number != 2 in the first subpacket
16     }
17     if (packet-> p != 0)
18     {
19         ...error: padding bit is set on first packet in compound
20     }
21     if ((packet->pt != RTCP_SR) && (packet->pt != RTCP_RR))
22     {
```

```

23     ...error: compound packet does not start with SR or RR
24 }
25 // Check all following parts of the compound RTCP packet. The RTP
26 // version number must be 2, and the padding bit must be zero on
27 // all except the last packet.
28 do
29 {
30     if (p == 1)
31     {
32         ...error: padding before last packet in compound
33     }
34     if (r-> p)
35     {
36         p = 1;
37     }
38     if (r-> version != 2)
39     {
40         ...error: version number != 2 in subpacket
41     }
42     l += (r->length + 1) * 4;
43     r = (rtcp_t *) (((uint32_t *) r) + r->length + 1);
44 } while (r < end);
45 // Check that the length of the packets matches the length of the
46 // UDP packet in which they were received...
47 if ((l != length) || (r != end))
48 {
49     ...error: length does not match UDP packet length
50 }
51     ...packet is valid
52 }

```

一个常见的实现问题将导致包的有效性测试失败：当填充复合RTCP包超出其自然长度时，需要确保只将填充添加到复合包中的最后一个包。一个常见的错误是在最后一个包中添加了填充（padding），而在复合包的第一个包的头中设置了P位。必须只在最后一个数据包中设置P位。

可以通过包类型字段，检测到RTCP包被发送到RTP端口上的错误。标准的RTCP包会在高位封装包的类型值；如果他们被错误的发送到RTP端口，那么高位的包类型字段，会落在RTP包头的M位。去掉顶部位后，标准的RTCP包类型对应于范围在72到76的RTP有效负载类型。这个范围在RTP的规范中属于保留位，不会用来传输有效的RTP包，因此如果检测到这个范围内的数据包，意味着流被发错了。类似的，发送到RTCP端口的RTP包可以通过包类型来区分，其类型超出了RTCP包类型的有效范围。

参与者数据库 (Participant Database)

RTCP会话中的每个应用程序将维护一个关于参与者和会话本身信息的数据库。RTCP时序从会话信息中获得，会话信息将被存储为一组变量：

- RTP带宽 -- 即在应用程序启动时配置的典型会话带宽
- RTCP带宽占比 -- 也就是RTCP报告在RTP带宽中的百分比。通常为5%，但是可以通过修改配置文件的方法来更改这一比例（也可以配置为0%，但是这并不意味着不传输RTCP数据包）。
- 该参与者发送和接收的所有RTCP数据包的平均大小
- 会话中成员的数量，该参与者上次发送RTCP数据包时的成员数量，以及在前一段报告间隔期间发送RTP数据包的成员比例。
- 程序最后一次发送RTCP包的时间，以及下一次预定的传输时间
- 一个标志位，指示了自发送最后两个RTCP数据包以来，程序是否发送了任何RTP数据包。
- 一个标志位，表明是否实现（implementation）发送了任何RTCP数据包。

此外，实现时还需要维护RTCP SR包中包含的变量：

- 发送的RTP数据的包和字节数
- 使用的最后一个序列号
- 正在使用的RTP时钟与NTP格式时间戳之间的对应关系

包含这些变量的会话的数据结构也适合存储：正在使用的SSRC、程序的SDES信息以及RTP和RTCP套接字的文件描述符。最后，会话数据结构应该包含一个数据库，用于保存每个参与者信息。

在实现方面，可以简单的存储会话数据：C语言的结构体，以及面向对象语言中的类。除了参与者特定的数据之外，结构体或类中的每个变量都是一个简单的类型：整型、文本字符串等。接下来描述参与者特定数据的格式。

为了正确的生成RTCP包，每个参与者还需要维护会话中其他成员的状态。良好的设计会使得参与者数据库成为系统运行的重要组成部分，不仅包含RTCP相关的信息，而且包含每个参与者的所有状态。参与者数据结构可能包含以下内容：

- SSRC标识符
- 源描述信息：CNAME是必须的；以及其他可能包含的信息（注意，这些值不是以null结尾的，在处理他们需要注意）
- 接收质量统计信息（丢包和抖动），以生成RTCP RR包。

- 从发送者报告收到的信息，以允许音视频同步（见第七章）。
- 最后一次接收到该参与者的信息，这样不活跃的参与者可以做超时处理
- 一个标志位，标识了该参与者是否在当前RTCP报告间隔内发送了数据
- 媒体播放缓冲区，以及所需要的任何编解码器的状态（参见第六章《媒体采集、播放和时序》）。
- 信道编码和错误恢复所需要的任何信息 -- 例如，在解码之前等待接收修复包的数据（参见第八章《错误隐藏》和第九章《错误恢复》）。

在RTP会话中，成员由其同步源标识符表示。因为可能存在许多参与者，并且他们的访问顺序很随机，所以参与者数据库的数据结构应该是一个哈希表，由SSRC标识符索引。在只处理单一媒体格式的应用程序中，这是足够的。然后，对音视频同步也需要具备根据它们的CNAME查找其源的能力。因此，参与者数据库应该是被一个双哈希表索引；一次由SSRC索引，一次由CNAME索引。

在一些实现中，选择SSRC标识符时，使用不完全随机数生成器。这意味着一个简单的哈希函数 -- 例如，使用SSRC的最低几位做为表的索引 -- 可能导致不平衡和低效的操作。尽管SSRC的值应该是随机的，但是他们应该与高效的哈希函数一起使用。有些人建议使用SSRC的MD5散列做为索引的基础，尽管这可能被认为有点过了。

在收到来自参与者验证过的包之后，应该将参与者添加到数据库中。这个验证的步骤很重要：除非确定参与者是有效的，否则程序不应为参与者创建状态。以下是一些指导方案：

- 如果接收到一个RTCP包并进行了校验，则应该将参与者输入到数据库中。对RTCP包的有效性检验很严格，伪造数据包是很难的。
- 除非是收到多个带有连续序列号的包，否则，不能只根据RTP包进行录入。对单个RTP包的有效性检验是不可靠的，而伪造的无效数据包有可能满足测试。

这意味着我们在实现过程中应该维护一个附加的轻量级的源的临时列表（仅仅接收到一个RTP包的源）。为了防止伪造的RTP和RTCP数据源占用太多内存，这个表应该有个主动超时机制，并应有固定的最大容量。很难防止攻击者故意生成许多不同的源来耗尽接收者的所有内存，但是这些预防措施可以防止在接收到错误定向的非RTP流时，意外的耗尽内存。

有效RTP包中的每个CSRC（贡献源）也算一个参与者，应该添加到数据库中。你应该期望只收到CSRC中的参与者的源描述信息（SDES）。

当将参与者添加到数据库时，应用程序还应该更新会话级别的成员计数和发送方百分比（sender fraction）。添加一个参与者也可能导致RTCP前向重估（forward reconsideration），稍后将对此进行讨论。

在收到Bye包或在指定的非活跃期（specified period of inactivity）后，可以将参与者从数据库中删除。这听起来很简答，但是有几个微妙的地方需要注意。

由于不能保证接收到数据包的顺序，因此可能在来自源的最后一个数据包被收到前，提前接收到Bye数据包。为了防止状态被破坏后，又重新建立连接，参与者应该在收到bye数据包之后将其标志为已离开，并且其状态应该保留几秒钟（本书作者在实现中使用的是固定2秒的延迟）。重要的一点是，延迟要比最大的期望重排序（expected reordering）和媒体播放延迟都大，以便使延迟数据包和播放缓冲区中的任何数据都能在延迟期间到达。

如果源在报告间隔期间内，未被收到超过5次消息，就可以被置为超时。如果报告间隔小于5秒，那么这里使用5秒做最小值（即使在发送RTCP包时使用更小的间隔）。

当收到一个Bye数据包或某个成员超时，将发生RTCP的逆向重估（reverse reconsideration takes place），本章后面的《Bye重处理》一节，会对此进行描述。

时序规则

每个参与者发送RTCP数据包的速率都是不固定的，会根据其会话的大小和媒体流的格式而变化。它的目的是将RTCP流量总量限制在一个固定的比例（通常是会话带宽的5%）。当会话大小增加时，每个参与者发送RTCP数据包的速率会降低，从而实现上述的目标。在双方使用RTCP拨打电话时，每隔几秒会发送一次RTCP报告；在有成千上万参与者的会议中——例如，一个因特网无线电台——每个听众的RTCP报告间隔可能是几分钟。

每个参与者根据本书后面描述的规则决定何时可以发送RTCP包。遵循这些规则非常的重要，特别是可能用在大型会话场景中的程序，如果实现了正确的规则，RTCP可以扩展到具有数千个成员的会话。否则，RTCP的流量将随着成员数的增长而线性增长，最终将导致严重的网络拥塞。

报告时间间隔

复合的RTCP包会使用随机定时器周期性发送。发送RTCP包之间每个参与者的平均等待时间称为报告间隔。它是根据以下几个因素计算的：

- **RTCP带宽的分配**，这是一个固定的分配策略 -- 通常是会话带宽的5%。会话带宽是会话期望的传输数据的速率；通常是单个音频或视频数据流的比特率，乘以同时发送方的典型数量（typical number）。会话带宽在会话期间是固定的，并在RTP应用程序启动时做为参数传递过去。

分配给RTCP的会话带宽比例，可以根据使用中的RTP配置文件而变化。重要的是会话所有成员使用相同的比例（same fraction），否则某些成员的状态可能会过早的超时。

- **发送和接收的RTCP数据包的平均大小**。平均大小不仅包含RTCP数据，还包括UDP和IP包头的大小（即，实现一个典型的IPv4，每个包需要添加28个字节）。
- **参与者的总数以及参与者中发送者的比例** 这需要一个所有参与者的数据库，标记它们是否是发送者（也就是说从他们那里收到了RTP包或者RTCP SR包）还是接收者（如果只收到了RTCP RR、SDES或者APP数据包）。前面的“参与者数据库”一节对此进行了详细的说明。

为了防止在没有发送数据时，就发送SR包的错误，监听数据的参与者应该只在收到数据包后，才将另一个参与者视为发送者。只发送数据且不监听他人数据（如媒体服务器）的程序可以使用RTCP SR包做为发送方的标识，但是它应该验证包和字节计数字段为非零，并且是每个SR包的这些字段都有变化。

如果发送者的数量大于零，但不到参与者总数的四分之一，那么报告时间间隔取决于我们是否正在发送。如果参与者正在发送，报告间隔被设置为，发送者的数量乘以RTCP数据包的平均大小，再除以分配给RTCP带宽的25%。如果参与者没有在发送，报告间隔被设置为接收者的数量乘以RTCP数据包的平均大小，在除以RTCP所需带宽的75%：

```
1  If ((senders > 0) and (senders < (25% of total number of participants))
2  {
3      If (we are sending)
4      {
5          Interval = average RTCP size * senders / (25% of RTCP bandwidth)
6      }
7      else
8      {
9          Interval = average RTCP size * receivers / (75% of RTCP bandwidth)
10     }
11 }
```

如果没有发送方，或者超过四分之一的成员是发送方，则报告间隔计算为RTCP数据包的平均大小乘以成员总数，再除以所需的RTCP带宽：

```
1 | if ((senders = 0) or (senders > (25% of total number of participants)) {  
2 |     Interval = average RTCP size * total number of members / RTCP bandwidth  
3 | }
```

这些规则确保发送方可以占用很大一部分RTCP带宽，至少占到总RTCP带宽的四分之一。因此，对发送方进行音视频同步和识别RTCP所需的数据包，可以相对较快的被发送，同时还有富余的带宽允许接收来自接收方的报告。

计算出的间隔会与绝对最小间隔进行比较，该最小间隔默认设置为5秒。如果时间间隔小于最小间隔，那么就把它设置为最小间隔。

```
1 | If (Interval < minimum interval)  
2 | {  
3 |     Interval = minimum interval  
4 | }
```

在某些情况下，希望发送RTCP的频率超过使用默认最小间隔。例如，如果数据率很高，而应用程序需要更及时的接收质量统计数据，则需要一个较短的默认间隔。最新修订版的RTP协议规范，允许在这种情况下减少最小间隔。

```
1 | Minimum interval = 360 / (session bandwidth in Kbps)
```

对于大于72Kbps的会话带宽，最小间隔可减小至小于5秒。在使用最小间隔时，重要的一点是要记住，一些参与者可能仍然在使用默认的5秒，在决定是否因为参与者处于不活跃状态而算其超时，要考虑到这一点。

计算出的间隔是RTCP数据包之间的平均时间。然后使用下面描述的传输规则，将该值转换为每个数据包实际发送的时间。每当会话中参与者的数量发生变化，或者发送者的比例发生变化时，都应该重新计算报告间隔。

基本传输规则

当应用程序启动时，根据对报告间隔的初始预估，安排第一个RTCP包进行传输。当发送第一个包时，将调度第二个包，以此类推。数据包之间的时间是随机的，在报告间隔的1/2到1.5倍之间，这是为了避免与参与者报告产生同步，以免它们每次都同时到达。最后，如果这是发送的第一个RTCP包，间隔减半，以提供新成员加入的更快的反馈，从而计算下一次发送时间，如下所示：

```

1 | I = (Interval * random[0.5, 1.5])
2 | if (this is the first RTCP packet we are sending)
3 | {
4 |     I *= 0.5
5 | }
6 | next_rtcp_send_time = current_time + I

```

例程random[0.5,1.5]是在0.5到1.5之间产生一个随机数。在某些平台上，它是调用系统的rand()来实现；在其他平台上，像drand48()这样的调用可能会产生更好的随机源。

以基本的传输规则为例，考虑一个使用RTP-over-IP多播发送128Kbps的MP3音频的Internet电台，其听众为1000人。使用最小报告间隔（5秒）和RTCP带宽占比（5%）的默认值，假设RTCP数据包的平均大小为90字节（包括UDP/IP包头）。当一个新的听众成员启动时，它不会知道其他的听众，因为他还没有收到任何RTCP数据。它必须假设发送方是唯一的其他成员，并相应地计算其初始报告间隔。发送者（单一来源）已知成员（来源和这个接受者）的比例超过25%，因此计算报告间隔如下：

```

1 | Interval = average RTCP size * total number of members / RTCP bandwidth
2 |           = 90octets * 2 / (5% of 128Kbps)
3 |           = 180 octets / 800 octets per second
4 |           = 0.225 seconds

```

因为0.225秒小于最小值，所以使用5秒的最小间隔做为间隔。然后将这个值随机化并减半，因为这是要发送的第一个RTCP包。因此，第一个RTCP包在应用程序启动后1.25到3.75秒之间发送。

在启动应用程序和发送第一个RTCP包之间的这段时间内，将从会话的其他成员接收到几个接收方报告，从而允许程序更新预估的成员数量。这个更新会用于调度第二个RTCP包。

我们将在后面看到，1000个听众足以使平均间隔大于最小间隔，因此，从所有听众接收到RTCP数据包的总速率为 $75\% * 800 \text{ 字节/秒} / 90 \text{ 字节/数据包} = 6.66 \text{ 数据包/秒}$ 。应用程序在2.86秒之后发送它的第一个RTCP包，已知的用户大小大约是 $2.86 \text{ 秒} * 6.66 \text{ 秒} = 19$ 。

因为发送者的比例现在小于已知成员的25%，所以第二个包的报告间隔是这样计算的：

```

1 | Interval = receivers * average RTCP size / (75% of RTCP bandwidth)
2 |           = 19 * 90 / (75% of (5% of 128 Kbps))
3 |           = 1710 / (0.75 * (0.05 * 16000 octets/second))
4 |           = 1710 / 600
5 |           = 2.85 seconds

```

再次，这个值被增加到最小间隔并被随机化。第二个RTCP包在第一个包之后2.5秒到7.5秒之间发送。

重复此过程，从发送第一个RTCP数据包到第二个RTCP数据包之间平均要监听到33个新的接收方，总共已知成员数为52。结果是平均间隔为7.8秒，由于大于最小间隔，因此可以直接使用。随后，在第二分组之后的3.9秒到11.7秒之间发送第三分组。数据包之间的平均间隔，随着其他接收方的发现而增加，直到收到完整的接收方为止。间隔的计算如下：

```
1 Interval = receivers * average RTCP size / (75% of RTCP bandwidth)
2           = 1000 * 90 / (75% of (5% of 128Kbps))
3           = 90000 / (0.75 * (0.05 * 16000 octets/second))
4           = 90000 / 600
5           = 150 seconds
```

150秒的间隔，相当于每秒 $1/150 = 0.0066$ 个包，如果有1000个监听器，则平均的RTCP接收速率为每秒6.66个包。

RTP的标准版本，仅使用这些基本的传输规则，虽然这些对于许多应用程序来说已经足够了，但是它们有一些限制，这些限制会在成员迅速变化的会话中会造成问题，重估（Reconsideration）概念的提出就是为了避免这些问题。

前向重估（Forward Reconsideration）

正如上一节所建议的，当会话比较大时，新成员需要一定数量的报告间隔，才能知道会话的总大小。在这个过程中，由于获取到的信息不完整，新成员发送数据包的速度要比“正确”的速度快（“correct” rate）。当许多成员同时加入时，这个问题就会变得非常严重，这种情况叫做“步进加入（step join）”。一个“步进加入”的连接可能发生的典型场景，就是在当应用程序同时自动启动多个参与者。

在“step join”的情况下，如果只使用基本的传输规则，每个参与者将在初始预估为0参与者的基础上加入，并调度它的第一个RTCP包。它将在最小间隔的平均一半之后，发送该数据包，并且它将根据当时观察到的参与者数量，安排下一个RTCP数据包，可能是几百，甚至是几千个。由于对组大小的初始预估比较低，所以当所有参与者加入会话时，RTCP流量会激增，这可能会阻塞网络。

Rosenberg研究了这一现象，并报告了一个案例，在这个案例中，有一万名成员同时加入了一个会话。根据他的模拟，在这样的一个步进连接（step join）中，所有10000个成员都试图在开始的2.5秒内发送一个RTCP包，这几乎是预期速率的3000倍。这一的数据包骤长的极端情况将导致网络阻塞 -- 这并不是低速率控制协议所期望的结果。

持续更新参与者的数量和发送者比例，然后使用这些数字重估每个RTCP包的发送时间，可以解决以上问题。在预定的传输时间到达后，根据组的大小重新预估并计算间隔时间，并使用此值计算新的发送时间，如果新的发送时间，比现在要晚，则不发送数据包，而是修改此包的新的发送时间。

这是一个听起来很复杂，但是实现起来很简单的方案。只考虑基本传输规则的情况下，可以这样写伪代码：

```
1 | if (current_time >= next_rtcp_send_time)
2 | {
3 |     send RTCP packet
4 |     next_rtcp_send_time = rtcp_interval() + current_time
5 | }
```

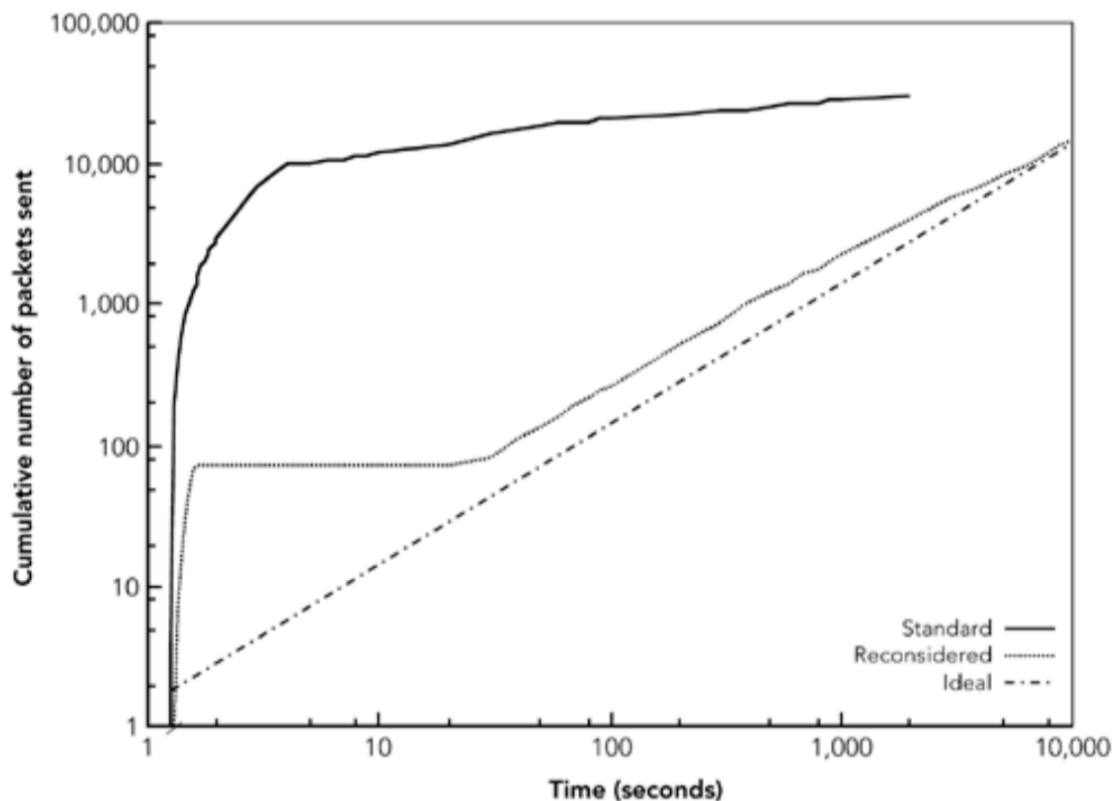
加上前向重估后，以上代码可以优化为：

```
1 | if (current_time >= next_rtcp_check_time)
2 | {
3 |     new_rtcp_send_time = (rtcp_interval() / 1.21828) + last_rtcp_send_time
4 |     if (current_time >= new_rtcp_send_time)
5 |     {
6 |         send RTCP packet
7 |         next_rtcp_check_time = (rtcp_interval() / 1.21828) + current_time
8 |     }
9 |     else
10 |    {
11 |        next_rtcp_check_time = new_send_time
12 |    }
13 | }
```

这里，函数rtcp_interval()是基于当前会话的大小的预估，返回报告间隔的随机抽样。注意，rtcp_interval()除以1.21828（欧朗常数 $e - 1$ ）。这是一个会受补偿因子影响的重估算法（reconsideration algorithm），它最终会收敛到一个低于期望5%的带宽占比的值。

Rosenberg研究了这种现象，并报告了一次有10,000名成员参加会议的情况。他的模拟显示，在这样的步进连接中，所有10,000个成员都尝试在前2.5秒内发送RTCP数据包，这几乎是所需速率的3,000倍。这样的数据包突发将导致极端的网络拥塞，这对于低速率控制协议而言并不是理想的结果。

Figure 5.13. The Effect of Forward Reconsideration on RTCP Send Rates (Adapted from J. Rosenberg and H. Schulzrinne, "Timer Reconsideration for Enhanced RTP Scalability," Proceedings of IEEE Infocom '98, San Francisco, CA, March 1998. © 1998 IEEE.)



作为另一个示例，考虑上一节“基本传输规则”中讨论的场景，其中一个新的听众，使用多播RTP的方式加入到一个已经建立的互联网电台。当听众加入会话时，第一个RTCP包，像之前一样被调度，在应用程序启动的1.25秒到3.75秒之间。当预定的传输时间到达时，就会出现差异：应用程序不是发送数据包，而是根据当前成员数量重新考虑发送时间。如前所述，假设一个随机的初始时间间隔为2.86秒，应用程序将从其他成员接收到大约19个RTCP包，新的平均时间间隔变为2.85秒：

```
1 Interval = number of receivers * average RTCP size / (75% of RTCP bandwidth)
2   = 19 * 90 / (0.75 * (0.05 * 16000 octets/second))
3   = 1710 / 600
4   = 2.85 seconds
```

结果小于最小值，因此使用最小值5秒，随机化并除以比例因子。如果结果值小于当前时间（在本例中是应用程序启动后的2.85秒），则发送数据包。如果不是，例如，新的随机值是5.97秒，则修改数据包发送时间为后者。

在新的计时器到期之后（本例中为应用程序启动后的5.97秒），重新开始处理。此时，接收方将从大约 $5.97 \text{秒} \times 6.66/\text{秒} = 40$ 个其他成员处接收到RTCP数据包，重新计算的RTCP间隔为6秒（随机化和缩放之前）。此过程会重复的进行，直到重新预估的发送时间小于当前时间。此时发送第一个RTCP包，并调度第二个RTCP包。

重估（Reconsideration）是很容易实现的，建议在实现的过程中都要包括它，即使它只有在参与者达到几百人的时候才有显著效果。不管会议规模多大，或者同时有多少参与者加入，包含前向重估的实现都是安全的。只使用基本传输规则的实现可能会过于频繁的发送RTCP包，从而使得在大型会话场景中造成网络拥塞。

反向重估

如果步进接入（step join）存在问题，那么许多参与者快速离开（步进离开）时，我们可以合理地预期将会出现问题。基本的传输规则确实是这样，虽然问题不在于发送RTCP太频繁而导致拥塞，而在于发送不够频繁，导致参与者过早的超时。

当大多数（但不是所有）成员离开大型会话时，问题就出现了。结果是，报告间隔迅速缩短，可能从几分钟缩短到几秒。然而，使用基本的传输规则，虽然更新了超时间隔，但是在更新之后不会重新调度数据包。结果是那些没有离开的成员被标记为超时；它们的包在新的超时时间内没有到达。

这个问题的解决方法类似于步进（step join）；当收到每个Bye数据包，更新参与者的预估数量，重新考虑下一个RTCP数据包的发送时间。与前向重估的不同之处在于，预估值会变得越来越小，因此下一个数据包的发送将比原来的发送时间更早。

当收到一个Bye数据包时，新的传输时间是根据Bye之后，仍然存在的成员比例，和在原计划传输时间之前的剩余时间来计算的。例程如下：

```
1 | if (BYE packet received)
2 | {
3 |     member_fraction = num_members_after_BYE / num_members_before_BYE
    time_remaining =
    next_rtcp_send_time - current_time
    next_rtcp_send_time = current_time + member_fraction *
    time_remaining
4 | }
```

结果就是新的传输时间比原始的早，但是比当前时间晚。因此，包会被安排的足够早，以使得其余的成员不会超时，从而防止错误地将参与者数量的预估值降为0。

在实践中，反向重估可以显著减少由于过早超时而导致的问题，但不能完全解决问题。在某些情况下，在步进离开后，组成员的预估值可能会在短时间内降为零，但很快就会恢复到正确值。在重估的实现中，彻底解决这一问题的复杂性会超过其收益。

反向重估的实现是一个次要问题：它只是在有几百名参与者且成员迅速变化的会话中才会出现，而不实现它会导致错误的超时，但是不会造成全网范围的问题。

Bye包重估

在RTP标准中，希望离开会话的成员需要立即发送一个Bye数据包，然后再退出。如果许多成员立即离开，这可能会导致大量的Bye数据包，并可能导致网络拥塞（就像如果不用前向重估RTCP数据包在步进加入时发生的情况一样）。

为了避免这个问题，当前的版本中的RTP只允许在参与者决定离开时的成员小于50个时，才允许立即发送Bye数据包。如果超过50人，则成员在准备离开时，如果收到其他成员的Bye数据包，则应延迟发送Bye数据包，这个过程称作Bye数据包重估。

Bye重估与前向重估类似，但其是根据收到的报文数量，而不是根据其他成员的数量。一个参与者想要离开一个会话，它会暂停RTP/RTCP数据包的正常处理流程，并根据前向重估规则调度一个Bye数据包，就像没有其他成员，或者就像这是第一个要发送的RTCP数据包一样。在等待预估的传输时间时，参与者会忽略所有的RTC/RTCP包，除了Bye包。对接收到的Bye包进行计数，并且当调度的Bye数据包的传输时间到达时，基于该计数重估所接收到的Bye数据包。该过程将继续持续到发送Bye包，然后参与者离开会话。

正如这个描述所表明的，在发送Bye数据包之前的延迟依赖于离开的成员数量。如果只有一个成员决定离开，Bye将延迟1.026到3.078秒（基于5秒的最小报告间隔在减半，因为Bye包会被当做初始的RTCP包对待）。如果很多参与者决定离开，在决定离开会话和能够发送Bye包之间可能会有相当长的延迟。如果需要快速退出，不发送Bye包就可以安全退出，其他参与者最终对其按超时处理。

使用Bye重估是一个相对次要的决定，只有当许多参与者同时离开一个会话，并且参与者关心离开通知时，Bye重估才有用。离开大型会话而不发送Bye包是安全的，而不是去实现一个Bye的重估算法。

对重估算法的评论 (Comments on Reconsideration)

重估算法规则的引入，使得RTCP能够扩展到成员迅速变化并且人数极多的会话中。作者建议所有的实现最好都包含重估算法，即使它们原本在极小规模的会议中使用。这样会避免产生一些设计者没有预见到的问题。

第一次读的时候，会感觉重估的规则复杂难以实现。实际上，只会增加少量的代码。作者的RTP和RTCP的实现中大约包含了2500行C语言代码（不包括套接字和加密部分）。正向和反向重估部分的代码只有15行。Bye重估代码比较复杂，但是也只有33行，这显然不是最大的问题。

重估规则的正确运行，在很大程度上取决于大量参与者行为的统计均值。在大型会话中，单个错误实现的影响不会很大，但是许多错误实现的话，就可能会造成严重的拥塞问题。对于小的会话，更大程度是一个理论上的问题，但是随着会话体量的增加，糟糕的RTCP实现会被放大，并可能造成网络拥塞，从而影响AV的质量。

实现过程中常见的问题

RTCP在实现的过程中最常见的问题是与基本传输规则以及带宽计算相关的：

- 没有随参与者数量同比伸缩。固定的报告间隔，将导致流量随着成员数量的增加而线性增长，最终超过发送音视频数据的量，最终造成网络拥塞
- 缺乏报告时间间隔的随机性。使用非随机报告间隔的实现方案，可能会无意间同步发送报告，从而导致RTCP包突增，导致接收方阻塞。
- 忘记在带宽计算的过程中的网络底层协议的开销。在计算报告间隔时，所有数据包都要包含UDP和IP包头（典型的基于IPv4实现，28个字节大小）。
- 填充（padding）错误。如果需要填充，应该只添加到复合RTCP包中的最后一个包。

在测试一个RTCP的实现时，一定要使用多种场景。大型和小型会话都可能会测出问题，在这些会话中，可以设置成员快速变化，可以设置大多数的参与者是发送者，可以设置少部分参与者是发送者，还可以加入步进接入和退出机制，来测试会话。测试大型会话本身就是复杂的。如果实现中可以构建一个独立于底层网络传输机制的系统，那么可以允许在单个测试机上模拟大型会话。

IETF 音视频工作组编写了一份RTP实现的测试策略文档，可以参考，这非常有帮助。

总结

本章着重描述了RTP的控制协议，RTCP。它由三部分组成：

1. RTCP包格式，以及生成复合包的方法
2. 将参与者数据库作为基于rtp应用程序的主要数据结构，及存储RTCP正确操作所需要的信息。
3. 控制RTCP包的时序规则：定期传输、适应会话大小和重估策略。

我们还简要讨论了安全性和隐私的问题（这些问题会在第十三章《安全性考虑》中深入讨论），及RTCP数据包的正确性校验中深入讨论。

RTP控制协议是RTP的一个组成部分，用于接收质量报告、源描述、成员控制和音视频同步。正确的实现RTCP可以显著的增强RTP会话，它允许接收方同步音视频以及识别会话的其他成员，并且允许发送方掌握足够的信息，以选择错误保护方案以实现最佳的通信质量。