1 概述和运输层服务

报文段：在发送端中运输层将从发送应用程序进程接收到的报文转换为运输层分组，这个分组成为报文段。

1.1 运输层和网络层的关系

网络层提供了主机之间的逻辑通信，而运输层为运行在不同主机上的进程之间提供了逻辑通信，运输层刚好位于网络层之上。

运输层提供的服务受制于网络层所能提供的服务。不过即使底层网络协议不能在网络层提供相应的服务，运输层协议也能提供某些服务。比如即使底层网络协议是不可靠的，运输层协议也能为应用程序提供可靠的数据传输服务。

1.2 因特网运输层概述

因特网为应用层提供了两种运输层协议：用户数据报协议 UDP 和传输控制协议 TCP。

UDP 提供的服务：

- 进程到进程的数据交付。
- 进程到进程的数据差错检查。

TCP 提供的服务：

- 进程到进程的数据交付。
- 进程到进程的数据差错检查。
- 可靠数据传输。
- 拥塞控制。

2 多路复用与多路分解

课本第 127 页有运输层工作的例子。

多路复用：在源主机从不同套接字中收集数据块，并为每个数据块封装上首部信息，从而生成报文段，然后将报文段传递到网络层。

多路分解：将运输层报文段中的数据交付到正确的套接字。

运输层多路复用的要求：

1. 套接字有唯一标识符。
2. 每个报文段有特殊字段来指示该报文段所要交付到的套接字。这些特殊字段包括源端口号字段和目的端口号字段。

端口号：一个 16 比特的数，大小在 0 ～ 65535 之间。其中 0 ～ 1023 的端口号称为周知端口号，用于保留给周知的应用层协议。

2.1 无连接的多路复用与多路分解

一个 UDP 套接字由一个二元组来标识：(目的 IP 地址，目的端口号)。

一般来说，运输层自动地为应用程序的客户端分配唯一的端口号，为服务器端分配一个特定的端口号。

课本第 129 页有 UDP 复用与分解的例子。

2.2 面向连接的多路复用与多路分解

一个 TCP 套接字由一个四元组来标识：(源 IP 地址，源端口号，目的 IP 地址，目的端口号)。

也就是说，两个具有不同源 IP 地址或源端口号的到达 TCP 报文段将被定向到两个不同的套接字。

课本第 130 页有 TCP 复用与分解的例子。

3 无连接运输：UDP

UDP 的三个特点：

1. UDP 从应用程序得到数据，附加上用于多路复用/分解服务的源和目的端口号字段，以及两个其他的小字段，然后将形成的报文段交给网络层。
2. 在接收主机上，UDP 使用目的端口号将报文段中的数据交付给正确的应用进程。
3. 没有拥塞控制。

称 UDP 为无连接的原因：在发送报文段之前，发送方和接收方的运输层实体之间没有握手。

DNS 是使用 UDP 的一个例子，见课本第 133 页。

UDP 优于 TCP 的方面：

1. 关于何时、发送什么数据的应用层控制更为精细。

2. 无需建立连接。
3. 不需要维护连接状态，可以支持更多的活跃客户。
4. 分组首部开销小，UDP 的首部只有 8 字节的开销。

3.1 UDP 报文段结构

UDP 报文段结构如下：

- 一个首部，有四个字段：
  - 源端口号字段和目的端口号字段，UDP 通过端口号可以使目的主机将应用数据交给运行在目的端系统中的相应进程。
  - 长度字段，指示了 UDP 报文段的字节数。
  - 检验和，用于检查该报文段是否出现了差错。
- 一个数据字段，用于装载应用层数据。

3.2 UDP 检验和

检验和的作用：确定当 UDP 报文段从源到达目的地移动时，其中的比特是否发生了改变。

求检验和的步骤：

1. 对报文段首部中的其他三个字段求和，如果加法有溢出，就将该溢出回卷。
2. 将第一步得到的结果求反码，然后存放在 UDP 报文段中的检验和字段。

利用检验和进行差错检测的方法：

1. 接收方将报文段首部的 4 个字段加在一起。
2. 检验第一步中得到的结果，如果所有比特全为 1，则说明没有出错。

需要知道的是，UDP 只提供了差错检测，但它对差错恢复无能为力。

4 可靠数据传输原理

可靠数据传输的定义：

1. 传输数据比特不会受到损坏或丢失。
2. 所有数据都是按照其发送顺序进行交付。

4.1 构造可靠数据传输协议

4.1.1 经完全可靠信道的可靠数据传输：rdt1.0

考虑最简单的情况：底层信道是完全可靠的。

课本第 138 页画出了 rdt1.0 的有限状态机，需要知道的是，发送方和接收方都有各自的 FSM。

在发送端，由较高层应用调用产生 rdt_send(data) 事件，然后产生两个动作：

1. 经由 make_pkt(data) 动作产生一个包含该数据的分组 packet。
2. 经由 udt_send(packet) 动作将分组发送到信道中。

在接收端，由较低层协议调用产生 rdt_rcv(packet) 动作，然后产生两个动作：

1. 经由 extract(packet, data) 动作从分组中取出数据。
2. 通过 deliver_data(data) 动作将数据上传给较高层的应用。

4.1.2 经具有比特差错信道的可靠数据传输：rdt2.0

假设底层信道：

1. 分组中的比特可能受损。
2. 所有发送的分组将按其发送的顺序被接收，不会发生丢包。

自动重传协议 (ARQ 协议)：使用控制报文“肯定确认 (ACK)”和“否定确认 (NAK)”。“肯定确认”让发送方知道哪些内容被正确接收，“否定确认”让发送方知道哪些内容接收有误并因此需要重复。

ARQ 协议中的其他三种协议功能：

1. 差错检测，使得接收方检测到何时出现了比特差错。
2. 接收方反馈，让接收方向发送方会送 ACK 与 NAK 分组。
3. 重传，接收方收到有差错的分组时，发送方将重传该分组文。

课本第 139 页画出了 rdt2.0 的有限状态机。

发送端状态机的描述：

1. 发送端首先处于等待来自上层的调用的状态。
2. 上层调用时发生 rdt_send(data) 的事件，产生 sndpkt=make_pkt(data, checksum) 和 udt_send(sndpkt) 的动作，随后进入等待 ACK 或 NAK 的状态。

3. 处于等待 ACK 或 NAK 的状态时，如果发生 rdt_rcv(rcvpkt)&&isNAK(rcvpkt) 的事件，就产生 udt_send(sendpkt) 的动作，并回到等待 ACK 或 NAK 的状态。如果发生 rdt_rcv(rcvpkt)&&isACK(rcvpkt) 的事件，就回到等待来自上层的调用的状态。

需要知道的是，当发送方处于等待 ACK 或 NAK 的状态，它不能从上层获得更多的数据。所以 rdt2.0 协议又称为停等协议。

接收端状态机的描述：

1. 接收端处于等待来自下层的调用的状态。

2. 如果发生 rdt_rcv(rcvpkt)&&corrupt(rcvpkt) 事件，说明收到的分组受损，然后产生 sndpkt=make_pkt(NAK) 和 udt_send(sndpkt) 的动作。如果发生 rdt_rcv(rcvpkt)&&notcorrupt(rcvpkt) 事件，说明收到的分组没有受损，然后产生 extract(rcvpkt, data) ，deliver_data(data)，sndpkt=make_pkt(ACK) 和 udt_send(sndpkt) 的动作。

rdt2.0 的缺陷：没有考虑到 ACK 或 NAK 分组受损的可能性。如果一个 ACK 或 NAK 分组受损，发送方无法知道接收方是否正确接收了上一块发送的数据。

解决方法：当发送方接收到含糊不清的 ACK 或 NAK 分组时，只需要重传当前数据分组即可。该方法引入的新问题：信道中出现了冗余分组，接收方不知道它接收到的分组是新分组还是一次重传。

针对新问题的解决方法：在数据字段添加一个新字段，让发送方对其数据分组编号，将发送数据分组的序号放在该字段。该字段只需要 1 比特：0 或 1。如果接收方接收到的分组序号与上一次接收到的分组序号相同，就说明这是一次重传。ACK 或 NAK 分组不需要指明确认的分组序号，因为这里假设不丢失分组，所以它们一定是为响应这一次发送的数据分组而生成的。

课本第 141 页有 rdt2.1 发送方的状态图，第 142 页有 rdt2.1 接收方的状态图。

rdt2.2 通过在 ACK 分组中加入分组序号，实现了无 NAK 的可靠数据传输协议。如果接收方收到受损的分组时，它发送一个带有上一次数据分组序号的 ACK 分组。发送方接收到同一分组的两个 ACK 分组时，就知道接收方没有正确接收到跟在被确认两次的分组后面的分组。

课本第 142 页有 rdt2.2 发送方的状态图，第 143 页有 rdt2.2 接收方的状态图。

4.1.3 经具有比特差错信道的丢包信道的可靠数据传输：rdt3.0

协议新关注的两个问题：怎么检测丢包以及发生丢包后该做些什么。

如何检测丢包：发送方等待足够长的时间，如果没有接收到接收方的响应，就默认丢包。

发生丢包后该做些什么：只需要重传该数据分组。

如果没有发生丢包又重传时，信道中将引入冗余数据分组。我们在 rdt2.1 中已经解决了这个问题。

实现一个基于时间的重传机制：

1. 每次发送一个分组时，就启动一个定时器。
2. 响应定时器中断，从而采取适当的动作。
3. 终止定时器。

因为分组序号在 0 和 1 之间交替，所以 rdt3.0 被称为比特交替协议。
课本第 144 页有 rdt3.0 发送方的状态图，第 145 页有 rdt3.0 的运行举例。
课本第 144 ～ 146 页有 rdt 协议的效率分析。

4.2 流水线可靠数据传输协议

流水线：不使用停等方式运行，允许发送方发送多个分组而无需等待确认，许多从发送方向接收方输送的分组可以被看成是填充到一条流水线中。
解决流水线的差错恢复的两种基本方法：回退 N 步和选择重传。

4.2.1 回退 N 步

回退 N 步协议：允许发送方发送多个分组而不需等待确认，也受限于在流水线中未确认的分组数不能超过某个最大允许数 N。
基序号：最早的未确认分组的序号。下一个序号：下一个待发分组的序号。
课本第 148 页有 GBN 发送方的状态图，第 149 页有 GBN 接收方的状态图。
GBN 发送方需要响应三种类型的事件：

- 上层的调用。首先判断是否超过 N，如果窗口未满，则产生一个分组并将其发送，并相应地更新分量。如果窗口已满，发送方只需将数据返回给上层。
- 收到一个 ACK。采取累积确认的方式，发送方接收到的 ACK 附带的序号为 n，就默认 n 之前且包括 n 的分组全部被接收方接收到。
- 超时事件。如果出现超时，发送方将重传已发送但未被确认过的分组。接收到一个 ACK 时，如果仍有已发送但未被确认的分组，则定时器被重新启动。如果没有已发送但未被确认的分组，该定时器被终止。

GBN 接收方需要响应的事件：

- 一个分组 n 根据正确的序号被接收到，则接收方为分组 n 发送 ACK，并将其数据交付给上层。

- 在所有其他情况下，接收方丢弃该分组，并为最近按序接收的分组重新发送 ACK。根据这个机制，发送方 getacknum(rcvpkt) 得到的肯定是最近按序接收的分组序号，这保证了数据的按序交付。

课本第 150 页有对 GBN 运行的举例。

因为 GBN 协议未确认的分组数不能超过某个最大允许数 N，这被视为一个窗口，GBN 协议也被称为滑动窗口协议。

4.2.2 选择重传

GBN 协议的缺陷：单个分组的差错就能够引起 GBN 重传大量分组，随着信道差错率的增加，流水线可能会被这些不必要重传的分组所充斥。

选择重传：通过让发送方仅重传那些它怀疑在接受方出错的分组而避免不必要的重传。

SR 发送方需要处理的事件：

- 从上层收到数据。如果序号位于发送方的窗口内，则将数据打包并发送。
- 超时。现在每个分组必须拥有其自己的逻辑定时器，因为超时发生后职能发送一个分组。
- 收到 ACK。如果 ACK 携带的分组序号位于窗口内，则 SR 发送方将那个被确认的分组标记为已接收。如果该分组的序号等于 send_base，则窗口基序号向前移动到具有最小序号的未确认分组处。

SR 接收方也建立了一个窗口 N，用于限制流水线中未完成、未被确认的分组数。SR 接收方需要处理的事件：

- 收到的分组落在接受方的窗口内，产生一个 ACK 分组回送给发送方。如果分组以前从未收到过，则缓存该分组。如果分组序号等于 rcv_base，则窗口向前移动到具有最小序号的未缓存分组处。
- 收到的分组序号小于 rcv_base，此时必须产生一个 ACK 分组回送给发送方，这是考虑到了之前传给发送方的 ACK 可能丢失。
- 其他情况，则忽略该分组。

课本第 152 页有 SR 操作的例子。

当分组序号范围有限时，窗口大小选择不当将产生严重的后果。课本第 153 页有一个相应的例子。窗口长度必须小于或等于分组序号空间大小的一半。

4.3 可靠数据传输机制的总结

实现可靠数据传输的机制：

- 检验和，用于检测在一个分组中的比特错误。
- 定时器，用于超时重传一个数据分组，因为一个数据分组或 ACK 分组可能在信道中丢失了。如果一个分组延时但未丢失，导致过早超时，信道中就会出现冗余数据分组。
- 序号，具有相同序号的数据分组可使接收方检测出一个分组的冗余副本。接收分组序号间的空隙还可以使接收方检测出丢失的分组。
- 肯定确认 ACK，用于告诉接收方一个分组或一组分组已被正确接收到了。
- 否定确认 NAK，用于告诉发送方某个分组未被正确地接收。
- 窗口、流水线，用于提高发送方的利用率。

4.4 分组重新排序问题

当连接发送方和接收方两端的信道是一个网络时，分组重新排序是可能发生的。

分组重新排序的表现：一个具有序号或确认号 x 的分组的旧副本可能会出现，即使发送方或接收方的窗口中都没有包含 x。这就相当于信道缓存了分组，并在将来任意时刻自然地释放出这个分组。因为序号 x 可以背重新使用，一旦该分组被释放，将导致错误的发生。

解决方法：确保序号 x 不被重新使用，知道发送方确信任何先前发送的序号为 x 的分组都不再在网络中为止。

5 面向连接的运输：TCP

5.1 TCP 的基本概念

TCP 的特点：

1. 提供全双工服务。
2. TCP 连接是点对点的。
3. TCP 连接的组成包括：一台主机上的缓存、变量与进程连接的套接字，以及另一台主机上的另一组缓存、变量和雨进程连接的套接字。两台主机间的网络元素没有为该连接提供任何缓存和变量。

最大传输单元 MTU：指的是从源主机到目的主机间所有链路上发送的最大链路层帧长度，典型值为 1500 字节。

最大报文段长度 MSS：需要保证一个 TCP 报文段加上 TCP/IP 首部可以适合一个链路层帧。TCP/IP 首部长度通常为 40 字节，所以 MSS 的典型值为 1460 字节。

5.2 TCP 报文段结构

TCP 报文段结构如下：

- 16 比特的源端口号和 16 比特的目的端口号。
- 32 比特的序号和 32 比特的确认号，用于实现可靠数据传输服务。TCP 对序号的使用是基于字节流而不是基于报文段的，所以一个报文段的序号是该报文段首字节的字节流序号。而确认号是目的主机期望从源主机收到的下一字节的序号。因为 TCP 只确认字节流中到第一个丢失字节为止的字节，所以 TCP 提供的是累积确认。课本第 159 页给出了使用序号和确认号的例子。
- 16 比特的接收窗口字段，用于流量控制。
- 16 比特的检验和字段。
- 4 比特的首部长度字段，单位为字。
- 6 比特的标志字段：
  - ACK 比特用于确认字段中的值是有效的。
  - RST、SYN 和 FIN 用于连接建立和拆除。
  - PSH 比特用于指示接收方应立即将数据上交给上层。
  - URG 比特用于指示报文段里存在着被发送端置为“紧急”的数据。
- 16 比特的紧急数据指针字段，用于指向紧急数据的最后一个字节。
- 可选与长度可变的选项字段。
- 数据字段。

课本第 157 页画出了 TCP 报文段结构。

5.3 往返时间的估计与超时

TCP 中使用超时重传机制，所以超时间隔必须大于该连接的往返时间。

TCP 估计往返时间的方法：在任意时刻，为一个已发送但目前未被确认的报文段估计 SampleRTT。TCP 仅在某一时刻做一次 SampleRTT 测量，而且绝不为已被重传的报文段计算 SampleRTT。

课本的第 161 页还讨论了 EstimatedRTT 和 DevRTT 这两个概念。TCP 的超时间隔计算公式如下：

$$\boldsymbol{TimeoutInterval = EstimatedRTT + 4 \times DevRTT} \tag{1}$$

5.4 可靠数据传输

TCP 发送方对三个事件的响应：

- 上层的调用。TCP 从应用程序接收数据，将数据封装在一个报文段中，并把该报文段交给 IP。
- 超时。TCP 将重传具有最小序号的未确认的报文，并重启定时器。需要注意的是，TCP 中只有一个定时器。
- 接收来自接收方的 ACK。ACK 携带序号 y，发送方记载着最早未被确认的字节的序号 SendBase。TCP 采用累积确认，所以 y 确认了序号在 y 之前且包括 y 的所有字节都已经收到，并且重启定时器。

课本第 164 ～ 165 页给出了可靠传输的例子。

5.4.1 超时间隔加倍

如果 TCP 响应超时事件，那么 TimeoutInterval 将是先前值的两倍。而如果定时器在另两个事件 (上层的调用、接收到 ACK) 中的任意一个启动，TimeoutInterval 的值将有 EstimatedRTT 和 DevRTT 决定。

5.4.2 快速重传

TCP 不使用否认确认，而是使用冗余 ACK 进行否定确认。当接收方接收到一个比期望序号大的失序报文段，接收方检测出间隔，并发送冗余 ACK，指示下一个期待字节的序号。

快速重传：一个收到 3 个冗余 ACK，TCP 就执行快速重传，也就是在该报文段的定时器过期之前重传丢失的报文段。

5.5 流量控制

一条 TCP 连接每一侧的主机都为该连接设置了接收缓存，如果接收方读取数据的速度慢于发送方发送数据的速度，那么该连接的接收缓存可能溢出。

流量控制服务：使得发送方的发送速率与接收方的读取速率相匹配。

实现方法：

1. 接收方定义三个变量，RcvBuffer 表示接收缓存的大小，LastByteRead 表示从缓存读出的数据流的最后一个字节的编号，LastByteRcvd 表示从网络中接收的数据流的最后一个字节的编号。
2. 接收方维持一个接收窗口的变量 rwnd，使得 $rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]$。
3. 接收方将当前 rwnd 值放入发给发送方的报文段中的接收窗口字段中，让发送方知道 rwnd 的值。
4. 发送方定义两个变量，LastByteSent 和 LastByteAckd，两者之差表示发送到连接中但未被确认的数据量。只要让这个值小于 rwnd，就能保证接收缓存不会溢出。

流量控制的缺陷：当 rwnd=0 时，将导致发送方不会再发报文给接收方，从而接收方也不会发确认报文给发送方，发送方中的 rwnd 值得不到更新，发送方就会被阻塞而不能再发送数据。

上述缺陷的解决方法：当 rwnd=0 时，发送方继续发送只有一个字节数据的报文段，这些报文段将会被接收方接收被返回相应的确认报文。

5.6 TCP 连接管理

客户 TCP 通过如下步骤与服务器 TCP 建立一条连接：

1. 客户端的 TCP 向服务器的 TCP 发送一个特殊的 TCP 报文段，称为 SYN 报文段。报文段中 SYN 比特置为 1，序号字段中随机选择一个初始序号 client_isn，数据字段中不包含应用层的数据。
2. SYN 报文段到达服务器主机后，服务器开始分配 TCP 缓存和变量，并发送 SYNACK 报文段。SYNACK 报文段中 SYN 比特置为 1，序号字段随机选择一个初始序号 server_isn，数据字段中不包含应用层的数据。
3. SYNACK 报文段到达客户端主机后，客户开始分配 TCP 缓存和变量，并发送确认报文。报文中 SYN 比特为 0，确认号为 server_isn+1，数据字段包含应用层的数据。

连接创建过程称为 3 次握手，在连接创建之后，每一个报文段中的 SYN 比特都将被置为 0。

课本第 171 ～ 172 页讲了 TCP 连接拆除的例子。

TCP 建立过程还有一种可能：服务器 TCP 接收到 SYN 报文段，但是 SYN 报文段中的端口号或源 IP 地址与服务器主机上的套接字不匹配，那么主机将向源发送一个重置报文段。该报文段中 RST 比特置为 1，用于告诉源主机“我没有那个报文段的套接字，请不要再发送该报文段了”。

6 拥塞控制原理

6.1 拥塞原因与代价

首先我们考虑随着主机增加其发送速率并使网络变得拥塞，这时候会发生的情况。

6.1.1 情况 1: 两个发送方和一台具有无穷大缓存的路由器

最简单的拥塞情况：两台主机都有一条连接，且这两条连接共享源与目的地之间的单跳路由，该路由器拥有无穷大的缓存。

主机 A 和主机 B 向路由器提供流量的速率为 λ_{in} 字节/秒，它们在一段容量为 R 的共享式输出链路上传输。

因为 A 和 B 共享输出链路，所以它们各自最大的吞吐量为 $R/2$。从吞吐量看，运行在总吞吐量为 R 时，链路被充分利用了。但是从时延看，随着主机 A 的发送速率接近 $R/2$，平均时延就会越来越大。

课本上第 175 页的图 3-44 描述了这一情况。

6.1.2 情况 2: 两个发送方和一台具有有限缓存的路由器

基本假设：路由器缓存的容量是有限的，当分组到达一个已满的缓存时会被丢弃。

供给载荷 λ'_{in} 的定义：运输层向网络发送初始数据以及重传数据的速率。

考虑三种情况：

1. 主机 A 能够以某种方式确定路由器中的缓存是否空闲。

   此时只有当缓存空闲时主机 A 才发送一个分组，所以不会产生丢包，λ_{in} 与 λ'_{in} 相等。

2. 主机 A 仅当确定一个分组已经丢失时才重传。

   当 $\lambda'_{in} = R/2$ 时，假设数据被交付给接收方应用程序的速率是 $R/3$，也就是说 0.5R 的数据中，有 0.166R 的数据是重传数据。

   网络阻塞的代价：发送方必须执行重传以补偿因为缓存溢出而丢失的分组。

3. 主机 A 会提前发生超时并重传分组，也就是重传载队列中被推迟但还未丢失的分组。

   此时路由器将转发重传的初始分组副本，也就做了无用功。假定每个分组被路由器平均转发两次，所以当 $\lambda'_{in} = R/2$ 时，假设数据被交付给接收方应用程序的速率是 $R/4$。

   网络阻塞的代价：发送方在遇到大时延的时候，它所进行的不必要重传会引起路由器利用其链路带宽来转发不必要的分组副本。

6.1.3 情况 3:4 个发送方和具有有限缓存的多台路由器及多跳路径

基本假设：有 4 台主机发送分组，每台主机都通过交叠的两条路径传输分组。

现在主机 A 发送分组到主机 C，第一跳路由器为 R1，第二跳路由器为 R2，A-C 连接与 D-B 连接共享路由器 R1，与 B-D 连接共享路由器 R2。

当 λ_{in} 较小时，路由器缓存溢出较少，λ_{in} 接近 λ'_{in}，λ_{in} 的增大会导致 λ_{out} 的增大。

当 λ_{in} 较大时，分组延时较大，此时，B-D 的流量到达 R2 的速率会比 A-C 流量的到达速率大得多。当主机 A 的供给载荷无穷大时，分组延时极大，R2 的缓存空间也就被 B-D 连接的分组占满了，A-C 连接在 R2 上的吞吐量趋近于 0。

每当有一个分组在第二跳路由器上被丢弃时，第一跳路由器所做的工作就是无用功。这里出现了新的网络拥塞的代价：当一个分组一个路由器丢弃了，那么每个上游路由器转发该分组而使用的传输容量就被浪费了。

6.2 拥塞控制方法

我们根据网络层是否为运输层拥塞控制提供了显式帮助来区分拥塞控制方法，有两种主要的拥塞控制方法：

- 端到端拥塞控制。这种方法中，网络层没有为运输层拥塞控制提供显式支持，端系统必须通过对网络行为的观察来推断。比如 TCP，它通过 3 次冗余确认来判断是否出现网络拥塞，并决定是否减小其窗口长度。
- 网络辅助的拥塞控制。这种方法中，路由器向发送方提供关于网络中拥塞状态的显式反馈信息。拥塞信息从网络反馈到发送方有两种方式：
  - 直接网络反馈，也就是由网络路由器向发送方发送一个阻塞分组。
  - 经由接收方的网络反馈，路由器标记或更新从发送方流向接收方的分组中的某个字段，用于指示拥塞的产生。如果接收方收到带有该标记的分组，接收方就会向发送方通知网络阻塞的情况。

6.3 网络辅助的拥塞控制例子：ATM ABR 拥塞控制

ATM ABR 拥塞控制面向 VC 状态来处理分组交换。

每台交换机维护着有关源到目的地 VC 的状态，从而提供了两种功能：

1. 交换机可以跟踪各个发送方的行为，比如跟踪它们的平均传输速率。
2. 交换机可以采取特定源的拥塞控制动作，比如向发送方发显式信令，从而减少发送方的速率。

ABR 是一种弹性数据传输服务：

- 当网络轻载时，ABR 服务会充分利用空闲的可用带宽。
- 当网络拥塞时，ABR 服务会将其传输速率抑制为某些预先确定的最小传输速率。

RM 信元：资源管理信元，用于在主机和交换机之间传递与拥塞有关的信息。每 32 个数据信元中会有一个 RM 信元。

ABR 提供了三种机制用于交换机向接收方发送与拥塞相关的信令信息：

- EFCI 比特，每个数据信元都包含 1 比特的显式转发拥塞指示比特。如果接收方收到的数据信元的 EFCI 比特都置为 1，接收方就会将 RM 信元中的 CI 比特置为 1，并将该 RM 信元送回发送方。
- CI 和 NI 比特，它们处于 RM 信元中，分别为拥塞指示比特和无增长比特，被交换机所设置。当轻微拥塞时，NI 比特置为 1，当严重拥塞时，CI 比特置为 1。
- ER 的设置，RM 信元包含了两字节的显式速率字段。当发生拥塞时，ER 字段将被设置为源至目的地的路径上的所有交换机中的最小可支持速率。

7 TCP 拥塞控制

TCP 拥塞控制的三个问题：

- 第一，一个 TCP 发送方如何限制它向其连接发送流量的速率？
- 第二，一个 TCP 发送方如何感知从它到目的地之间的路径上存在拥塞？
- 第三，当发送方感知到端到端的拥塞时，采用何种算法来改变其发送速率？

7.1 如何限制发送流量的速率

算法如下：

1. 定义一个拥塞窗口变量 cwnd。
2. 发送方中的两个变量 LastByteSent 和 LastByteAckd，两者之差表示发送到连接中但未被确认的数据量。保证这个值小于 cwnd，就能间接地限制发送方的发送速率。

7.2 如何感知拥塞

丢包事件的定义：要么出现超时，要么收到来自接收方的 3 个冗余 ACK。

当出现过度拥塞时，路由器的缓存会溢出，从而丢弃一个数据报，也就引起了发送方的丢包事件，从而让发送方得到拥塞的指示。

7.3 采用何种算法来改变其发送速率

假设不出现丢包事件，TCP 使用确认来增大它的拥塞窗口长度：

- TCP 将确认的到达作为一切正常的指示，从而增加拥塞窗口的长度。
- 如果确认到达的速率较慢，则拥塞窗口的增长速率也较慢。

上述算法使得 TCP 称为自计时的。
TCP 设置发送速率的指导性原则：

- 一个丢失的报文意味着拥塞，因此当丢失报文段时就应该降低 TCP 发送方的速率。
- 一个确认报文段到达时，应该增加发送方的速率。
- 需要探测拥塞开始出现的速率。ACK 到达时不断增加速率直到出现丢包事件，随后发送方减小速率，然后再开始新一轮的带宽探测。

TCP 拥塞控制算法的三个主要部分：慢启动、拥塞避免和快速恢复。

7.3.1 慢启动

慢启动的算法如下：

1. cwnd 的值初始化为 1 个 MSS，并发送一个报文段。
2. 当收到非冗余 ACK 时，将 cwnd 的值增加一个 MSS，并发送与 cwnd 相等数量的报文段，如此反复下去。
3. 出现超时事件时，将 ssthresh 设置为 cwnd ／ 2，将 cwnd 的值设为 1，重传丢失的报文，并重新开始慢启动。
4. 出现 3 个冗余 ACK 时，将 ssthresh 设置为 cwnd ／ 2，将 cwnd 的值设为 ssthresh +3*MSS，重传丢失的报文，并进入快速恢复状态。
5. $\boldsymbol{cwnd \geq ssthresh}$ 时，进入拥塞避免状态。

7.3.2 拥塞避免

算法如下：

1. 收到非冗余 ACK 时，cwnd 增加 MSS*(MSS/cwnd) 个字节，并发送与 cwnd 相等数量的报文段。
2. 当出现超时事件时，将 ssthresh 设置为 cwnd ／ 2，将 cwnd 的值设为 1，重传丢失的报文，并进入慢启动状态。

3. 出现 3 个冗余 ACK 时，将 ssthresh 设置为 cwnd ／ 2，将 cwnd 的值设为 ssthresh +3*MSS，重传丢失的报文，并进入快速恢复状态。

7.3.3 快速恢复

算法如下：

1. 当收到冗余 ACK 时，cwnd 增加一个 MSS 字节，发送与 cwnd 相等数量的报文段，并重新进入快速恢复状态。
2. 当收到非冗余 ACK 时，cwnd 的值设为 ssthresh，并进入拥塞避免状态。
3. 当出现超时事件时，将 ssthresh 设置为 cwnd ／ 2，将 cwnd 的值设为 1，重传丢失的报文，并进入慢启动状态。

7.3.4 TCP 拥塞控制：回顾

假设不会发生超时事件，丢包仅由 3 个冗余 ACK 引起。

TCP 的拥塞控制此时为：每个 RTT 内发送 cwnd/MSS 个报文，则每个 RTT 内 cwnd 线性增加一个 MSS，然后出现 3 个 ACK 事件时 cwnd 减半。

TCP 拥塞控制也就称为加性增、乘性减拥塞控制方式，也就是 AIMD 拥塞控制方式。

7.4 TCP 的吞吐量

TCP 稳态行为的描述：当速率增长到 W/RTT 时，网络丢弃来自连接的分组，从而导致发送速率减半。然后每过一个 RTT 发送速率就增加 MSS/RTT，知道速率在此达到 W/RTT。

对于这个高度理想化的 TCP 稳态动态性模型，它的平均吞吐量为 $\frac{0.75\times W}{RTT}$。