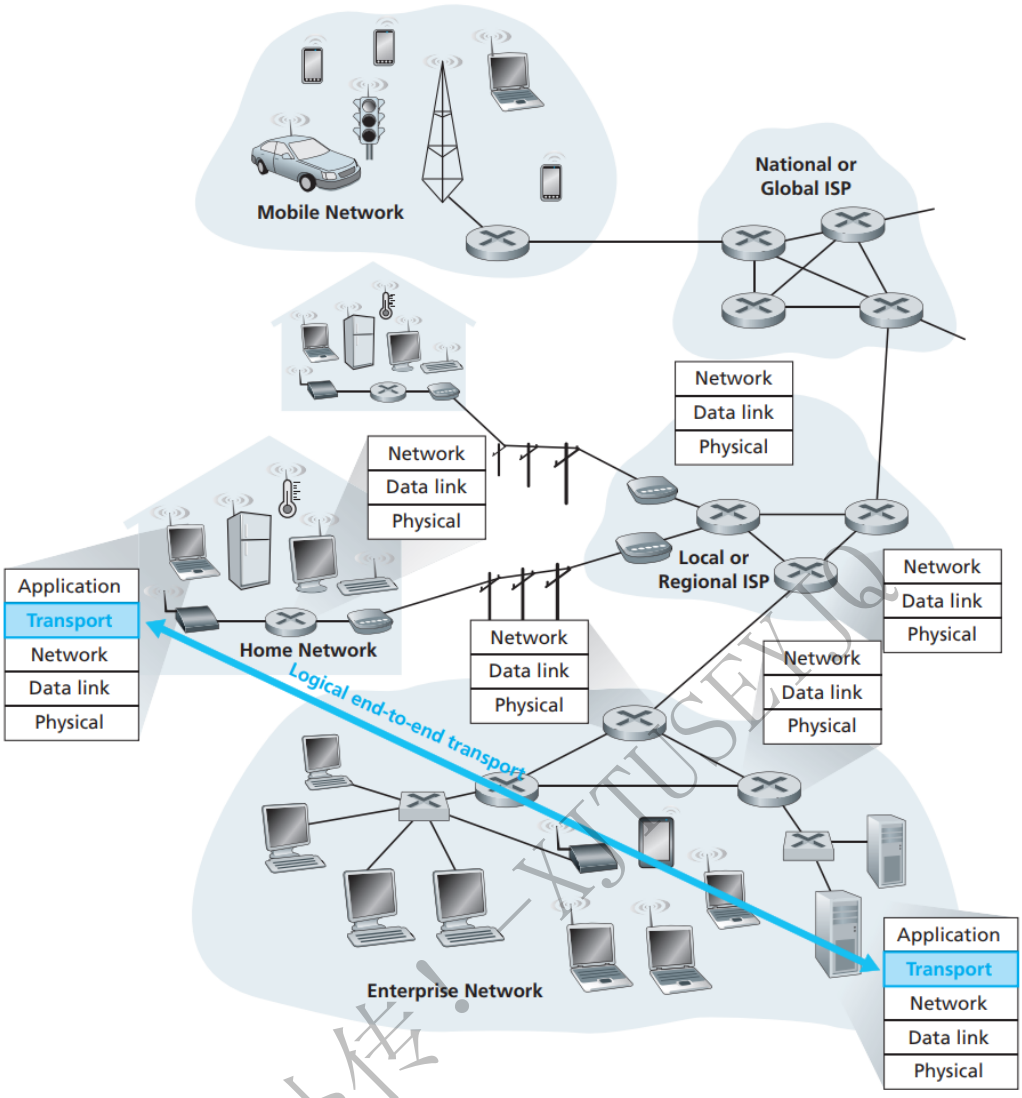
[计算机网络]第三章——传输层

3.1 概述和传输层服务

传输层协议为运行在不同主机上的**应用进程之间**提供了**逻辑通信(logic communication)**功能。从应用程序的角度看，通过逻辑通信，运行不同进程的主机好像直接相连一样;实际上，这些主机也许位于地球的两侧，通过很多路由器及多种不同类型的链路相连。应用进程使用传输层提供的逻辑通信功能彼此发送报文，而无须考虑

承载这些报文的物理基础设施的细节。



如图所示，传输层协议是**在端系统中而不是在路由器中实现的**。在发送端，传输层从发送应用程序进程接收到的报文转换成传输层报文段(segment)。 实现的方法(可能)是将应用报文划分为较小的块，并为每块加上一个传输层首部以生成传输层报文段。然后，在发送端系统中，传输层将这些报文段传递给网络层，网路层将其封装成网络层分组(即数据报)并向目的地发送。

网络路由器仅作用于该数据报的网络层字段;即它们不检查封装在该数据报的传输层报文段的字段。在接收端，网络层从数据报中提取传输层报文段，并将该报文段向上交给传输层。传输层则处理接收到的报文段，使该报文段中的数据为接收应用进程使用。

网络应用程序可以使用多种的传输层协议。例如，因特网有两种协议，即TCP和UDP

传输层概述

因特网网络层协议有一个名字叫IP,即网际协议。IP为主机之间提供了逻辑通信。IP 的服务模型是尽力而为交付服务(best- effort delivery service)。这意味着IP尽它“最大的努力”在通信的主机之间交付报文段，但它并不做任何确保。特别是，它**不确保报文段的交付，不保证报文段的按序交付**，不保证报文段中数据的完整性。由于这些原

因，IP被称为不可靠服务(unreliable service)。 在此还要指出的是,每台主机至少有一个网络层地址，即所谓的IP地址。

在对IP服务模型有了初步了解后，总结一下UDP和TCP所提供的服务模型。UDP和TCP最基本的责任是，**将两个端系统间IP的交付服务扩展为运行在端系统上的两个进程之间的交付服务**。将主机间交付扩展到进程间交付被称为传输层的多路复用(transport-layer multiplexing)与多路分解(demultiplexing)。

UDP和TCP还可以通过在其报文段首部中包括差错检查字段而**提供完整性检查**。进程到进程的**数据交付和差错检查是两种最低限度的传输层服务，也是UDP所能提供的仅有的两种服务**。特别是，与IP一样，UDP也是一种不可靠的服务，即不能保证一个进程所发送的数据能够完整无缺地(或全部!)到达目的进程。

关于TCP的讲解，接下来会仔细说明。

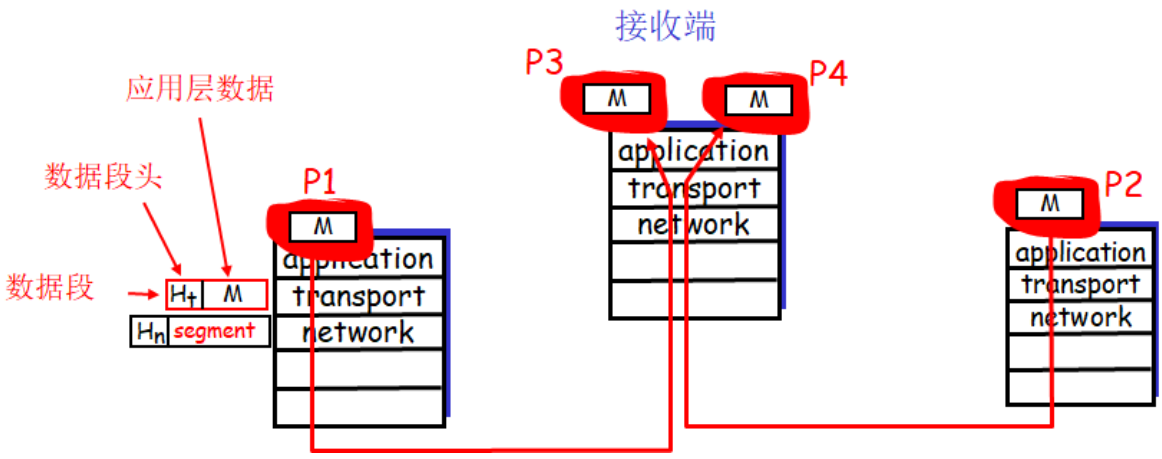
传输层的功能

1. 提供应用进程之间的逻辑通信（网络层提供主机之间的逻辑通信）
2. 提供复用与分用
3. 差错检测
4. 提供无连接的或面向连接的服务

3.2 多路复用与多路分解

将主机间交付扩展到进程间交付被称为传输层的**多路复用(transport- layer multiplexing)与多路分解(demultiplexing)。**

一个进程有一个或多个套接字，相当于从网络向进程传递数据和从进程向网络传递数据的门户，在接受方中的传输层并没有直接将数据交付给进程，而是给了套接字。



每个传输层报文段中具有几个字段。在接收端，传输层检查这些字段，标识出接收套接字，进而将报文段定向到该套接字。**将传输层报文段中的数据交付到正确的套接字的工作称为多路分解**（demultiplexing）。

> 类似于打开微信和qq，qq的消息不会给微信，微信的消息不会给qq

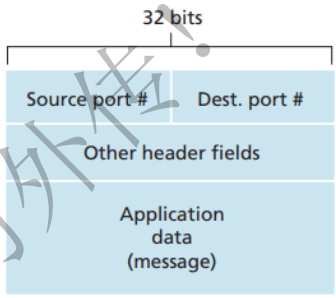
在源主机从不同套接字中收集数据块，并为每个数据块封装上首部信息（这将在以后用于分解）从而生成报文段，然后将报文段传递到网络层，所有这些工作称为 多路复用（multiplexing）

> 发送端进行多路复用；接收端进行多路分解
>
> 例如小明从邮递员收到新建，并通过查看收信人姓名而将信件交付给他的朋友时执行的就是多路分解；而当小美从朋友手中收集信件并交给邮递员时，执行的就是多路复用

传输层多路复用的要求

> 1 套接字有唯一标识符
>
> 2 每个segment有特殊字段来指示所要交付到的套接字，而这些特殊字段就是源端口号字段和目的端口号字段

端口号是一个16位的数，大小在0~65535之间，0~1023范围之间的是周知端口号（如HTTP：80，FTP：21，telnet：23），用户使用的端口号要大于1024：1024~65535



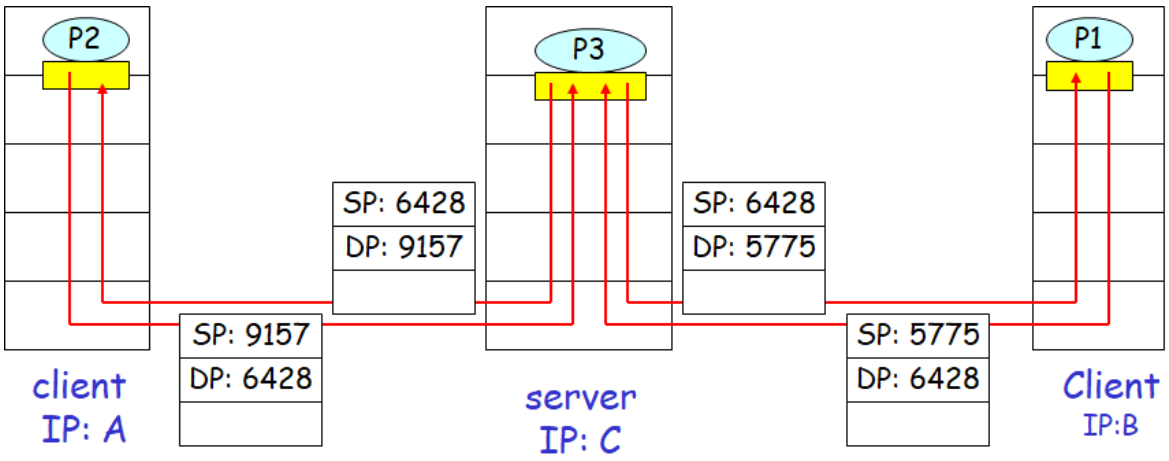
无连接的多路复用与多路分解

通常，客户端应用程序的端口号可以自动分配

此时，一个UDP套接字是由一个二元组全面标识的，该二元组包括一个目的IP地址和一个目的端口号

> 具有不同源IP地址和源端口号，但是具有相同目的IP地址和目的端口号的两个报文段会通过同一个套接字被送到同一个目的进程中。

报文段中的源端口号和源IP地址可以作为报文段回发时的返回地址使用



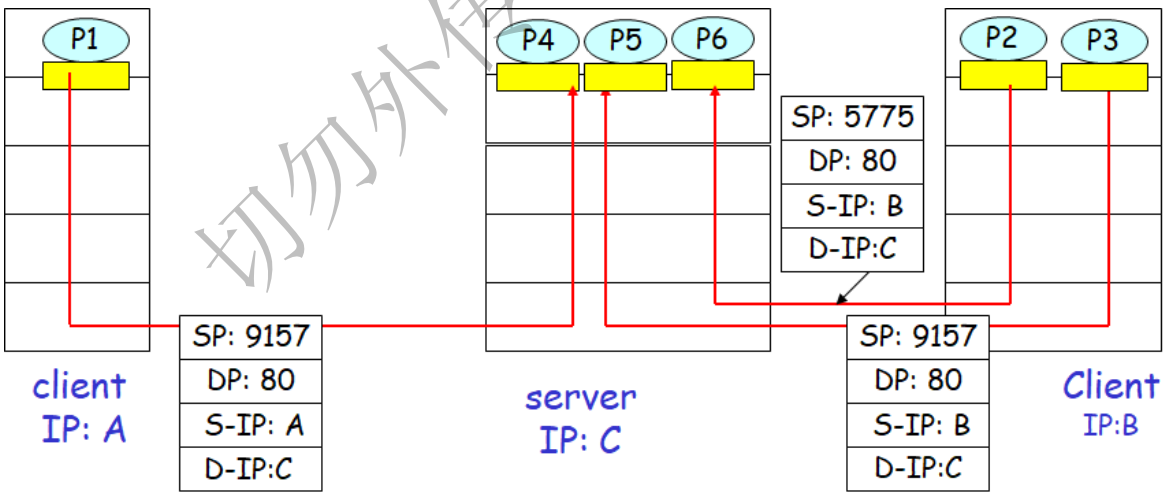
有连接的多路复用与多路分解

一个TCP套接字需要一个四元组

> 源IP地址和目的IP地址
>
> 源端口号和目的端口号

> 具有不同源IP地址和源端口号，但是具有相同目的IP地址和目的端口号的两个报文段会被送到不同的套接字中

每个带有端口号的客户端应用程序都指向服务器上的套接字



3.3 UDP|User Datagram Protocol :用户数据报协议

UDP的段叫数据报

UDP只具有传输协议能够做的最少工作：**多路复用/多路分解；差错检测**。几乎是直接跟IP打交道

使用UDP时，在发送报文段之前，发送方和接收方的传输层实体之间没有握手，因此UDP被称为是*connectionless*

提供的也是"best effort"服务

特征

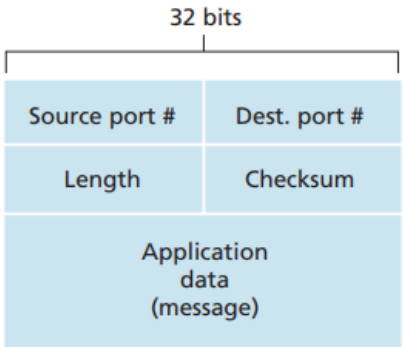
> [1] 实现简单：发送方、接收方没有连接状态
>
> [2] 数据段首部head小(8字节)，传输开销小，时延较短
>
> [3] 速度快：不用控制

典型应用

> Remote file server (NFS)
>
> Streaming multimedia流式多媒体
>
> Internet telephony
>
> Network management
>
> Routing protocol(RIP)
>
> Name translation (DNS)
>
> Multicasting
>
> Real-time involved apps(RTP)
>
> TFTP
>
> DHCP

UDP报文段结构

段头只有8个字节：源端口号、目的端口号、长度、校验和各2B



UDP校验和checksum

UDP检验和提供了差错检测功能。这就是说，检验和用于确定当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(例如，由于链路中的噪声干扰或者存储在路由器中时引入问题)。

发送方

1 将段内容视为16位整数序列

> 0110011001100110
> 0101010101010101
> 0000111100001111

2 对段内容相加，取低16位，按位取反，得到校验和

> 相加得到1100101011001010
>
> 按位取反得到0011010100110101

3 发送方将校验和0011010100110101输入UDP校验和字段

接收方

1 将段内容视为16位整数序列

> 0110011001100110
> 0101010101010101
> 0000111100001111

2 对段内容相加，取低16位

> 相加得到1100101011001010

3 与checksum再相加，检查是否全为1（这里相加就是1111111111111111）

> NO -检测到错误
>
> YES -没有检测到错误
>
> 但是，不能纠正

这种检错能力很弱

3.4 TCP|Transmission Control Protocol传输控制协议

TCP概述

特点

1 点对点：一个发送方，一个接收方(不能用于多播)

2 可靠的、字节有序的流式发送数据：没有“报文边界”

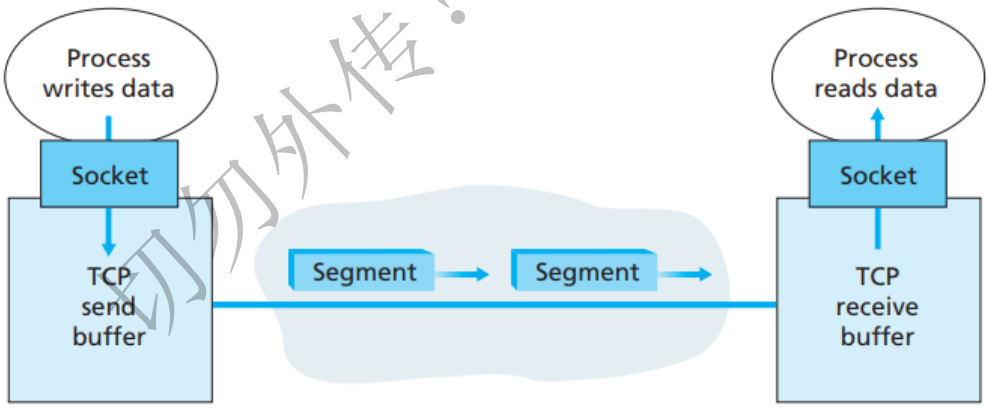
3 流水线式：TCP拥塞和流量控制设置窗口大小

4 需要开辟发送和接收缓冲区

5 全双工数据full duplex data：在同一连接中双向数据流；(UDP也是)

MSS:最大段大小(536字节)

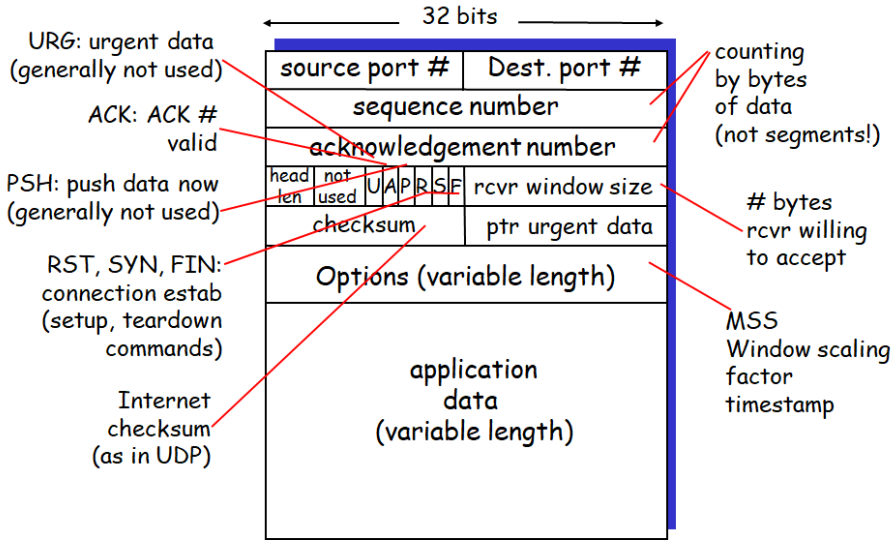
三控一管：**连接管理、可靠性控制、流量控制、拥塞控制**



所谓的连接只是逻辑上的，发送方和接收方开辟缓存，设置变量，交换序列号

段文格式

仅从TCP报文段的首部是无法得知目的IP地址的。因此，TCP必须告诉IP层此报文段要发送给哪一个目的主机（给出其IP地址）。**此目的IP地址填写在IP数据报的首部**中。



TCP段头指的是前面五行，一共**20个字节**

[1] 源端口号和目的端口号：各占2B

[2] **序列号**：TCP是面向字节流的，传送时按照一个个字节传送，所以在一个TCP连接中传送的字节流需要编号，这样才能保证按序交付

> 例如，某报文段的序号从301开始，而携带的数据共有100B.这就表明本报文段数据的第一个字节的序号是301,最后一个字节的序号是400.显然，下一个报文段（如果还有）的数据序号应当从401开始，即下一个报文段的序号字段应为401,这个字段名也称为“报文段序号”。

[3] **确认号**acknowledgement number：占4B。TCP是含有确认机制的，所以**接收端需要给发送端发送确认号**，这个确认号只需记住一点：若确认号等于N,则表明到序号N-1为止的所有数据都已经正确收到。

> 例如，B正确收到了A发送过来的一个报文段，其序号字段值是501,而数据长度是200B（序号501~700),这表明B正确收到了A发送的到序号700为止的数据。因此，B期望收到A的下一个数据序号是701,于是B将发送给A的确认报文段中的确认号设置为701.注意，现在的确认号不是501,也不是700,而是701.

[4] 首部长度

[5] 保留字段：占6位。保留为今后使用，但目前应置为0,该字段可以忽略不计。

[6] 紧急 URG:当URG=1时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送（相当于高优先级的数据）

[7] **确认比特ACK**:只有当ACK=1时，确认号字段才有效；当ACK=0时，确认号无效。TCP规定，一旦连接建立了，所有传送的报文段都必须把ACK置1.

[8] 推送比特PSH:TCP收到推送比特置1的报文段，就尽快地交付给接收应用进程，而不再等到整个缓存都填满后再向上交付。

[9] 复位比特RST:当RST=1时，表明TCP连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立传输连接。

[1] [0] 同步比特SYN:同步比特SYN置为1,表示这是一个连接请求或连接接收报文，后面的TCP连接会详细讲到。

[1] [1] 终止比特FIN:释放一个连接。当FIN=1时，表明此报文段的发送端的数据已发送完毕，并要求释放传输连接。

[1] [2] **窗口字段**：占2B.窗口字段用来控制对方发送的数据量，单位为字节（B).记住一句话：**窗口字段明确指出了现在允许对方发送的数据量**。例如，设确认号是701,窗口字段是1000.这就表明，从701号开始算起，发送此报文段的一方还有接收1000B数据的接收缓存空间。

[1] [3] 校验和字段：占2B.校验和字段检验的范围包括首部和数据两部分。在计算校验和时，和UDP一样，要在TCP报文段的前面加上12B的伪首部（只需将UDP伪首部的第4个字段的17改为6,其他和UDP一样）。

[1] [4] 紧急指针字段：占2B.前面已经讲过紧急指针指出在本报文段中的紧急数据的最后
一个字节的序号。

[1] [5] 选项字段：长度可变。TCP最初只规定了一种选项，即最大报文段长度MSS.MSS告诉对方TCP:“我的缓存所能接收的报文段的数据字段的最大长度是MSS字节。”

[1] [6] 填充字段：为了使整个首部长度是4B的整数倍。

可靠性控制（重要）

丢包重传

[1] **发送方**:

重新发送丢失的片段，未被正确接收前一直存在发送方的缓存区

需要开辟发送缓冲区（发送窗口）：开始指针：send_base（指向发送缓存的最左侧），窗口大小：n，下一个序列号：nextseqnum

> 当有数据需要发送时，检查nextseqnum是否有效，有效就将其封装成TCP的段发送，发送后将nextseqnum右移，一直移动到nextseqnum-send_base=N，此时窗口就满了，应用层再给数据就无法发送了。接收方确认数据收到，然后发送窗口右移。

如何知道段丢失了吗?

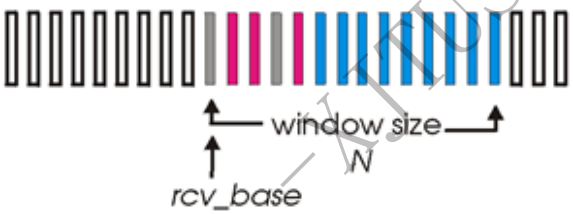
> 接收方发送确认段序号
>
> 计时器时间到了还没收到确认信号就认为段丢了，重传

如何知道哪些部分丢失了？序列号

2 **接收方**:

对期望的那个段进行确认 ——返回的段序号是它期待的那个段的段序号

需要开辟接收缓冲区：开始指针：rcv_base（指向的段序号就是接收方期待的段的段序号）以及窗口大小N



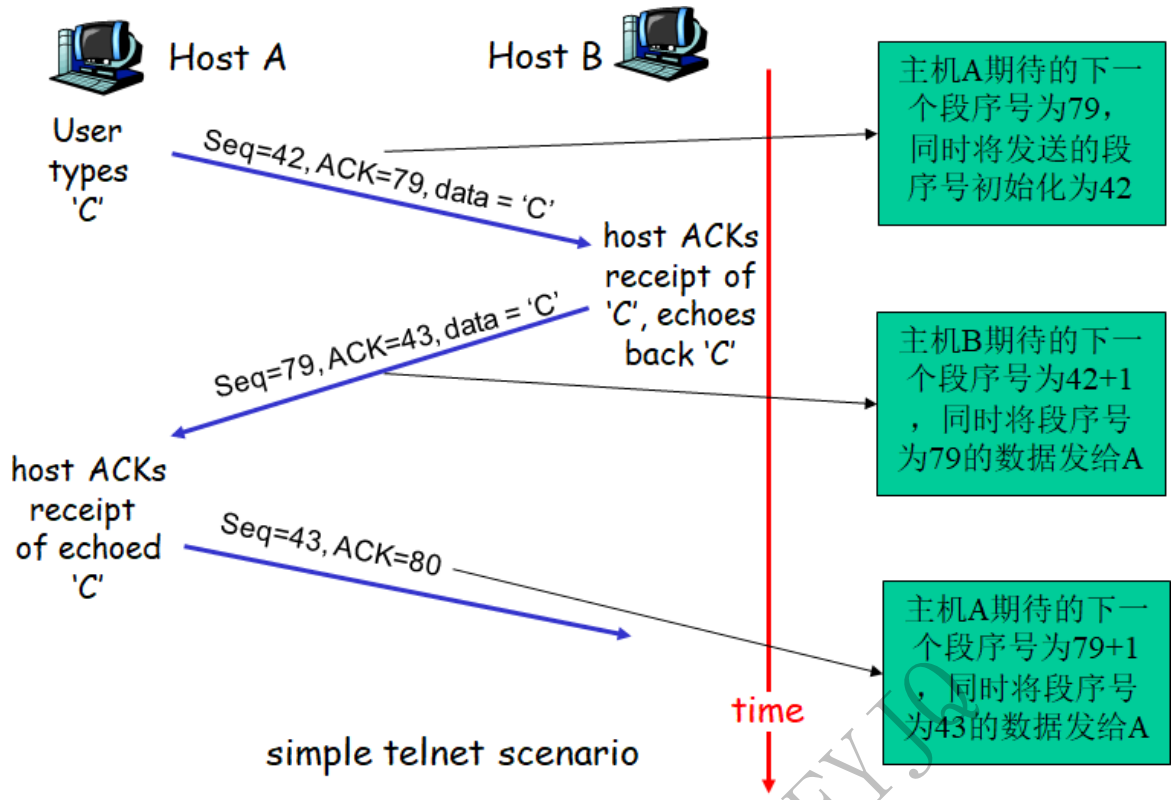
> 第一个灰色：rcv_base指向的段序号就是接收方期待的段的段序号
>
> 紫红色：收到了这个段，没有差错，先缓存起来，但是不能送给应用层，如果送给应用层会乱序（不可靠），因为期待的那个段还没有到，到了一起送
>
> 第二个灰色：还没有收到，下一个期待的段
>
> 蓝色：空闲接收缓存

示例

后一个段的段序号=前一个段的段序号+前一个段的数据域长度

Seq：段数据的第一个字节的字节流“数”，随机选择一个初始序列号

ACK：从另一端期望的下一个字节的Seq



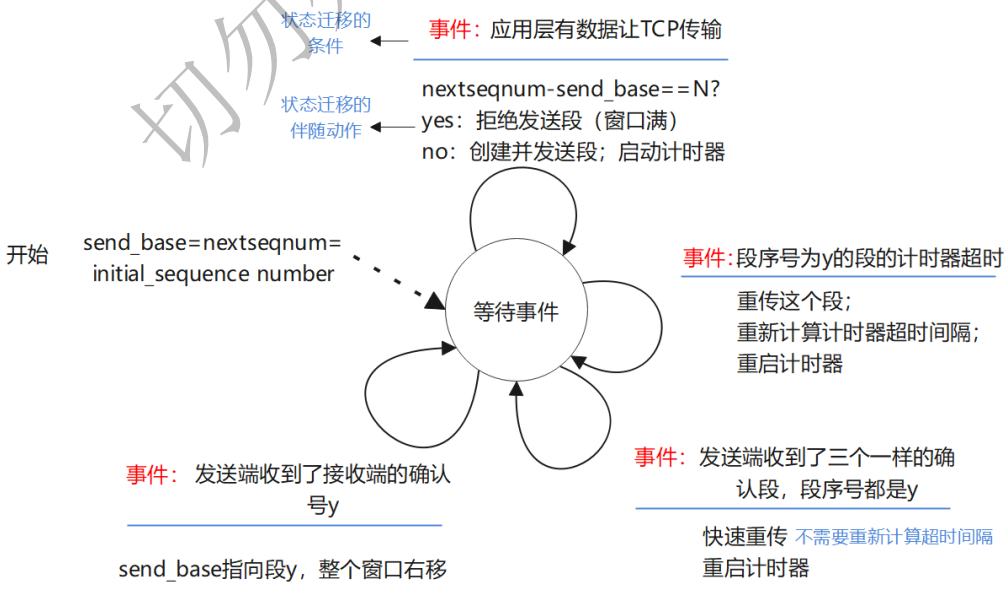
算法表示

发送端

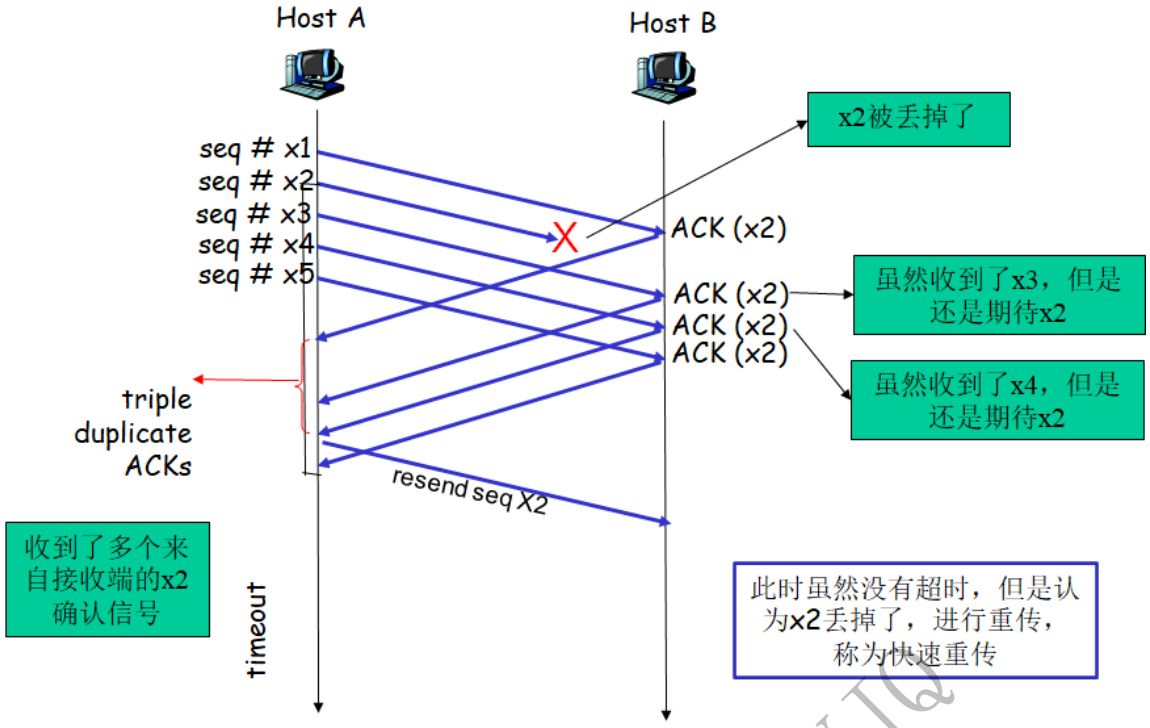
发送端简化后的可靠性控制算法：单向数据传输；无流量，拥塞控制

有限状态机图如下：

只有一个状态：等事件，当一个事件发生时，状态迁移到自己继续等事件



快速重传的解释如下：代表了轻度拥塞，超时是重度拥塞

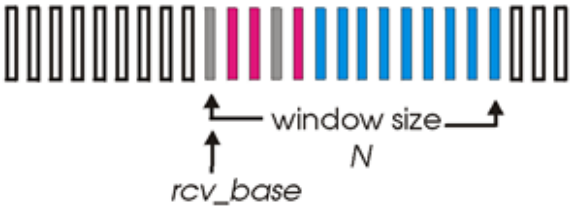


伪代码表示如下

```
/*发送端可靠性控制伪代码，要记住*/
/*假设发送方不受TCP流量和拥塞限制，来自上层数据的长度小于MSS，且数据传输只在一个方向进行*/
send_base = init_sequence number
nextseqnum = init_sequence number
loop(永远){
    switch(事件)
    事件:应用层有数据让TCP传输
        if(nextseqnum-send_base<N){
            创建段序号为nextsqnum的段
            启动计时器
            将段发给IP层
            nextseqnum = nextseqnum + length(data)/*段序号是跳跃式的*/
        }else{
            拒绝发送段
        }
    事件:段序号为y的段的计时器超时
        重传这个段y
        重新计算计时器超时间隔
        重启计时器
    事件:接收到ACK，字段值为y
        if(y>send_base){/*段在发送窗口内*/
            取消掉段y之前所有的段的计时器
            send_base = y/*窗口右移*/
        }else{/*这里指的是y=send_base，接收还没有收到y*/
            对ACK字段为y的计数器+1
            if(计数器的值==3){
                快速重传段y
                重启段y的计时器
            }
```

```
        }
}
```

接收端

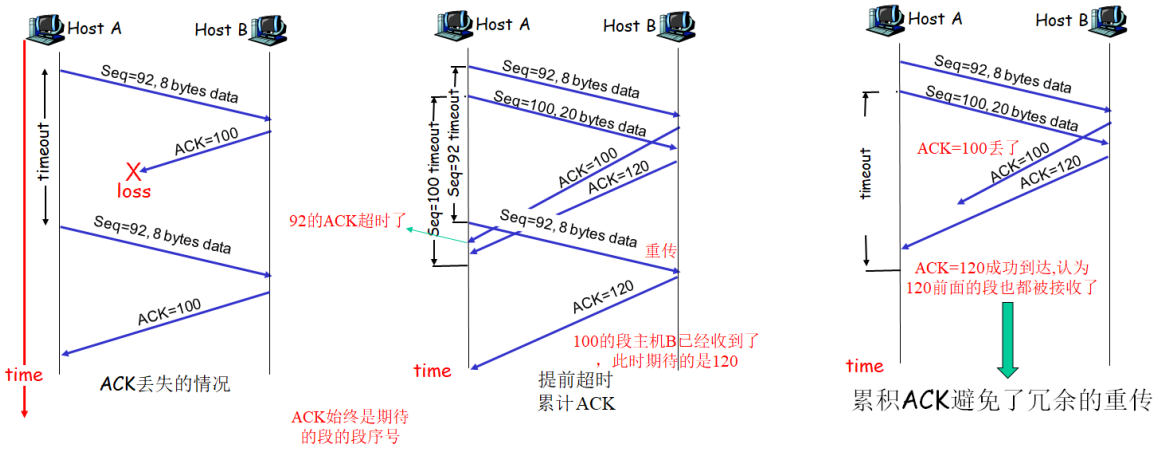


> 第一个灰色：rcv_base指向的段序号就是接收方期待的段的段序号
>
> 紫红色：收到了这个段，没有差错，先缓存起来，但是不能送给应用层，如果送给应用层会乱序（不可靠），因为期待的那个段还没有到，到了一起送
>
> 第二个灰色：还没有收到，下一个期待的段
>
> 蓝色：空闲接收缓存

产生**ACK**

编号	事件	TCP接收端动作
①	有序到达一个段，中间没有间隙，所有的其他段都被确认过了	做一个延迟，等待下一个段500ms，如果下一个段到来了一起确认，没有到来的话发送ACK
②	有序到达一个段，中间没有间隙，有一个ACK在做延时	不能再做延时，立即发送ACK(一个)
③	乱序到达一个段，段序号比期待的段序号要高，此时会产生一个gap（如红色部分）	立即发送一个ACK（跟前一个一样），ACK为期待的段的段序号（rcv_base指向的段的段序号）
④	到达一个段，这个段部分或全部的填满了gap	如果是rcv_base指向的段（左边灰色的），则这个段变为红色，连同身后红色的段一起送给应用层，接收窗口右移到下一个期待的段（右边灰色的）；否则（右边灰色的），则这个段变为红色，返回一个ACK，ACK为期待的段的段序号（rcv_base指向的段的段序号）
⑤	如果收到一个段位于窗口左侧	将其丢弃（这种情况是老师上课补充的）

举例分析如下



全双工通信的话发送端和接收端都放一份上面的两种可靠性算法

TCP往返时间和超时

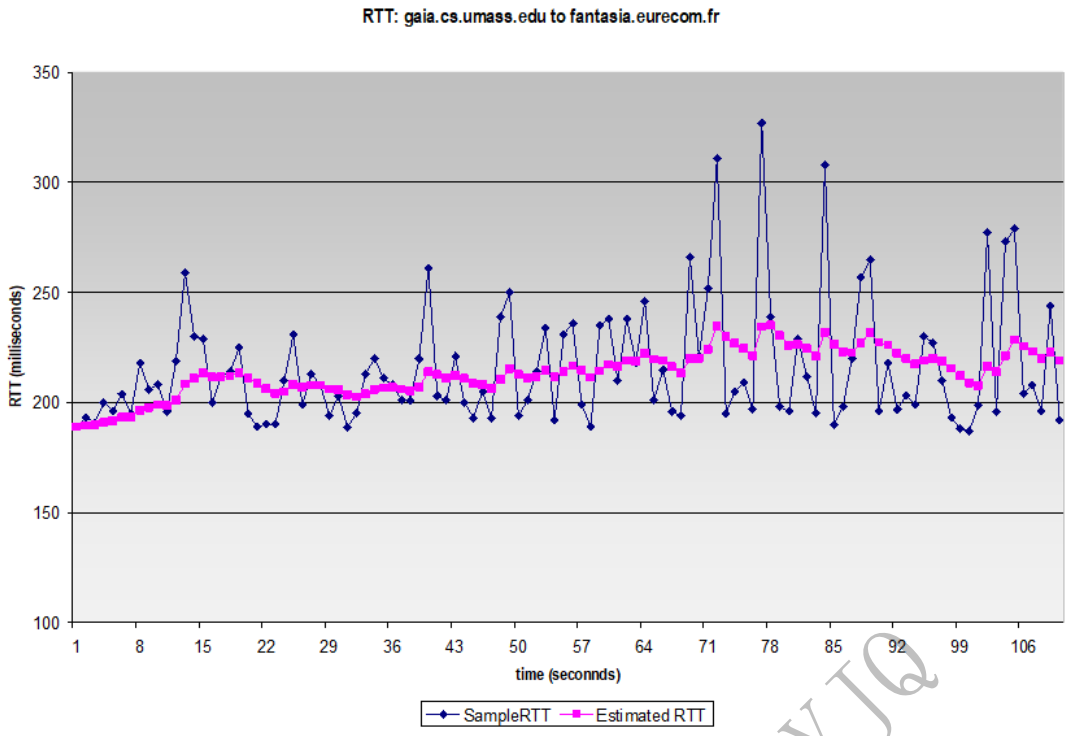
如何设置TCP超时时间间隔：略大于一个RTT

可以预测RTT：

> SampleRTT（实际测量的RTT):从段传输到ACK接收的时间
>
> 忽略重传，累积的分段
>
> SampleRTT会因路由器拥塞和终端系统负载变化而变化，因此，我们希望估计的RTT“更平稳”。

$$EstimatedRTT_n = (1-\alpha) * EstimatedRTT_{n-1} + \alpha * SampleRTT$$

指数加权运动平均；给定样本的影响以指数速度递减；α的典型值为0.125(右移三位，速度更快)



自适应超时时间设置：EstimtedRTT加上“安全margin”

$$Timeout = EstimatedRTT + 4 * Deviation$$

其中，

$$Deviation_n = (1-\beta) * Deviation_{n-1} + \beta * |SampleRTT - EstimatedRTT|$$

β通常取0.25

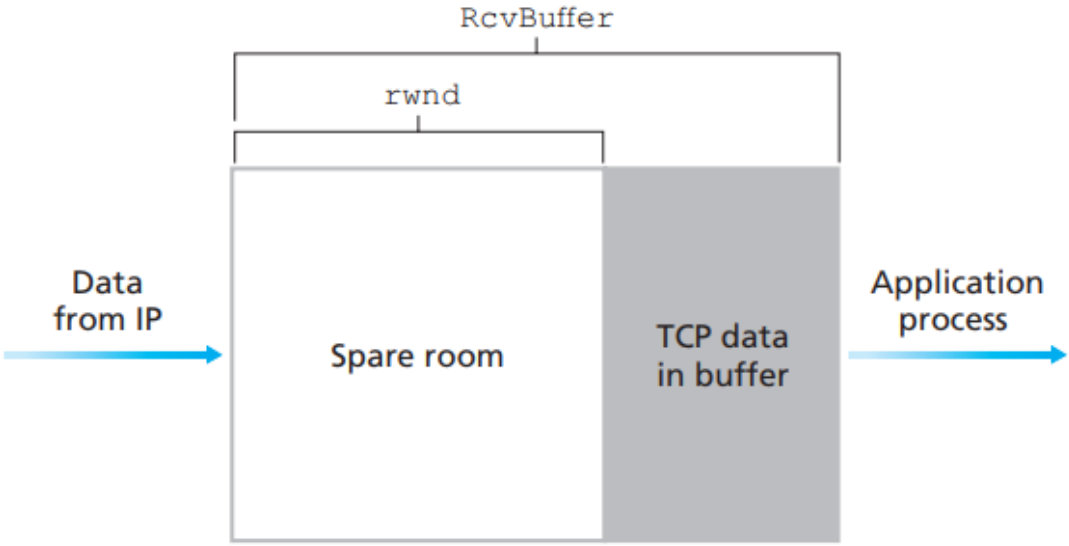
这个公式也可以用于预测接收窗口rwnd的大小

流量控制Flow Control

前面讲过，一条TCP连接的每一侧主机都为该连接设置了接收缓存。当该TCP连接收到正确、按序的字节后，它就将数之前重传丢失的报文段据放人接收缓存。相关联的应用进程会从该缓存中读取数据，但不必是数据刚一到达就立即读取。事实上，接收方应用也许正忙于其他任务，甚至要过很长时间后才去读取该数据。如果某应用程序读取数据时相对缓慢，而发送方发送得太多、太快，发送的数据就会很容易地使该连接的接收缓存溢出。

TCP为它的应用程序提供了流量控制服务，以消除发送方是接收方缓存溢出的可能性。流量控制是一个速度匹配服务，即发送方的发送速率和接收方应用程序的读取速率相匹配。

TCP通过让发送方维护一个称为**接收窗口receive window**的变量来提供流量控制，接收窗口用于给发送方一个指示——**接收方还有多少可用的缓存空间**。接收缓存中的空闲空间。



TCP段头中有一个字段表示接收窗口的大小。

假设主机A通过一条TCP连接向主机B发送一个大文件，主机B为该连接分配了一个接收缓存，并用RcvBuffer来表示其大小，主机B上的应用进程不时地从该缓存中读取数据，有如下变量定义

> LastByteRead：主机B上的应用进程从缓存读出的数据流的最后一个字节的编号。（被读走的最后一个段序号）
>
> LastByteRcvd：从网络中到达的并且已放入主机B接收缓存中的数据流的最后一个字节的编号。（刚收到的段的段序号）

由于 TCP 不允许已分配的缓存溢出，下式必须成立：

$$LastByteRcvd - LastByteRead \leq RcvBuffer$$

接收窗口(空闲缓存)用RcvWindow表示，根据缓存可用空间的数量来设置：

$$RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]$$

主机 B 通过把**当前的RcvWindow值放入它发给主机 A 的报文段接收窗口字段**中，通知主机 A 它在该连接的缓存中还有多少可用空间。开始时，主机B设定RcvWindow = RcvBuffer。

主机A 轮流跟踪两个变量，LastByteSent 和 LastByteAcked，这两个变量的意义很明显。注意到这两个变量之间的差LastByteSent - LastByteAcked，就是**主机A发送到连接中但未被确认的数据量**。通过将未确认的数据量控制在值 rwnd 以内，就可以保证主机 A 不会使主机B的接收缓存溢出。因此，主机A在该连接的整个生命周期须**保证**：

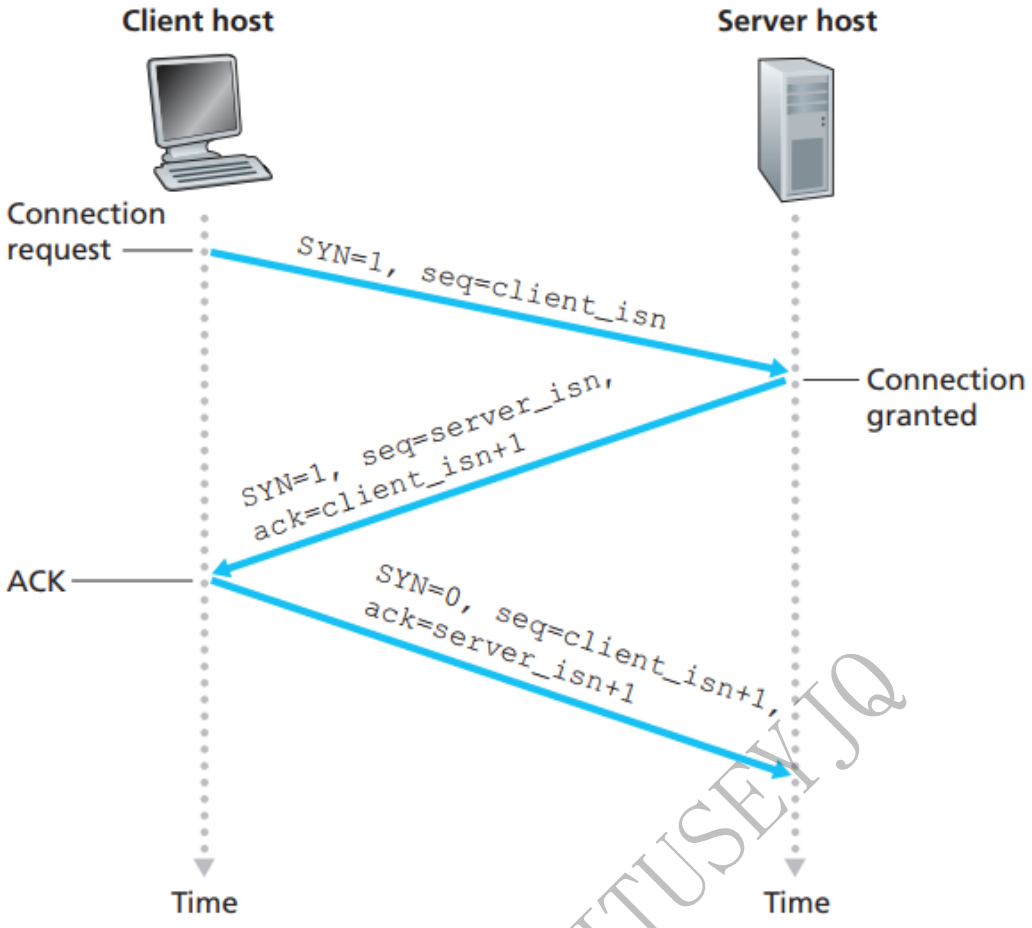
$$LastByteSent - LastByteAcked \leq rwnd$$

对于这个方案还存在一个小小的技术问题。为了理解这一点，假设主机B的接收缓存已经存满，使得rwnd=0。在将rwnd=0通告给主机A之后，还要假设主机B没有任何数据要发给主机A。此时，考虑会发生什么情况。因为主机B上的应用进程将缓存清空，TCP并不向主机A发送带有rwnd新值的新报文段；事实上，**TCP仅当在它有数据或有确认要发时才会发送报文段给主机A**。这样，主机A不可能知道主机B的接收缓存已经有新的空间了，即主机A被阻塞而不能再发送数据！为了解决这个问题，TCP规范中要求：**当主机B的接收窗口为0时，主机A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确认。**最终缓存将开始清空，并且确认报文里将包含一个非0的rwnd值**。

TCP连接管理(重要)

连接的开启：三个握手

> [1] 发送端(客户端)给接收端(服务器端)发送一个SYN段(在 TCP 标头中 SYN 位字段为 **1** 的 TCP/IP 数据包), 该段中也包含客户端的初始序列号(序列号Sequence number = J)。
>
> > 同步比特SYN:同步比特SYN置为1,表示这是一个连接请求或连接接收报文，后面的TCP连接会详细讲到。
>
> [2] 接收端返回给发送端 SYN +ACK 段(在 TCP 标头中SYN和ACK位字段都为 1 的 TCP/IP 数据包)， 该段中包含接收端的初始序列号(序列号 = K)；同时使 确认号 Acknowledgement number = J + 1来表示确认已收到客户端的 SYN段(序列号 = J)。
>
> > 第二次握手，接收端开辟缓存
>
> [3] 发送端给接收端响应一个ACK段(在 TCP 标头中 ACK 位字段为 **1** 的 TCP/IP 数据包), 该段中使 确认号 = K + 1来表示确认已收到服务器的 SYN段(序列号 = K)。
>
> > 第三次握手，发送端开辟缓存



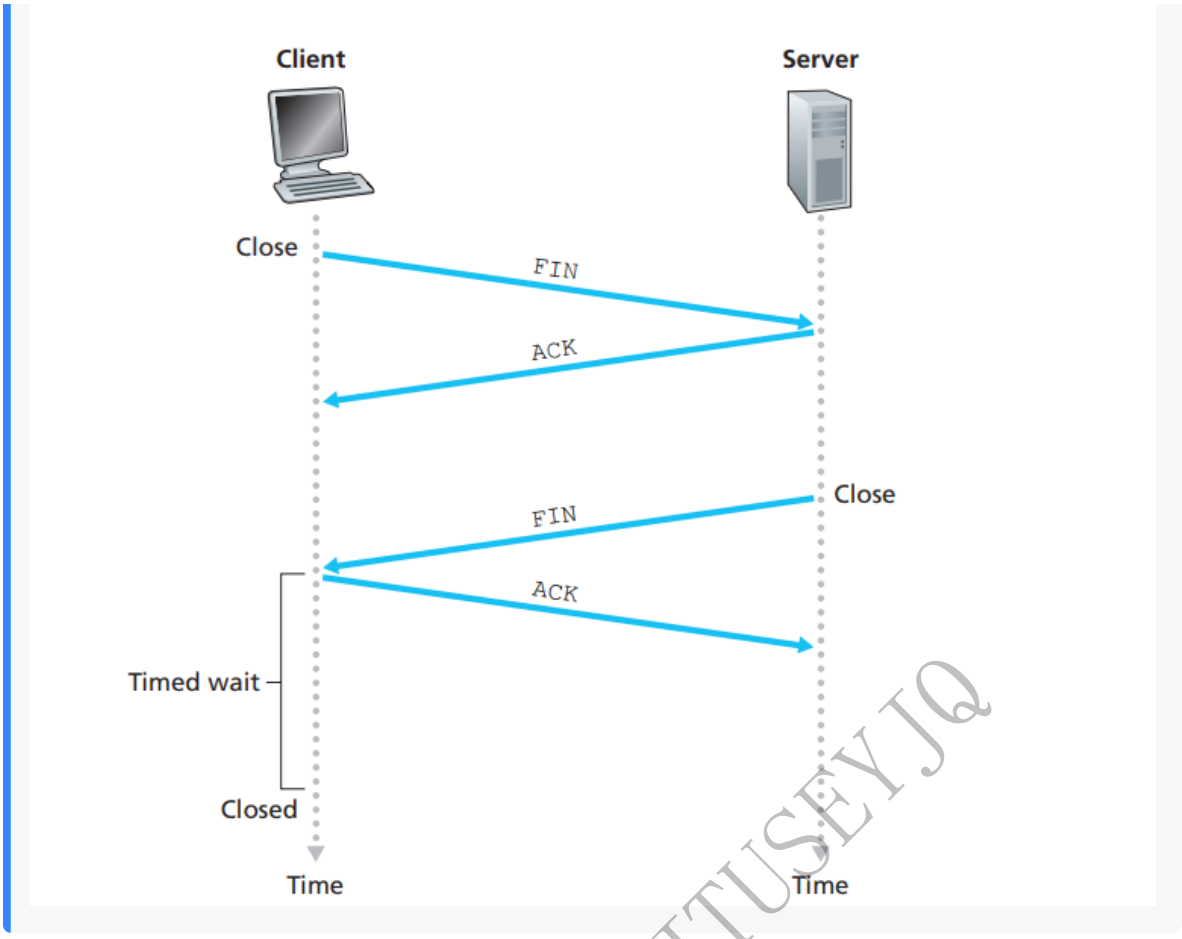
一旦完成这三个步骤，客户和服务器主机就可以互相发送包括数据的报文段了。

DoS攻击：半连接攻击(第三个握手永远不做，没完没了发送连接请求，使服务器端不断开辟缓存，使服务器崩溃)

> 很多台客户端攻击叫DDos

连接的关闭：两次握手（四次挥手）

> [1]客户端系统向服务器发送TCP FIN控制段
>
> > 终止比特FIN:释放一个连接。当FIN=1时，表明此报文段的发送端的数据已发送完毕，并要求释放传输连接。
>
> [2]服务器接收到FIN，返回ACK。关闭连接，发送FIN。
>
> [3]客户端收到FIN，以ACK回应。
>
> 计时器定时等待
>
> [4]服务器，接收ACK。连接关闭。
>
> timed wait后TCP才真正的关闭



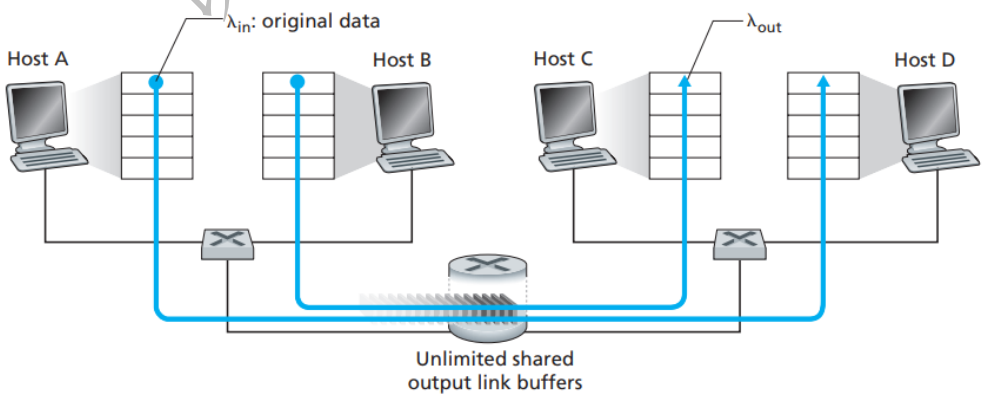
拥塞控制Congestion Control

拥塞指的是路由器拥塞：有太多的发送端发送数据，发的太快

表现：丢包(路由器缓冲区溢出)；长延迟(在路由器缓冲区中排队)

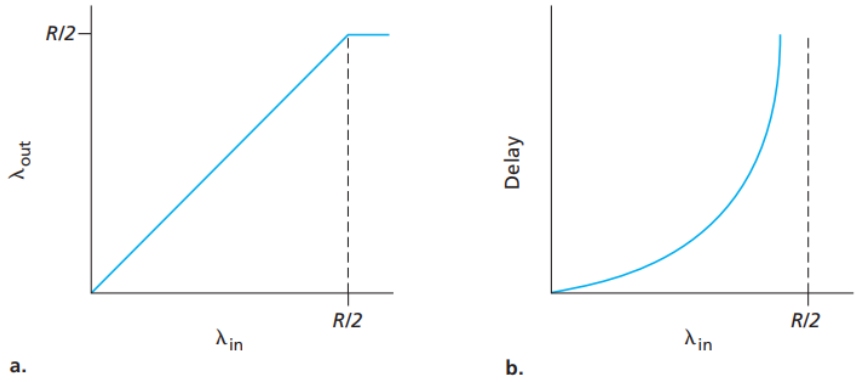
拥塞原因与代价（了解即可）

情况一：两个发送端和一台无限大缓存路由器

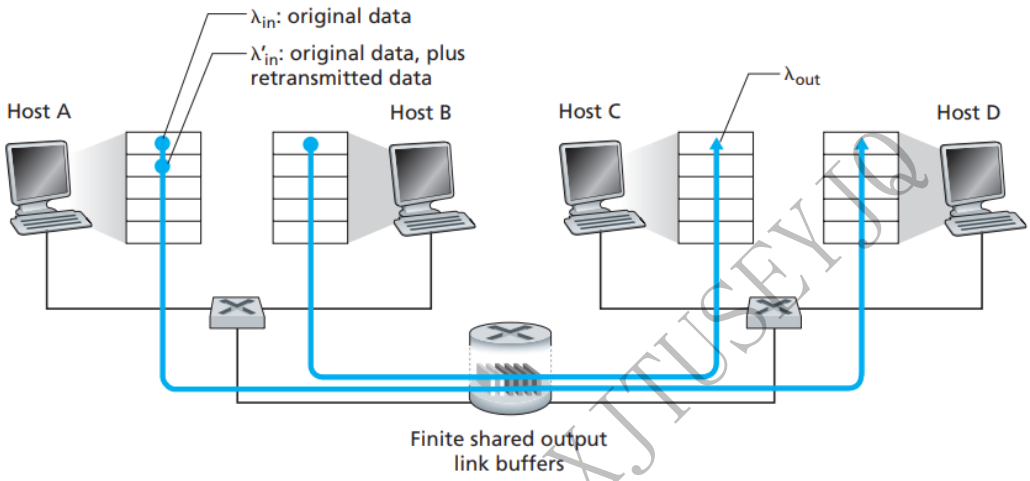


λ_{in}：主机A(B)中的应用程序以λ_{in}字节/秒的平均速率将数据发送到连接中

来自主机A和主机B的包通过一台路由器，在一段容量为R的共享式输出链路上传输，路由器缓存无限大说明不会丢包。当两个包的发送速度大于路由器的交换能力时$2\lambda_{in} > R$，就会产生拥塞。



情况二：两个发送端和一台有限缓存路由器



此时，路由器的缓存是有限的，也符合实际情况，当包到达一个已满的缓存时会被丢弃，从而被发送端重传。

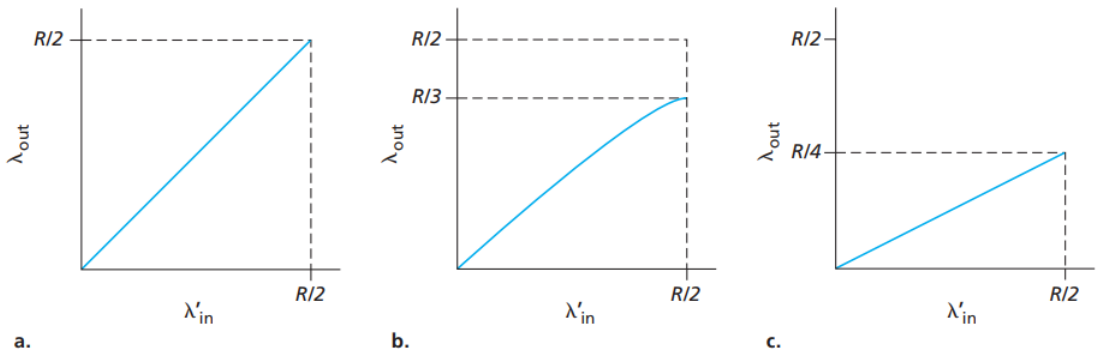
λ_{in}：表示应用程序将初始数据发送到套接字中的速率

λ'_{in}：表示传输层向网络中发送报文段（**含有初始数据和重传数据**）的速率，也称为网络的供给载荷（offered load）

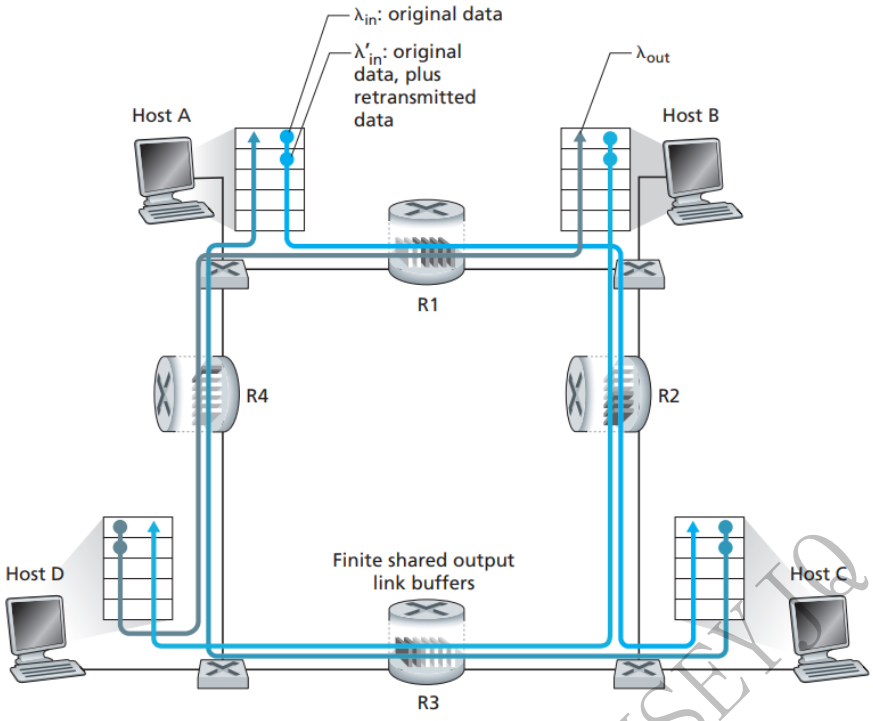
1 λ_{in}较小时，$\lambda_{in} = \lambda_{out}$

2 出现丢包重传，$\lambda'_{in} > \lambda_{out}$

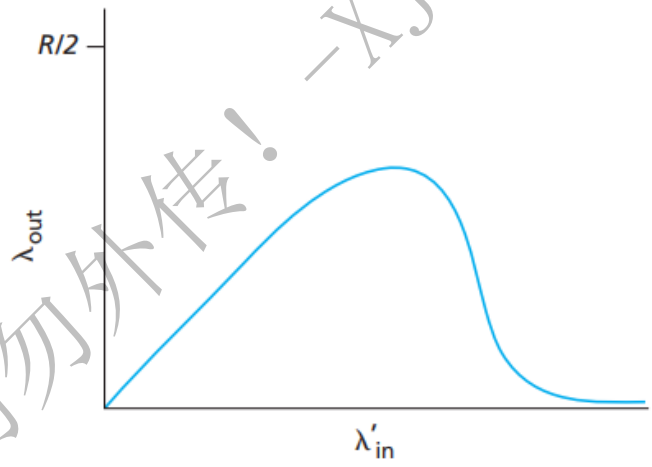
3 延时较大，会进行不必要的重传，实际有效资源只用到了一半（被垃圾包占用了）



情况三：4个发送方和具有有限缓存的多台路由器以及多跳路径



对角发送。无论在哪一个路由器丢包，都会造成资源浪费。



拥塞控制方法分类

1 网络帮助的拥塞控制

ATM(异步传输模式)的内核设备不叫路由器，叫ATM交换机（胖内核，瘦端系统，内核功能强大），拥塞以后通知发送端——**网络帮助的拥塞控制**

> 交换机直接通知给发送端
>
> 交换机通知给接收端，接收端再通知发送端（用的更多）

2 端到端的拥塞控制

因特网是瘦内核，胖端系统的网络，端系统功能强大（如DNS就放在端系统），路由器的功能尽量简单，拥塞了不通知发送端，靠端系统自行感知。

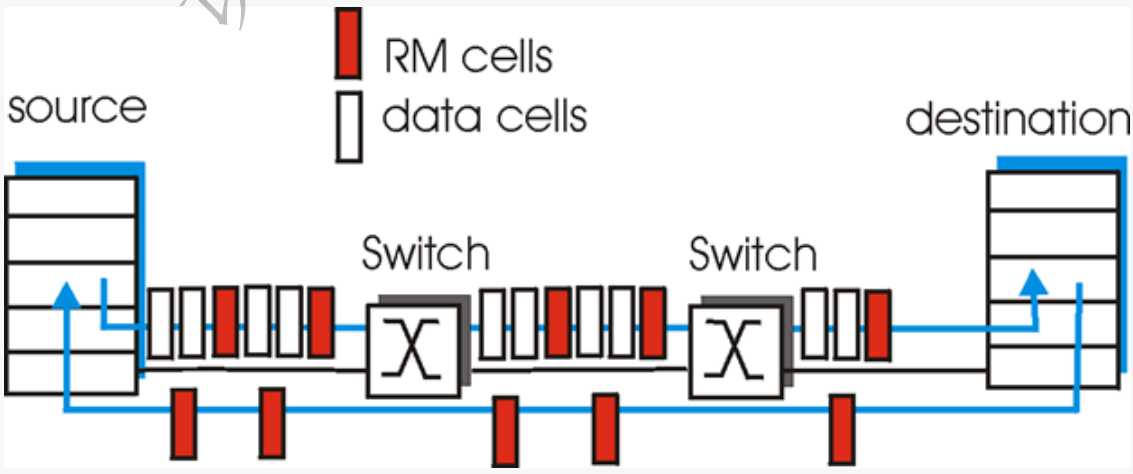
> 超时了说明网络重度拥塞（端到端的延时跟距离有关）
>
> 收到三个相同ACK（丢包）说明网络轻度拥塞——通常采用此方法

ATM的拥塞控制(过时了，了解即可)

ATM提供的四种业务：ABR、UBR、CBR、VBR

> 1 **ABR**: Available Bit Rate 有效位率服务,主要用于视频服务; 可以保证一个最小带宽,可能丢包.
>
> 2 CBR : Constant Bit Rate 主要用于实时语音通信; 不会丢包,不需要拥塞控制.
>
> 3 VBR : Variable Bit Rate 不会丢包,不需要拥塞控制.
>
> 4 UBR : Unspecified Bit Rate 使用时有资源则使用,无资源则丢包,免费使用,无拥塞控制

ATM 的通信数据单元称为信元(Cell)。

> 头是5个字节（存访拥塞指示信息），数据域是48个字节
>
> 分为两种：
>
> 1 data cells:数据信元
>
> 2 RM (resource management) cells资源管理信元：存访拥塞信息，通常每几十个数据信元放一个资源管理信元
>
> 
>

ATM网络针对ABR的拥塞控制方法有三种

> 1 **CI和NI bits**

CI bit：Congestion Indication 拥塞指示ATM 信元头中有一位即为CI 位,当发生拥塞时将 CI 位置1。发送端在接收到CI 位置1 的信息后将会降低发送速率。
NI bit：No Increasein rate 告诉发送端不要再增加发送速率。代表网络即将进入拥塞或发生轻度拥塞。

每32 个信元中插入一个资源管理信元,交换机对资源管理信元中的CI 与NI 位进行置位,发送端根据这两位进行速率控制。

[2] ER Setting (Explicit Rate Setting),明确速率设置

交换机根据可用带宽,在资源管理信元中有一个字段,告诉发送端可以多大速率发送数据.如果**经过多个交换机,会取一个最小值**。

[3] EFCI(Explicit Forward Congestion Indication)明确转发拥塞指示

在数据信元中的位,如果拥塞则置1。

TCP拥塞控制（重要）

对于TCP中的拥塞，一共有两种判断

超时了说明网络重度拥塞（端到端的延时跟距离有关）

收到三个相同ACK（丢包）说明网络轻度拥塞——通常采用此方法

探测拥塞

为了探测网络是否拥塞，先发一个段探测一下，如果这个段的确认信息正确返回，则没有问题；下一个RTT开始时发送两个段，如果还没有问题，下一个RTT开始时发送四个段，……，每个RTT发送窗口以2的倍数增加，经过若干次探测之后，此时还没丢包，发送窗口不能再x2，每个RTT开始时窗口大小+1 （慢慢地增加）

发送窗口以2的倍数增加的过程叫**慢启动**，经过若干RTT后+1过程叫做**拥塞避免**，这两个结合起来就是TCP的拥塞算法

算法中的重要变量

[1] 从慢启动到拥塞避免的分界线用一个变量threshold（阈值）表示

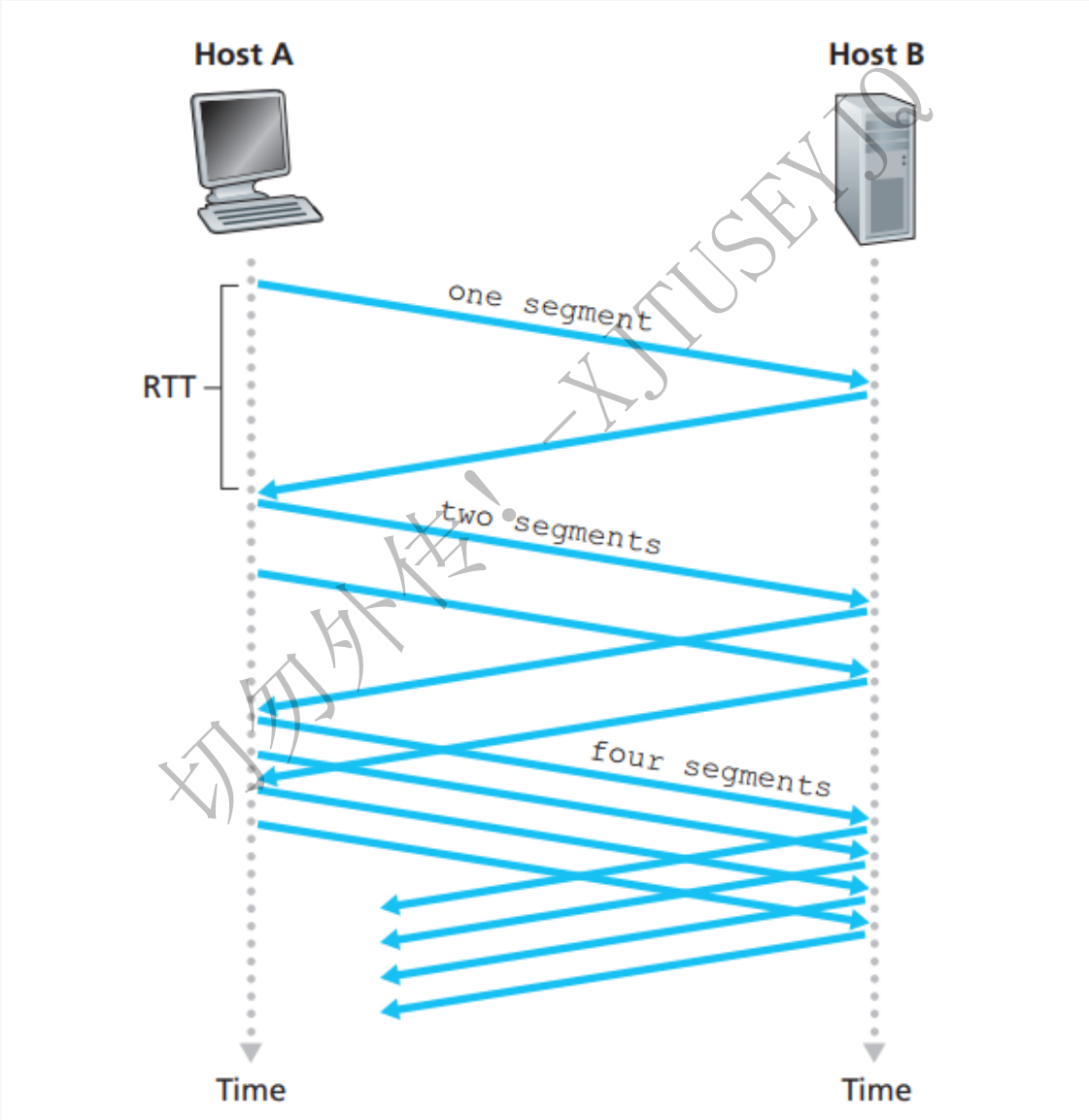
[2] Congwin：拥塞窗口大小（拥塞控制时使用的发送窗口）

TCP慢启动

慢启动算法伪代码表示

```
初始化：threshold=适当的值(10、20 ... 不要太大)
初始化：Congwin=1
for(每个确认段)
    Congwin++
until(丢包orCongWin≥threshold)
```

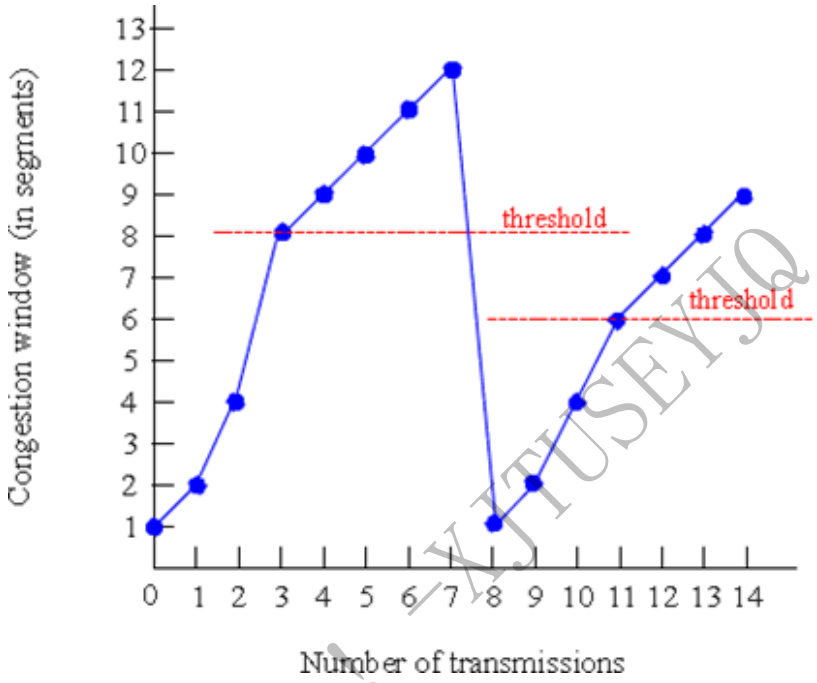
💻 这里是说没收到一个对新的报文段的确认后，拥塞窗口就+1，第一轮收到1个确认，第二轮2个，第三轮4个，以此类推，按轮次加倍



TCP拥塞避免

Tahoe 拥塞避免算法伪代码

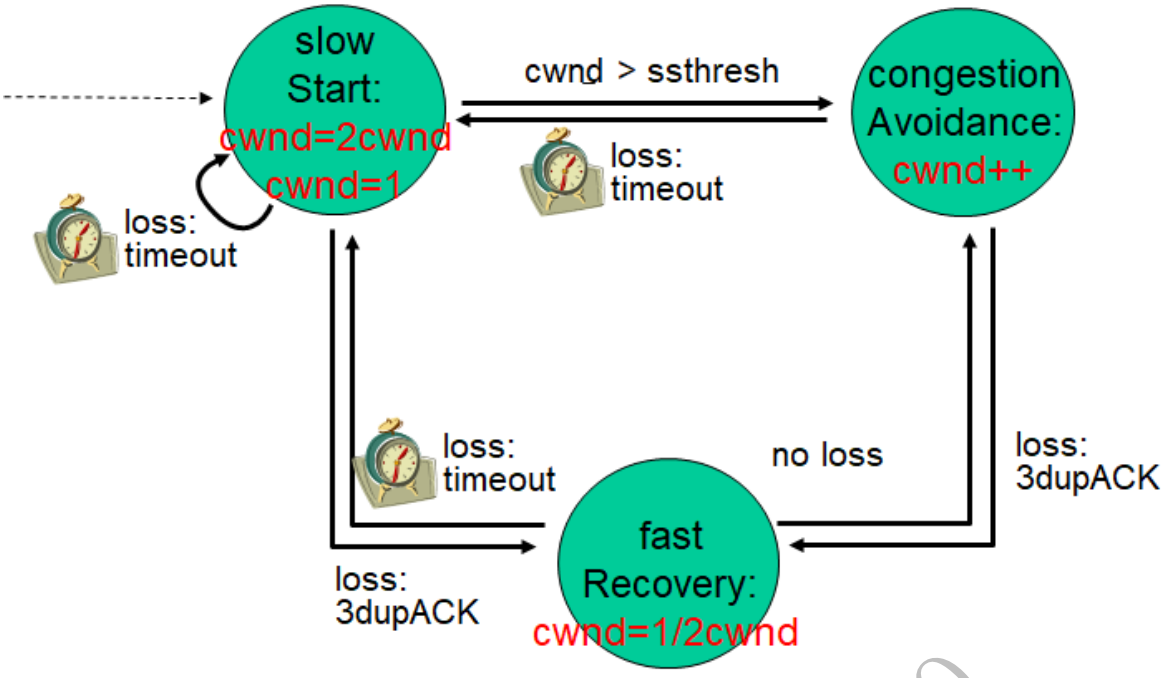
```
/*慢启动结束*/
while (没有丢包) {
  每w个段被确认:
      Congwin++/*每个RTT，窗口+1*/
}/*线性增加*/
/*丢包了*/
threshold = Congwin/2
Congwin = 1
进行慢启动
```



轻度拥塞也会时发送窗口变为1，显得不太合理

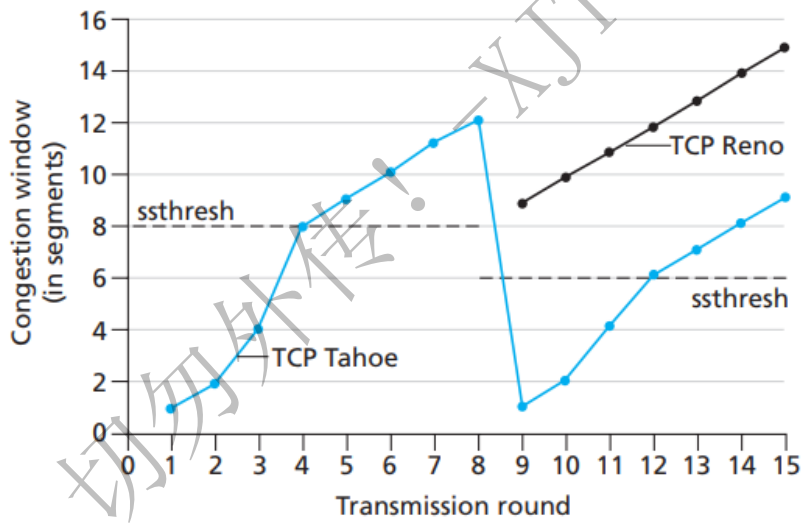
Reno 拥塞避免算法伪代码

```
/*慢启动结束*/
while (没有丢包) {
  每w个段被确认:
      Congwin++/*每个RTT，窗口+1*/
}/*线性增加*/
/*丢包了*/
threshold = Congwin/2
if(因为超时丢包){/*重度拥塞*/
    Congwin = 1
    进行慢启动/*回到慢启动*/
}
if(因为收到三个相同确认段丢包){/*轻度拥塞*/
    Congwin = Congwin /2/*回到while*/
}/*快速恢复*/
```



问题：如果又收到三个相同的确认段，此时应该再减半

两个算法的比较



Reno算法的吞吐率更高，震荡率更小

拥塞避免中的吞吐率

设W是丢包时的窗口大小（以段为单位）

> 当窗口大小是W，吞吐率=W/RTT
>
> 当发生了丢包，窗口大小减少为W/2，吞吐率=$W/2RTT$
>
> 平均窗口大小为(W+W/2)/2=0.75W，平均吞吐率=$0.75W/RTT$

如果直到丢包率L，MSS为最大段大小(可要可不要），平均吞吐率为

$$\approx \frac{1.22MSS}{RTT\sqrt{L}}$$

推导过程如下：

> 在拥塞避免期间，发送窗口大小从w/2变化到w(段)
>
> 第一个RTT，窗口大小=w/2
>
> 第二个RTT，窗口大小=w/2+1
>
> 第三个RTT，窗口大小=w/2+2
>
> ……
>
> 当丢包时，窗口大小=w/2+w/2=w
>
> 在此期间发送的段的总数(包数)
>
> $$\frac{w}{2}(\frac{w}{2}+1)+(1+2+,\ldots+\frac{w}{2})=\frac{3w^2}{8}+\frac{3w}{4}\approx\frac{3w^2}{8}$$
>
> 总共丢了一个包，丢包率为
>
> $$L=\frac{8}{3w^2}\Rightarrow w=\sqrt{\frac{8}{3L}}$$
>
> 平均吞吐率则为
>
> $$\frac{3}{4}w/RTT=\frac{3}{4RTT}\sqrt{\frac{8}{3L}}\approx\frac{1.22MSS}{RTT\sqrt{L}}$$

算法总结

TCP拥塞避免算法：AIMD：radditive increase, multiplicative decrease

> 线性增加，指数减少
>
> 每一个RTT增加一次窗口；每次丢包减少为原来窗口大小的1/2

这个算法具有四个特性（也称为TCP的四个特性）

> [1] 有效性：Effectiveness
>
> [2] 收敛性
>
> [3] 公正性：Fairness
>
> > 如果N个TCP会话共享同一条瓶颈链路，则每个会话的链路容量应为1/N

> 没有绝对的公平

4 友好性：Friendliness

> 如果TCP和UDP用户共同使用带宽。如果两者发送数据的速度>R(路由器最大交换能力),TCP就会将发送速率降一半，腾出资源给UDP，最终TCP只能发送一个段，资源几乎都给了UDP

参考资料

[1] James F.Kurose，Keith W.Ross.Computer Networking—A Top-Down Approach（第6版）.北京：高等教育出版社出版者，2013年。

[2] 西安交通大学Computer Networking2022年春 课程PPT 朱利

[3] 天勤第11版 2023版计算机网络高分笔记