第3章:运输层

目的:

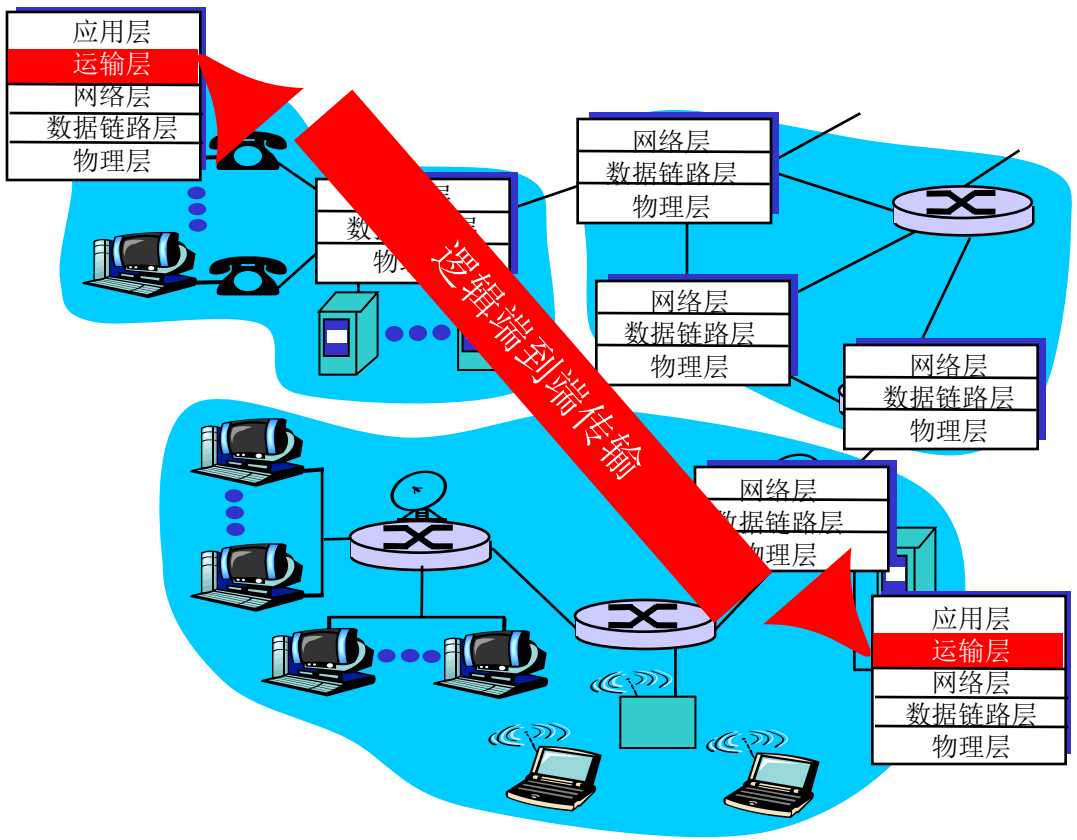
- 理解运输层服务依据的原理:
  - 复用/分解
  - 可靠数据传输
  - 流量控制
  - 拥塞控制
- 学习因特网中的运输层协议:
  - UDP: 无连接传输
  - TCP: 面向连接传输
  - TCP 拥塞控制

第3章 主要内容

- 3.1 运输层服务
- 3.2 多路复用与多路分解
- 3.3 无连接传输: UDP
- 3.4 可靠数据传输原理
  - rdt1
  - rdt2
  - rdt3
  - 流水线协议
- 3.5 面向连接的传输: TCP
  - 报文段结构
  - 可靠数据传输
  - 流量控制
  - 连接管理
- 3.6 拥塞控制原理
- 3.7 TCP拥塞控制
  - 机制
  - TCP吞吐量
  - TCP公平性

3.1 概述和运输服务

- 在运行不同主机上应用进程之间提供*逻辑通信*
- 运输协议运行在端系统中
  - 发送方：将应用报文划分为*段*，传向网络层
  - 接收方：将段重新装配为报文，传向应用层
- 应用可供使用的运输协议不止一个
  - 因特网：TCP和UDP



运输层 vs. 网络层

- *网络层:* 主机间的逻辑通信
- *运输层:* 进程间的逻辑通信
  - 依赖、强化网络层服务

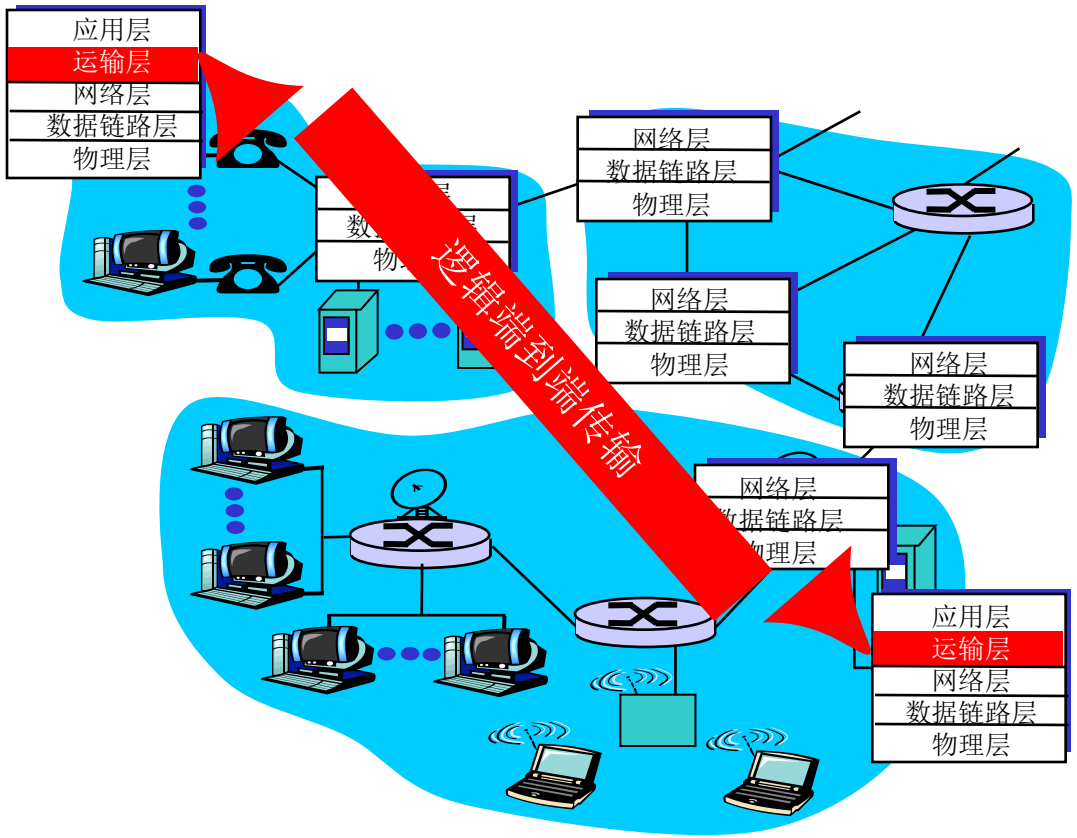
家庭类比:

12个孩子向12个孩子发信

- 进程 = 孩子
- 应用报文= 信封中的信
- 主机 = 家庭
- 运输协议 = Ann和Bill
- 网络层协议= 邮政服务

因特网运输层协议

- 可靠的、按序的交付 (TCP)
  - 拥塞控制
  - 流量控制
  - 连接建立
- 不可靠、不按序交付: UDP
  - “尽力而为”IP的不提供不必要服务的扩展
- 不可用的服务:
  - 时延保证
  - 带宽保证



第3章 主要内容

- 3.1 运输层服务
- 3.2 复用与分解
- 3.3 无连接传输: UDP
- 3.4 可靠数据传输的原则
  - rdt1
  - rdt2
  - rdt3
  - 流水线协议
- 3.5 面向连接的传输: TCP
  - 报文段结构
  - 可靠数据传输
  - 流量控制
  - 连接管理
- 3.6 拥塞控制的原则
- 3.7 TCP拥塞控制
  - 机制
  - TCP吞吐量
  - TCP公平性
  - 时延模型

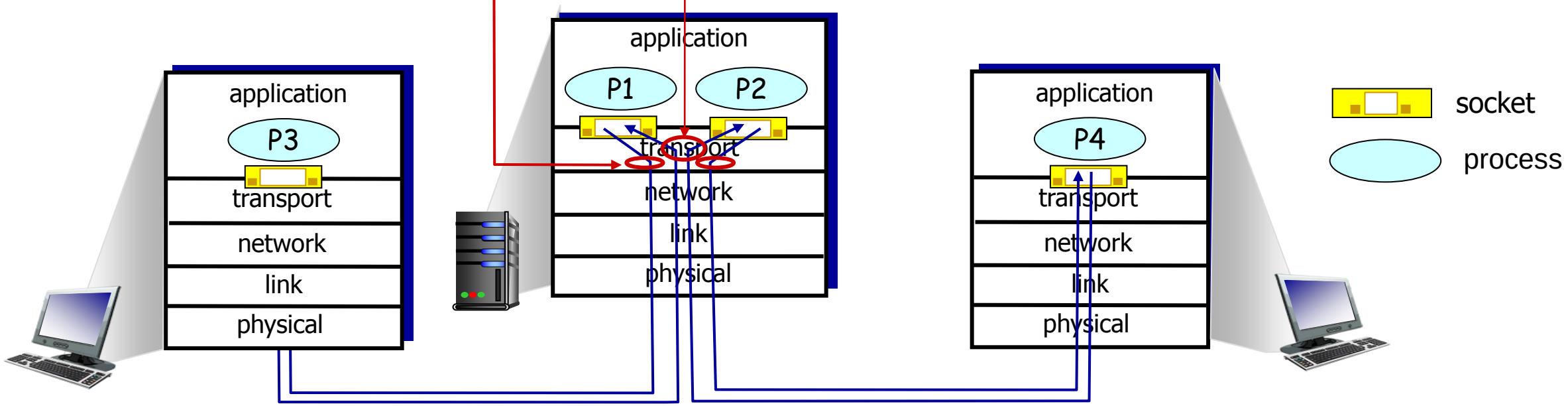
3.2 多路复用/多路分解

multiplexing at sender:

处理多个来自sockets的数据，增加运输层头部(后者将用于分解)

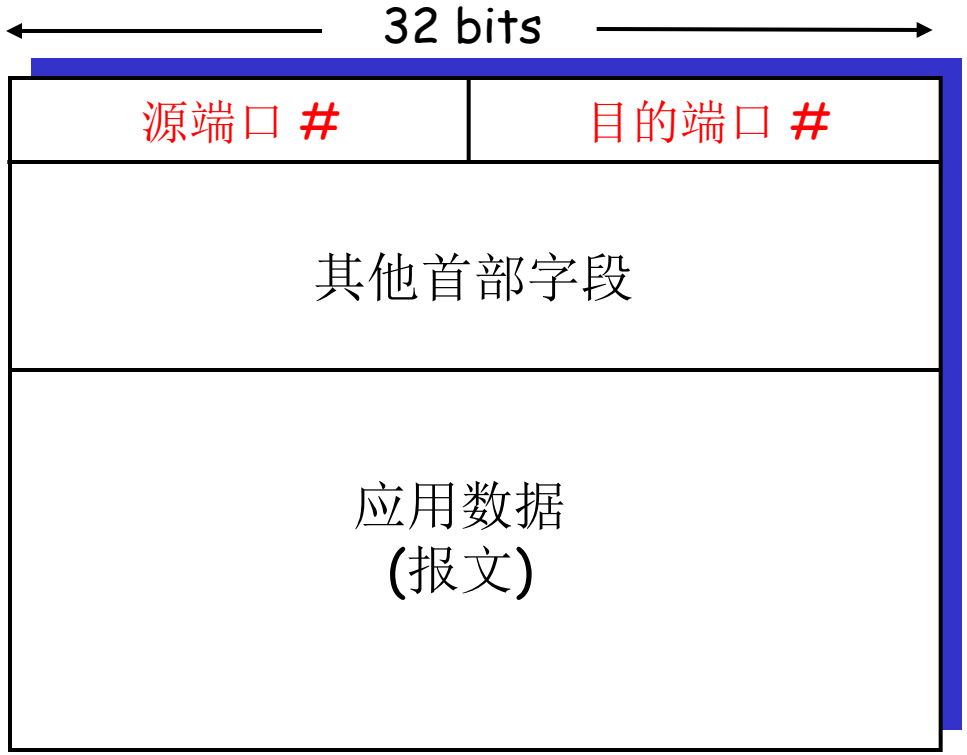
demultiplexing at receiver:

基于运输层头部信息将收到的数据段提交给正确的



分解工作过程

- 主机接收IP数据报
  - 每个数据报承载1个运输层段
  - 每个段具有源、目的端口号
    (回想: 对特定应用程序的周知端口号)
- 主机使用IP地址 &端口号将段定向到适当的套接字



TCP/UDP 段格式

无连接分解

- 生成具有本地端口号的套接字:

```
DatagramSocket mySocket1 = new
   DatagramSocket(99111);
DatagramSocket mySocket2 = new
   DatagramSocket(99222);
```

- UDP套接字由二元组标识：

(目的地IP地址, 目的地端口号)

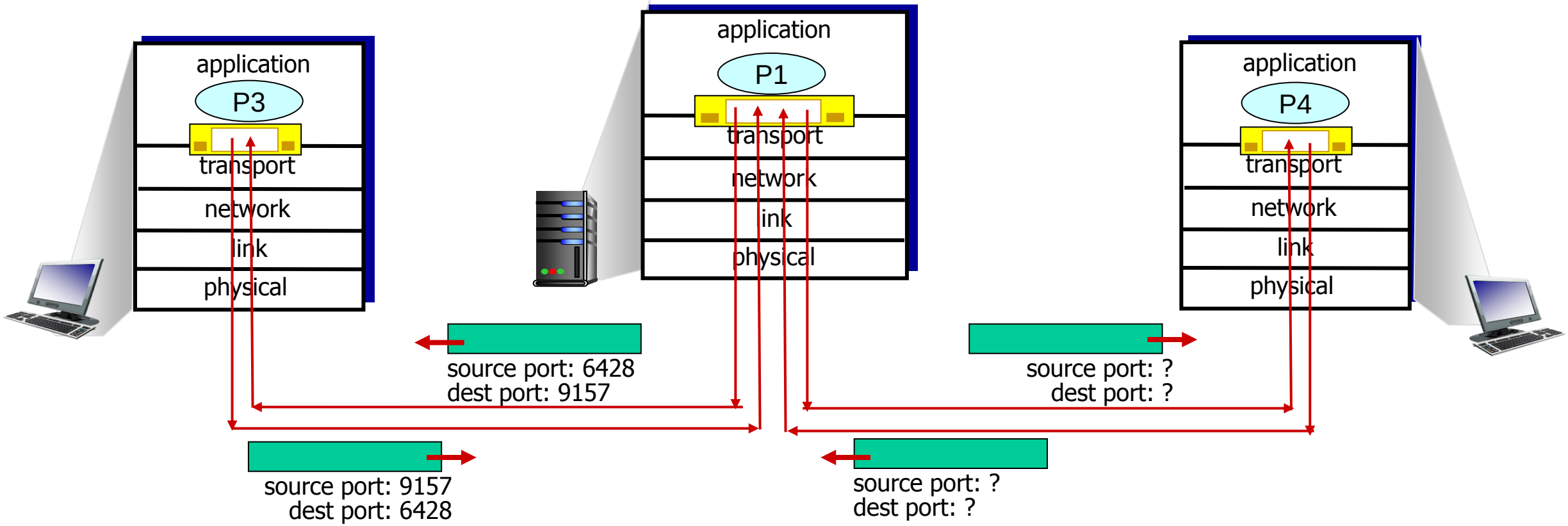
- 当主机接收UDP段时:
  - 在段中检查目的地端口号
  - 将UDP段定向到具有该端口号的套接字
- 具有**不同源IP地址和/或源端口号**，但是**具有相同目的IP地址和相同目的端口号**的IP数据报将定向到相同的套接字

无连接的多路分解的示例

```
DatagramSocket mySocket2
 = new DatagramSocket
 (9157);
```

```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

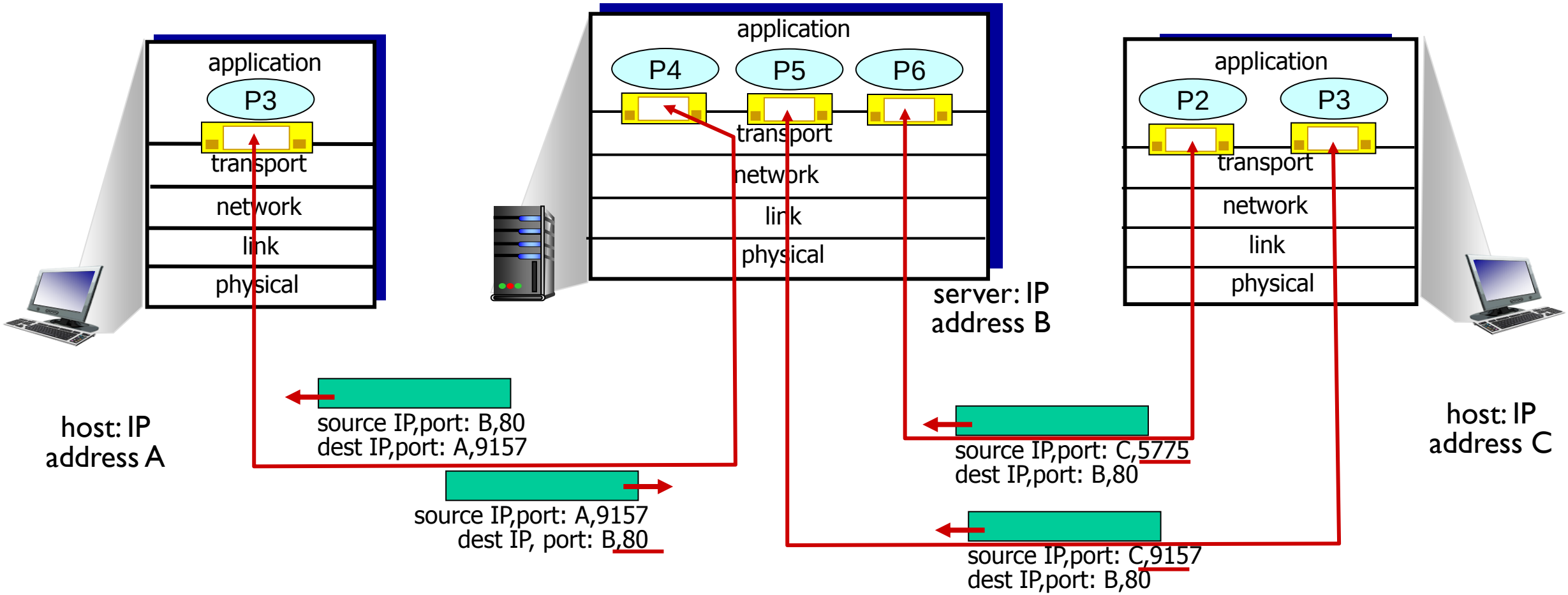
```
DatagramSocket mySocket1
 = new DatagramSocket
 (5775);
```



面向连接的多路分解

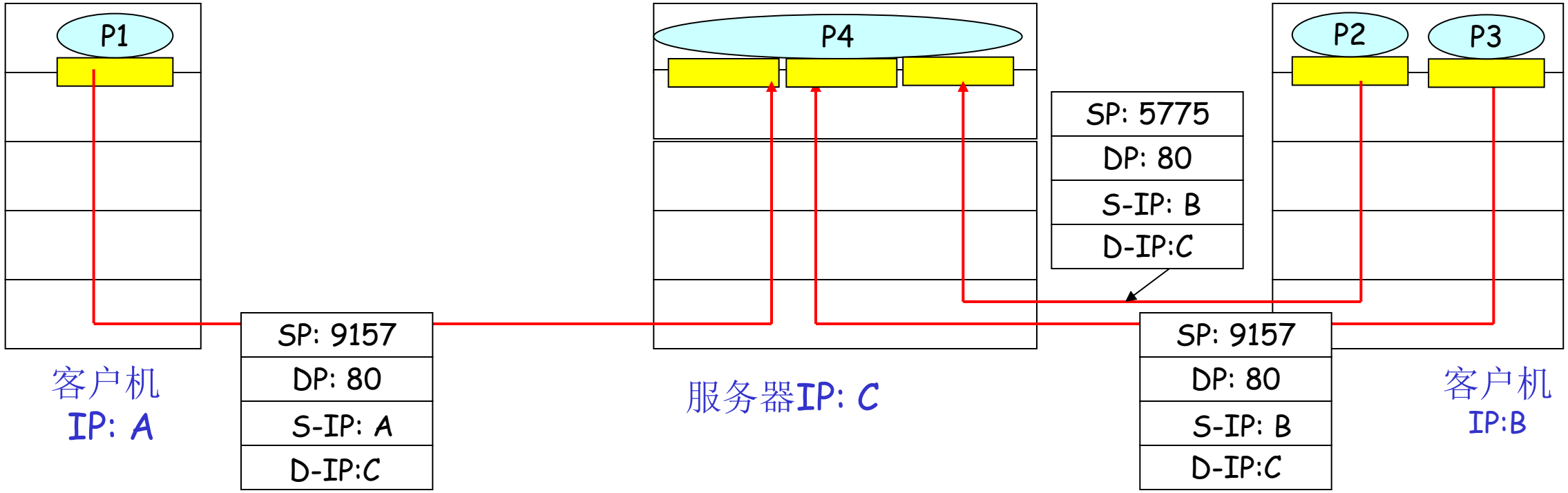
- TCP socket标识是由 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- 分解:接收方使用4个值来将收到的报文段提交到正确的socket
- 服务器主机同时运行了多个TCP套接字:
  - each socket identified by its own 4-tuple
- Web服务器为每个连接客户端进程创建不同的套接字
  - non-persistent HTTP will have different socket for each request

面向连接的多路分解示例



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

面向连接分解: 多线程Web服务器



第3章 主要内容

- 3.1 运输层服务
- 3.2 复用与分解
- 3.3 无连接传输: UDP
- 3.4 可靠数据传输的原则
  - rdt1
  - rdt2
  - rdt3
  - 流水线协议
- 3.5 面向连接的传输: TCP
  - 报文段结构
  - 可靠数据传输
  - 流量控制
  - 连接管理
- 3.6 拥塞控制的原则
- 3.7 TCP拥塞控制
  - 机制
  - TCP吞吐量
  - TCP公平性
  - 时延模型

3.3 UDP: 用户数据报协议 [RFC 768]

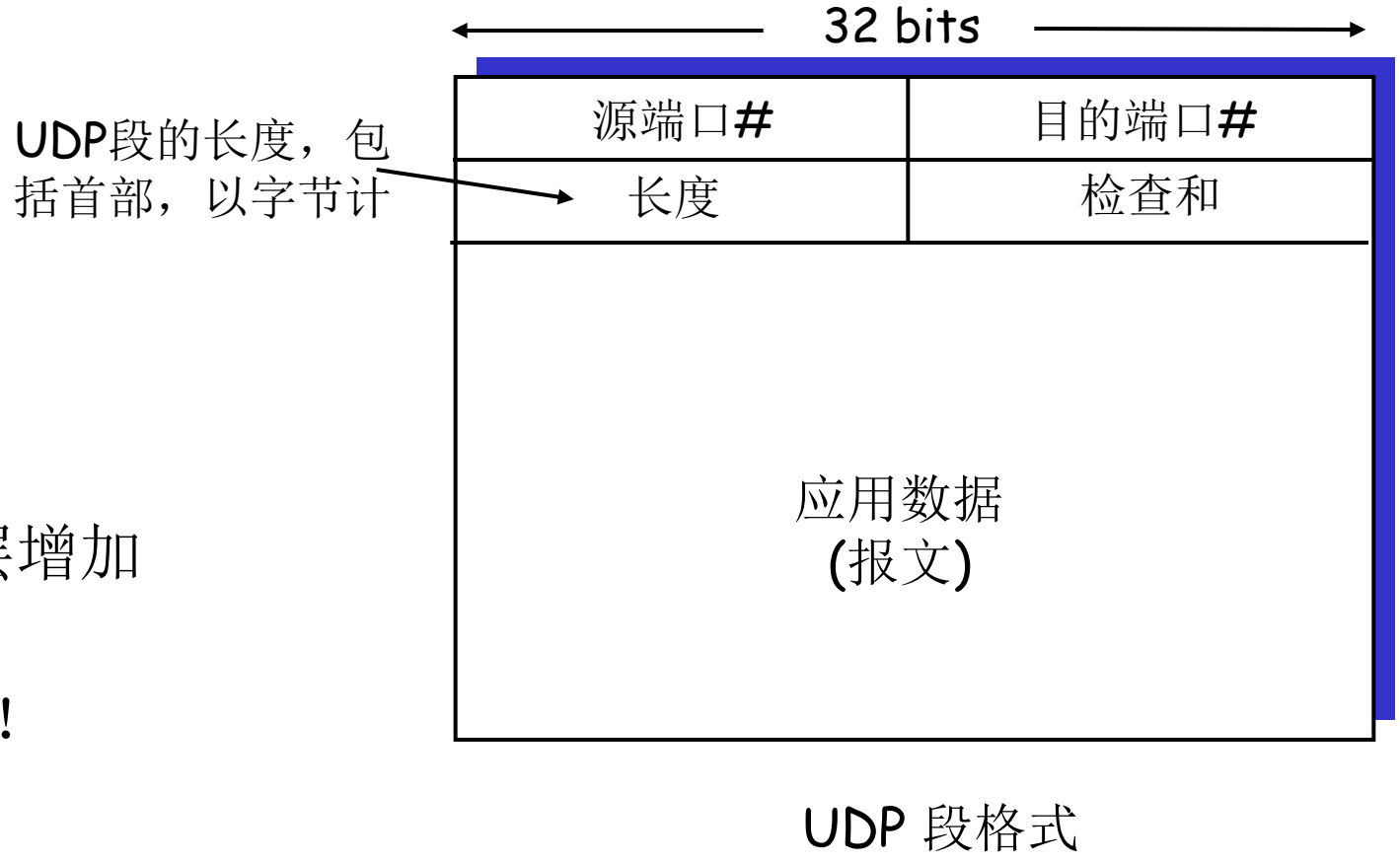
- “没有不必要的,“基本要素” 互联网传输协议
- “尽力而为”服务，UDP段可能：
  - 丢包
  - 对应用程序交付失序
- *无连接:*
  - 在UDP发送方和接收方之间无握手
  - 每个UDP段的处理独立于其他段

为何要有 UDP协议?

- 无连接创建(它将增加时延)
- 简单：在发送方、接收方无连接状态
- 段首部小
- 无拥塞控制: UDP能够尽可能快地传输

3.3.1 UDP报文段结构

- 常用于流式多媒体应用
  - 丢包容忍
  - 速率敏感
- 其他UDP应用
  - DNS
  - SNMP
- 经UDP的可靠传输：在应用层增加可靠性
  - 应用程序特定的差错恢复！



UDP 段格式

3.3.2 UDP校验和

目的: 在传输的段中检测“差错” (如比特翻转)

发送方:

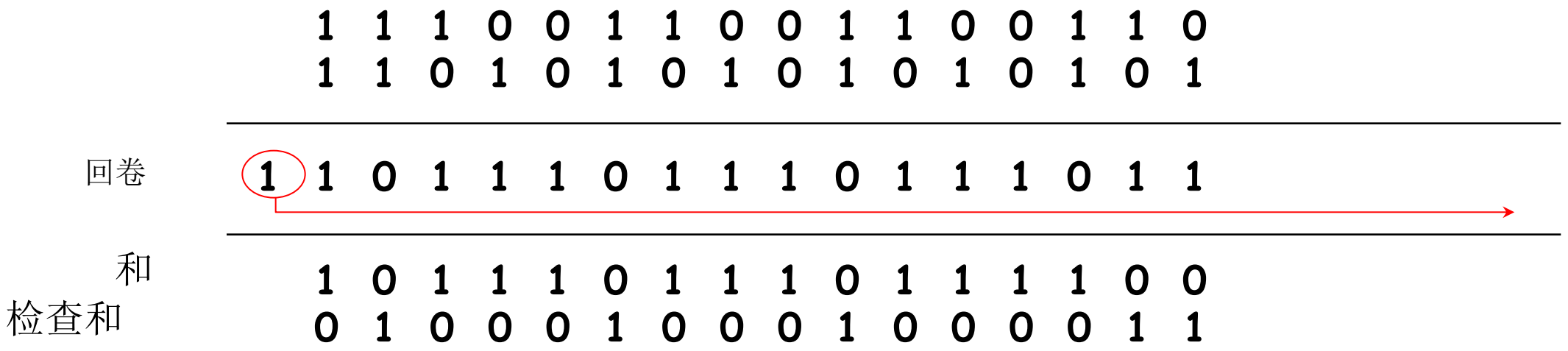
- 将段内容处理为16比特整数序列
- 检查和: 段内容的加法(反码和)
- 发送方将检查和放入UDP检查和字段

接收方:

- 计算接收的段的检查和
- 核对计算的检查和是否等于检查和字段的值:
  - NO - 检测到差错
  - YES - 无差错检测到。*虽然如此，还可能有差错吗？* 详情见后.....

互联网检查和例子

- 注意
  - 当数字作加法时，最高位进比特位的进位需要加到结果中
- 例子: 两个16-bit整数相加



课堂练习

消息的目的IP和目的端口号
源IP和源端口号

P192-R3，R7，R8
P194-P3

R3. Consider a TCP connection between Host A and Host B. Suppose that the TCP segments traveling from Host A to Host B have source port number x and destination port number y. What are the source and destination port numbers for the segments traveling from Host B to Host A?

R7. Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each send a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?

R8. Suppose that a Web server runs in Host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different Hosts, A and B. Are all of the requests being sent through the same socket at Host C? If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

课堂练习

P192-R3，R7，R8
P194-P3

P3. UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

第3章 主要内容

- 3.1 运输层服务
- 3.2 复用与分解
- 3.3 无连接传输: UDP
- 3.4 可靠数据传输的原则
  - rdt1
  - rdt2
  - rdt3
  - 流水线协议
- 3.5 面向连接的传输: TCP
  - 报文段结构
  - 可靠数据传输
  - 流量控制
  - 连接管理
- 3.6 拥塞控制的原则
- 3.7 TCP拥塞控制
  - 机制
  - TCP吞吐量
  - TCP公平性
  - 时延模型

研讨课-1：数据可靠传输机制

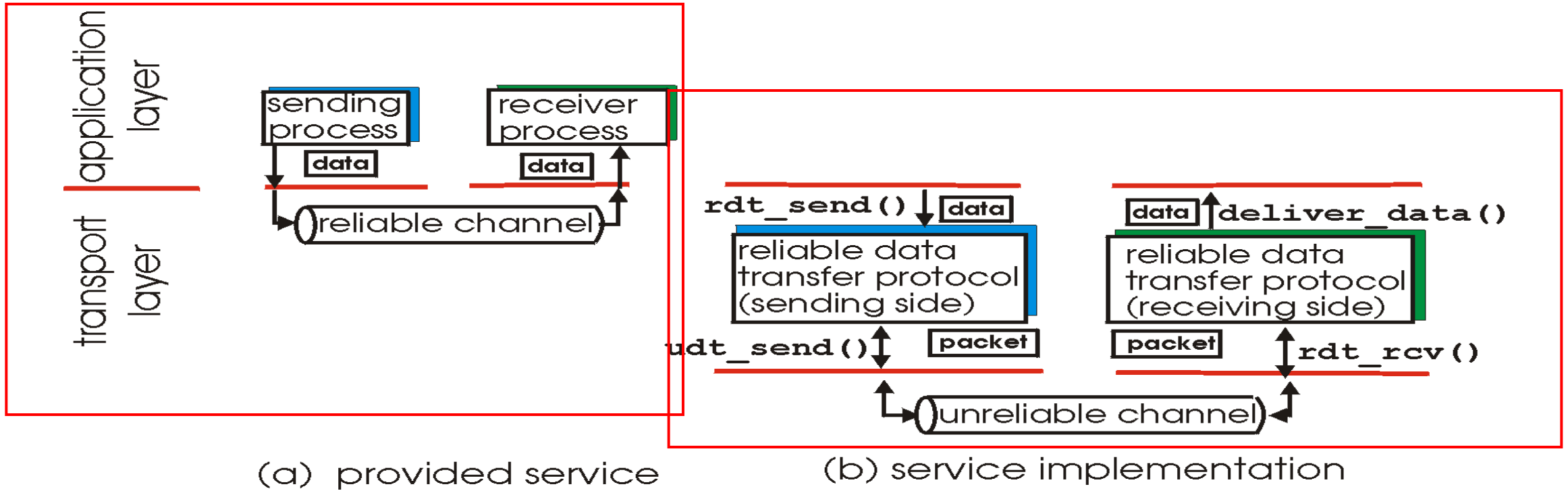
3.4 可靠数据传输原理

3.5 TCP：可靠输出传输

- 目的
  - 通过研讨让学生理解在有比特差错和丢包的网络场景下，如何设计数据的可靠传输机制
  - 理解TCP协议中的可靠数据传输机制
- 采用的形式
  - 抛出问题+教师引导+学生讨论的形式

准备知识：可靠数据传输的原则

- 在应用层、运输层、数据链路层的重要性
  - **重要的网络主题中的最重要的10个之一!**

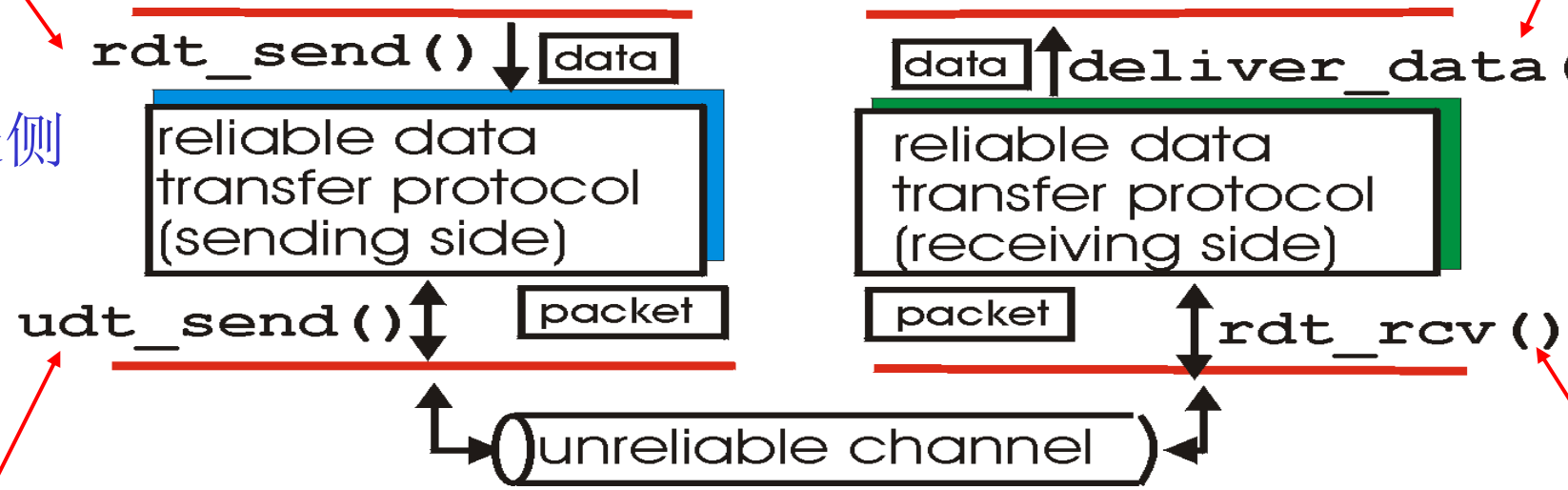


- 不可靠信道的特点决定了可靠数据传输 协议 (rdt) 的复杂性

准备知识：可靠数据传输基本概念

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data(): called by rdt to deliver data to upper



udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

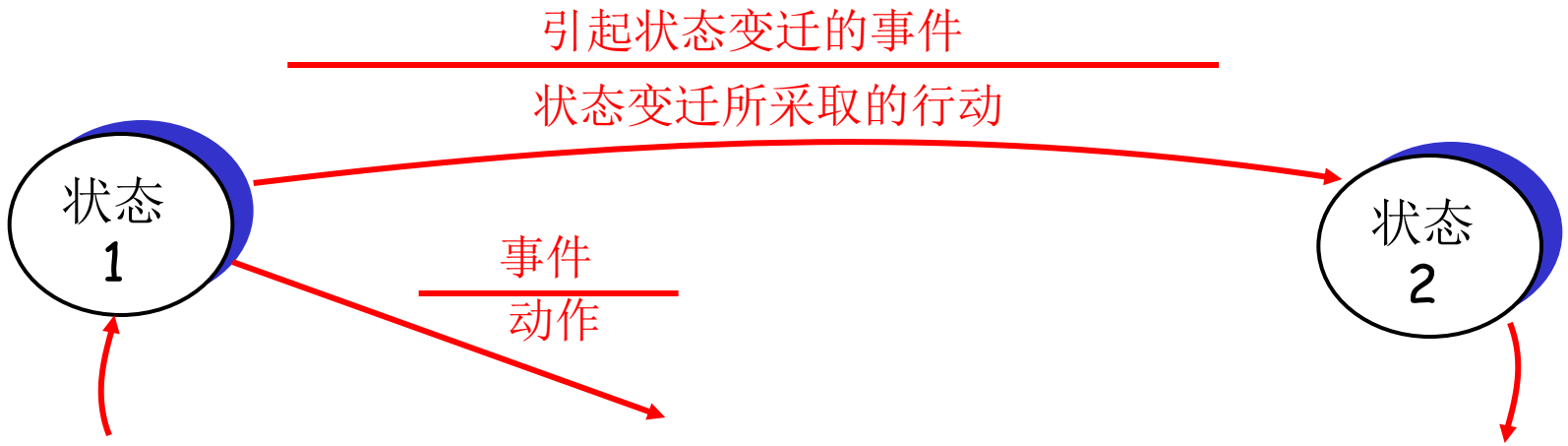
rdt_rcv(): called when packet arrives on rcv-side of channel

准备知识：可靠数据传输基本概念

我们将:

- 增强研发发送方，可靠数据传输协议 (rdt) 的接收方侧
  - 仅考虑单向数据传输
  - 但控制信息将在两个方向流动！
- 使用有限状态机 (FSM)来定义发送方和接收方

状态: 当位于这个“状态时"，下个状态惟一地由下个事件决定

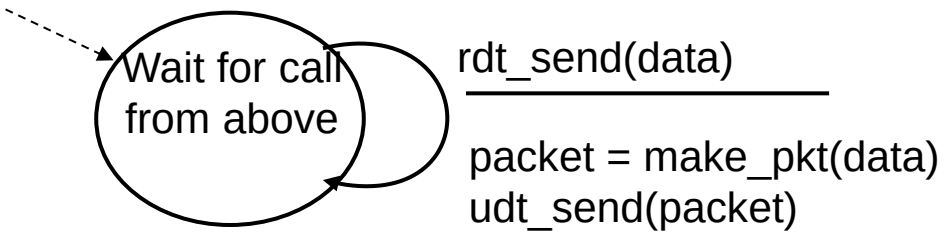


抛出问题

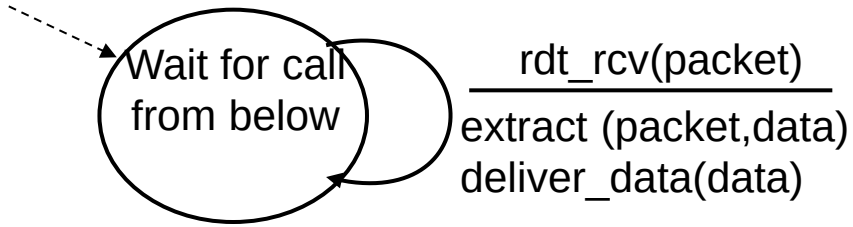
- 简单场景：收发双方**经可靠信道**传输数据分组
  - 无需提供额外的保证机制
- 较复杂场景：收发双方**经有比特差错的信道**传输数据分组
  - 需要提供的机制：
- 真实场景：收发双方**经有比特差错和丢包的信道**传输数据分组
  - 需要提供的机制：
- 因特网的TCP协议中采用的机制有哪些？
  - 分析采用的原因

简单场景：经可靠信道传输

- 场景解读：底层信道非常可靠
  - 无比特差错
  - 无分组丢失
- 机制设计：装发送方、接收方的单独FSM:
  - 发送方将数据发向底层信道
  - 接收方从底层信道读取数据



发送方



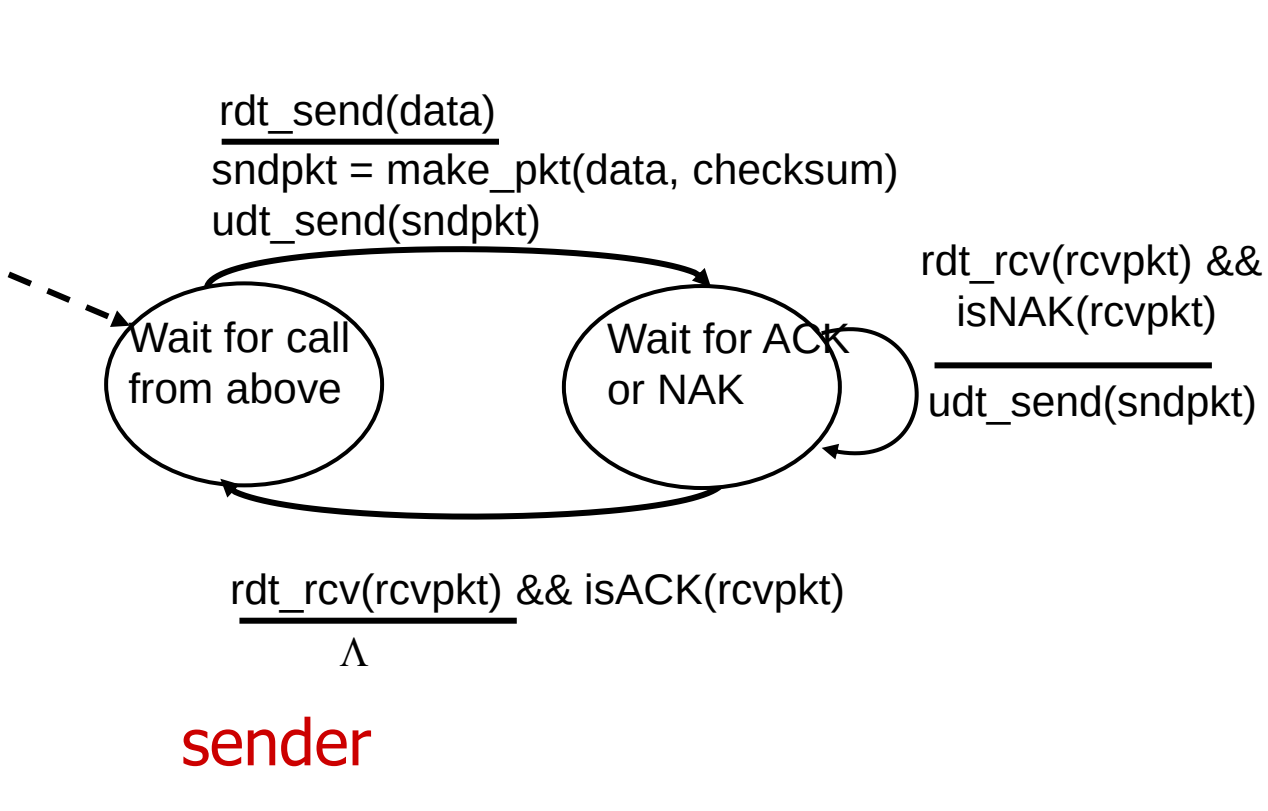
接收方

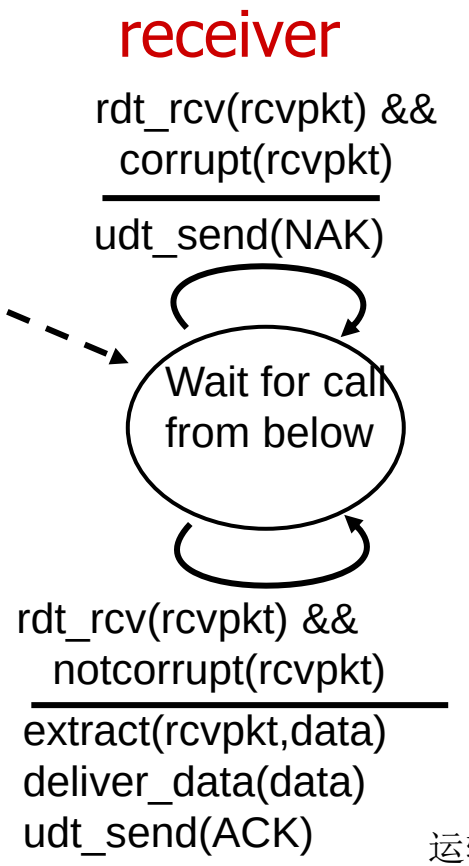
较复杂场景：经有比特差错的信道传输

- 场景解读：
  - **底层信道可能会将分组中的某些比特反转**
  - **无分组丢失**
- 机制设计：
  - 怎么检测到比特错误：检验和机制(checksum)
  - 检测到错误后的处理方法：**由接收方通知发送方重传**
    - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK（显式的告诉发送人成功收到了分组）
    - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors（接收人显式告诉发送人分组出错）
    - sender retransmits pkt on receipt of NAK（当收到NAK报文时，发送人重传出错的分组）

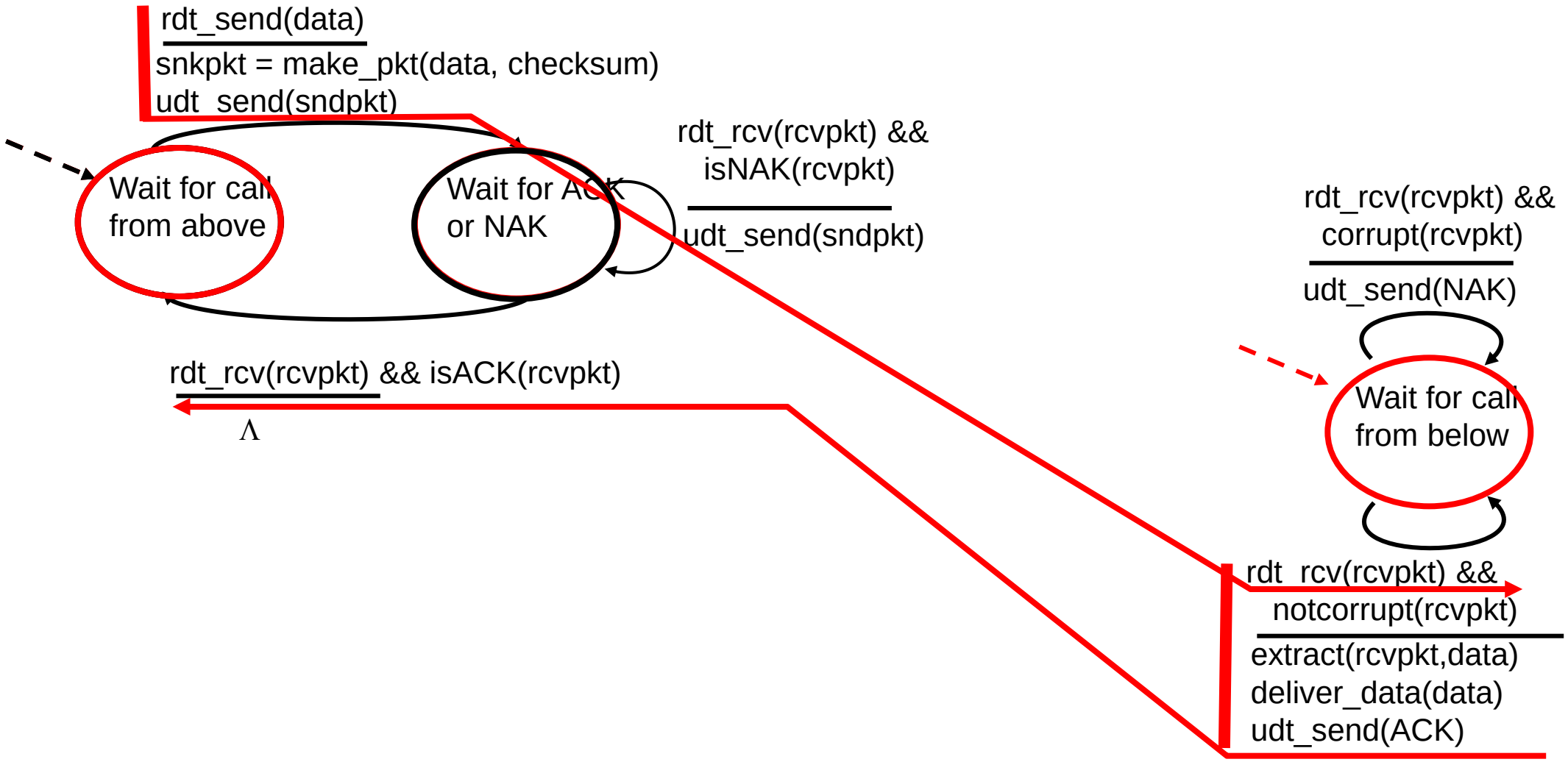
较复杂场景：经有比特差错的信道传输

- rdt2.0机制
  - error detection（出错检测）
  - receiver feedback: control msgs (ACK, NAK) rcvr->sender（发送控制消息报文）

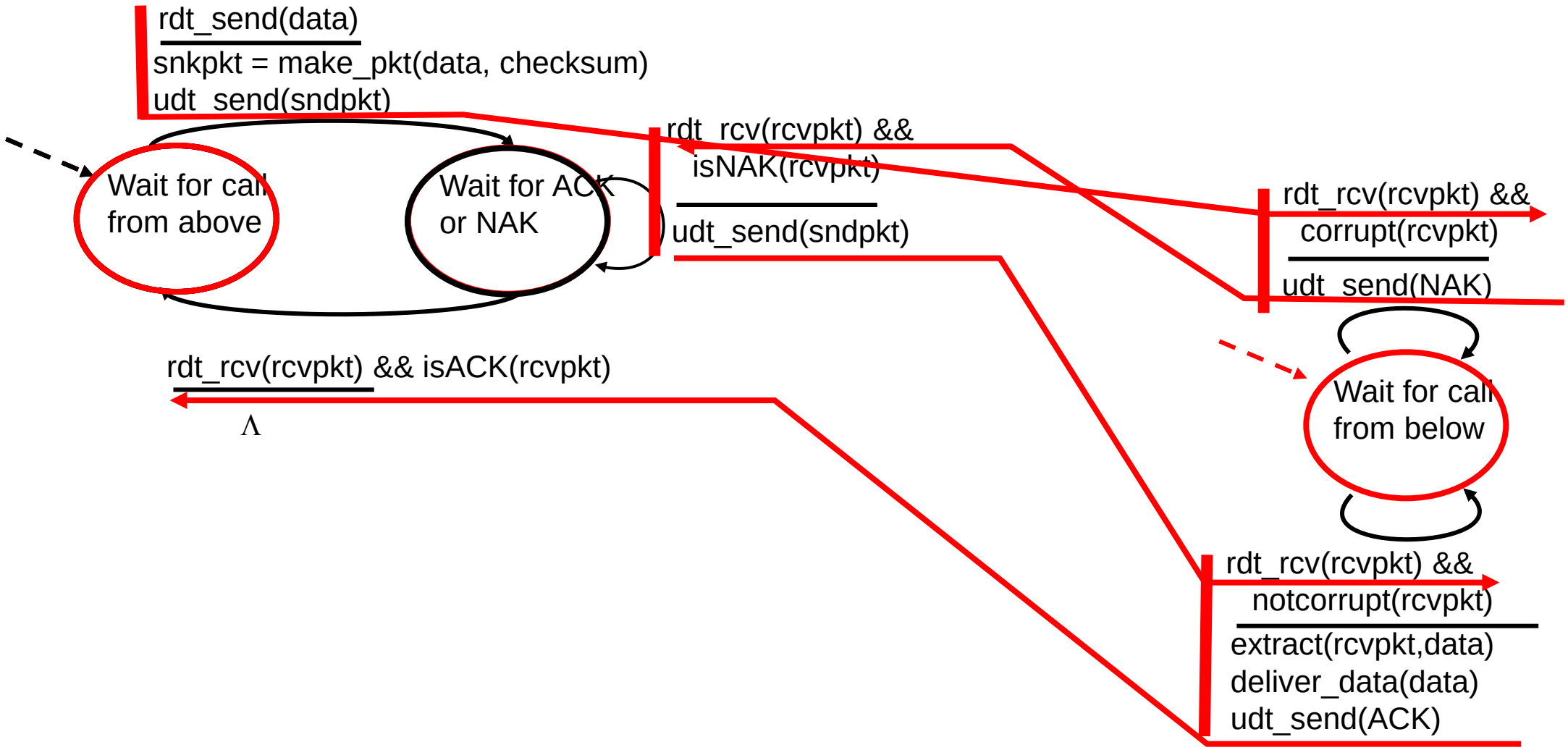




rdt2.0: operation with no errors



rdt2.0: error scenario



rdt2.0有重大的缺陷!

如果ACK/NAK受损，将会出现何种情况？

- 发送方不知道在接收方会发生什么情况！
- 不能只是重传：可能导致冗余

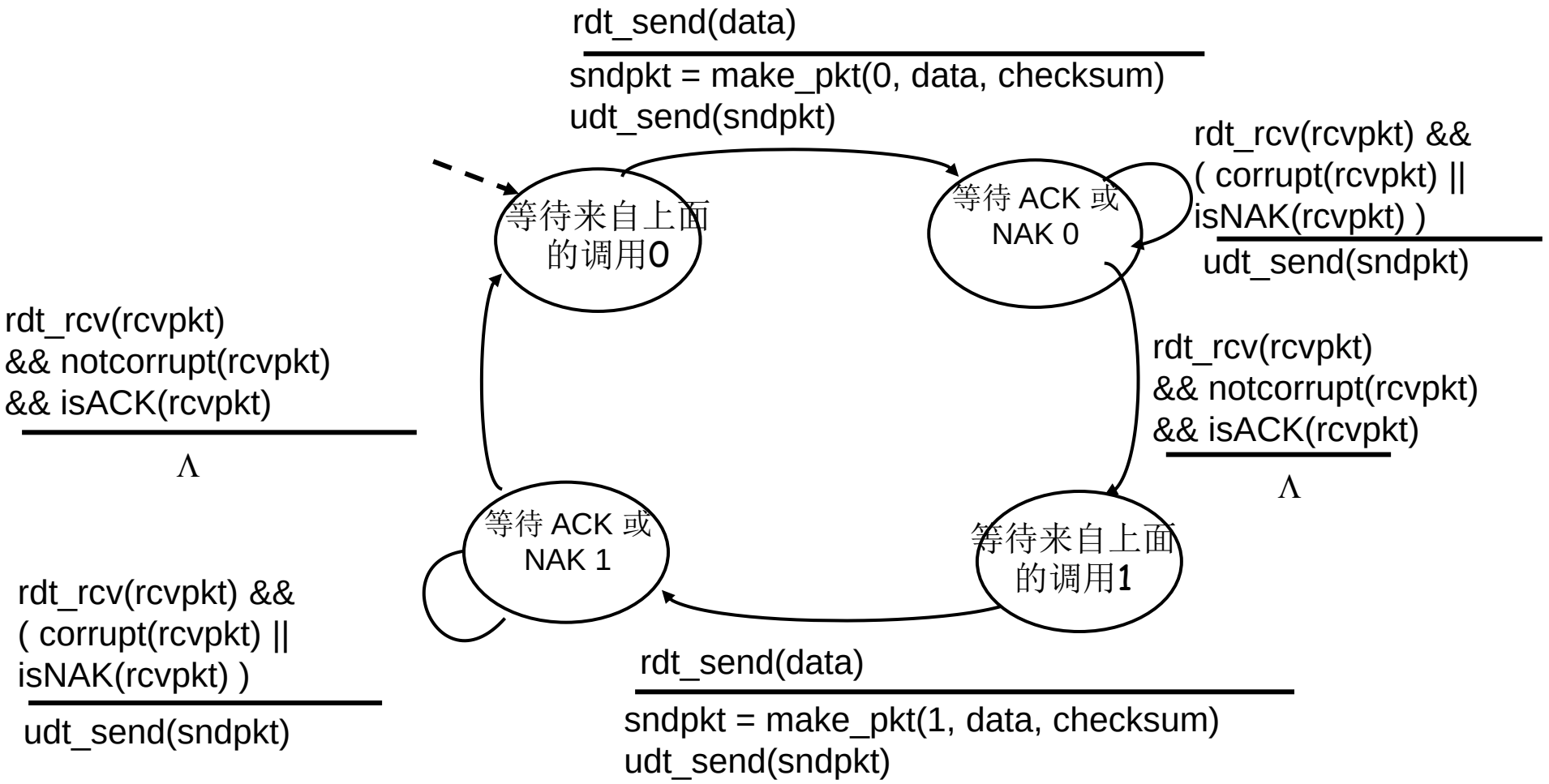
处理冗余:

- 发送方对每个分组增加*序列号*
- 如果ACK/NAK受损，发送方重传当前的分组
- 接收方丢弃(不再向上交付)冗余分组

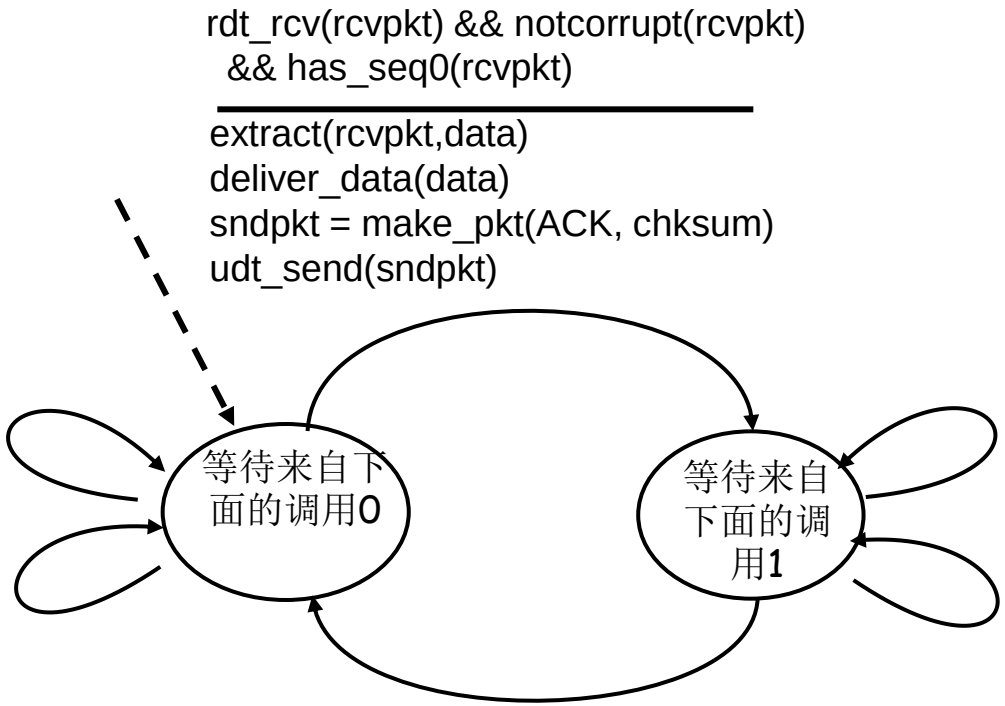
停止等待

发送方发送一个分组，然后等待接收方响应

rdt2.1: 发送方, 处理受损的ACK/NAK



rdt2.1: 接收方,处理受损的ACK/NAK



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq1(rcvpkt)

sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq0(rcvpkt)

sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

rdt2.1: 讨论

发送方:

- 序号seq # 加入分组中
- 两个序号seq. #'s (0,1) 将够用. (为什么?)
- 必须检查是否收到的ACK/NAK受损
- 状态增加一倍
  - 状态必须“记住”“当前的”分组具有0或1序号

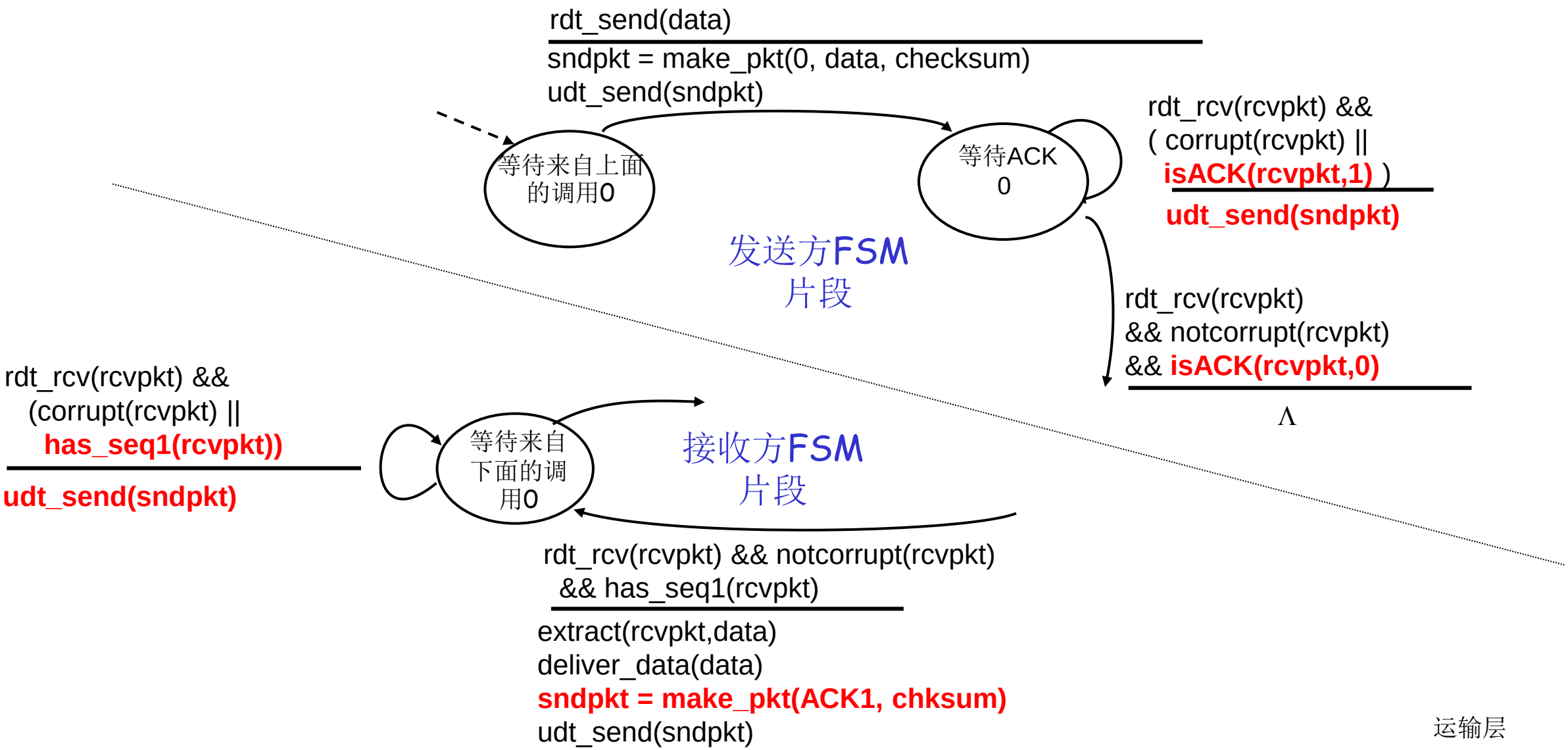
接收方:

- 必须检查是否接收到的分组是冗余的
  - 状态指示（0或1）表明所期待的分组序号seq #
- 注意: 接收方不能知道是否它最后的ACK/NAK在发送方已经成功接收

rdt2.2: 一种无NAK的协议

- 与rdt2.1一样的功能，仅使用ACK
- 代替NAK，接收方对最后正确接收的分组发送ACK
  - 接收方必须明确地包括被确认分组的序号
- 在发送方冗余的ACK导致如同NAK相同的动作：重传当前分组

rdt2.2: 发送方, 接收方片段



rdt3.0: 具有比特差错和丢包的信道

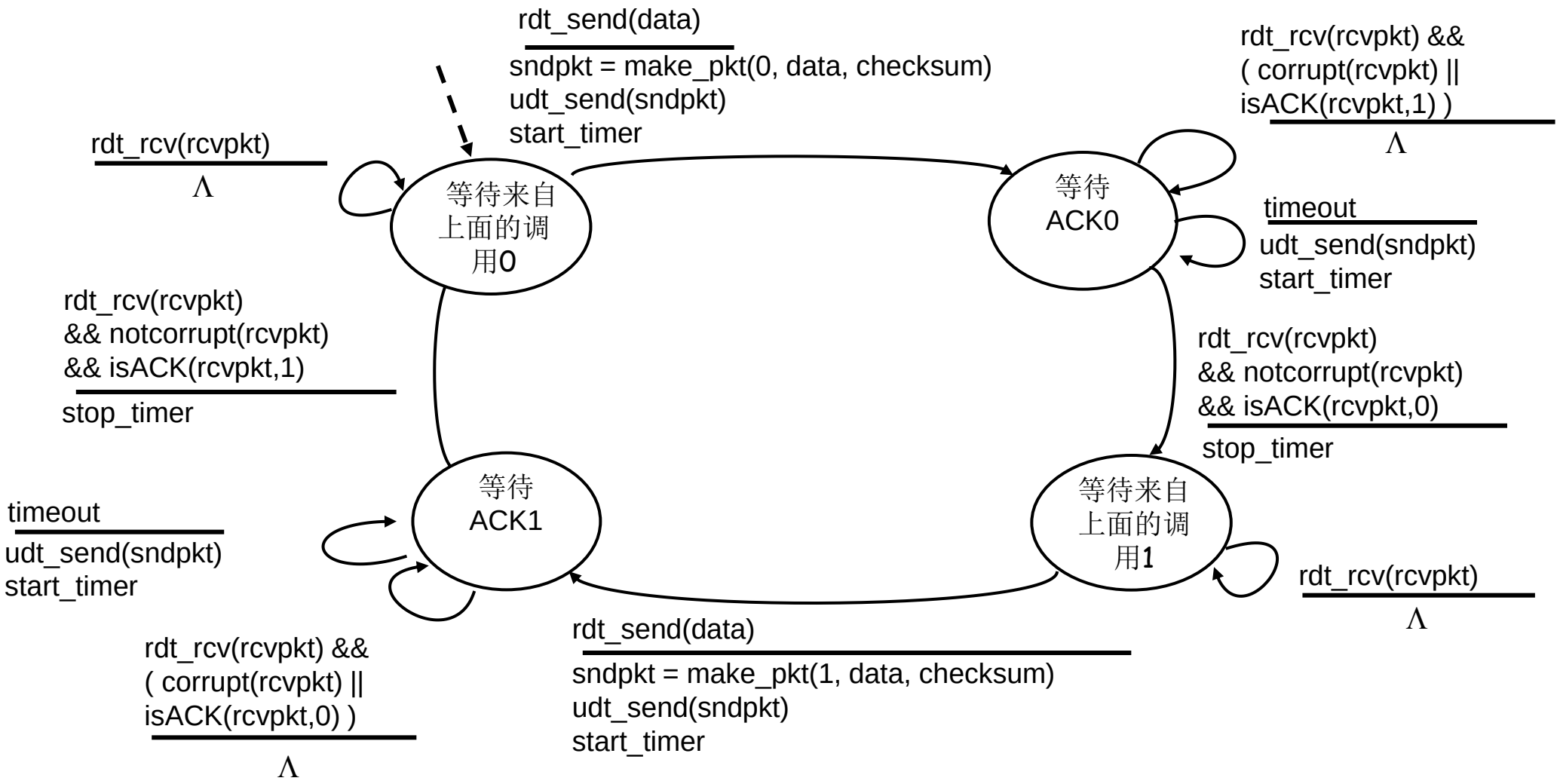
新假设: 下面的信道也能丢失分组(数据或ACK)

- 检查和、序号、重传将是有帮助的，但不充分

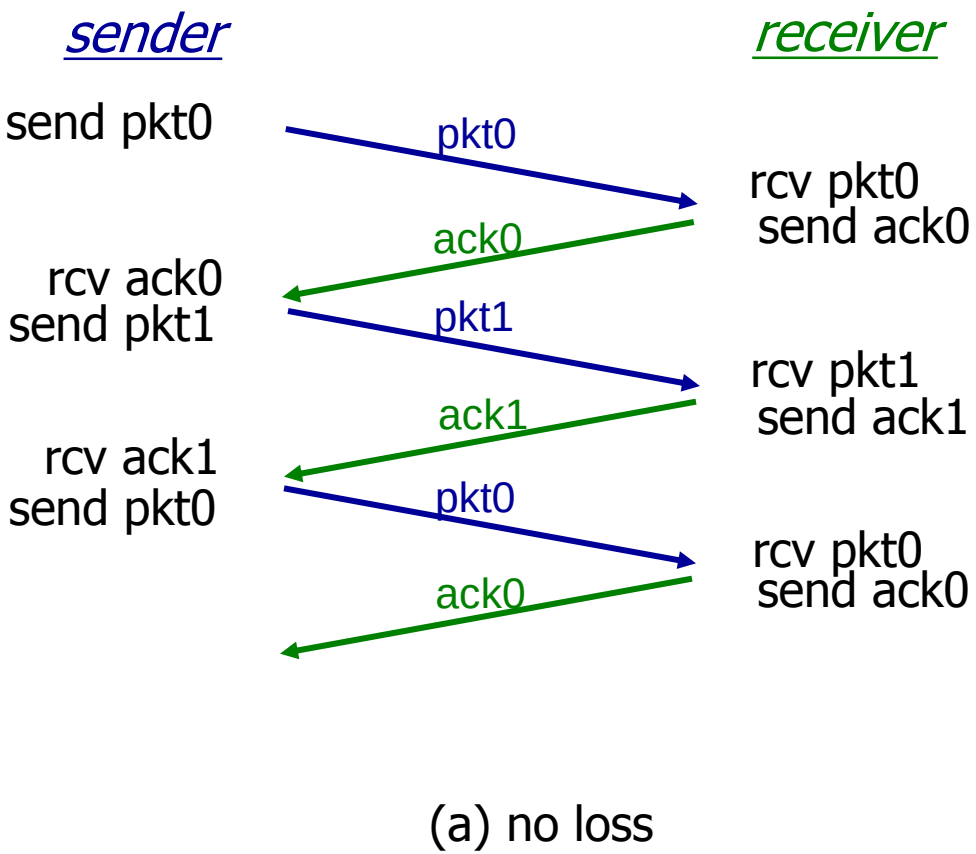
方法: 发送方等待ACK一段“合理的”时间间

- 如在这段时间没有收到ACK则重传
- 如果分组(或ACK)只是延迟(没有丢失):
  - 重传将是冗余的，但序号的使用已经处理了该情况
  - 接收方必须定义被确认的分组序号
- 需要倒计时定时器

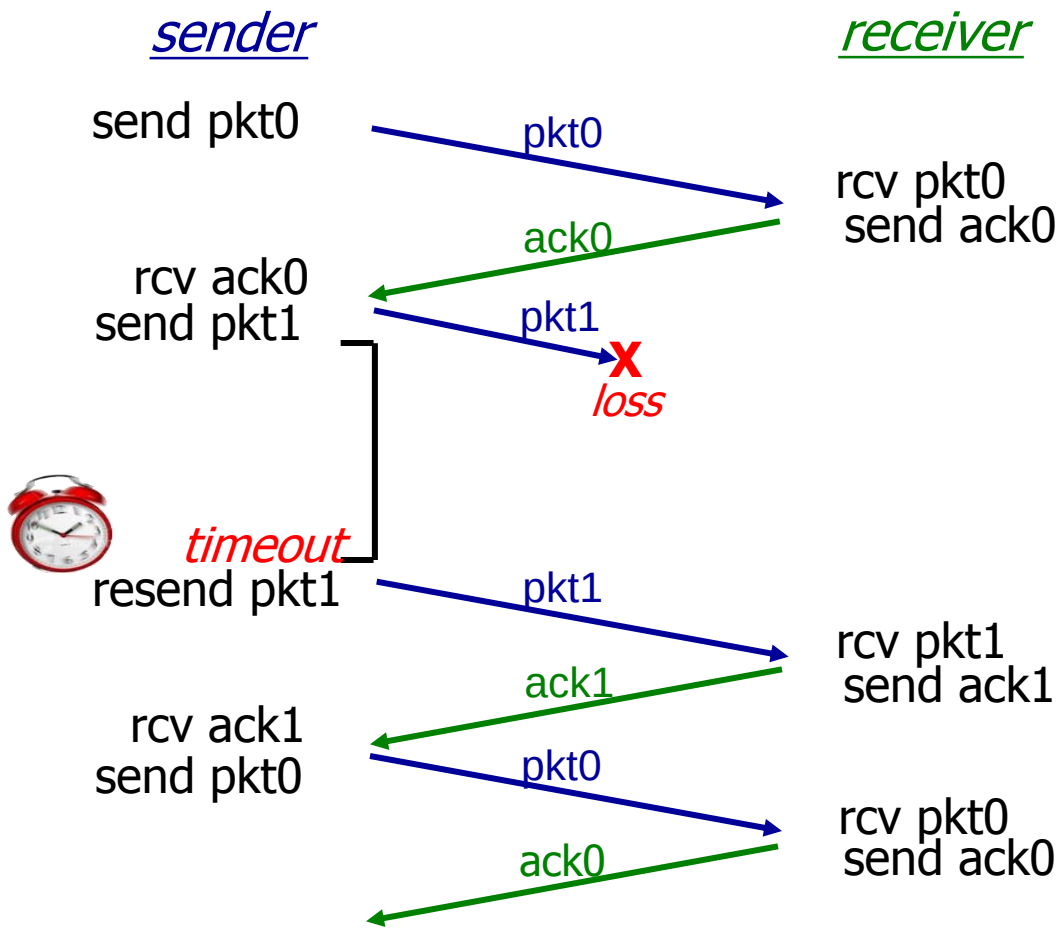
rdt3.0发送方


rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
等待来自
上面的调
用0
等待
ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,0)
stop_timer
等待
ACK1
等待来自
上面的调
用1
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) ||
isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

rdt3.0 in action

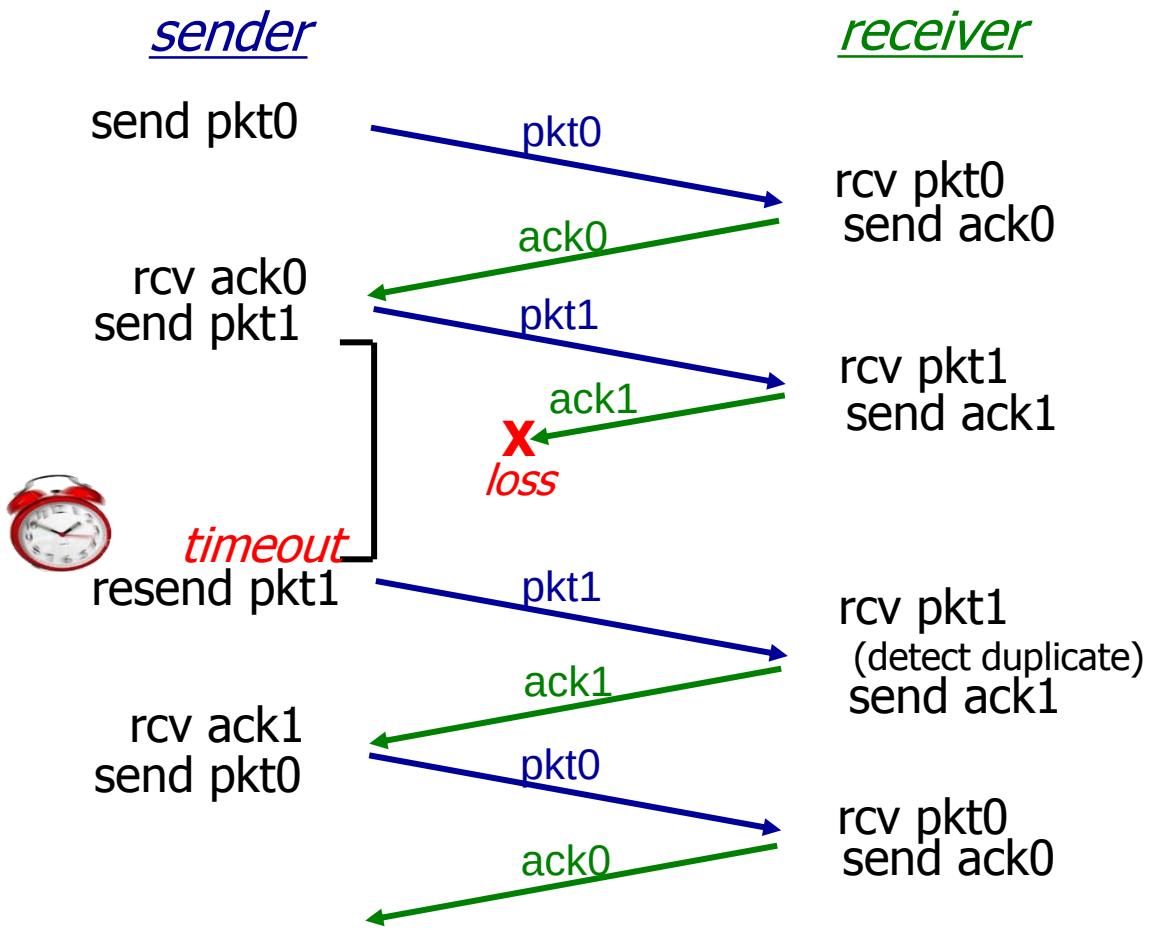


(a) no loss

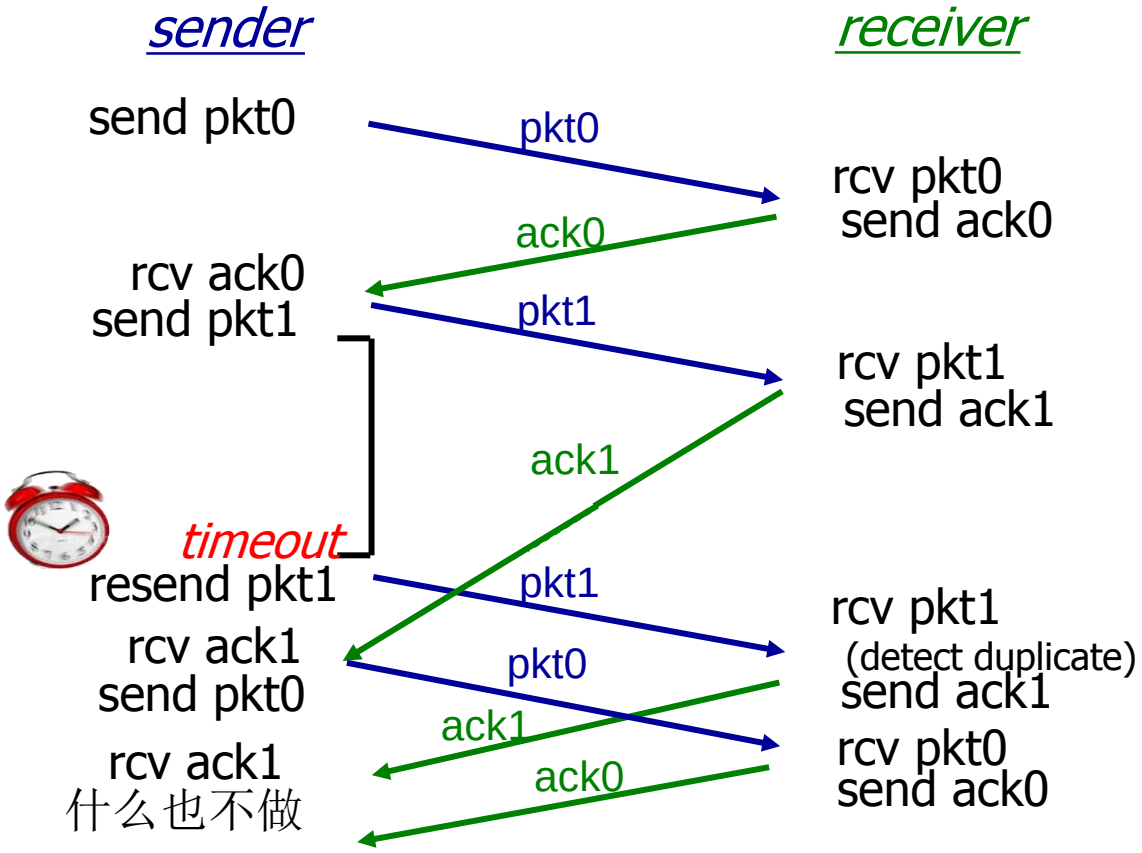


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

课堂练习

- 请同学们自己画出rdt3.0接收方的FSM

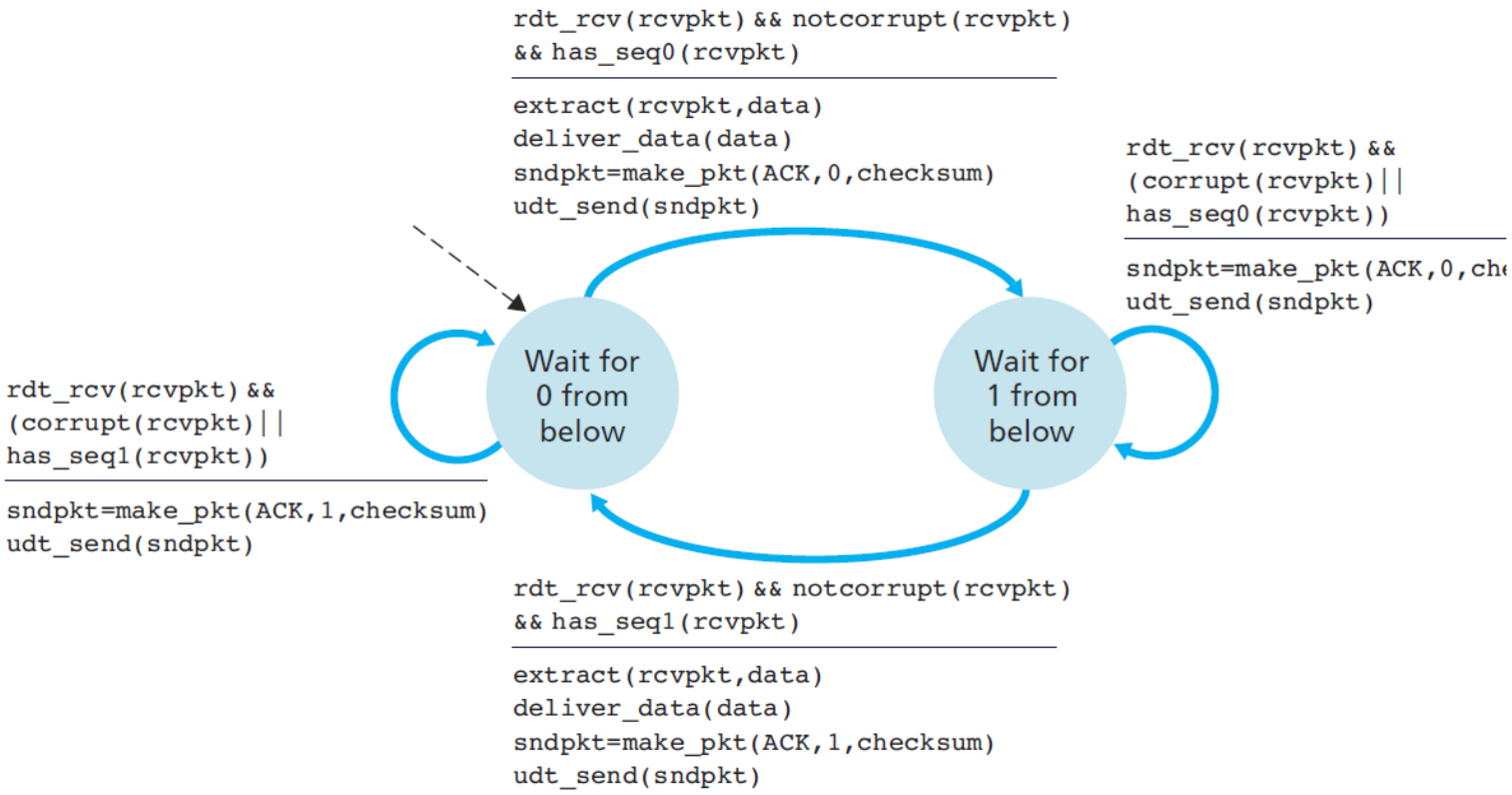


Figure 3.14 ♦ rdt2.2 receiver

上次研讨课内容回顾

- 通过探索的方式，逐层深入的讨论了真实场景下的可靠传输机制，最终得到了rdt3.0(停-等方式)协议；
- 探索了以下可靠传输机制：
  - 差错检查
  - 正确确认和否定确认
  - 重传机制
  - 定时器机制
- 要求大家能够掌握怎么通过有限状态机来描述协议

（简单协议描述，看得懂）

研讨题目1:

- 考虑一种仅使用否定确认的可靠数据传输协议。假定发送方只是偶尔发送数据。只用NAK的协议是否会比使用ACK的协议更好？为什么？现在我们假设发送方要发送大量的数据，并且该端到端连接很少丢包，在第二种情况下，只用NAK的协议是否会比使用ACK的协议更好，为什么？（）

对题目的理解:

- 在有丢包的情况下，只使用NAK，发送方和接收方如何获知丢包?
- 偶尔发送数据理解为数据分组发送间隔时间较长，在检测丢包时会产生什么情况?

分析rdt3.0协议的性能

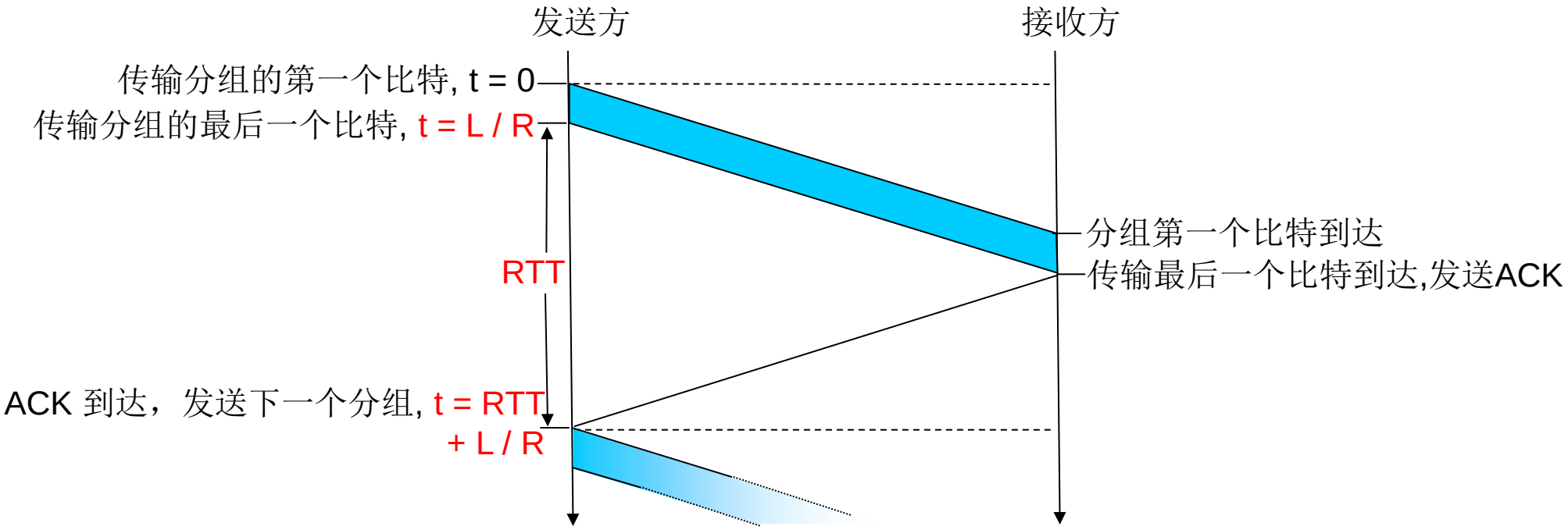
- rdt3.0能够工作，但性能不太好
- 例子: 1 Gbps链路, 15 ms端到端传播时延, 1KB分组:

$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ us}$$

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

  - U_{sender}: 利用率 – 发送方用于发送时间的比率
  - 每30 msec 1KB 分组 -> 经1 Gbps 链路有33kB/sec 吞吐量（1kBbit/0.03008sec）
  - 网络协议限制了物理资源的使用!

rdt3.0: 停等协议的运行

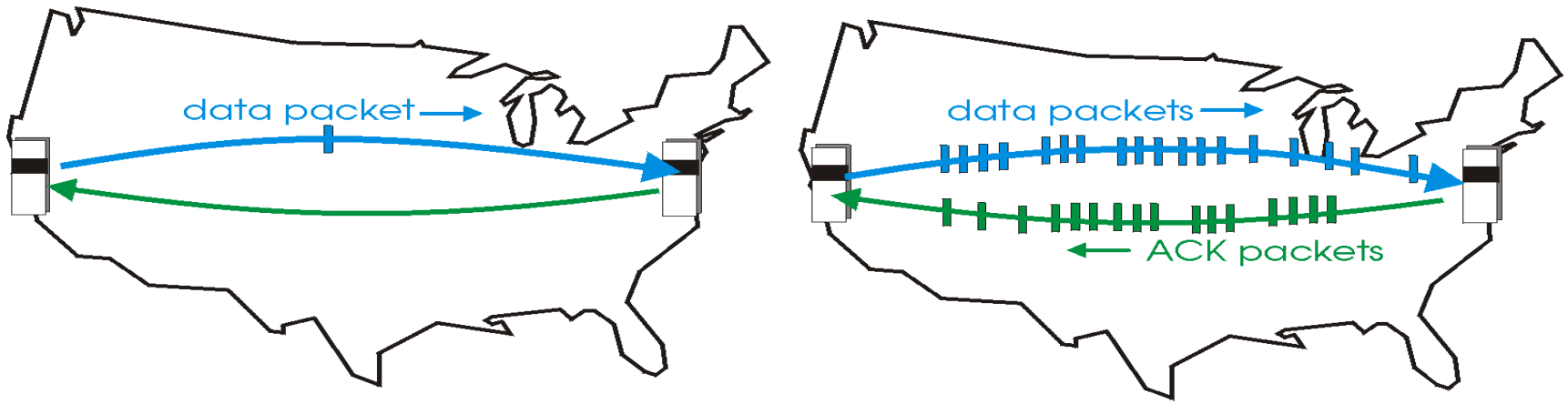


$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

3.4.2 流水线协议

流水线: 发送方允许发送多个、“传输中的”，还没有应答的报文段

- 序号的范围必须增加
- 发送方和/或接收方设有缓冲

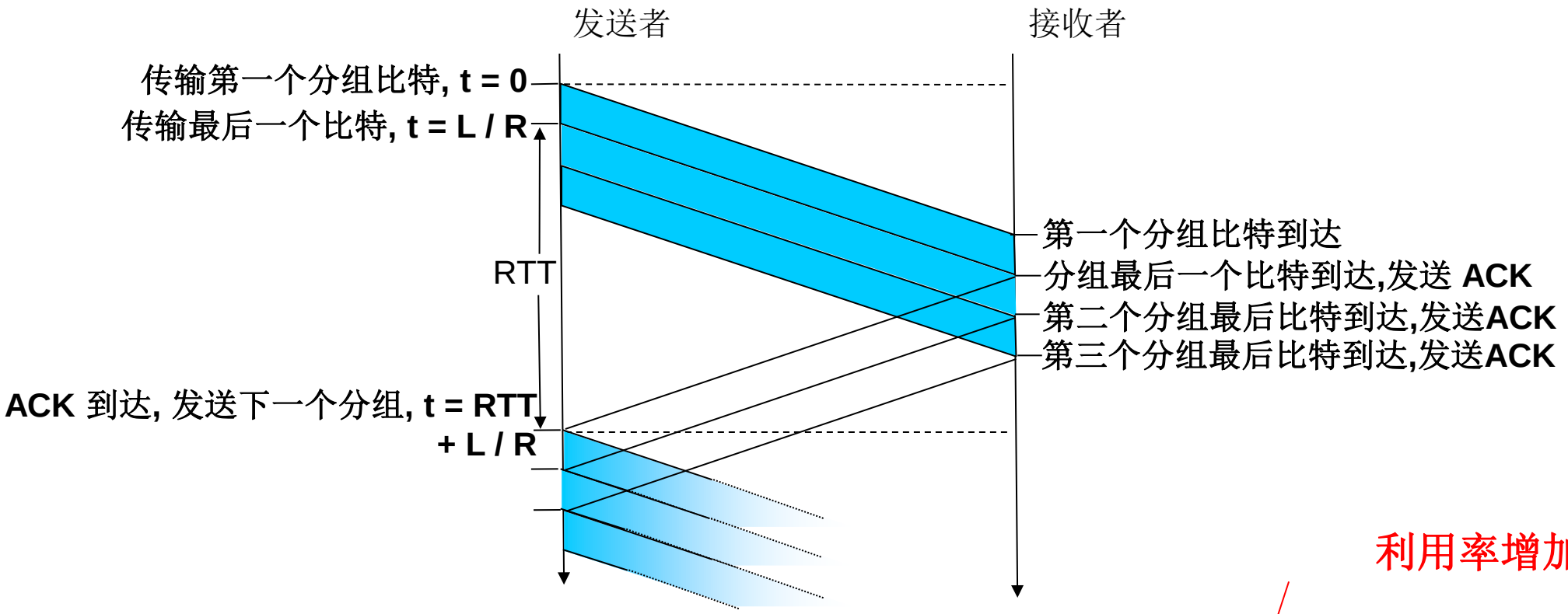


(a) a stop-and-wait protocol in operation　　(b) a pipelined protocol in operation

- 流水线协议的两种形式:

  回退N帧法（go-Back-N），选择性重传（S-R），

流水线协议: 增加利用率



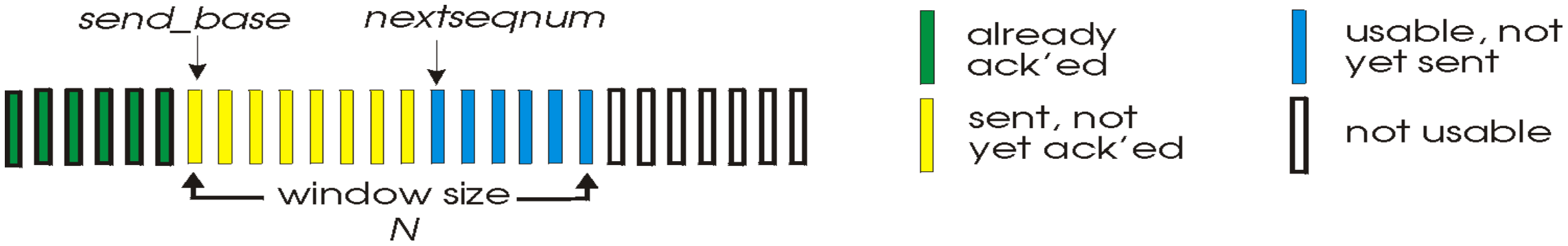
利用率增加3倍!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

3.4.3 Go-Back-N

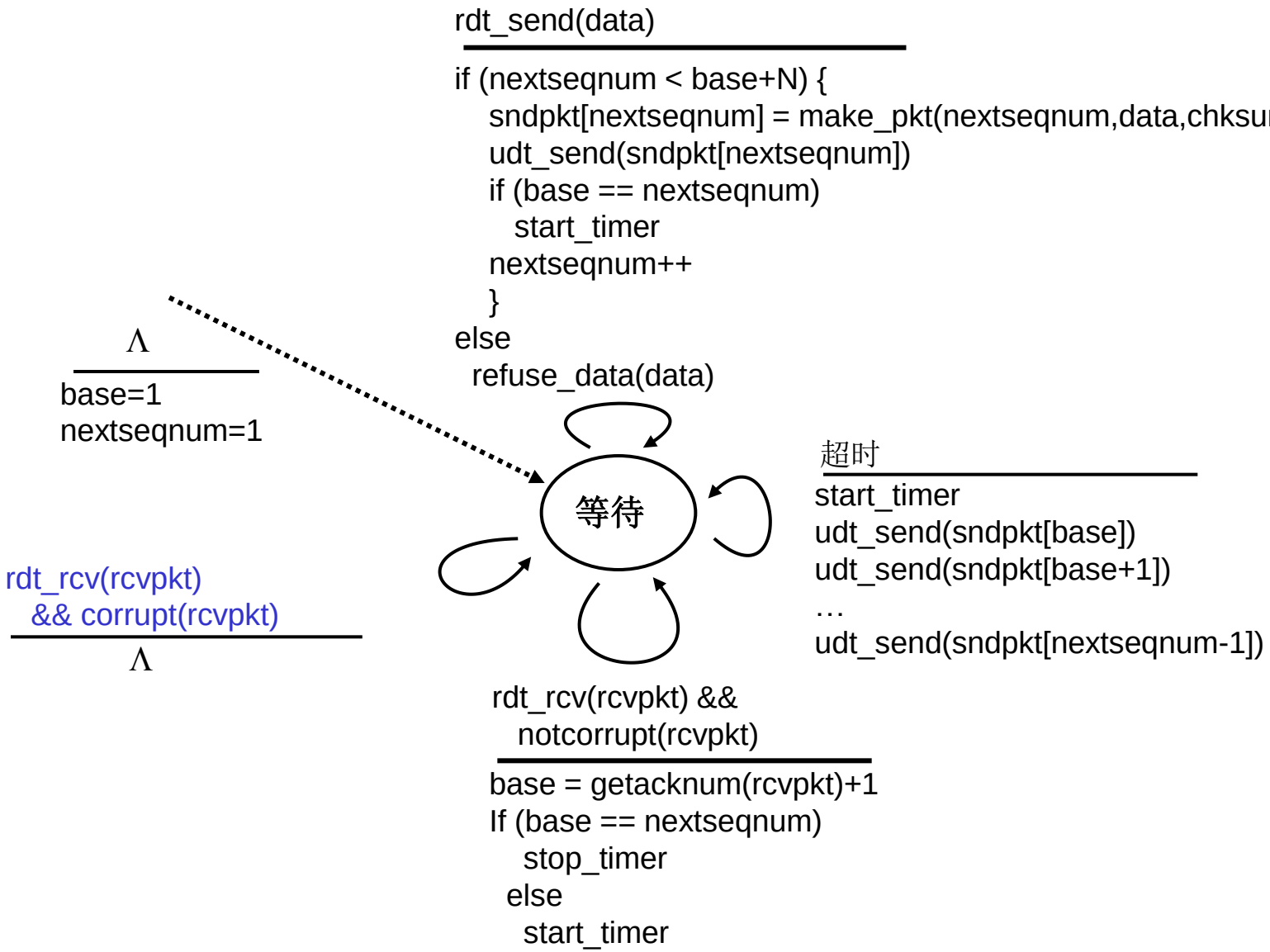
发送方:

- 在分组首部需要K比特序号，2^k=N
- “窗口”最大为N, 允许N个连续的没有应答分组

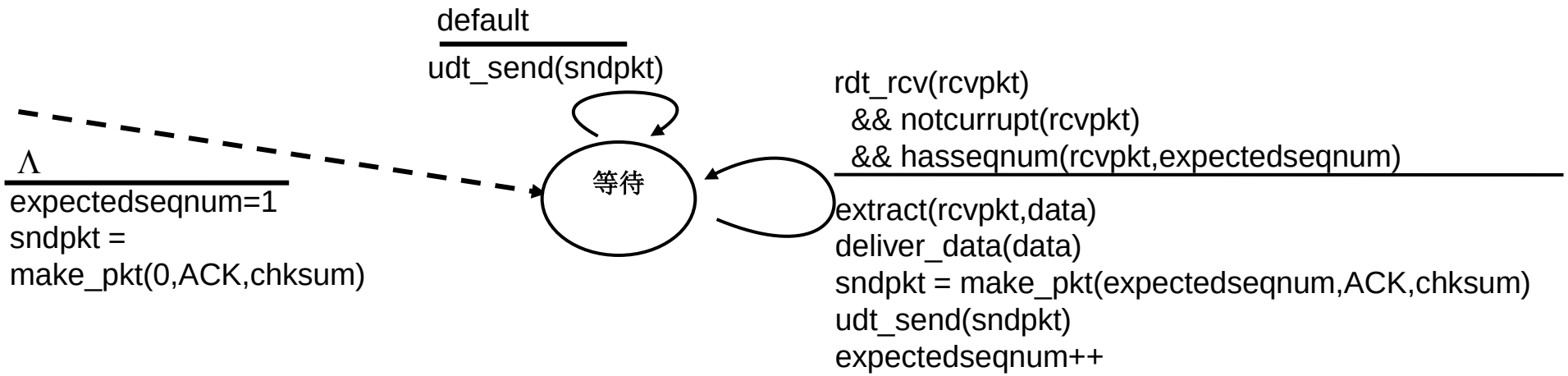


- ACK(n): ACKs all pkts up to, including seq # n - *“cumulative ACK ”*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

GBN: 发送方扩展的 FSM

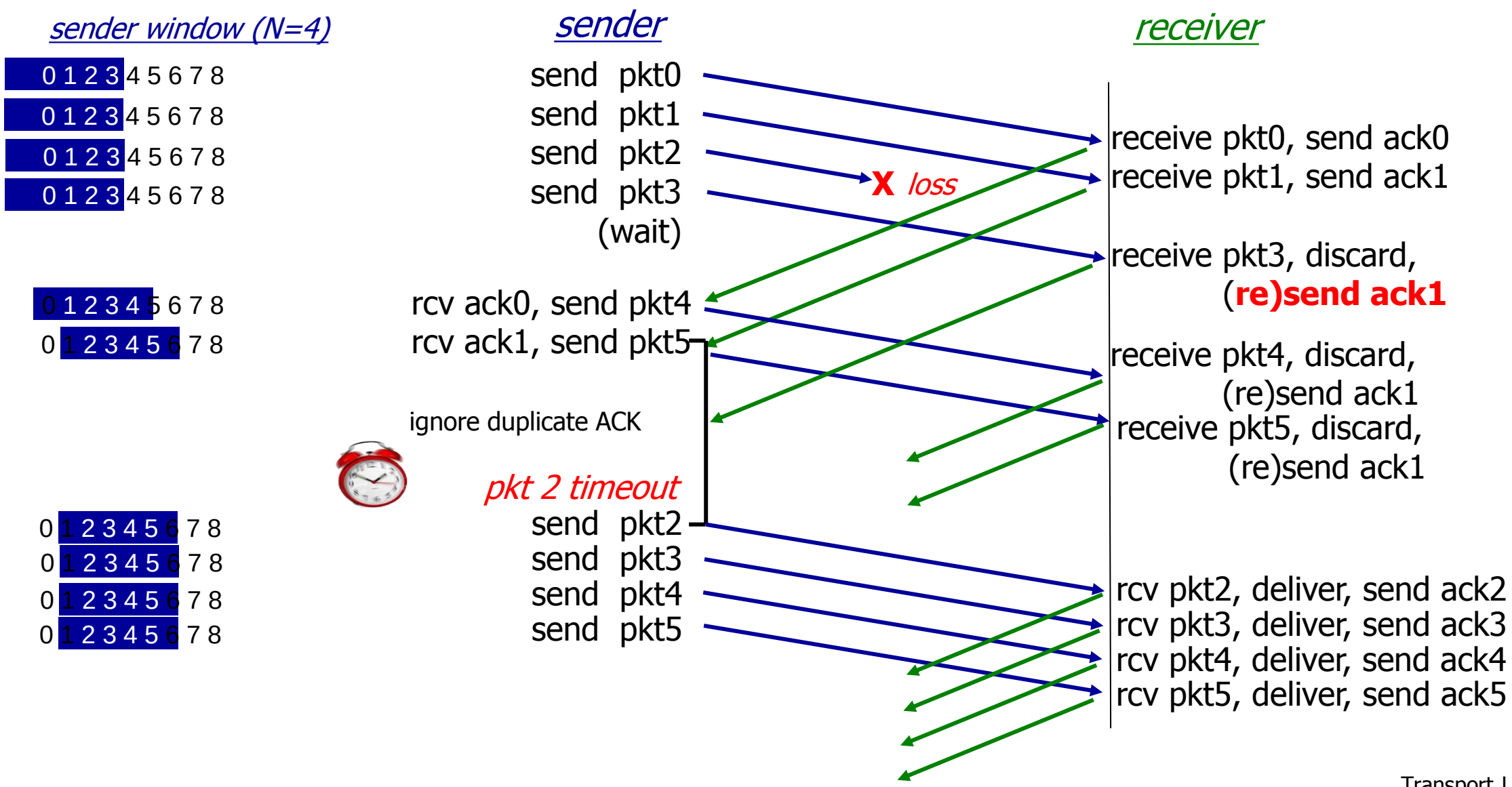


GBN: 接收方扩展 FSM



- 只有ACK: 对发送正确接收的分组总是发送具有最高按序序号的ACK
  - 可能产生冗余的ACKs
  - 仅仅需要记住期望的序号值（**expectedseqnum**）
- 对失序的分组:
  - 丢弃 (不缓存) -> 没有接收缓冲区!
  - 重新确认具有按序的分组

GBN in action



课堂练习

- R12

R12. Visit the Go-Back-N Java applet at the companion Web site.

a. Have the source send five packets, and then pause the animation before any of the five packets reach the destination. Then kill the first packet and resume the animation. Describe what happens.

b. Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.

c. Finally, try sending six packets. What happens?

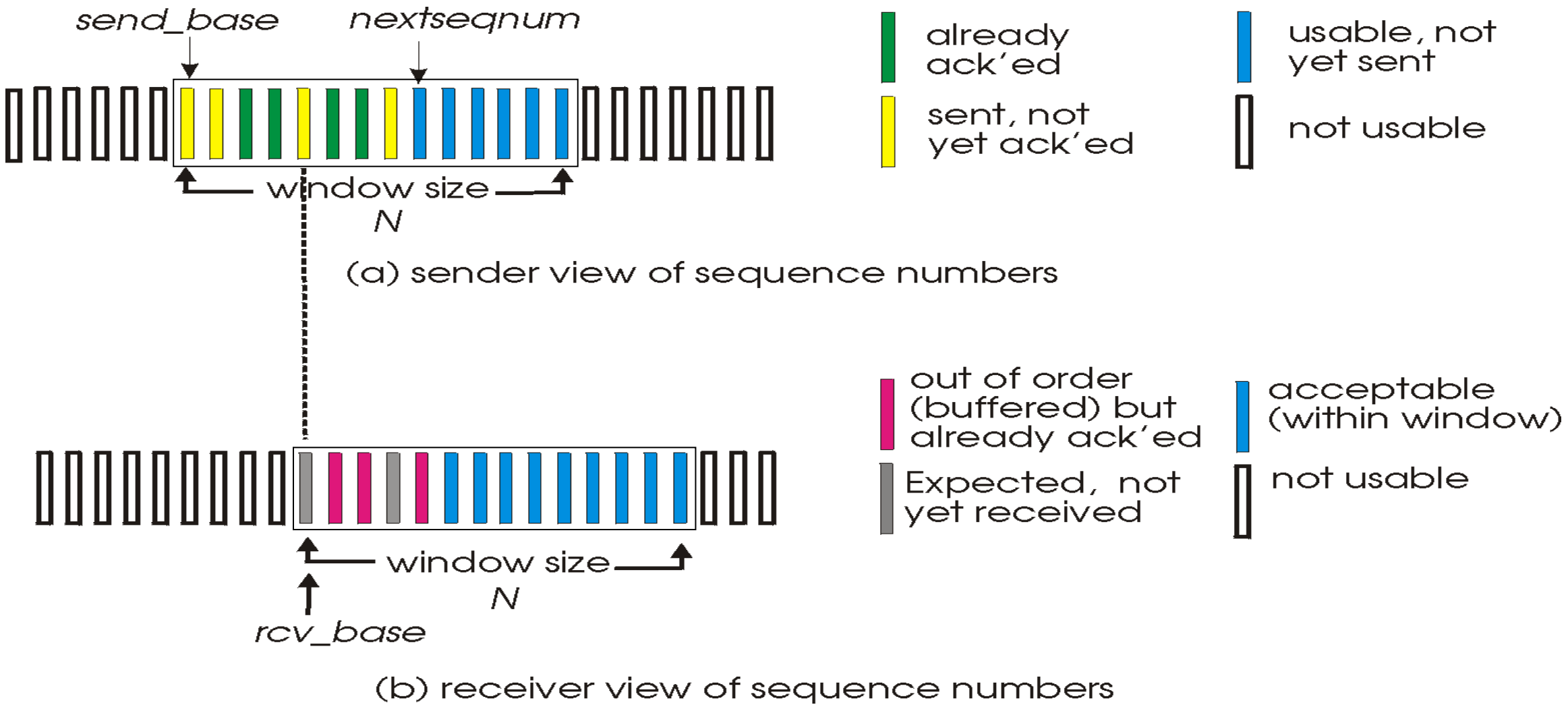
http://gaia.cs.umass.edu/kurose_ross/interactive/

3.4.4 选择性重传（Selective Repeat）

GBN改善了信道效率，但仍然有不必要重传问题，为此进一步提出了SR机制

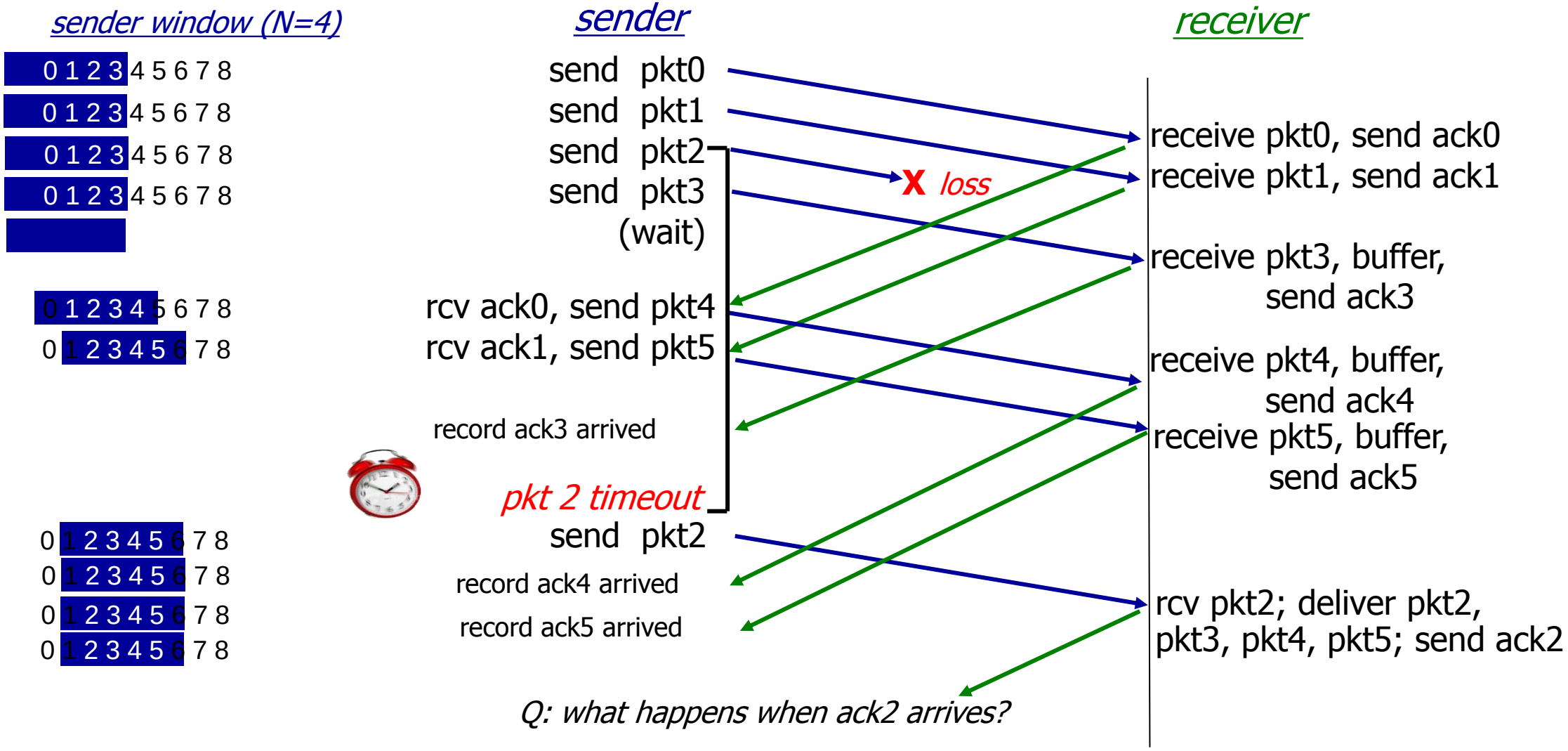
- SR接收方分别确认所有正确接收的报文段
  - 需要缓存分组，以便最后按序交付给给上层
- SR发送方只需要重传没有收到ACK的分组
  - 发送方定时器对每个没有确认的分组计时
- SR发送窗口
  - N个连续的序号
  - 也需要限制已发送但尚未应答分组的序号

Selective repeat: sender, receiver windows



GBN只有发送窗口；SR既有发送窗口，也有接收窗口

Selective repeat in action



选择性重传

发送方

上层传来数据：

- 如果窗口中下一个序号可用, 发送报文段

timeout(n):

- 重传分组n, 重启其计时器

ACK(n) 在[sendbase, sendbase+N]:

- 标记分组 n 已经收到
- 如果n 是最小未收到应答的分组，向前滑动窗口base指针到下一个未确认序号

接收方

分组n在 [rcvbase, rcvbase+N-1]

- 发送 ACK(n)
- 失序: 缓存
- 按序: 交付 (也交付所有缓存的按序分组),向前滑动窗口到下一个未收到报文段的序号

分组n在[rcvbase-N,rcvbase-1]

- ACK(n)

其他:

- 忽略

课堂练习

- R12、R13

R12. Visit the Go-Back-N Java applet at the companion Web site.

a. Have the source send five packets, and then pause the animation before any of the five packets reach the destination. Then kill the first packet and resume the animation. Describe what happens.

b. Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.

c. Finally, try sending six packets. What happens?

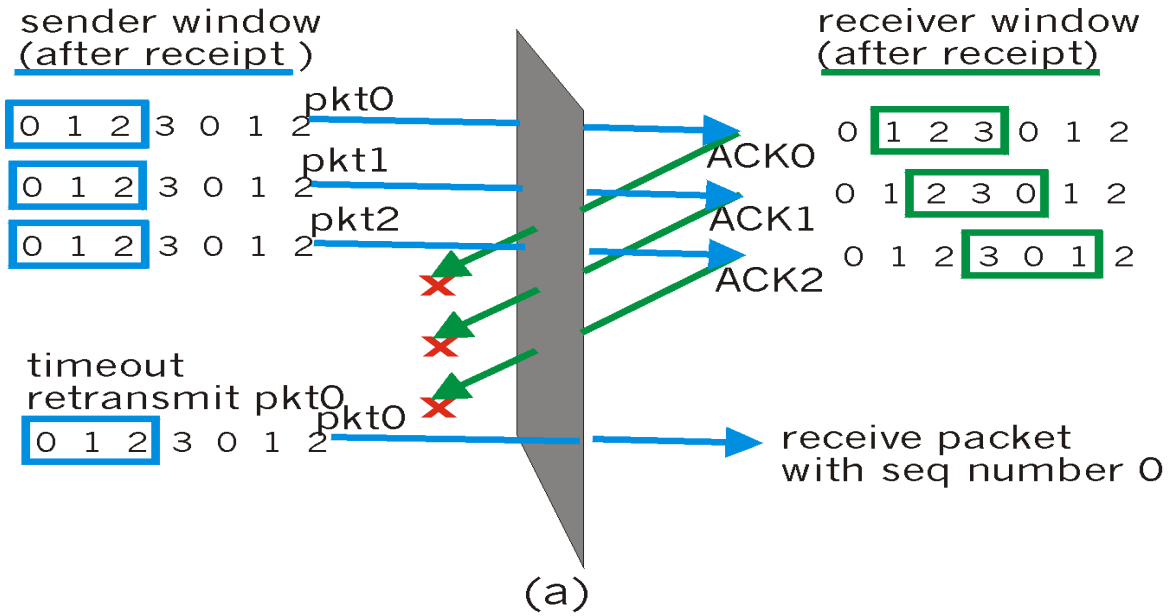
- R13: 使用选择重传来完成上述问题，与GBN有什么不同

http://gaia.cs.umass.edu/kurose_ross/interactive/

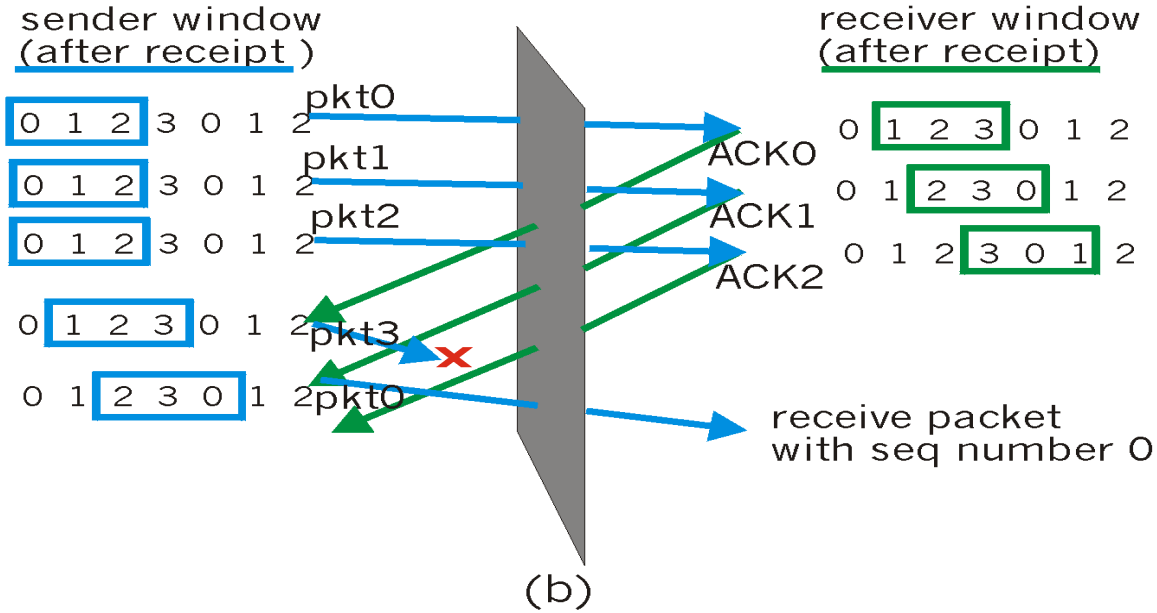
选择重传: 困难的问题

例子:

- 序号: 0, 1, 2, 3
- 窗口长度 = 3



- 接收方：在(a)和(b)两种情况下接收方没有发现差别!
- 在 (a)中不正确地将新的冗余的当为新的，而在(b)中不正确地将新的当作冗余的



问题: 序号长度与窗口长度有什么关系?

回答: 窗口长度小于等于序号空间的一半

请同学回答，这两种问题的场景是什么，带来了什么样的问题

可靠数据传输机制及用途总结

机制	用途和说明
检验和	用于检测在一个传输分组中的比特错误。
定时器	用于检测超时/重传一个分组，可能因为该分组（或其ACK）在信道中丢失了。由于当一个分组被时延但未丢失（过早超时），或当一个分组已被接收方收到但从接收方到发送方的ACK丢失时，可能产生超时事件，所以接收方可能会收到一个分组的多个冗余拷贝。
序号	用于为从发送方流向接收方的数据分组按顺序编号。所接收分组的序号间的空隙可使该接收方检测出丢失的分组。具有相同序号的分组可使接收方检测出一个分组的冗余拷贝。
确认	接收方用于告诉发送方一个分组或一组分组已被正确地接收到了。确认报文通常携带着被确认的分组或多个分组的序号。确认可以是逐个的或累积的，这取决于协议。
否定确认	接收方用于告诉发送方某个分组未被正确地接收。否定确认报文通常携带着未被正确接收的分组的序号。
窗口、流水线	发送方也许被限制仅发送那些序号落在一个指定范围内的分组。通过允许一次发送多个分组但未被确认，发送方的利用率可在停等操作模式的基础上得到增加。我们很快将会看到，窗口长度可根据接收方接收和缓存报文的能力或网络中的拥塞程度，或两者情况来进行设置。

研讨题目3：TCP的数据可靠传输机制

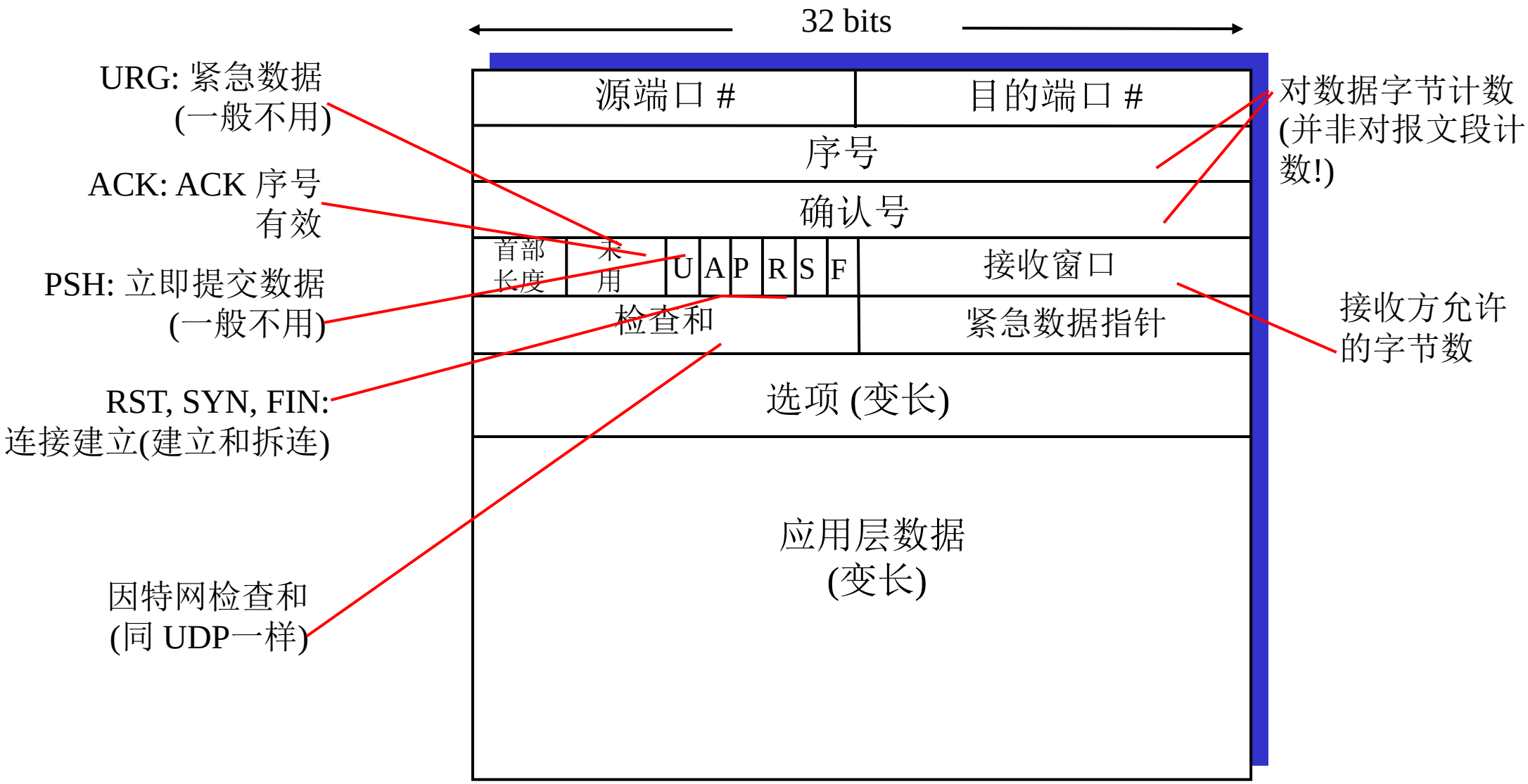
- **TCP的可靠传输机制的设计**
  - **首先请两名同学分别介绍你为TCP协议设计的可靠传输机制（可准备你自己的PPT）；**
  - **包括如下内容：**
    - **差错检查？**
    - **序号？**
    - **正确确认？**
    - **否定确认？**
    - **重传？**
    - **定时器?**
    - **流水线？GBN机制，还是SR机制，收发双方是否需要缓冲区？**
    - **尝试画出发送方和接收方的有限状态机**

研讨题目3： TCP的数据可靠传输机制

- **TCP的可靠传输机制的设计**
  - 请两名同学针对前面2名同学的设计进行点评或提问（）；
    - 差错检查？
    - 序号？
    - 正确确认？
    - 否定确认？
    - 重传？
    - 定时器?
    - 流水线？GBN机制，还是SR机制，收发双方是否需要缓冲区？

VINTON CERF, ROBERT KAHN因为设计了TCP协议获得了2004年的图灵奖！！！

(1) TCP 报文段结构


32 bits
源端口 #
目的端口 #
序号
确认号
首部长度
未用
U
A
P
R
S
F
接收窗口
检查和
紧急数据指针
选项 (变长)
应用层数据
(变长)
URG: 紧急数据
(一般不用)
ACK: ACK 序号
有效
PSH: 立即提交数据
(一般不用)
RST, SYN, FIN:
连接建立(建立和拆连)
因特网检查和
(同 UDP一样)
对数据字节计数
(并非对报文段计数!)
接收方允许
的字节数

TCP序号和确认号

- 序号：一个报文段的序号是其首字节的字节流编号。

例子：假设数据流由一个500,000的字节组成，每个段的数据为1000字节，则需要将数据流分为500个报文段，第1个报文段的序号为0，第2个报文段的序号为1000，如图

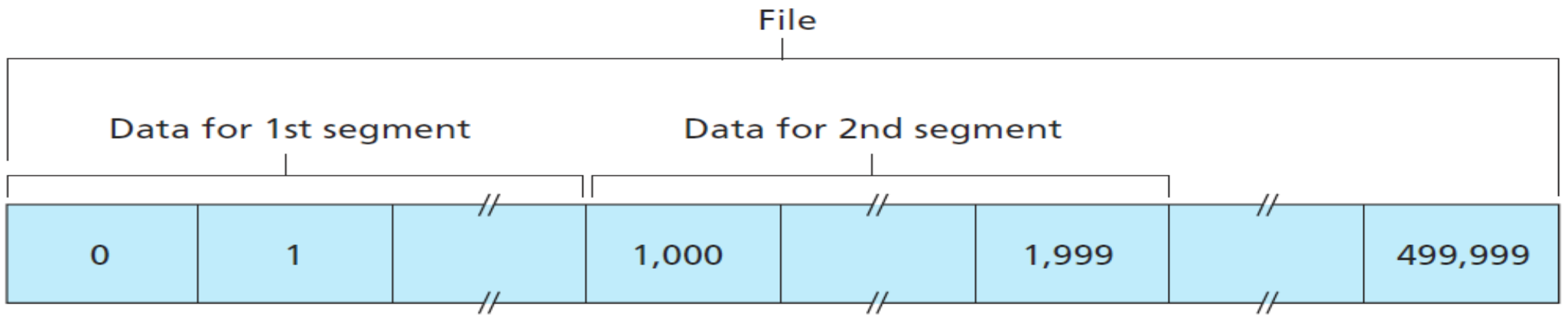


Figure 3.30 ◆ Dividing file data into TCP segments

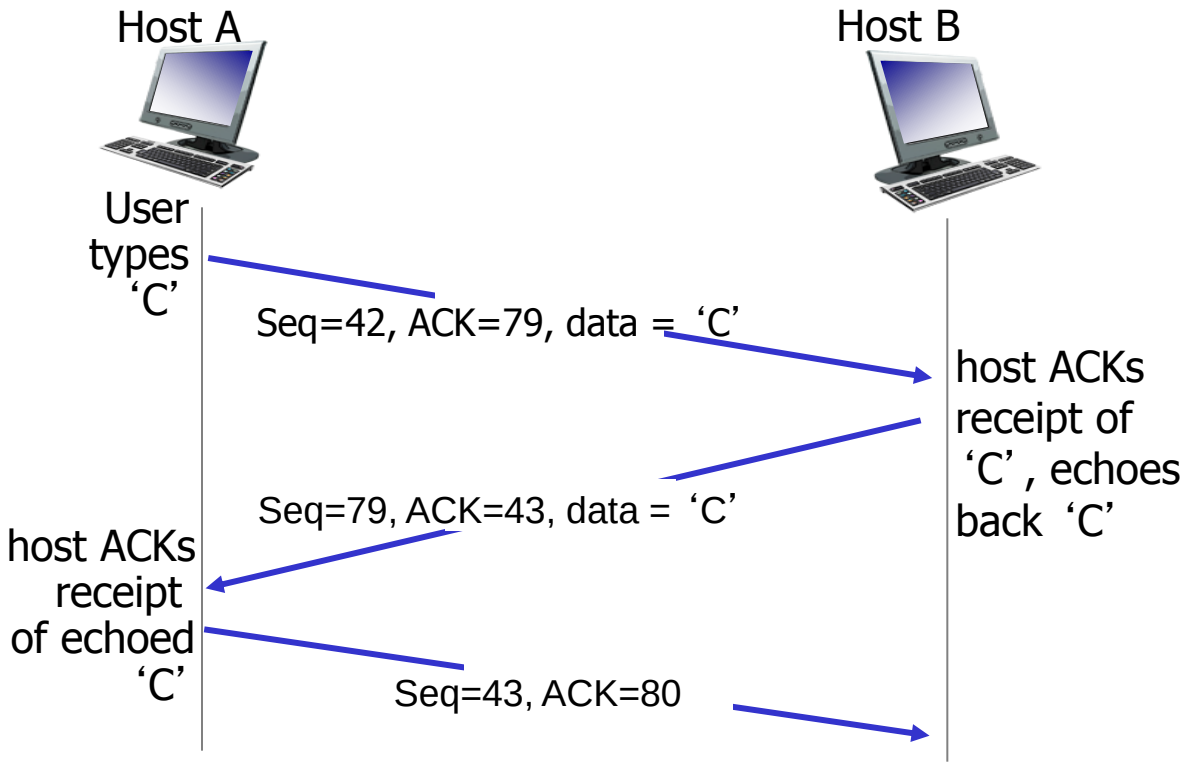
TCP序号和确认号

- 确认号：期望从对方收到下一个字节的序号
  - 累计确认机制：TCP只确认数据流中至第1个未收到字节为止的字节。

例子：假设主机A已经收到主机B的编号为0-535的所有字节，因此主机A发给主机B的确认包中的确认号为536.

什么是累计确认：假设主机A收到的字节为0-500,510-535，那么主机A发送给主机B的确认包中的确认号为501.

TCP seq. numbers, ACKs



simple telnet scenario

(2) TCP正确确认机制(ACK)

- **TCP的接收方的主要任务：**
  - **接收segment，判断是否损坏(校验和)；**
  - **对收到的segment进行ACK确认（累计确认）**
- **TCP中只有正确确认：**
  - **检测到segment损坏，不发生动作，不通知重传；**
  - **根据收到的segment的情况，通过发送不同的ACK(n)来通知接收方需要发送的segment；**

TCP ACK 产生 [RFC 1122, RFC 2581]

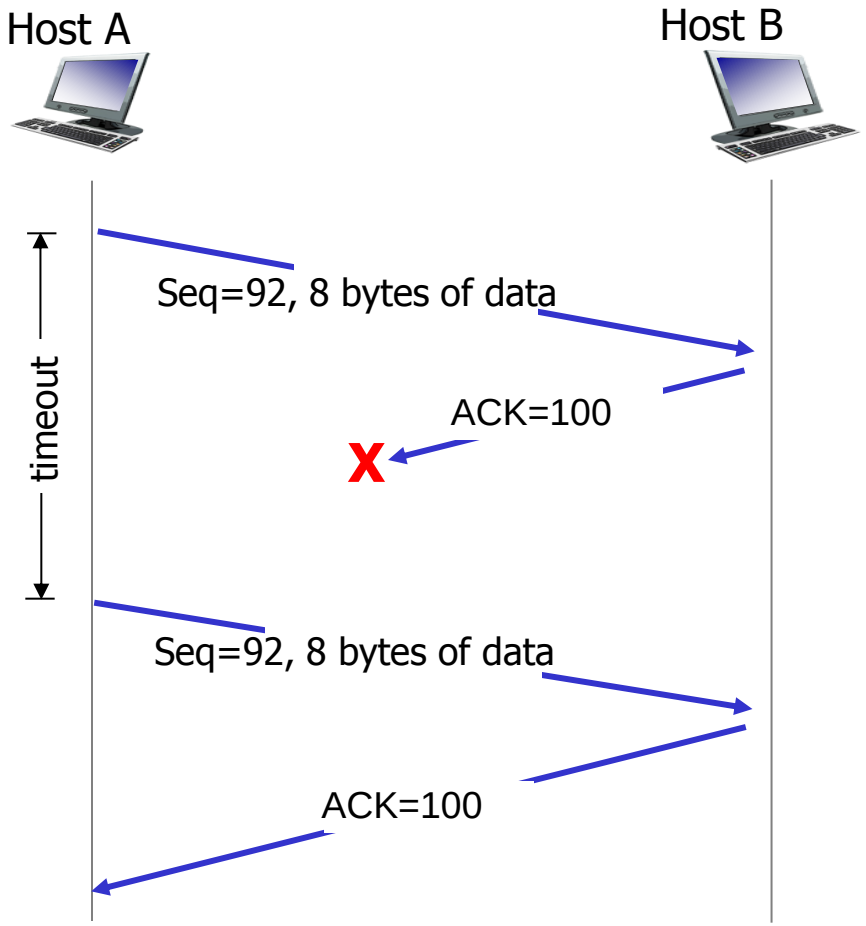
接收方事件	TCP 接收方行为
所期望序号的报文段按序到达。所有在期望序号及以前的数据都已经被确认	延迟的ACK。对另一个按序报文段的到达最多等待500 ms。如果下一个按序报文段在这个时间间隔内没有到达，则发送一个ACK
有期望序号的报文段按序到达。另一个序号大于期望序号的报文段等待发送ACK	立即发送单个累积ACK，以确认两个按序报文段
比期望序号大的失序报文段到达，检测出数据流中的间隔。	立即发送冗余ACK，指明下一个期待字节的序号（也就是间隔的低端字节序号）
部分或者完全填充已接收到数据间隔的报文段到达	倘若该报文段起始于间隔的低端，则立即发送ACK

(3) 发送方的重传机制

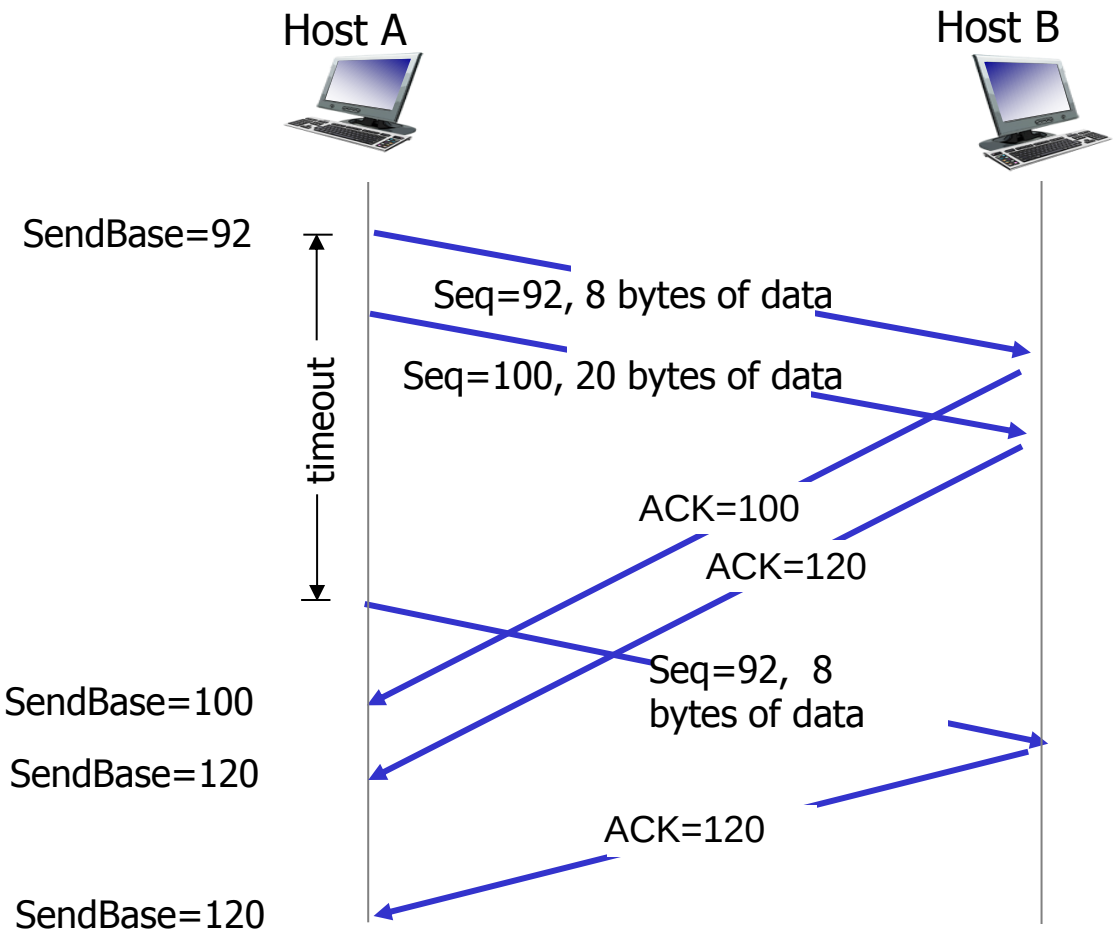
- TCP的发送方的主要任务：
  - 发送数据报；
  - 根据收到的ACK确认，判断是否启动重传机制；
- 在什么情况下，启动重传(判断数据报丢失)：
  - 定时器超时，则认为对应的数据报丢失，立即重传；
  - 收到3个相同的ACK(n)确认，则认为对应的数据报丢失，立即重传；

首先看定时器超时，引起重传的几种场景。。。
再分析3个相同ACK确认是什么情况。。。

TCP: retransmission scenarios

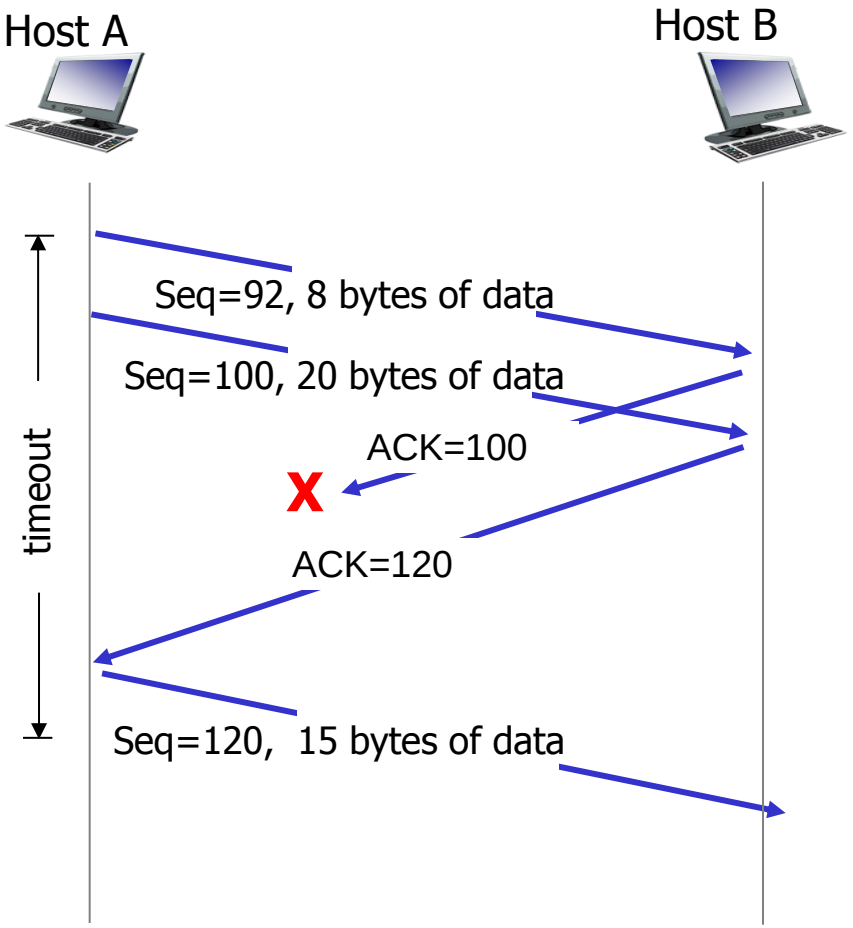


lost ACK scenario



premature timeout

TCP: retransmission scenarios



cumulative ACK

TCP快速重传

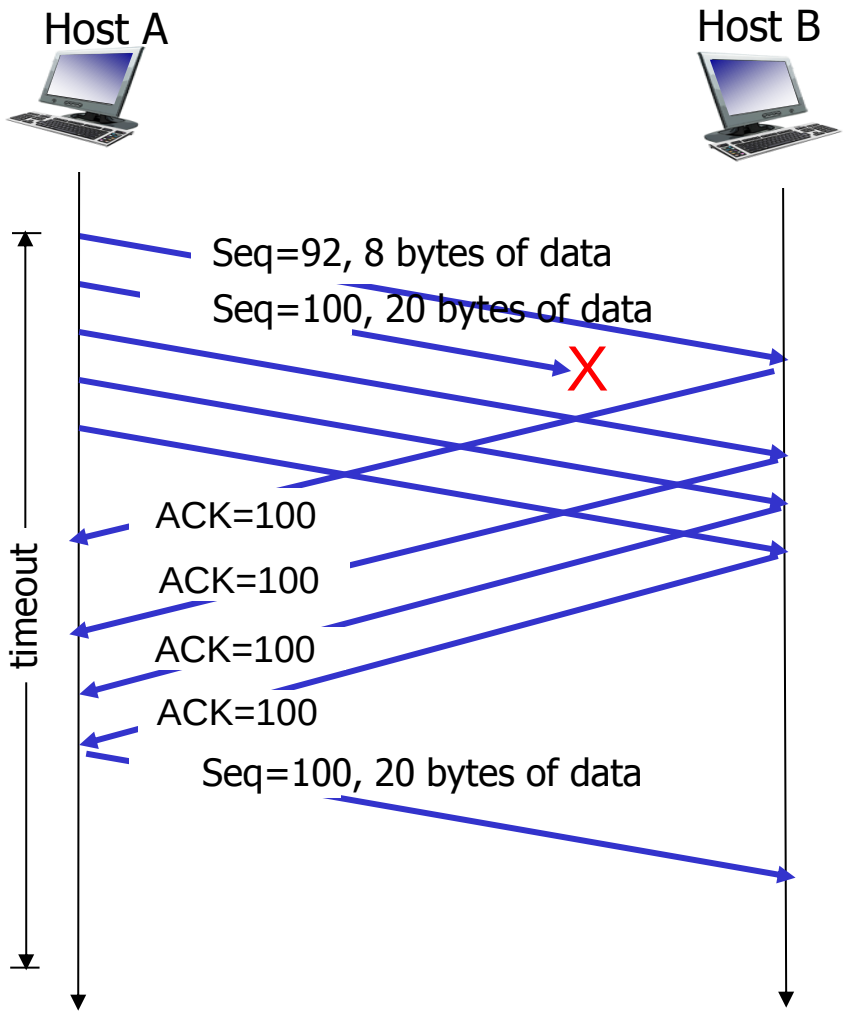
- 定时器的超时时间通常比较长:
  - 在重发丢失分组之前有较长的延时
- 可以通过重复的ACK确认来检测丢失的报文段
  - 发送方通常连续的发送多个报文段
  - 如果有报文段丢失，将很可能有多个重复的ACKs。

TCP fast retransmit

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), unacked segment with smallest seq #

- likely that unacked segment lost, so don't wait for timeout

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

(4) TCP的定时器机制

- **TCP的发送方只维护一个定时器(timer)；**
  - 启动：每当发送一个报文且这个报文是最早的发送且还未确认的，则启动定时器；
  - 终止/重启：
    - 当收到确认且这个确认是最早的发送且还未确认的报文段的确认，则终止定时器，且将发送窗口向前移动，并对新的最早发送且未确认的报文段启动定时器；
  - 超时后的动作：重传定时器对应的数据段，并对重传的数据段重启定时器；
  - 超时时间间隔的合理设置：
    - 设置什么值更为合理？
    - 如果再次超时，这个时间间隔是否需要调整？

    （随机请一位同学来回答）

TCP round trip time, timeout

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- ***too short***: premature timeout, unnecessary retransmissions
- ***too long***: slow reaction to segment loss

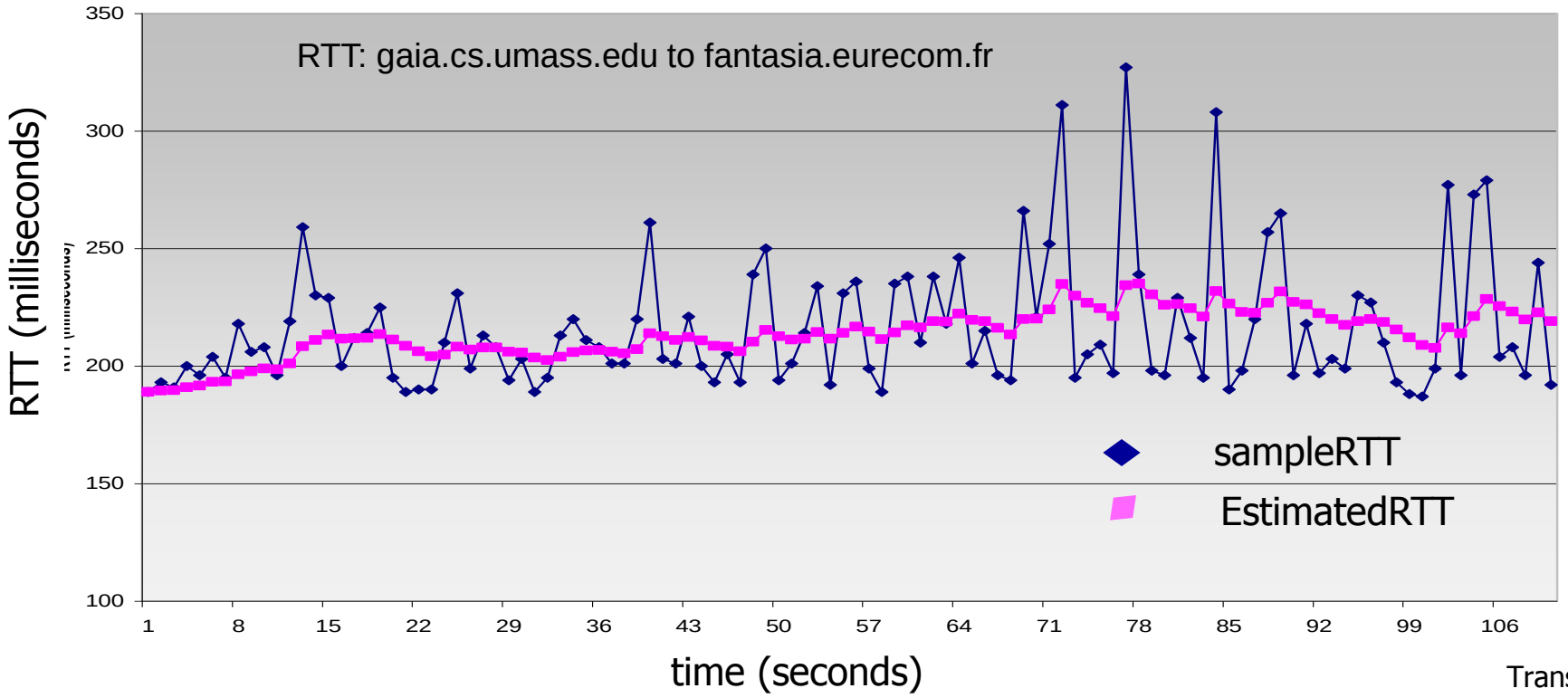
Q: how to estimate RTT?

- **`SampleRTT`**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current **`SampleRTT`**

TCP round trip time, timeout

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

$$\texttt{DevRTT} = (1-\beta)*\texttt{DevRTT} + \beta*|\texttt{SampleRTT}-\texttt{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4*\texttt{DevRTT}$$

estimated RTT

"safety margin"

(5) TCP流水线机制

- **TCP采用了一种GBN和SR的混合机制**
  - **累计确认**
  - **单个定时器**
  - **将失序达到的报文段缓存**
  - **每次重传只发送一个报文段(无论是定时器超时，还是收到3个重复ACK)**

总结：TCP 发送方事件

1.从应用层接收数据:

- 根据序号创建报文段
- 序号是报文段中第一个数据字节的数据流编号
- 如果未启动，启动计时器 (考虑计时器用于最早的没有确认的报文段)
- 超时间隔: **TimeOutInterval= EstimatedRTT + 4*DevRTT**

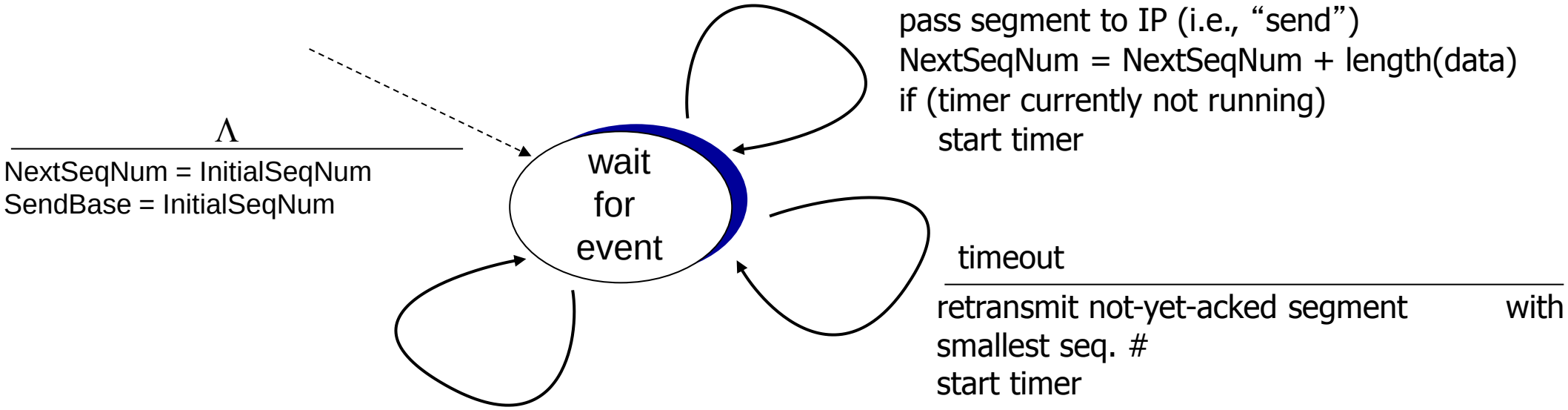
2.超时:

- 重传导致超时的报文段
- 重新启动计时器

3.收到确认：

- 如果确认了先前未被确认的报文段
  - 更新被确认的报文段序号
  - 如果还有未被确认的报文段，重新启动计时器
- 如果收到冗余的对已确认的报文段的确认

TCP sender (simplified)（无快速重传）



课堂练习

P27. Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

b. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?

c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?

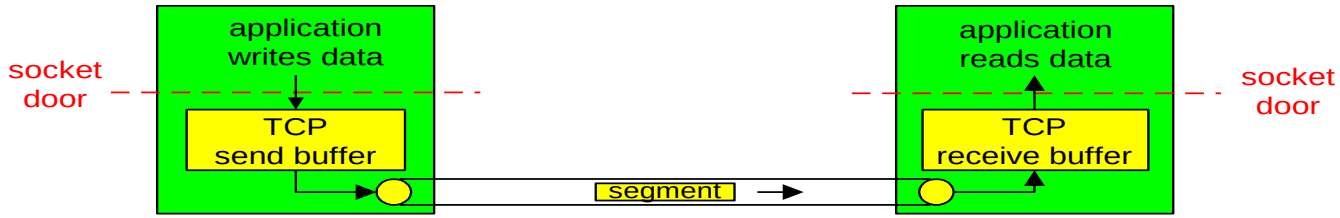
课堂练习

d. Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.

P31. Suppose that the five measured `SampleRTT` values (see Section 3.5.3) are 106 ms, 120 ms, 140 ms, 90 ms, and 115 ms. Compute the `EstimatedRTT` after each of these SampleRTT values is obtained, using a value of $\alpha = 0.125$ and assuming that the value of `EstimatedRTT` was 100 ms just before the first of these five samples were obtained. Compute also the `DevRTT` after each sample is obtained, assuming a value of $\beta = 0.25$ and assuming the value of `DevRTT` was 5 ms just before the first of these five samples was obtained. Last, compute the TCP `TimeoutInterval` after each of these samples is obtained.

TCP概述 RFCs: 793, 1122, 1323, 2018, 2581

- **点到点:**
  - 一个发送方, 一个接收方
  - 连接状态与端系统有关，不为路由器所知
- **可靠、有序的字节流:**
  - 没有 “报文边界”
- **流水线:**
  - TCP拥塞和流量控制设置滑动窗口协议
- ***发送和接收缓冲区***



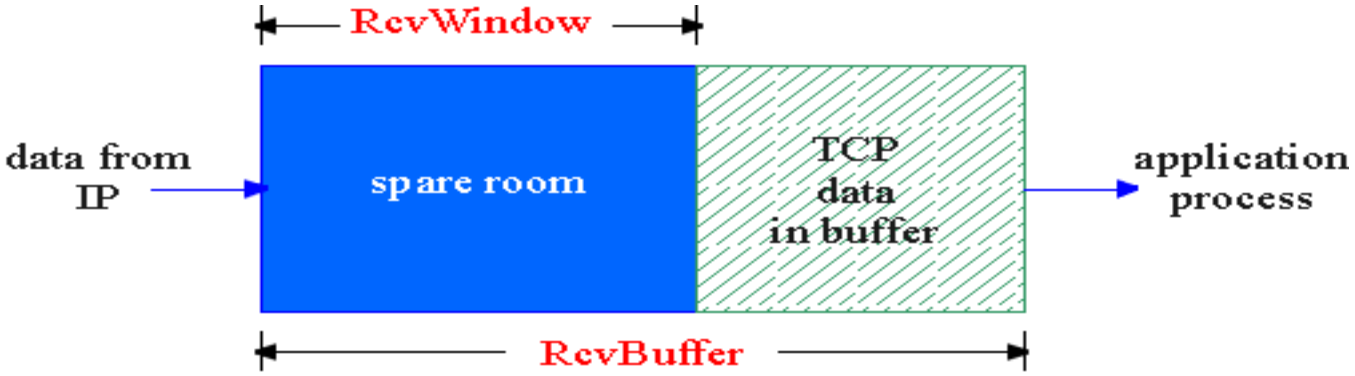
- **全双工数据:**
  - 同一连接上的双向数据流
  - MSS: 最大报文段长度
  - MTU:最大传输单元
- **面向连接:**
  - 在进行数据交换前，初始化发送方与接收方状态，进行握手(交换控制信息),
- **流量控制:**
  - 发送方不能淹没接收方
- **拥塞控制:**
  - 抑止发送方速率来防止过分占用网络资源

最大报文段长度MSS

- TCP可从发送缓存中取出并放入报文段(segment)的数据量受限于最大报文段长度MSS。
- MSS通常根据最初确定的本地发送主机发送的最大链路层帧长度MTU来设置。
  - MSS+TCP/IP头部的长度（40字节）= MTU
- 以太网和PPP链路中MTU为1500字节，因此MSS通常为1460字节。

3.5.5 TCP 流量控制

- TCP连接的接收方有1个接收缓冲区:



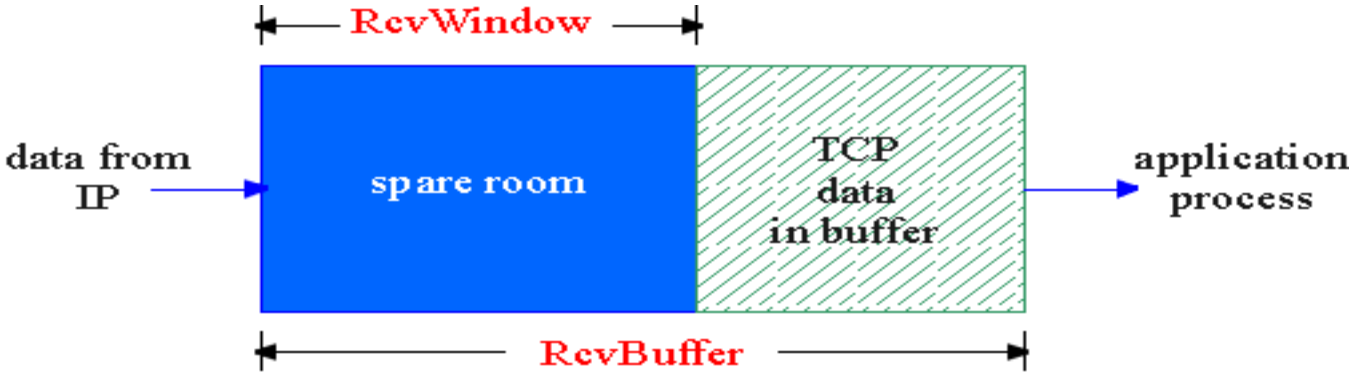
- 应用进程可能从接收缓冲区读数据缓慢

流量控制

发送方不能发送太多、太快的数据让接收方缓冲区溢出

- 匹配速度服务: 发送速率需要匹配接收方应用程序的提取速率

3.5.5 TCP流控: 工作原理



(假设 TCP 接收方丢弃失序的报文段)

- 缓冲区的剩余空间

= RcvWindow

= RcvBuffer-[LastByteRcvd - LastByteRead]

- 接收方在报文段接收窗口字段中通告其接收缓冲区的剩余空间
- 发送方要限制未确认的数据不超过**RcvWindow**

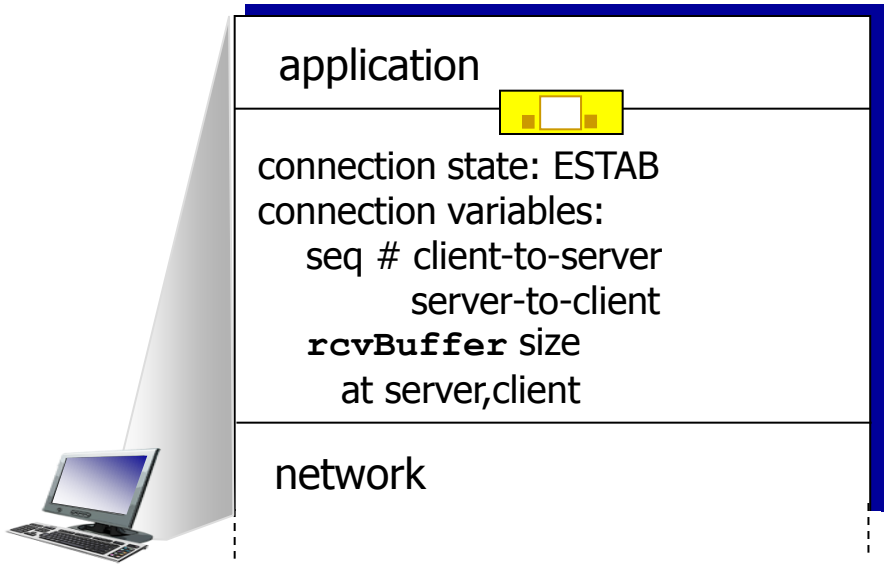
LastByteSent-LastByteAcked <或= RcvWindow

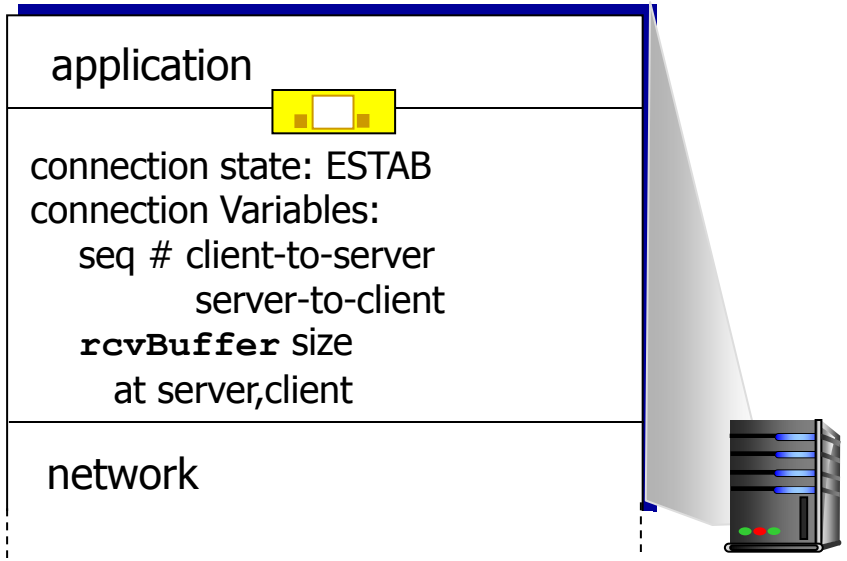
  - 保证接收缓冲区不溢出

3.5.6 连接管理

before exchanging data, sender/receiver “handshake”:

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

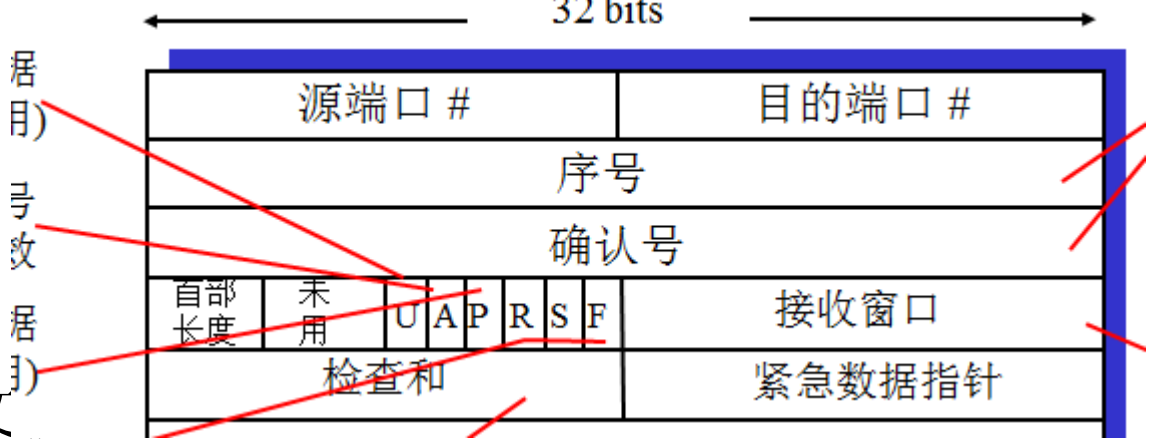





```
Socket clientSocket =
  newSocket("hostname","port number");
```

```
Socket connectionSocket =
  welcomeSocket.accept();
```

3.5.6 TCP 3-way handshake



三次握手:

步骤 1: 客户机向服务器发送 TCP SYN报文

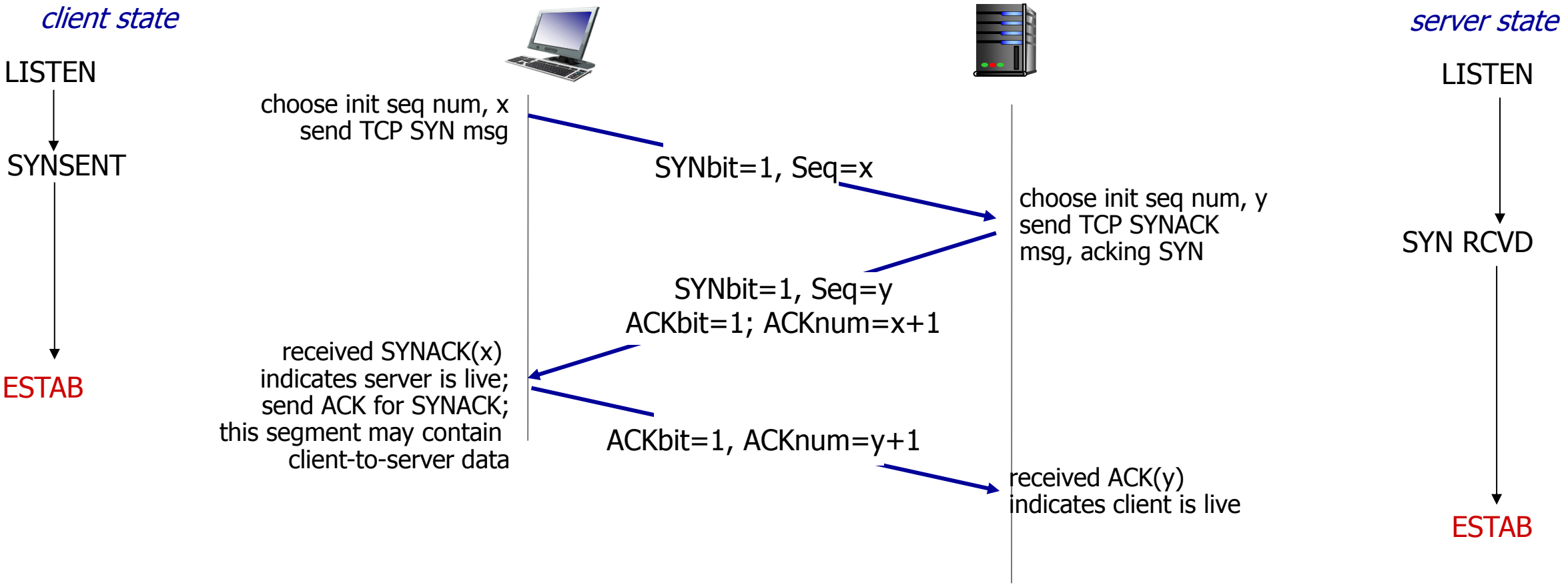
- 指定初始序号（随机产生）
- 没有数据

步骤 2: 服务器收到SYN报文段, 用SYNACK报文段回复

- 服务器为该连接分配缓冲区和变量
- 指定服务器初始序号

步骤 3: 客户机接收到 SYNACK, 用ACK报文段回复,可能包含数据

3.5.6 TCP 3-way handshake



TCP 关闭连接(续)

关闭连接: 服务器和客户端的任何一方都可以发送消息主动关闭连接

send TCP segment with FIN bit = 1

客户关闭套接字: `clientSocket.close();`

步骤 1: 客户机向服务器发送TCP FIN控制报文段

步骤 2: 服务器收到FIN，用ACK回答。关闭连接，发送FIN

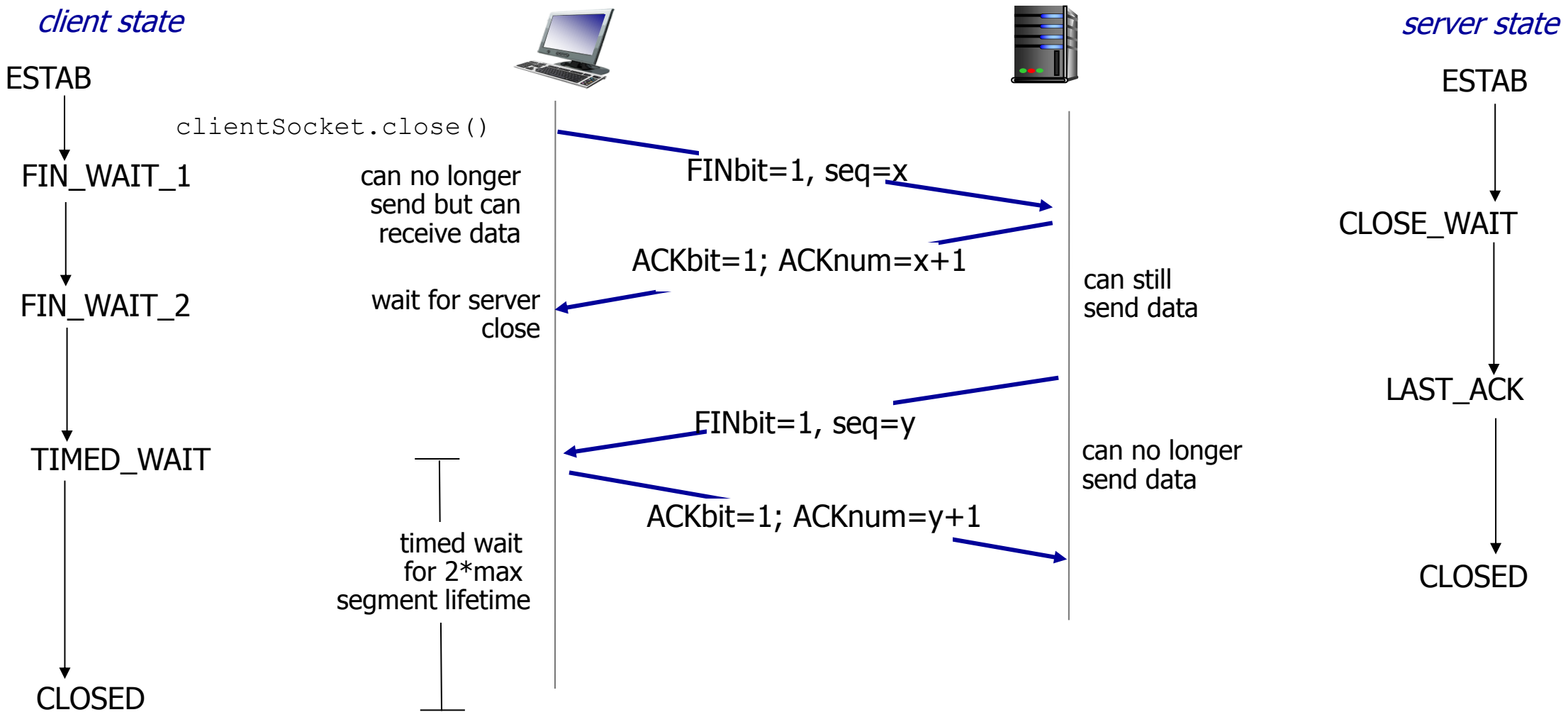
步骤 3: 客户机收到FIN, 用ACK回答

- 进入 “超时等待” - 将对接收到的FIN进行确认

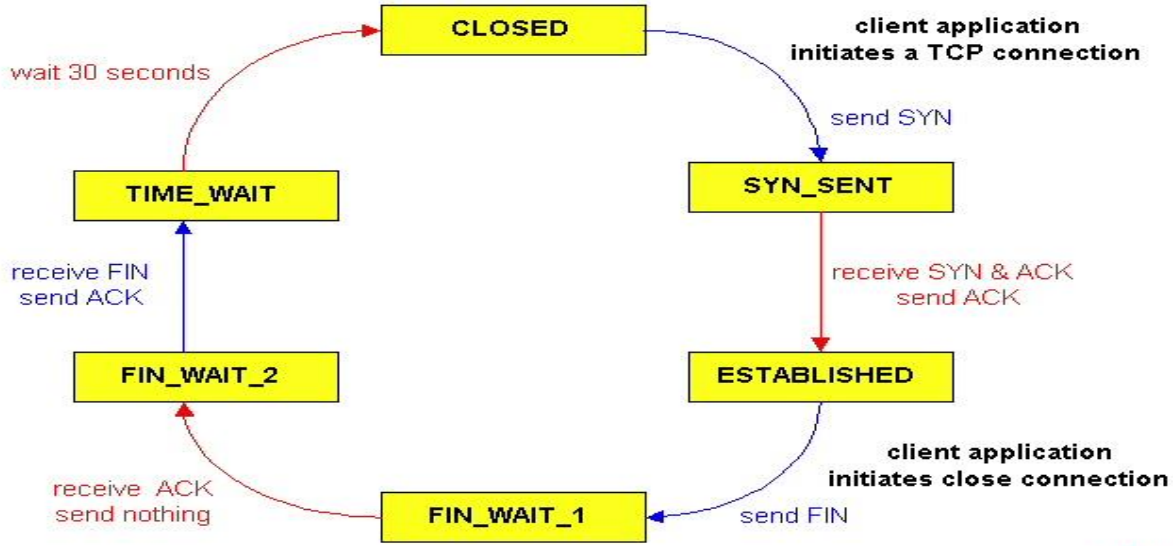
步骤 4: 服务器接收ACK，连接关闭

注意: 少许修改, 可以处理并发的FIN

TCP 关闭连接(续)

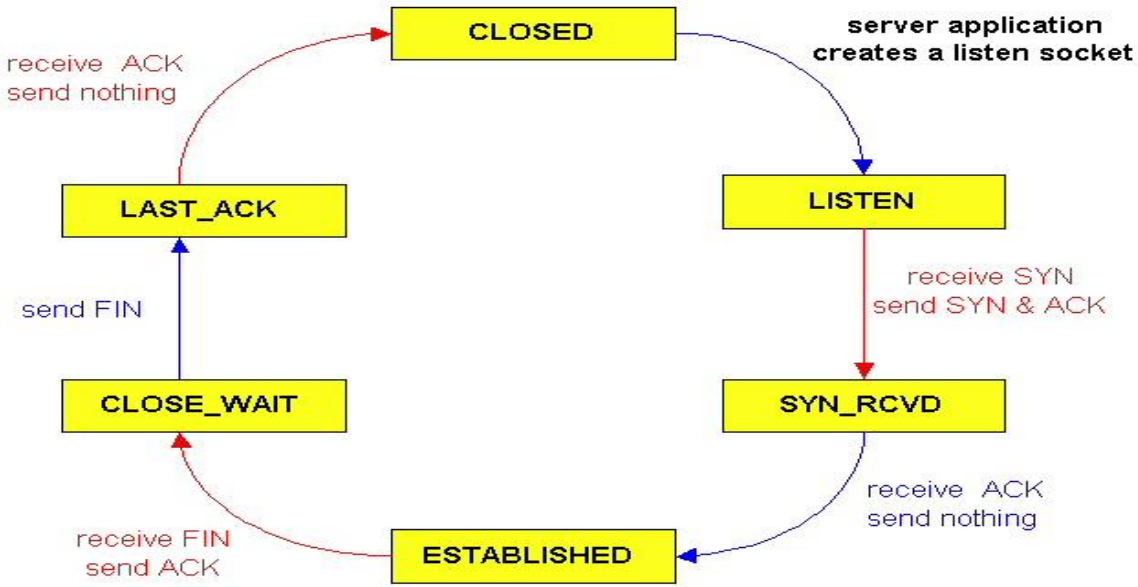

client state
ESTAB
clientSocket.close()
FIN_WAIT_1
can no longer send but can receive data
FINbit=1, seq=x
ACKbit=1; ACKnum=x+1
FIN_WAIT_2
wait for server close
can still send data
FINbit=1, seq=y
TIMED_WAIT
can no longer send data
ACKbit=1; ACKnum=y+1
timed wait for 2*max segment lifetime
CLOSED
server state
ESTAB
CLOSE_WAIT
LAST_ACK
CLOSED

TCP 连接管理 (续)



TCP 客户
生命周期

TCP 服务器
生命周期



SYN洪泛攻击

- TCP的三次握手机制为SYN洪泛攻击提供了环境。
  - 攻击者发送大量的TCP SYN报文段，而不去完成第3次握手的步骤，服务器不断为这些半开连接分配资源，导致服务器的连接资源被耗尽。
  - 一种有效的防御系统，称为SYN cookie，已经部署在大多数的主流操作系统中
    - 当服务器收到一个SYN报文时，它并不知道这个报文段是来自一个合法的用户，还是一个SYN洪泛攻击者，因此服务器不会为该报文段生成一个半开连接；
    - 服务器生成一个初始TCP序列号，这个序列号是TCP的源、目的IP地址、源、目的端口号和一个服务器才知道的秘密数字的hash值，称为cookie；
    - 服务器发送携带了cookie作为序列号的SYNACK分组

SYN洪泛攻击

- 如果客户是合法的，则客户将会返回一个ACK报文；
- 服务器收到这个ACK报文后，将其中的源、目的IP地址、源、目的端口号及以前生成的秘密数进行hash，得到的hash值加1，判断是否与该ACK报文的确认号字段是否一致，如果一致，才能判断客户是合法的。
- 最终服务器生成一个具有套接字的全开连接。

如果客户是攻击者，则不会返回ACK报文，那么服务器并不会对该客户的TCP连接事先分配任何资源

课堂练习

R14. 是非判断题：

a. 主机 A 经过一条 TCP 连接向主机 B 发送一个大文件。假设主机 B 没有数据发往主机 A。因为主机 B 不能随数据捎带确认，所以主机 B 将不向主机 A 发送确认。

b. 在连接的整个过程中，TCP 的 rwnd 的长度决不会变化。

c. 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 A 发送但未被确认的字节数不会超过接收缓存的大小。

d. 假设主机 A 通过一条 TCP 连接向主机 B 发送一个大文件。如果对于这条连接的一个报文段的序号为 m，则对于后继报文段的序号将必然是 $m+1$。

e. TCP 报文段在它的首部中有一个 rwnd 字段。

f. 假定在一条 TCP 连接中最后的 SampleRTT 等于 1 秒，那么对于该连接的 TimeoutInterval 的当前值必定大于等于 1 秒。

g. 假设主机 A 通过一条 TCP 连接向主机 B 发送一个序号为 38 的 4 个字节的报文段。在这个相同的报文段中，确认号必定是 42。

课堂练习

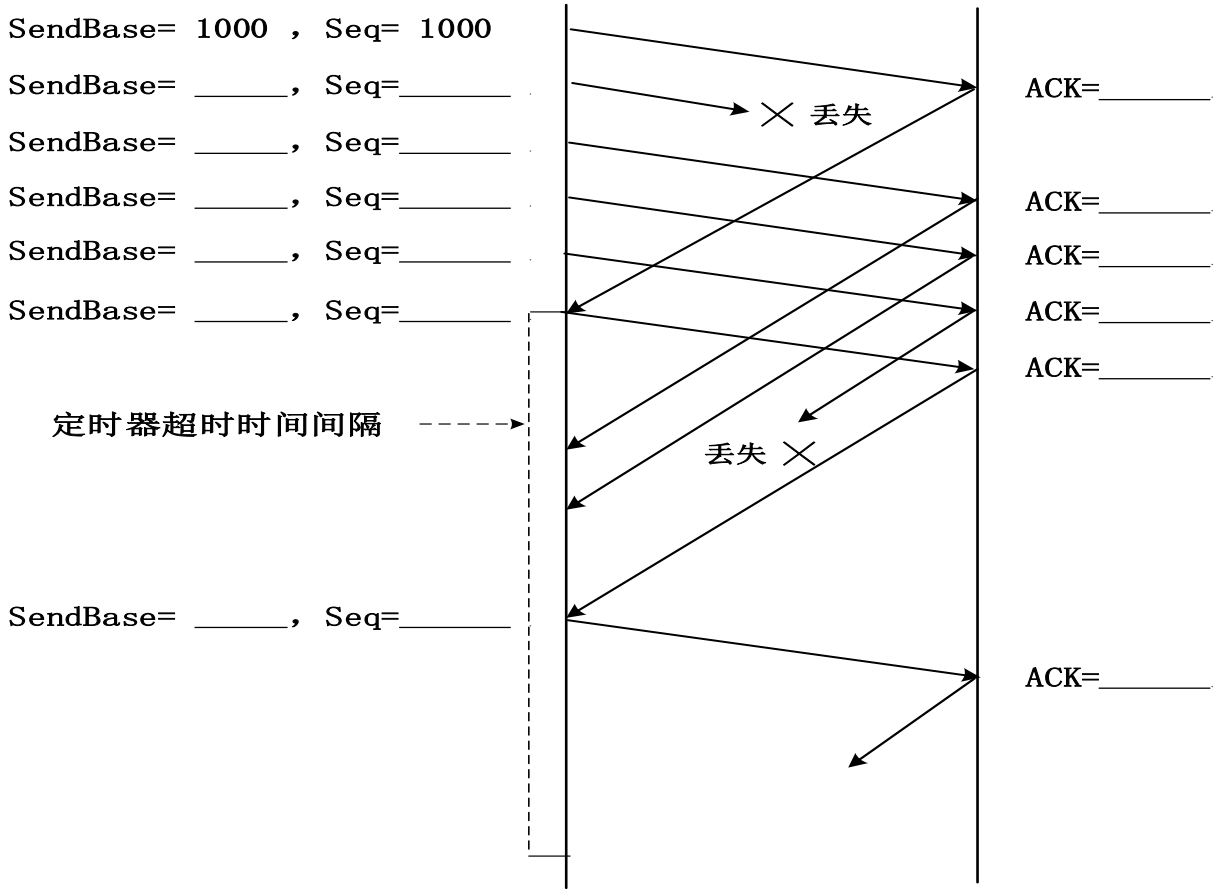
R15. 假设主机 A 通过一条 TCP 连接向主机 B 发送两个紧接着的 TCP 报文段。第一个报文段的序号为 90，第二个报文段序号为 110。

a. 第一个报文段中有多少数据？

b. 假设第一个报文段丢失而第二个报文段到达主机 B。那么在主机 B 发往主机 A 的确认报文中，确认号应该是多少？

Seq=90，20
Seq=110
B->A ack=90

假设主机A和主机B之间建立了TCP连接，并且主机A有大量的数据需要向B发送。发送发主机A支持快速重传，而接收方B会缓存正确接收但失序的报文段，每次发送的报文段的数据字段长度都为100字节。下面的描述中，Seq代表序号，ACK代表确认号，发送方窗口的基序号为SendBase，发送方窗口长度为500字节，请填写下图中各变量的值



3.6 拥塞控制原理(自学)

拥塞:

- 非正式地表述：太多的源发送太多太快的数据，使网络来不及处理
- 不同于流量控制！
- 表现：
  - 丢包 (路由器缓冲区溢出)
  - 长时延 (路由器缓冲区中排队)
- 网络中的前10大问题之一!

拥塞控制方法

控制拥塞的两类方法:

端到端的拥塞控制:

- 不能从网络得到明确的反馈
- 从端系统根据观察到的时延和丢失现象推断出拥塞
- 这是TCP所采用的方法

网络辅助的拥塞控制:

- 路由器为端系统提供反馈
  - 一个bit指示一条链路出现拥塞(SNA, DECnet, TCP/IP ECN, ATM)
  - 指示发送方按照一定速率发送

3.7 TCP 拥塞控制

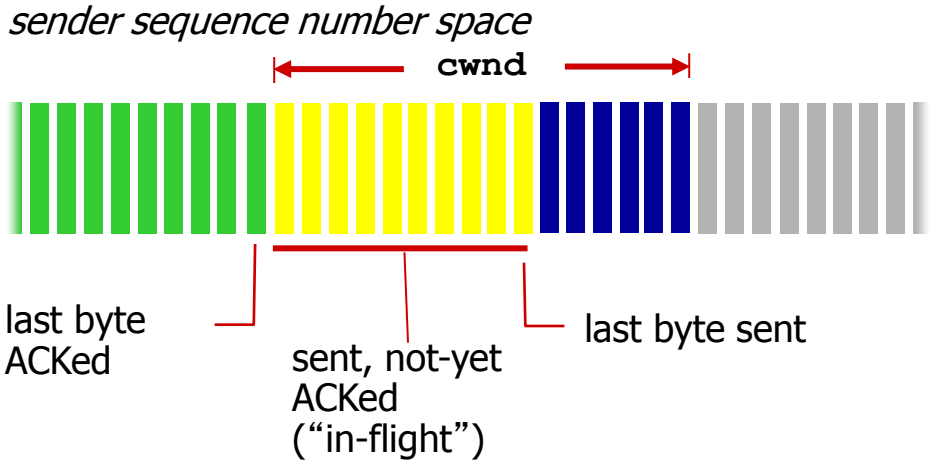
- 端到端控制 (没有网络辅助)
- 发送方限制传输:

  LastByteSent-LastByteAcked ≤ cwnd
- 粗略地,

$$\text{速率} = \frac{\text{cwnd}}{\text{RTT}} \quad \text{Bytes/sec}$$

- 拥塞窗口是动态的, 具有感知到的网络拥塞的函数

网络设计十大问题之一



发送方如何感知网络拥塞?

- 丢失事件 = 超时 *或者* 3个重复ACK
- 发生丢失事件后，TCP发送方降低速率(拥塞窗口)

TCP加增倍减 AIMD

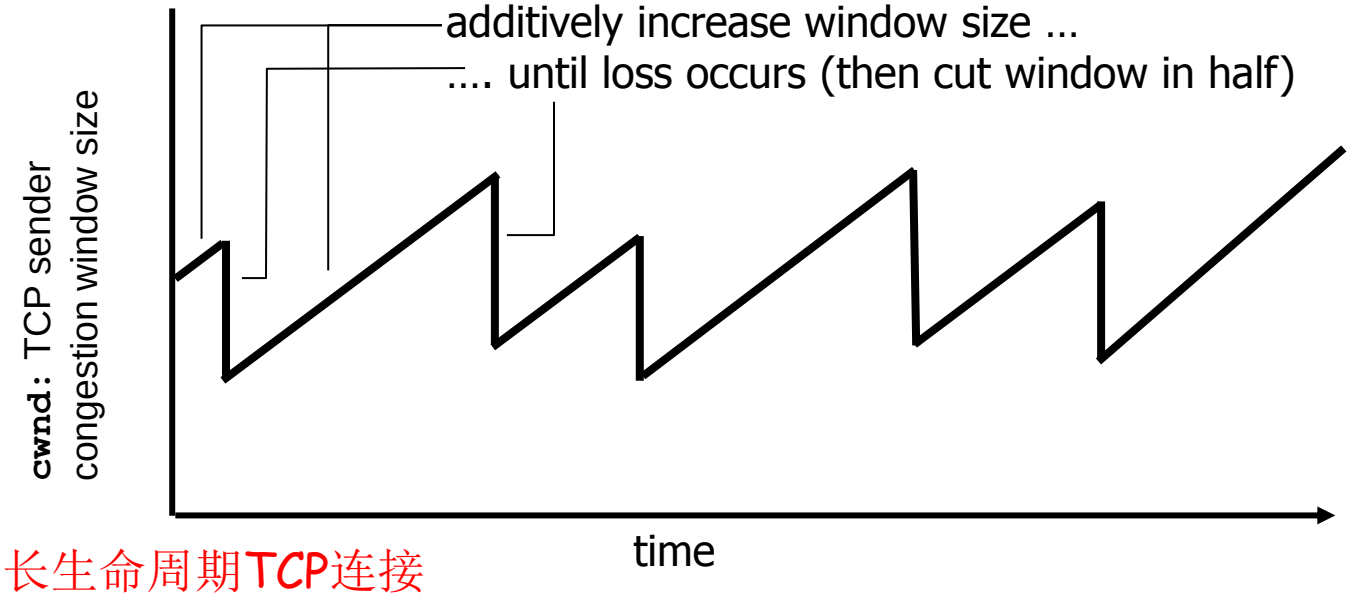
乘性减:

丢包事件后，拥塞窗口值减半

加性增:

如没有检测到丢包事件，每个RTT时间拥塞窗口值增加一个MSS (最大报文段长度)

AIMD saw tooth behavior: probing for bandwidth



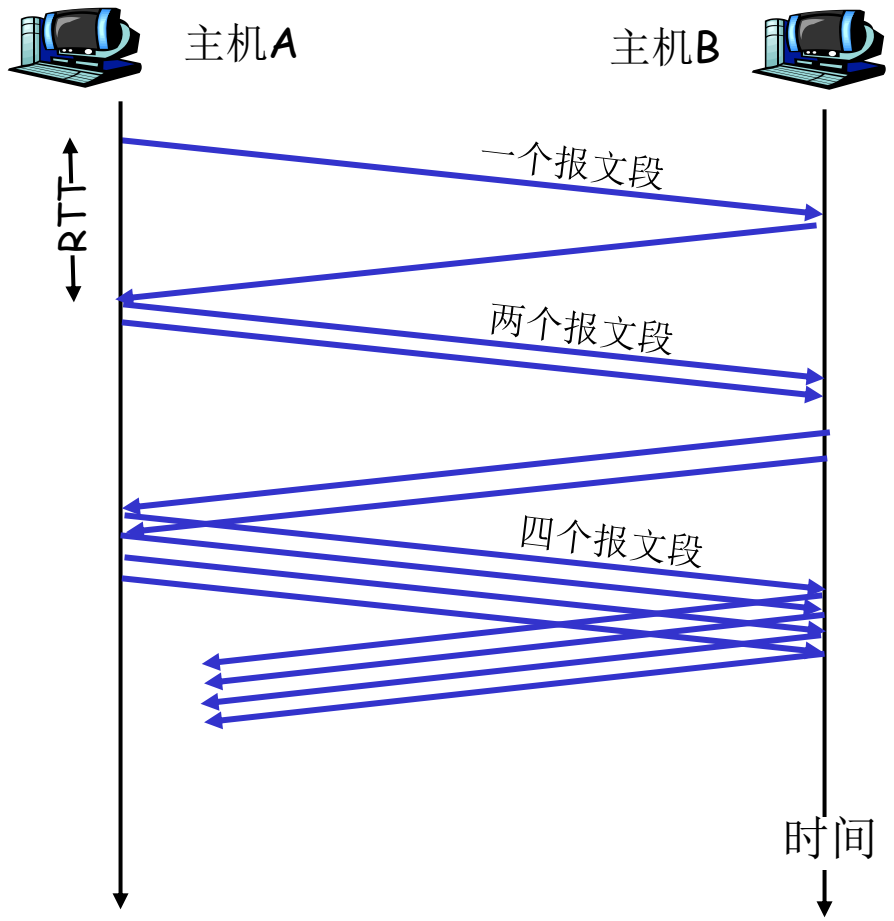
长生命周期TCP连接

1、TCP慢启动（Slow Start）

- 在连接开始时, 拥塞窗口值 = 1 MSS
  - 例如: MSS= 500 bytes & RTT = 200 msec
  - 初始化速率 = 20 kbps
- 可获得带宽可能 >> MSS/RTT
  - 希望尽快达到期待的速率
- 当连接开始，以指数倍增加速率，直到第一个丢失事件发生

1、TCP 慢启动(续)

- 当连接开始的时候，速率呈指数式上升，直到第1次报文丢失事件发生为止:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received（**每收到1个ack确认，cwnd增加1个MSS**）
- 总结: 初始速率很低，但指数倍增加



2、从慢启动进入拥塞避免状态(CA)

问题: 什么时候从指数增长转变为线性增长?

回答: **cwnd**达到它超时以前1/2的时候.

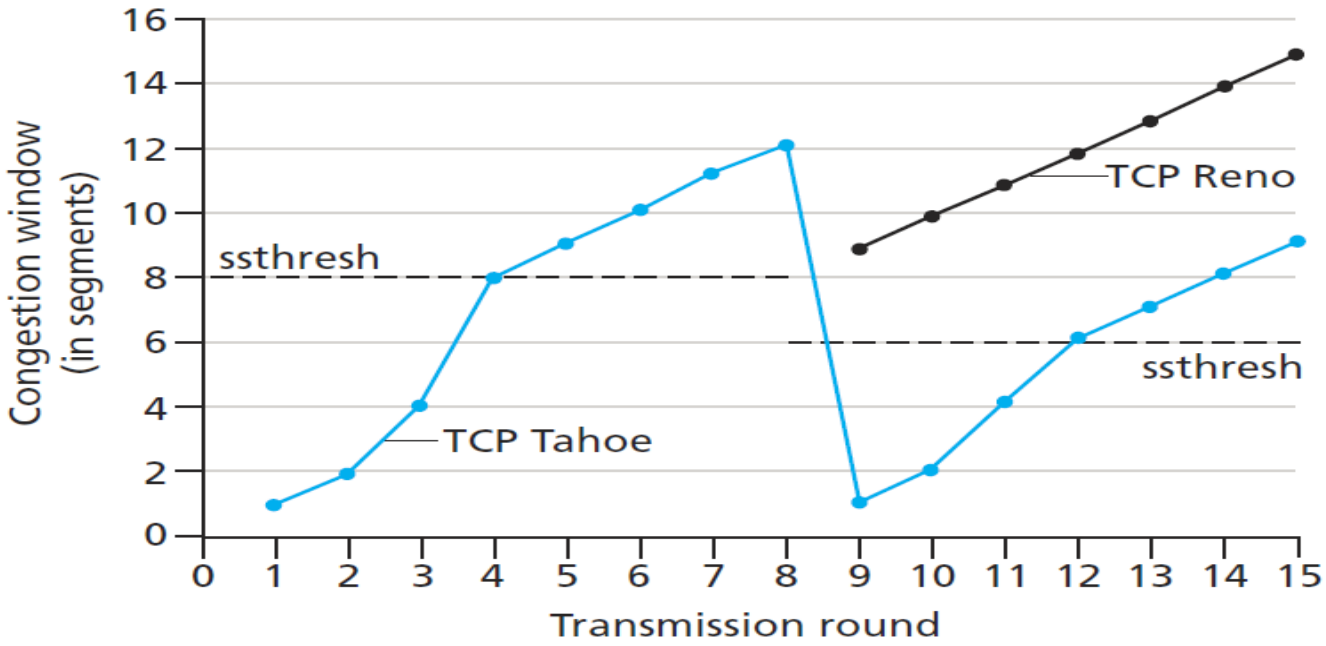


Figure 3.53 ◆ Evolution of TCP's congestion window (Tahoe and Reno)

实现方法:

- 设置一个阈值ssthresh（慢启动阈值）
- 在丢包事件发生时，阈值ssthresh设置为发生丢包以前的cwnd的一半

2、拥塞避免状态

- 当处于慢启动状态，拥塞控制窗口cwnd大于等于ssthresh时，发送方进入拥塞避免状态；
- 当处于快速恢复状态，如果收到一个新的ACK，进入拥塞避免状态

cwnd = ssthresh, 重复计数ACK清0

- 在该状态拥塞窗口cwnd在每个RTT，值增加一个MSS，即每新收到一个ACK:

cwnd = cwnd + MSS(MSS/cwnd)

注意与慢启动的区别

- 例如，MSS = 1460字节， cwnd = 14600字节，则在一个RTT之内可以发送10个报文段，每个ACK到达，拥塞窗口增加1/10MSS，在收到10个报文段的ACK后，拥塞控制窗口增加1个MSS

拥塞如何检测及发生拥塞后的处理

- 收到3个冗余确认后:
  - cwnd减半
  - 窗口再线性增加
- 但是超时事件以后：返回慢启动状态
  - cwnd值设置为1 MSS
  - 窗口再指数增长
  - 到达一个阈值 (Threshold) 后，再线性增长

基本思想:

- 3个冗余ACK指示网络还具有某些传送报文段的能力
- 如果在收到3个冗余ACK之前，就发生超时，则认为拥塞更为严重

3、快速恢复状态

- 当发送方收到来自于接收方3个重复的ACK报文时，进入快速恢复状态；3个重复的*ACK*报文触发的动作包括：**当前的状态可以是慢启动或者拥塞避免**

进入

ssthresh = cwnd/2
cwnd = ssthresh+3*MSS
重传丢失的报文

- 当处于快速恢复状态时（此时网络可能发生拥塞）：
  - 如果收到一个重复的ACK，**cwnd = cwnd + 1 MSS**
  - **当收到一个新的*ACK*（即期望报文的*ACK*），TCP在降低拥塞控制窗口后进入拥塞避免状态； cwnd = ssthresh**
  - **当发生超时事件，说明已经发生拥塞，**

说明有新的分组达到接收方，但不是期望的分组

说明拥塞状态已经缓解

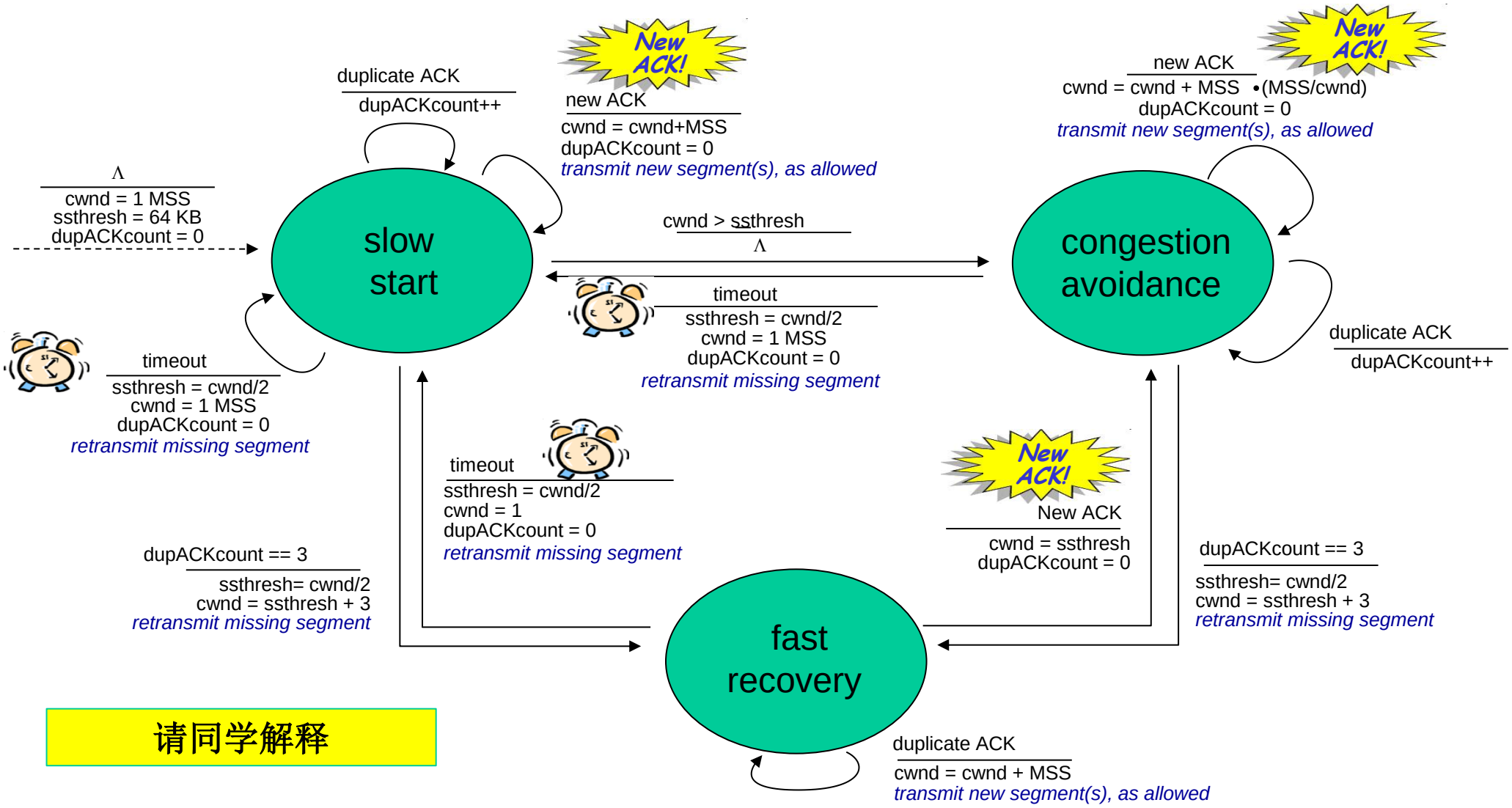
进入慢启动状态

ssthresh = cwnd/2
cwnd = 1 MSS
重传丢失的报文

TCP 拥塞控制：小结

- 当cwnd < ssthresh时，发送者处于慢启动阶段, cwnd指数增长
- 当cwnd >= ssthresh时，发送者处于拥塞避免阶段，cwnd线性增长
- 当出现3个冗余确认时，ssthresh设置为cwnd/2，且cwnd设置为ssthresh+3*MSS
- 当超时发生时， ssthresh设置为cwnd/2，并且cwnd设置为1 MSS.

Summary: TCP Congestion Control



请同学解释

TCP 发送方拥塞控制

状态	事件	TCP发送方拥塞控制动作	注释
慢启动（SS）	收到前面未确认数据的ACK	cwnd = cwnd + MSS, 如果（cwnd>= 慢启动阈值）设置状态为**拥塞避免**	导致每个RTT cwnd翻倍
拥塞避免（CA）	收到前面未确认数据的ACK	cwnd = cwnd + MSS * MSS/ cwnd	加性增，每RTT导致cwnd增加1个MSS
SS或CA	由3个冗余ACK检测到的丢包事件	阈值 = cwnd/2，cwnd = 阈值+3 MSS，设置状态为快速恢复	快速恢复，实现乘性减。CongWin将不低于1个MSS
SS或CA	超时	阈值 = CongWin/2，CongWin =1 MSS，设置状态为“慢启动”	进入慢启动
SS或CA	冗余ACK	对确认的报文段增加冗余ACK计数	CongWin和阈值不改变

课堂练习

P40. Consider Figure 3.58. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.

a. Identify the intervals of time when TCP slow start is operating.

b. Identify the intervals of time when TCP congestion avoidance is operating.

c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

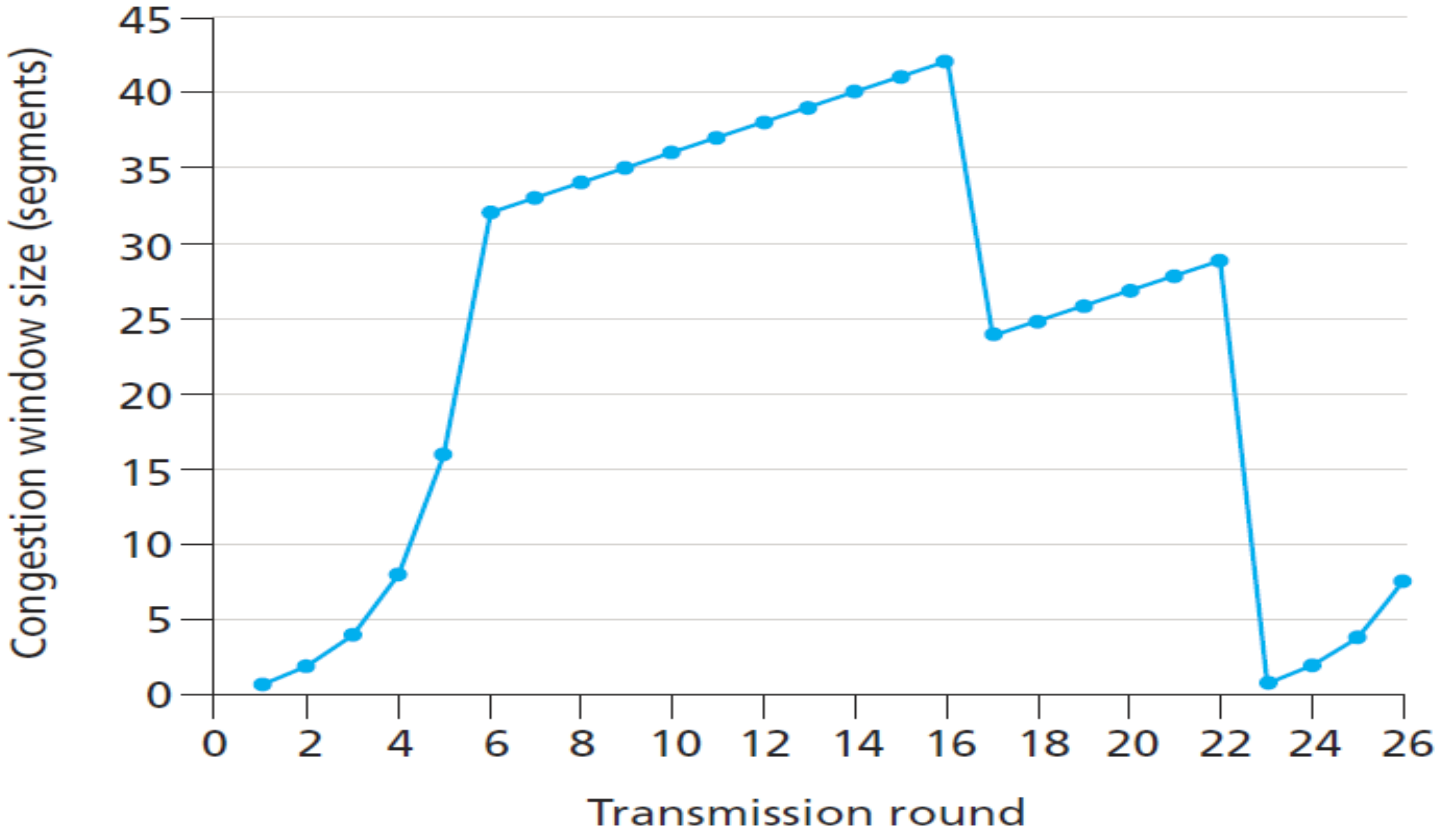
e. What is the initial value of `ssthresh` at the first transmission round?

f. What is the value of `ssthresh` at the 18th transmission round?

g. What is the value of `ssthresh` at the 24th transmission round?

h. During what transmission round is the 70th segment sent?

i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of `ssthresh`?



j. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the `ssthresh` and the congestion window size at the 19th round?

k. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?

5、对TCP吞吐量的宏观描述

- 一个TCP连接的平均吞吐量(即平均速率)可能是多少。
- 在一个特定RTT的往返时间间隔内，作为窗口长度和RTT的函数，TCP的平均吞吐量是什么？
  - 忽略慢启动

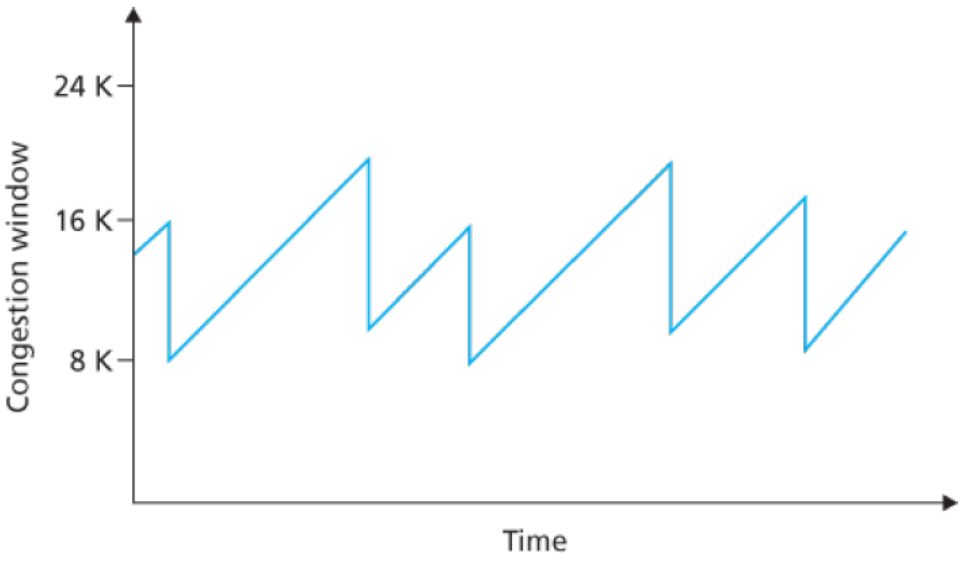
设当丢包发生时窗口长度是W

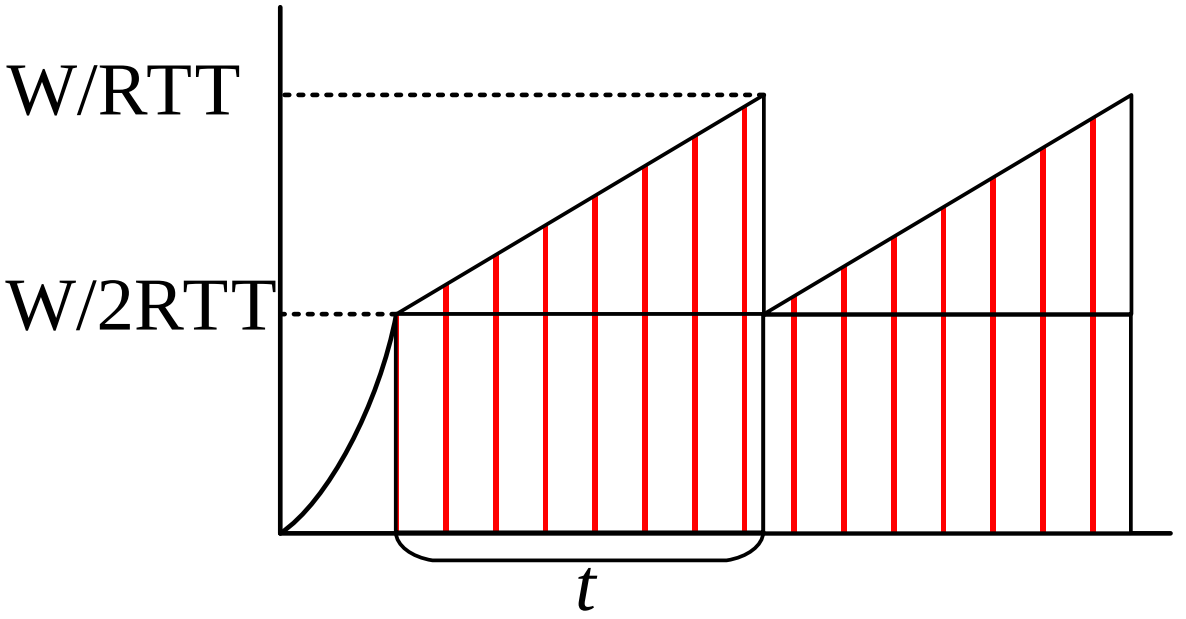
如果窗口为 W，吞吐量是 W/RTT（每RTT可以发送W字节的数据）

当丢包发生后，窗口降为 W/2，吞吐量为 W/2RTT.

一个连接的平均吞吐量为0.75 W/RTT

平均吞吐量的计算





这个梯形就是t时间的流量，平均吞吐量计算如下：

吞吐量=[$\frac{1}{2}$（W/RTT+W/2RTT）*t]/t = 0.75 W/RTT

6、经高带宽路径的TCP

- 举例: 1500 字节的报文段, 100ms RTT, 要达到10 Gbps 的吞吐量
- 要求窗口长度 W = 83,333 个报文段
- 根据丢包率，则一个连接的平均吞吐量为:

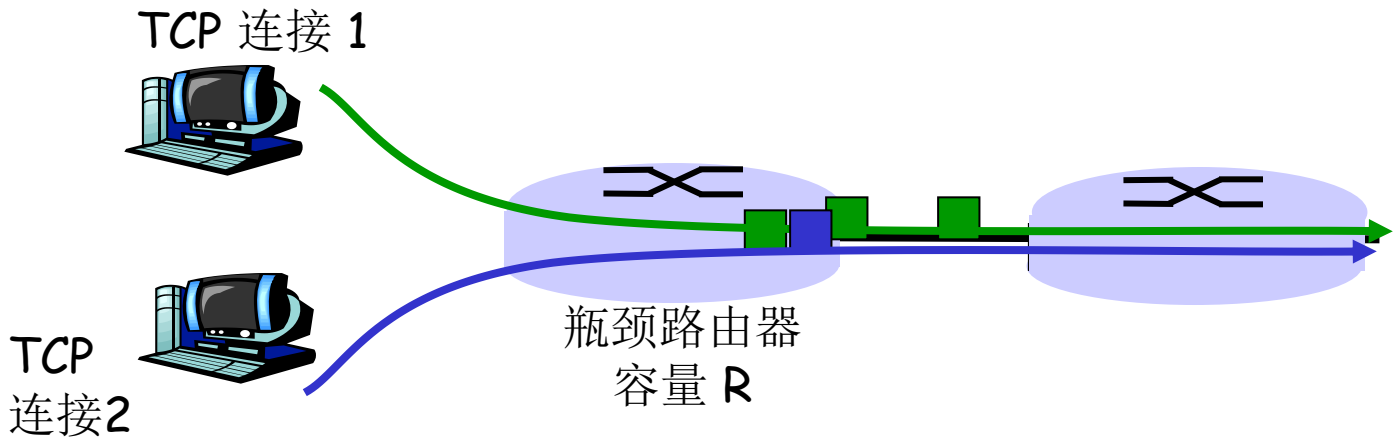
$$T = \frac{1.22 \cdot MSS}{RTT\sqrt{L}} \qquad L = \left(\frac{1.22 \cdot MSS}{RTT * T}\right)^2 = \left(\frac{1.22 * 1500 * 8(bit)}{100(ms) * 10 * 10^9 (bit/s)}\right)^2$$

$$= \left(\frac{14640}{10^9}\right)^2 = (1.464 \times 10^{-5})^2 = 2.14 \times 10^{-5}$$

- 丢包率 $L = 2 \cdot 10^{-10}$ **(难以达到)**
- **需要高速下的TCP新版本![Ha 2008]**

TCP 公平

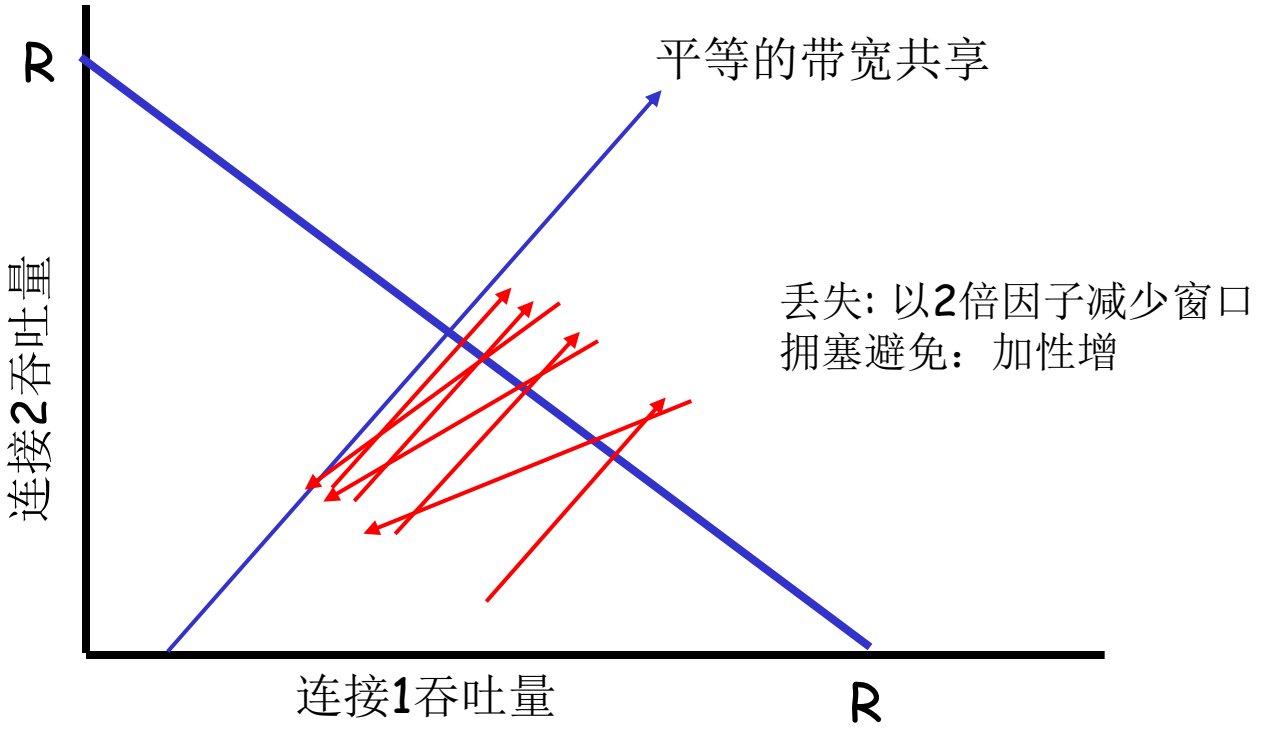
公平目标: 如果K个 TCP 会话 共享带宽为 R的链路瓶颈, 每个会话应有 R/K的平均链路速率



为什么TCP能保证公平性?

两个竞争会话:

- 随着吞吐量的增加，按照斜率1加性增加
- 等比例地乘性降低吞吐量



公平性(续)

公平性和UDP

- 多媒体应用通常不用TCP
  - 不希望拥塞控制抑制速率
- 使用UDP
  - 音频/视频以恒定速率发送, 能容忍报文丢失
- 研究领域:

  TCP友好(TCP friendly)

公平性和并行TCP 连接

- 不能防止2台主机之间打开多个并行连接.
- Web浏览器以这种方式工作
- 例子：支持9个连接的速率R的链路：
  - 新应用请求一个TCP连接, 则得到R/10的带宽。
  - 新应用请求11个TCP连接, 则得到R/2 的带宽!

怎么在UDP的基础上实现与TCP的友好性是曾经的一个热点问题

Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

- 下列关于TCP拥塞控制机制描述错误的是_____。

A. 当TCP连接刚建立时，处于慢启动状态，此时，cwnd的值为1个MSS，每收到1个ACK确认，将cwnd的值增加1个MSS；

B. TCP使用的是端到端拥塞控制机制，而不是网络辅助的拥塞控制机制；

C. 当发送端连续收到3个重复的ACK确认，进入快速恢复状态。

D. 当cwnd的值大于慢启动阈值ssthresh时，进入拥塞避免状态，在该状态，每收到1个ACK确认，将cwnd的值增加1个MSS。

- 下列哪项不是TCP协议的特性_____。

A. 提供可靠服务　　B. 提供无连接服务

C. 提供端到端服务　　D. 提供全双工服务

实验2—思考

任务1(研究TCP)：在这个实验中，同学们将使用web浏览器访问来自某Web服务器的一个文件。能够从该web服务器下载一个Wireshark可读的分组踪迹，记载你从服务器下载文件的过程。在这个服务器踪迹文件里，你将发现并访问该web服务器所产生的分组。你将分析客户端和服务器端踪迹文件，以探索TCP协议的各种细节，特别是将评估在你的计算机与web服务器之间TCP连接的性能。你将跟踪TCP窗口行为、推断分组丢失、重传、流控、拥塞控制行为并估计往返时间。

任务2 (研究UDP)：在这个实验中，同学们将进行分组捕获并分析那些使用UDP的你喜爱的应用程序(例如，DNS或skype这样的多媒体应用)。如我们在3.3节中所学的那样，UDP是一种简单的、不提供不必要服务的运输协议。在这个实验中，将研究在UDP报文段中的各首部字段以及校验和计算。