第八章

端到端访问与传送层

主要内容

- **传送服务**
- **传送层寻址**
- **连接管理**
- **缓存和流控**
- **互联网传送协议**
  - TCP协议
  - UDP协议

传送服务

- **引入传送层的原因**
  - 消除网络层的不可靠性
  - 提供从源端主机的进程到目的端主机的进程的可靠的、与实际使用的网络无关的数据传送
- **传送服务**
  - 传送实体（transport entity）：完成传送层功能的硬软件
  - 传送层实体利用网络层提供的服务向上层提供有效、可靠或尽力而为的服务

传送服务（续）

- 传送层提供两种服务
  - 面向连接的传送服务：连接建立，数据传送，连接释放
  - 无连接的传送服务
- 1 ~ 4层称为传送服务提供者（transport service provider）
- 4层以上称为传送服务用户（transport service user）

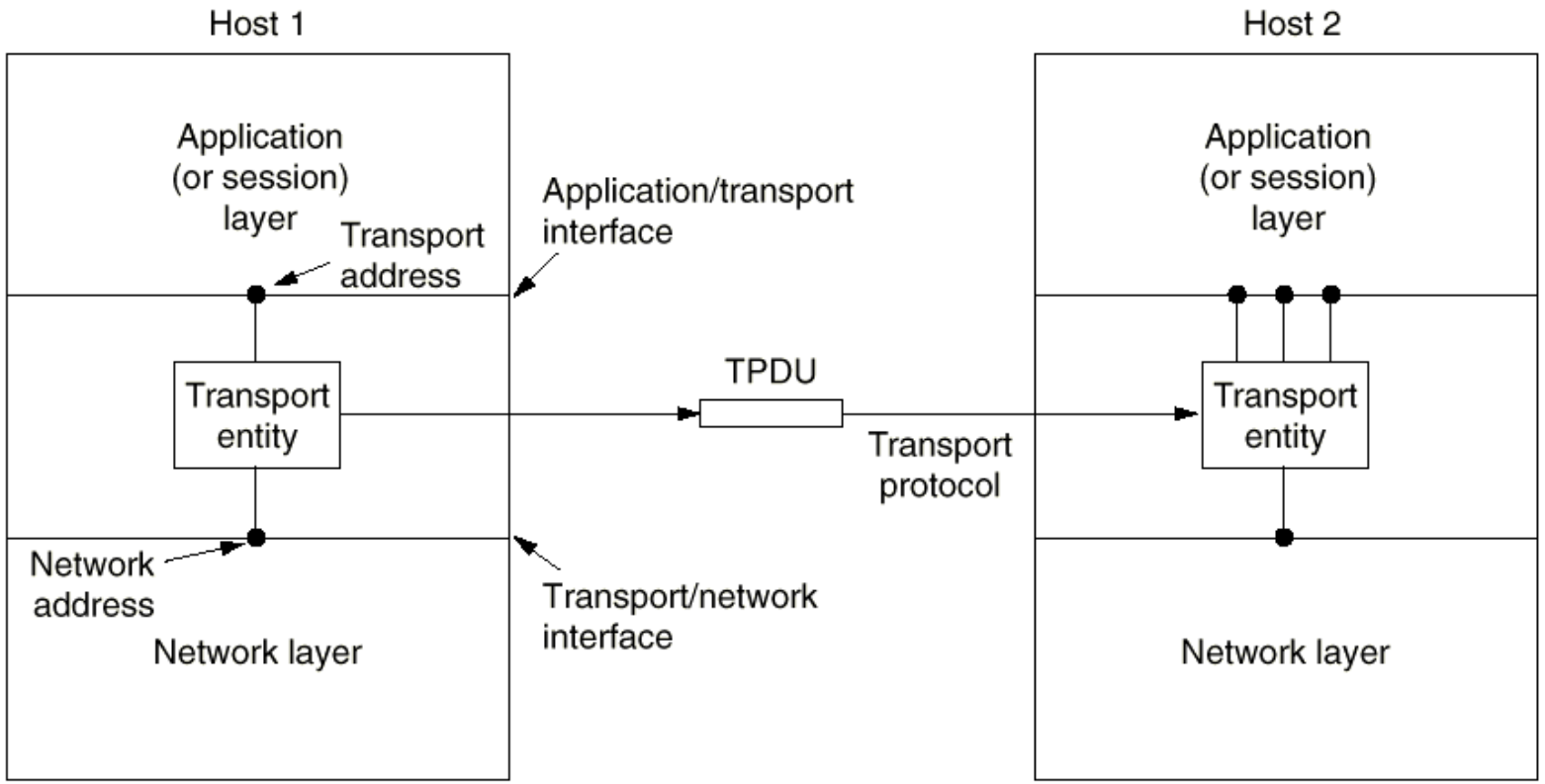


Fig. 6-1. The network, transport, and application layers.

传送服务原语

- **传送服务原语（Transport Service Primitives）**
  - 传送用户（应用程序）通过传送服务原语访问传送服务
  - 一个简单传送服务的原语
    - Client/Server 模式

Primitive	TPDU sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA TPDU arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Fig. 6-3. The primitives for a simple transport service.

TPDU

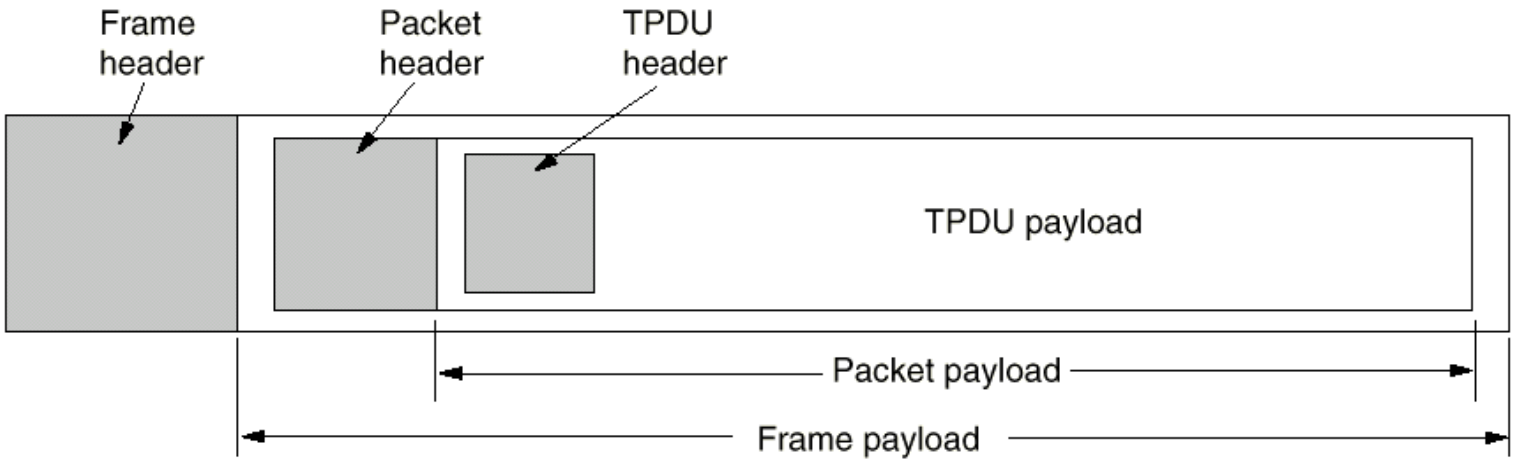


Fig. 6-4. Nesting of TPDUs, packets, and frames.

简单连接管理

- **拆除连接方式有两种**
  - 不对称方式：任何一方都可以关闭双向连接
  - 对称方式：每个方向的连接单独关闭，双方都执行 DISCONNECT 才能关闭整条连接
- **简单连接管理状态图**

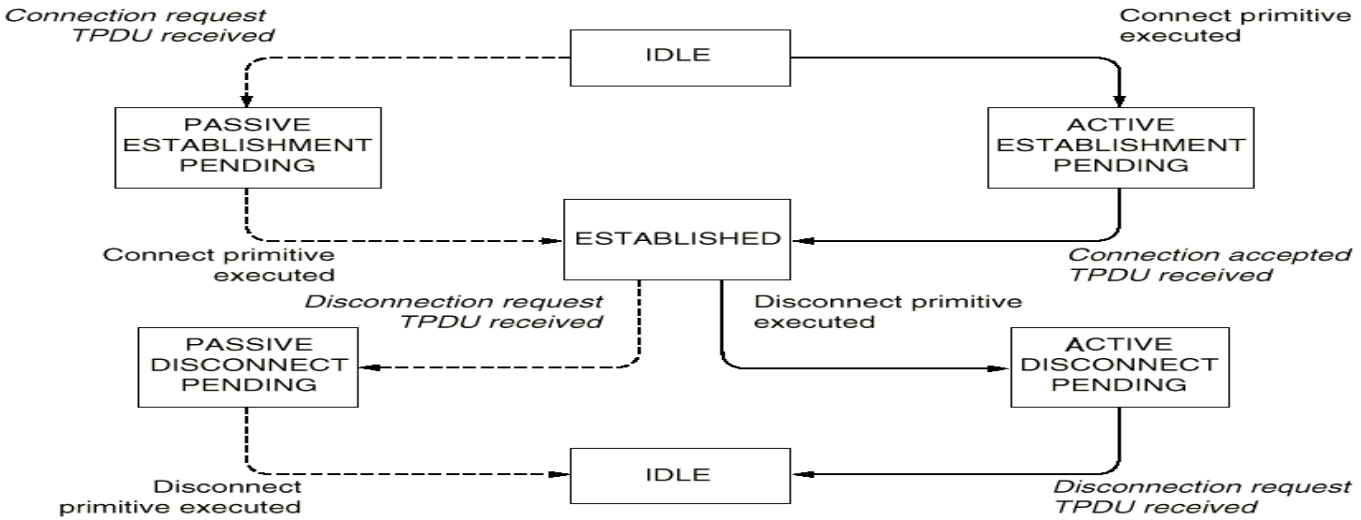


Fig. 6-5. A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Berkeley Sockets

- **Berkeley Sockets**
  - 连接释放是对称的

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Fig. 6-6. The socket primitives for TCP.

Berkeley Sockets（续）

- **应用举例**
  - 一个服务程序和几个远程客户程序利用面向连接的传送层服务完成通信
    - 建立连接
    - 数据传送
    - 释放连接

Berkeley Sockets（续）

- **建立连接**
  - 服务程序
    - 调用 socket 创建一个新的套接字，并在传送层实体中分配表空间，返回一个文件描述符用于以后调用中使用该套接字
    - 调用 bind 将一个地址赋予该套接字，使得远程客户程序能访问该服务程序
    - 调用 listen 分配数据空间，以便存储多个用户的连接建立请求
    - 调用 accept 将服务程序阻塞起来，等待接收客户程序发来的连接请求。
      - 当传送层实体接收到建立连接的 TPDU 时，新创建一个和原来的套接字相同属性的套接字并返回其文件描述符。
      - 服务程序创建一个子进程处理此次连接，然后继续等待发往原来套接字的连接请求

Berkeley Sockets（续）

- **建立连接（续）**
  - 客户程序
    - 调用socket创建一个新的套接字，并在传送层实体中分配表空间，返回一个文件描述符用于在以后的调用中使用该套接字
    - 调用connect阻塞客户程序，传送层实体开始建立连接，当连接建立完成时，取消阻塞

Berkeley Sockets（续）

- **数据传送**
  - 双方使用 send 和 receive 完成数据的全双工发送
- **释放连接**
  - 每一方使用 close 原语单独释放连接

传送层编址

- **编址（Addressing）**
  - 方法：标识传送服务访问点 TSAP（Transport Service Access Point），将应用进程与这些 TSAP 相连。
  - 在互联网中，TSAP 标识为 (IP address, local port)
- **远方客户程序如何获得服务程序的TSAP?**
  - 方法1：预先约定、广为人知的，例如 telnet 是（IP地址，端口23）
  - 方法2：client/server 为同一开发者，预先约定
  - 方法3：从名称服务器（name server）或目录服务器（directory server）获得 TSAP

名称服务器

- **一个特殊的进程称为名称服务器或目录服务器（TSAP 众所周知）**
- **协议工作过程**
  - 客户进程与名称服务器建立连接，发送服务名称，获得服务进程的 TSAP，释放与名称服务器的连接
  - 客户进程与服务进程建立连接

进程服务器

- **当服务程序很多时，可以使用进程服务器（process server）**
  - 进程服务器是一个同时在多个端口上监听的进程（例如：inetd）
  - 客户进程向它实际想访问的服务进程的 TSAP 发出连接建立请求
  - 如果没有服务进程在此 TSAP 上监听，则客户进程和进程服务器建立连接
  - 进程服务器创建所请求的服务进程，并使该进程继承和客户进程的连接
  - 进程服务器返回继续监听
  - 客户进程与所希望的服务进程进行数据传送

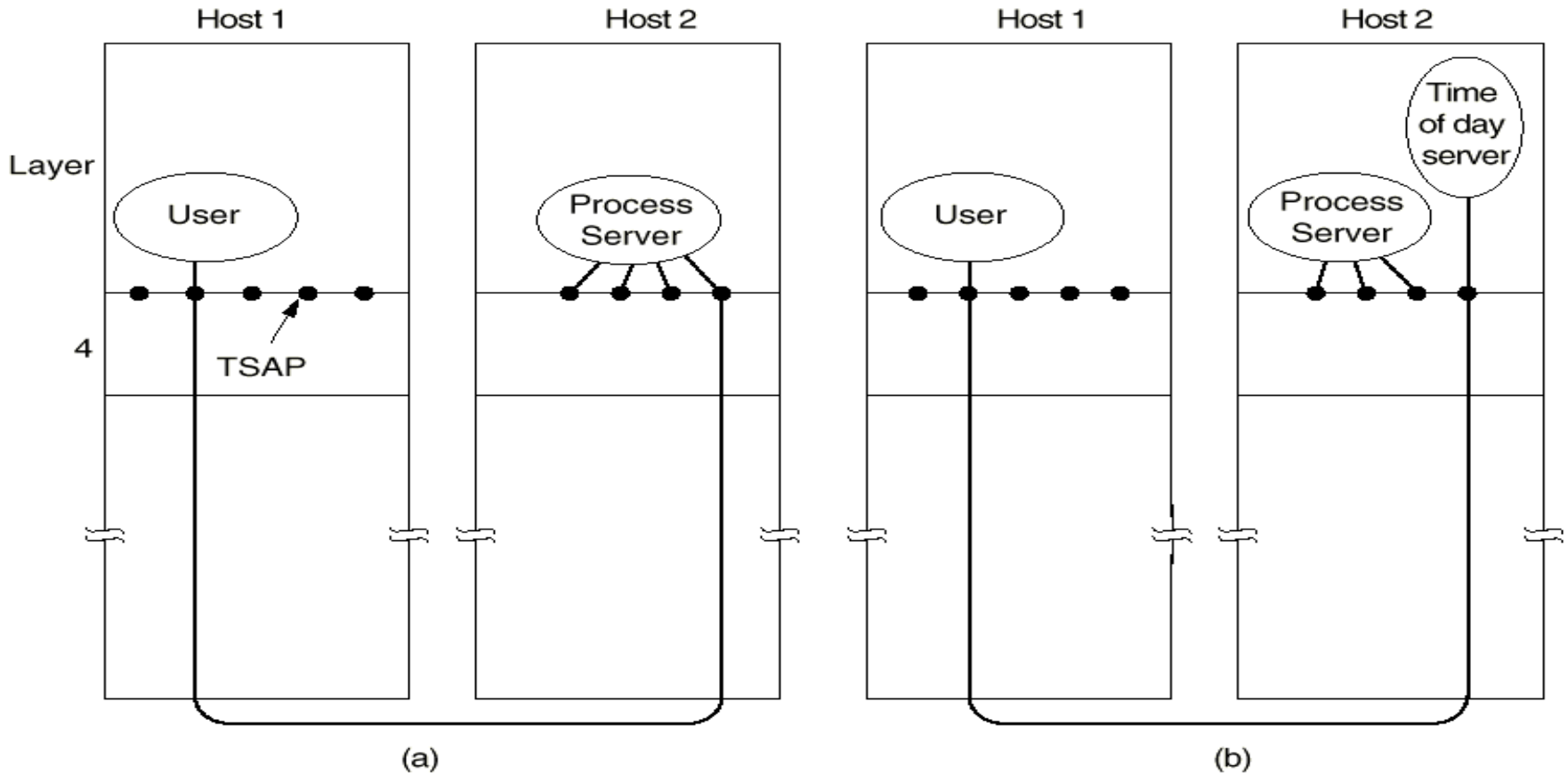


Fig. 6-9. How a user process in host 1 establishes a connection with a time-of-day server in host 2.

连接管理

- **建立连接**
  - 网络可能丢失、重复包，特别是延迟重复包的存在，导致传送层建立连接的复杂性
  - 两次握手不能满足要求
    - 两次握手：A 发出连接请求 CR TPDU，B 发回连接确认 CC TPDU
    - 失败的原因：网络层会丢失或缓存连接请求 TPDU，产生重复包
  - 解决延迟重复包的关键是丢弃过时的包
  - 采用三次握手

三次握手

- **三次握手（three-way handshake）**
  - A 发出序号为 X 的 CR TPDU
  - B 发出序号为 Y 的 CC TPDU，并确认 A 的序号为 X 的 CR TPDU
  - A 发出序号为 X+1 的第一个数据 TPDU，并确认 B 的序号为 Y 的 CR TPDU
  - Fig. 6-11 （DATA 序号应为 X+1）
- **三次握手方案解决了由于网络层会丢失、存储和重复包带来的问题**

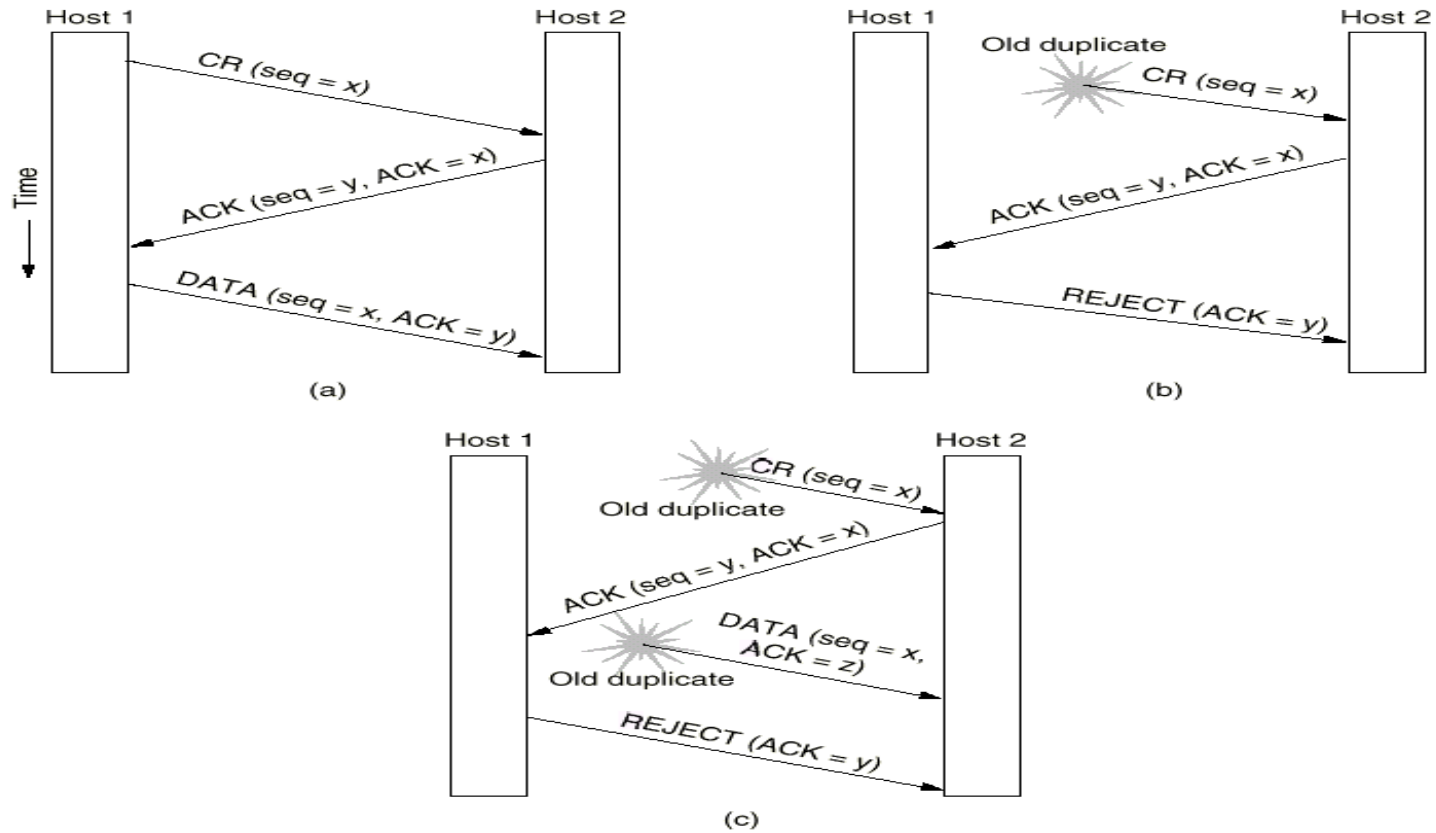


Fig. 6-11. Three protocol scenarios for establishing a connection using a three-way handshake. CR and ACC denote CONNECTION REQUEST and CONNECTION ACCEPTED, respectively. (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK.

连接管理（续）

- **释放连接**
  - 非对称式：一方释放连接，双向连接断开，存在丢失数据的危险
  - 对称式：每个方向的连接单独关闭，不会丢失数据，但是存在两军问题

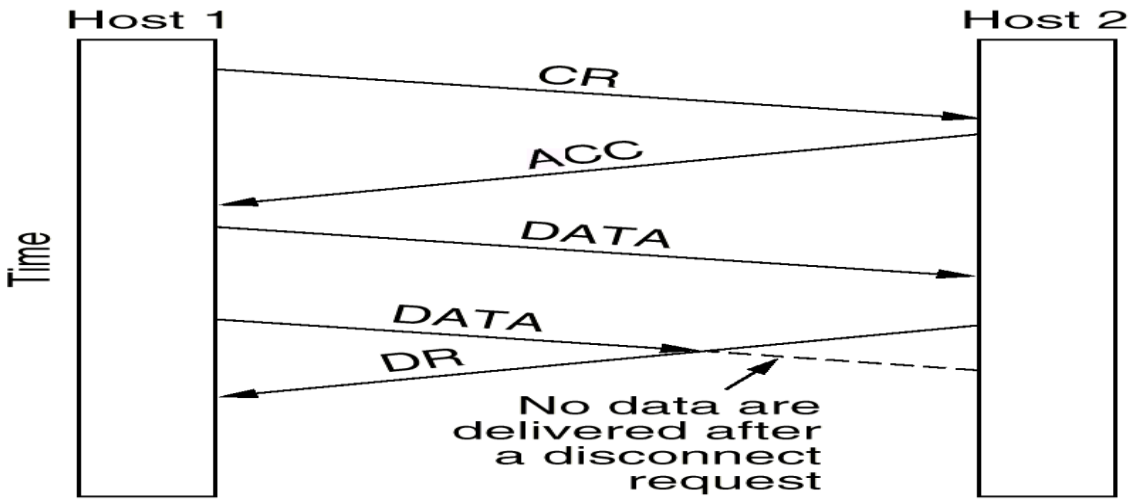


Fig. 6-12. Abrupt disconnection with loss of data.

两军问题

- 由于存在两军问题（two-army problem），可以证明不存在安全的通过 N 次握手实现对称式连接释放的方法

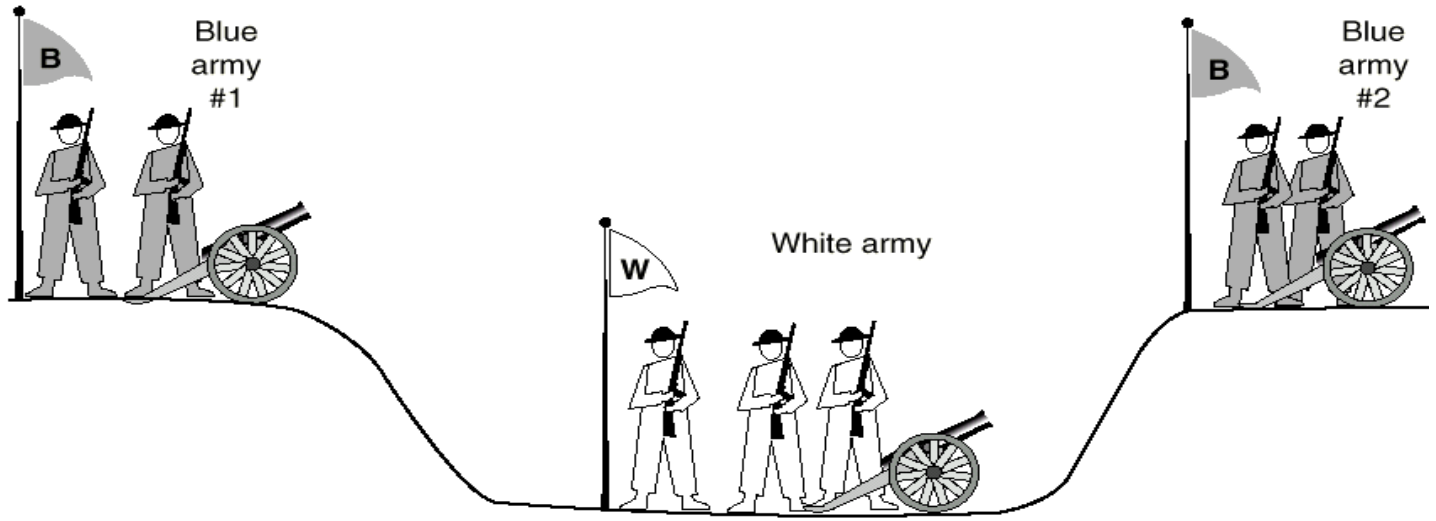


Fig. 6-13. The two-army problem.

连接管理（续）

- **但是在实际的通信过程中，使用三次握手 + 定时器的方法释放连接在绝大多数情况下是成功的。**

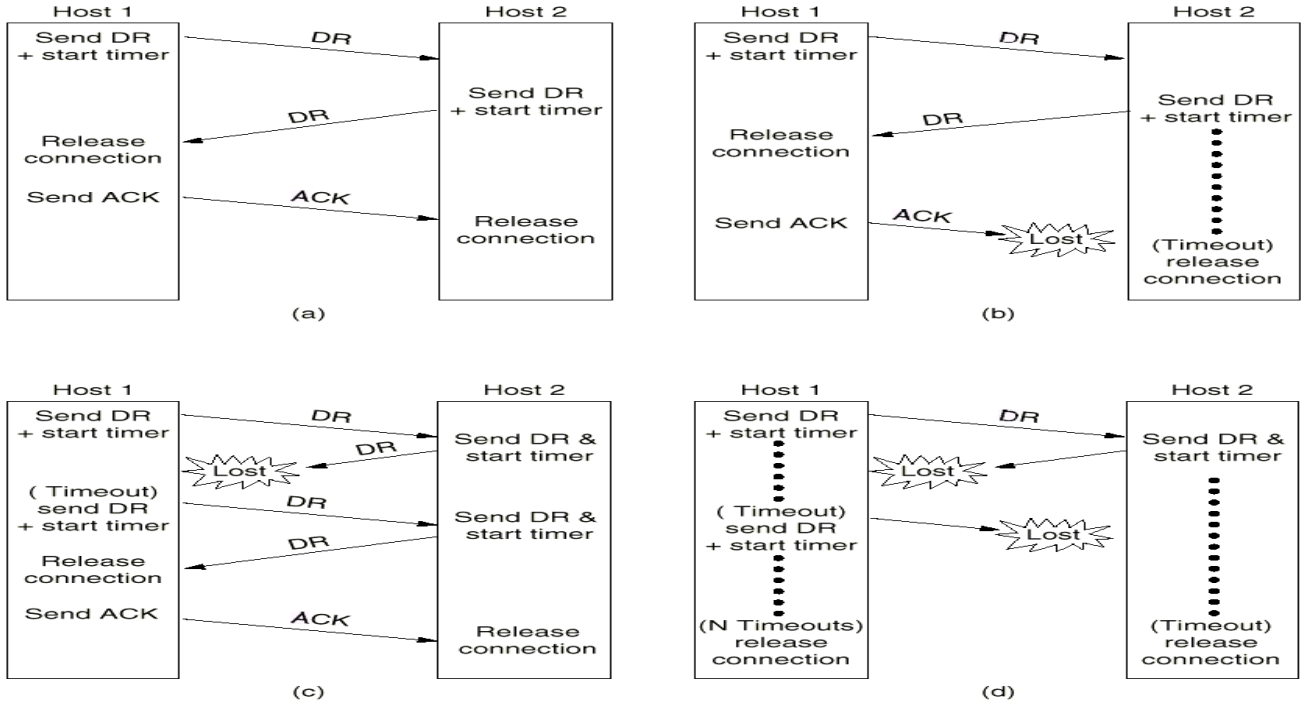


Fig. 6-14. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.

缓存

- **由于网络层服务是不可靠的，传送层实体必须缓存所有连接发出的TPDU，而且为每个连接单独做缓存，以便用于错误情况下的重传**
- **接收方的传送层实体可以做也可以不做缓存**
- **缓存的设计有三种**
  - 固定大小缓存
  - 可变大小缓存
  - 缓存池

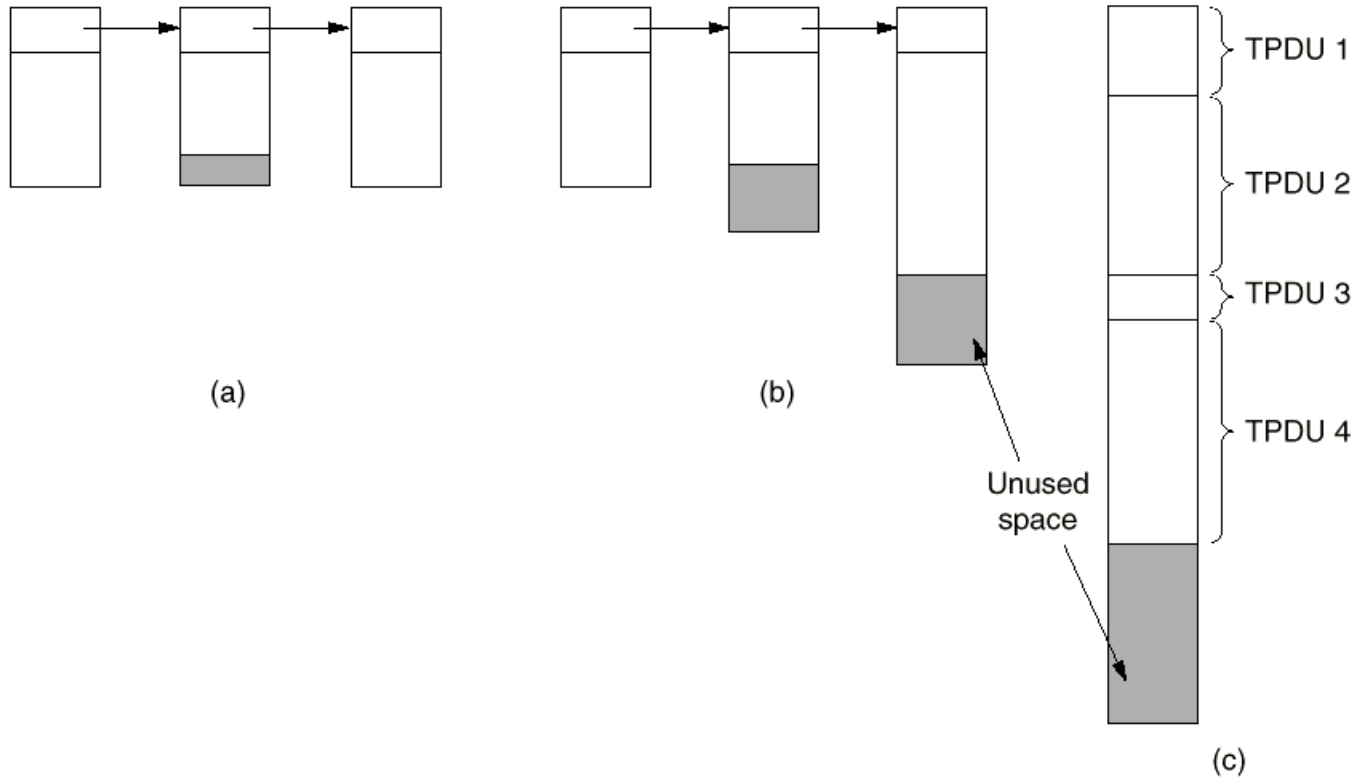


Fig. 6-15. (a) Chained fixed-size buffers. (b) Chained variable-size buffers. (c) One large circular buffer per connection.

流控

- **流控：Flow Control**
- **传送层利用可变滑动窗口协议来实现流控**
  - 可变滑动窗口协议：是指发送方的发送窗口大小是由接收方根据自己的实际缓存情况给出的
- **为了避免控制 TPDU 丢失导致死锁，端系统应该周期性地发送 TPDU**

	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	•••	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	•••	<ack = 6, buf = 4>	←	Potential deadlock

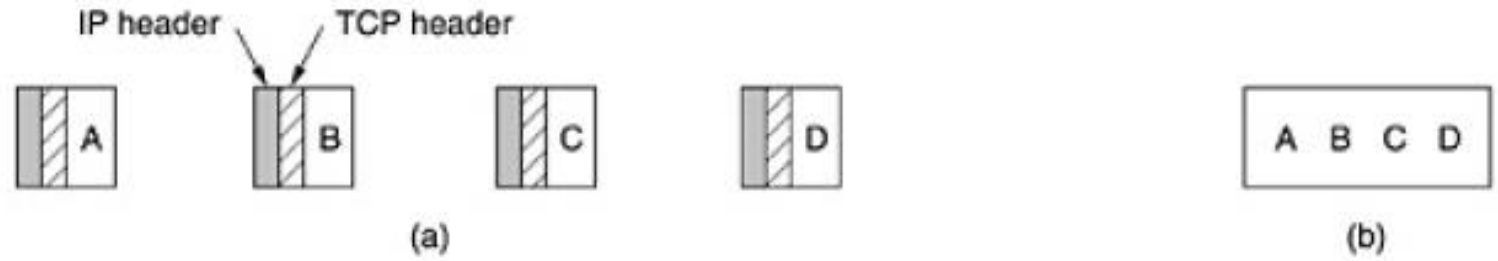
Fig. 6-16. Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU.

互联网传送协议

- **传送控制协议 TCP（Transmission Control Protocol）**
  - 面向连接的、可靠的、端到端的、基于字节流的传送协议
  - RFC 793，1122，1323，2018，2581等
- **用户数据协议 UDP（User Data Protocol）**
  - 无连接的端到端传送协议
  - RFC 768

TCP 协议

- **应用程序访问 TCP 服务是通过在收发双方创建套接字来实现的**
  - 套接字地址用（IP地址，端口号）来表示
    - 256 以下的端口号被标准服务保留，例如 FTP（21），TELNET（23）
  - 每条连接用(套接字1,套接字2)来表示，是点到点的全双工通道
- **TCP 不支持组播（multicast）和广播（broadcast）**
- **TCP 连接是基于字节流的，而非消息流，消息的边界在端到端的传送中不能得到保留**



TCP 协议（续）

- 对于应用程序发来的数据，TCP 可以立即发送，也可以缓存一段时间以便一次发送更多的数据。为了强迫数据发送，可以使用 PUSH 标记
- 对于紧急数据，可以使用 URGENT 标记
- 按字节分配序号，每个字节有一个 32 位的序号
- 传送实体之间使用的 TPDU 称为段（segment）
- 每个段包含一个 20 字节的头（选项部分另加）和 0 个或多个数据字节。
- 段的大小必须满足 65535 字节的 IP 包数据净荷长度限制，还要满足数据链路层最大传送单元（MTU）的限制，例如以太网的 MTU 为 1500 字节
- TCP 实体使用滑动窗口协议，确认序号等于接收方希望接收的下一个字节序号

TCP 协议（续）

- **TCP协议需要解决的主要问题**
  - 连接管理
    - 建立连接：三次握手
    - 释放连接：三次握手 + 定时器
  - 可靠传送
    - 滑动窗口
  - 流控制和拥塞控制
    - 可变滑动窗口
    - 慢启动（slow start）、拥塞避免（congestion avoidance）…

TCP 头

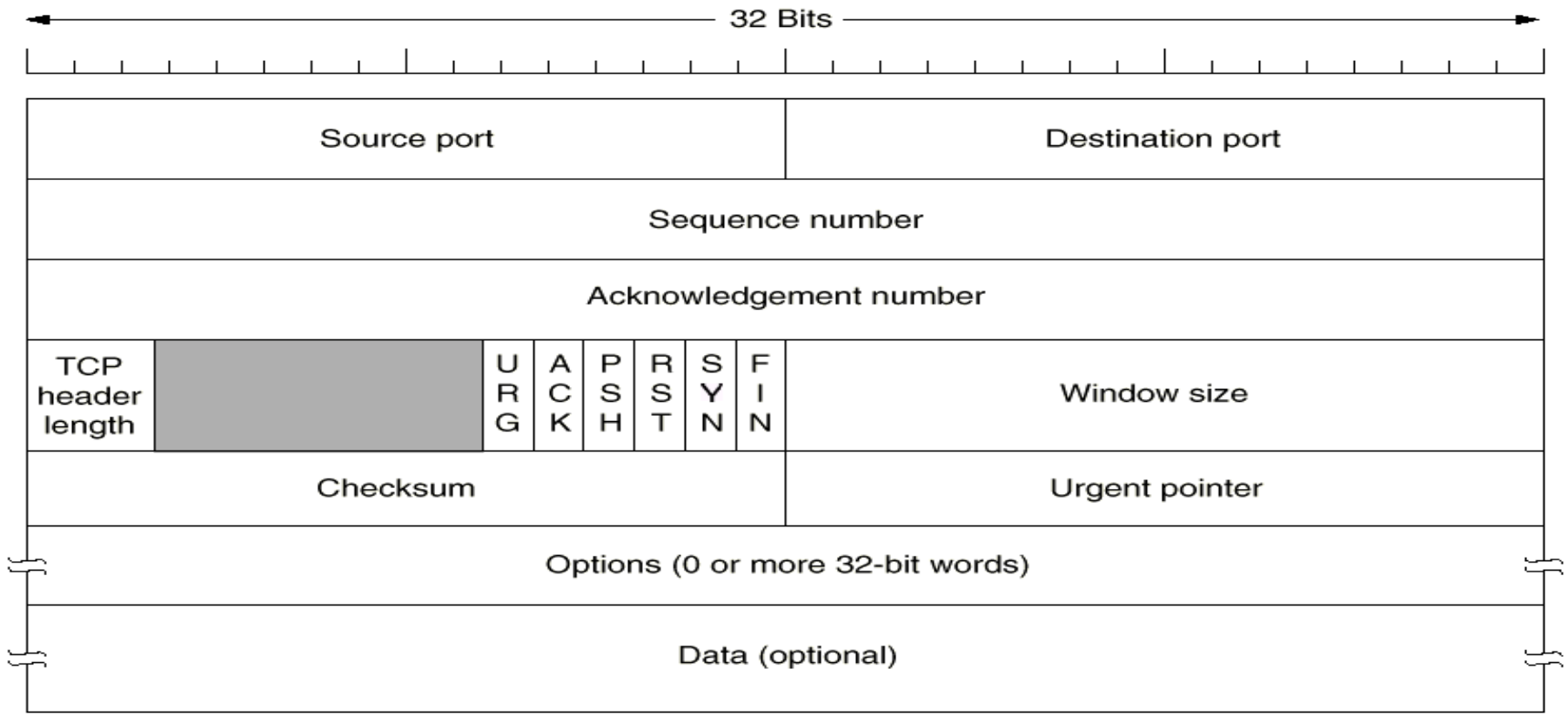


Fig. 6-24. The TCP header.

TCP 头（续）

- **源端口和目的端口：**各16位
- **序号和确认号：**以字节为单位编号，各32位
- **TCP头的长度：**4位，长度单位为32位字，包含选项域
- **6位的保留域**
- **6位的标识位：置1表示有效**
  - URG：和紧急指针配合使用，发送窗口为零时仍可发送紧急数据
  - ACK：确认号是否有效
  - PSH：指示发送方和接收方将数据不做缓存，立刻发送或接收
  - RST：由于不可恢复的错误重置连接
  - SYN：用于连接建立指示
  - FIN：用于连接释放指示

TCP 头（续）

- **窗口大小：**用于基于可变滑动窗口的流控，指示发送方从确认号开始可以再发送窗口大小的字节流
- **校验和：**为增加可靠性，对TCP头，数据和伪头计算校验和
- **选项域**

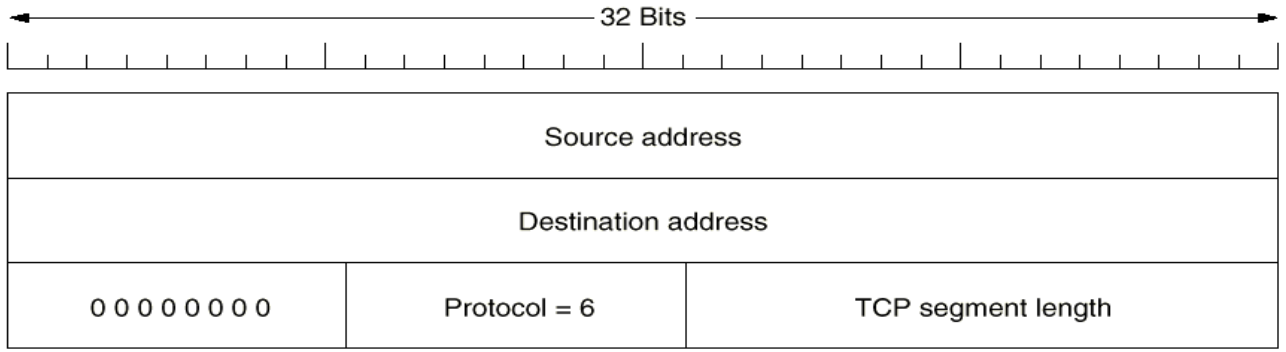


Fig. 6-25. The pseudoheader included in the TCP checksum.

TCP 连接管理

- **三次握手建立连接**
  - 服务器方执行 LISTEN 和 ACCEPT 原语，被动监听
  - 客户方执行 connect 原语，产生一个 SYN 为 1 和 ACK 为 0 的 TCP 段，表示连接请求
  - 服务器方的传送实体接收到这个 TCP 段后，首先检查是否有服务进程在所请求的端口上监听，若没有，回答 RST 置位的 TCP 段
  - 若有服务进程在所请求的端口上监听，该服务进程可以决定是否接受该请求。在接受后，发出一个SYN置1和ACK置1的TCP段表示连接确认，并请求与对方的连接
  - 客户方收到确认后，发出一个SYN置0和ACK置1的TCP段表示给对方的连接确认
  - 若两个主机同时试图建立彼此间的连接，则只能建立一条连接

TCP 连接管理（续）

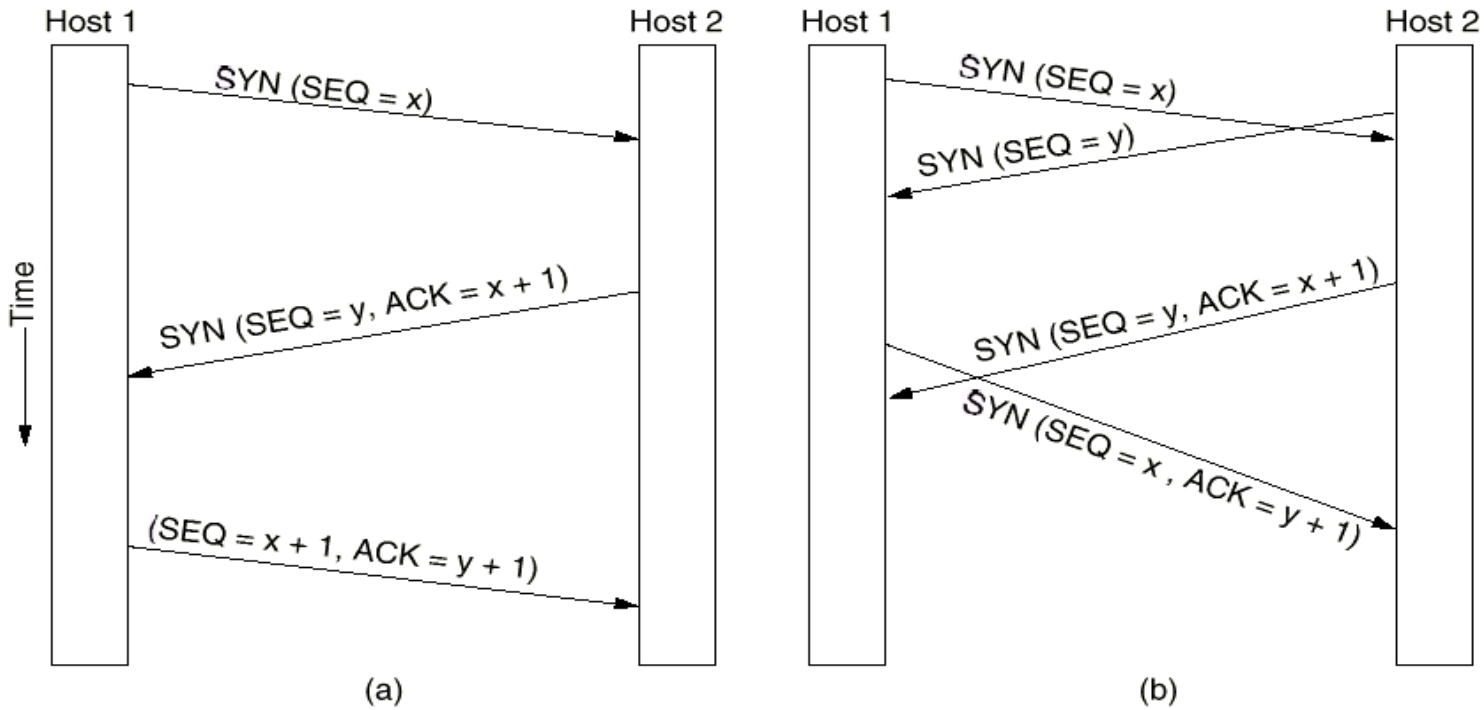


Fig. 6-26. (a) TCP connection establishment in the normal case. (b) Call collision.

TCP 连接管理（续）

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN> -->	SYN-RECEIVED
3. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK> <--	SYN-RECEIVED
4. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK> -->	ESTABLISHED
5. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK><DATA> -->	ESTABLISHED

Basic 3-Way Handshake for Connection Synchronization

Note that the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

TCP 连接管理（续）

- **对称方式连接释放**

- **释放连接时，发出 FIN 位置 1 的 TCP 段并启动定时器，在收到确认后关闭连接。若无确认并且超时，也关闭连接**

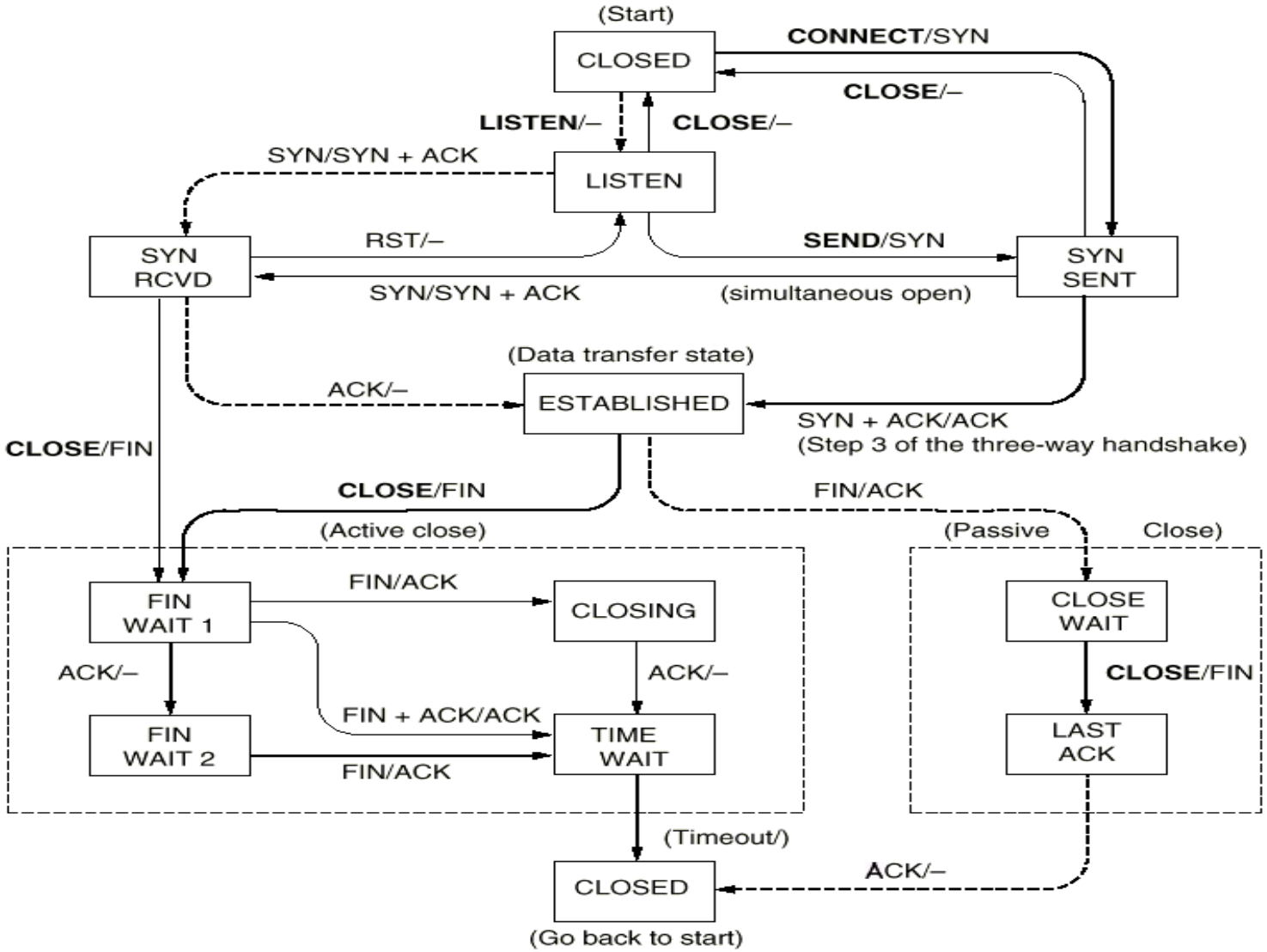


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

TCP 连接管理有限状态机

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Fig. 6-27. The states used in the TCP connection management finite state machine.

TCP 传送策略

- **TCP的窗口管理机制**
  - 基于确认和可变窗口大小
  - 窗口大小为 0 时，正常情况下，发送方不能再发 TCP 段，但有两个例外：
    - 紧急数据可以发送
    - 为防止死锁，发送方可以发送 1 字节的 TCP 段，以便让接收方重新声明确认号和窗口大小

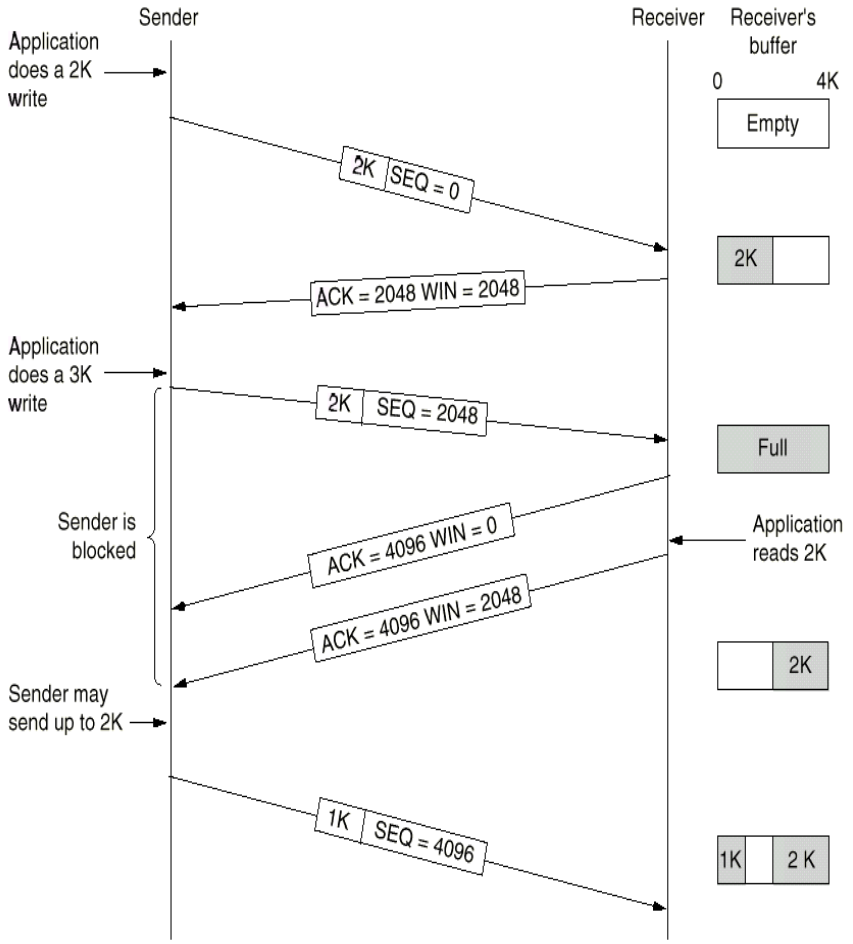


Fig. 6-29. Window management in TCP.

改进传送层的性能

- **策略1**：发送方缓存应用程序的数据，等到形成一个比较大的段再发出
- **策略2**：在没有可能进行“捎带”的情况下，接收方延迟发送确认段
- **策略3**：使用 Nagle 算法
  - 当应用程序每次向传送实体发出一个字节时，传送实体发出第一个字节并缓存所有其后的字节直至收到对第一个字节的确认
  - 然后将已缓存的所有字节组段发出并对再收到的字节缓存，直至收到下一个确认

改进传送层的性能（续）

- **策略4：**使用 Clark 算法解决傻窗口症状（silly window syndrome）
  - 傻窗口症状：当应用程序一次从传送层实体读出一个字节时，传送层实体会产生一个一字节的窗口更新段，使得发送方只能发送一个字节
  - 解决办法：限制收方只有在具备一半的空缓存或最大段长的空缓存时，才产生一个窗口更新段

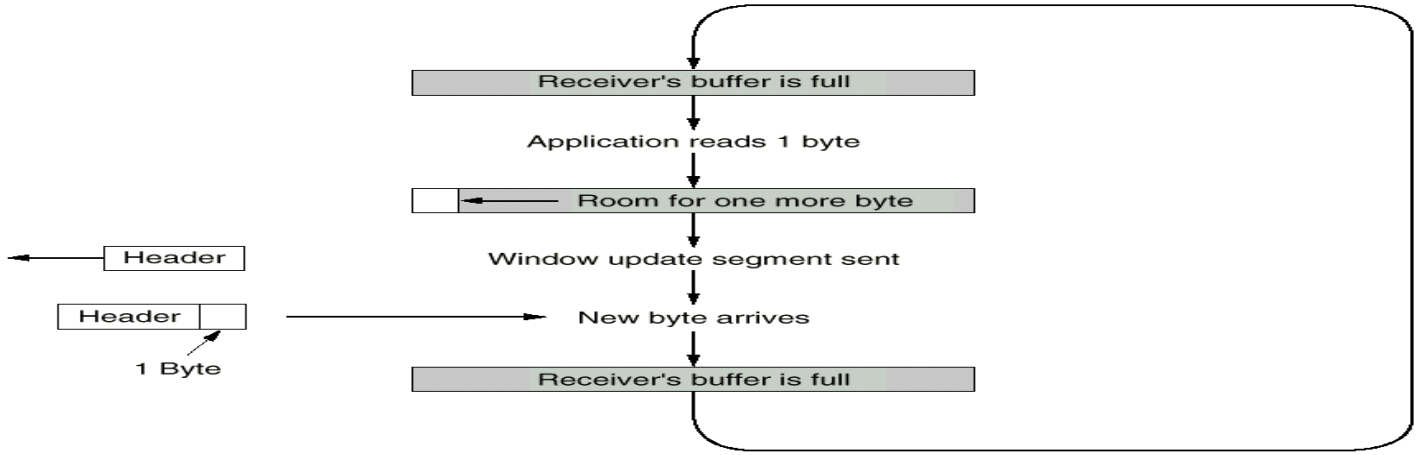


Fig. 6-30. Silly window syndrome.

TCP 拥塞控制

- **导致拥塞的两个潜在因素是：**网络转发能力和接收方接收能力
- **出现拥塞的两种情况**
  - 快网络小缓存接收者
  - 慢网络大缓存接收者

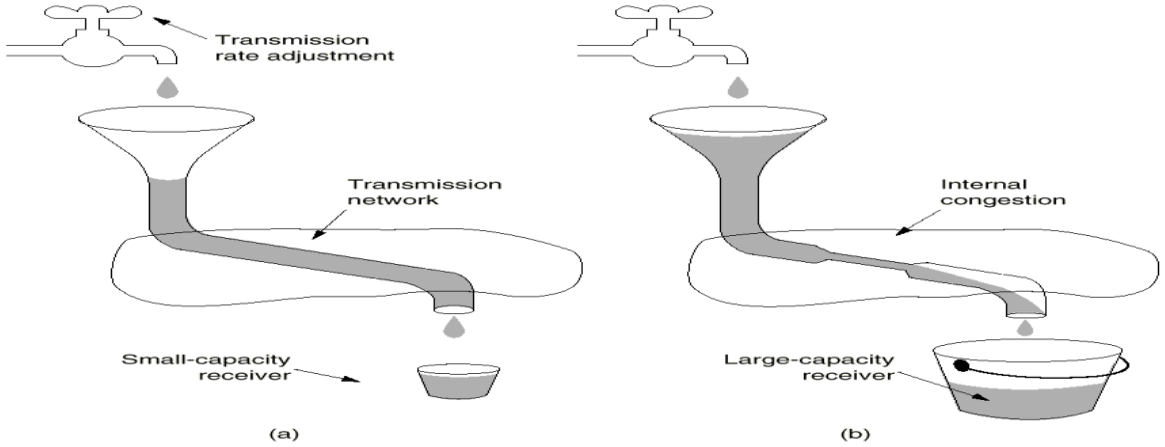


Fig. 6-31. (a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

TCP 拥塞控制（续）

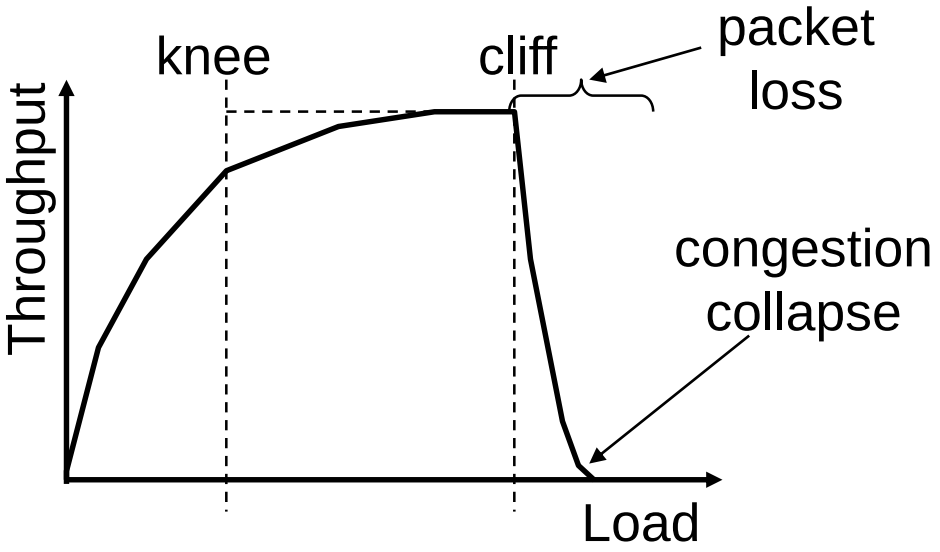
- **TCP 处理第一种拥塞的措施**
  - 采用可变滑动窗口
  - 在确认中声明最大可接收窗口
- **TCP 处理第二种拥塞的措施**
  - 采用拥塞窗口
  - 拥塞窗口按照慢启动（slow start）和拥塞避免（congestion avoidance）等算法变化
- **TCP 按两个窗口的最小值发送**

TCP拥塞控制（续）

- **1988年以前的TCP**
  - 采用 Go-back-N 滑动窗口
  - 只有流控制，没有拥塞控制
- **1986年互联网遭受第一次严重拥塞**
  - Link LBL to UC Berkeley
    - 400 yards, 3 hops, 32 Kbps
    - throughput dropped to 40 bps
    - factor of ~1000 drop!
- **1988, Van Jacobson 提出 TCP 拥塞控制机制**
  - Slow Start，Congestion Avoidance，Fast Retransmit

慢启动算法

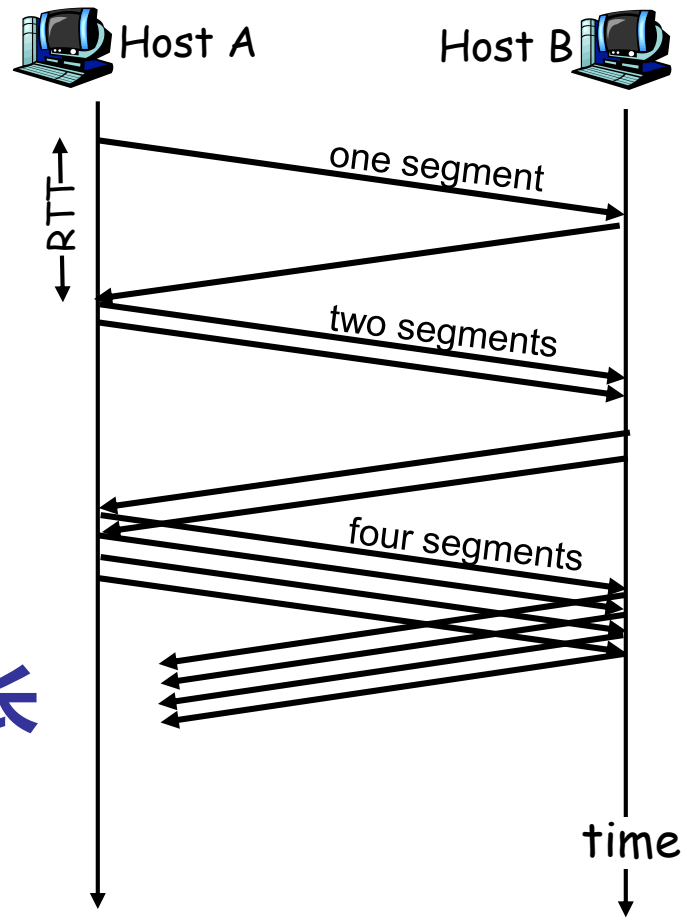
- 连接建立时拥塞窗口（congwin）初始值为该连接允许的最大发送段长 MSS，阈值（threshold）为 64K
- 发出一个最大段长的 TCP 段，若收到正确确认，则拥塞窗口变为两个最大段长
- 发出（ 拥塞窗口/最大段长）个最大长度的 TCP 段，若都得到确认，则拥塞窗口加倍
- 重复上一步，直至发生丢包产生的超时事件，或拥塞窗口大于阈值



慢启动算法（续）

慢启动算法

initialize: Congwin = 1
for (each segment ACKed)
    Congwin++
until (loss event OR
    CongWin ≥ threshold)



- **窗口大小在每个 RTT 周期内指数增长**

拥塞避免算法

- 当拥塞窗口大于阈值时，拥塞窗口开始线性增长，一个 RTT 周期增加一个最大段长，直至发生丢包产生的超时事件
- 超时事件发生后，阈值设置为当前拥塞窗口大小的一半，拥塞窗口重新设置为一个最大段长
- 执行慢启动算法

拥塞避免算法（续）

Congestion avoidance

```
/* slowstart is over        */
/* Congwin ≥ threshold */
Until (loss event) {
  every w segments ACKed:
      Congwin++
  }
threshold = Congwin/2
Congwin = 1
perform slowstart
```

w is the current value of the congestion window, and w is larger than threshold. After w acknowledgments have arrived, TCP replaces w with w+1.

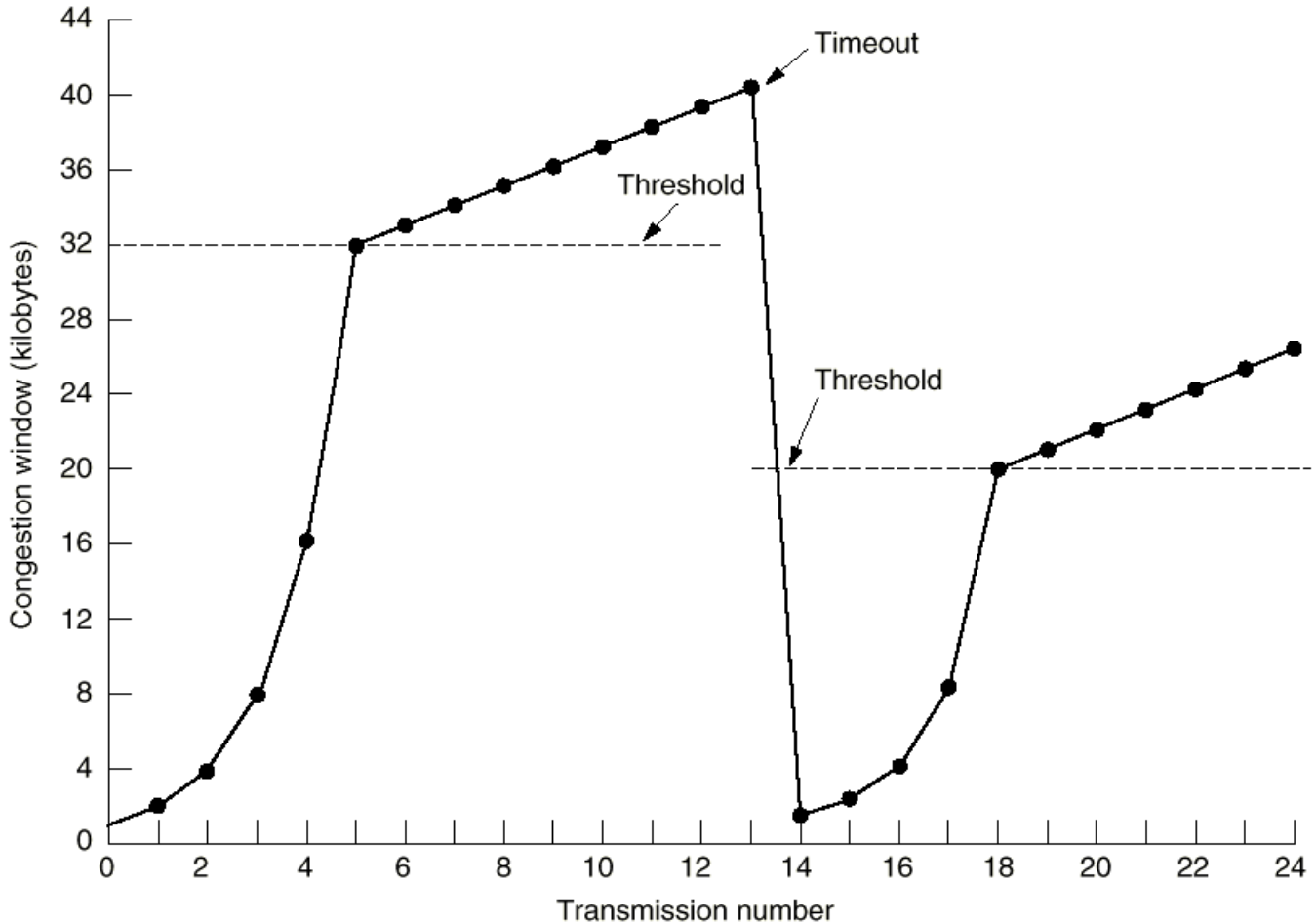
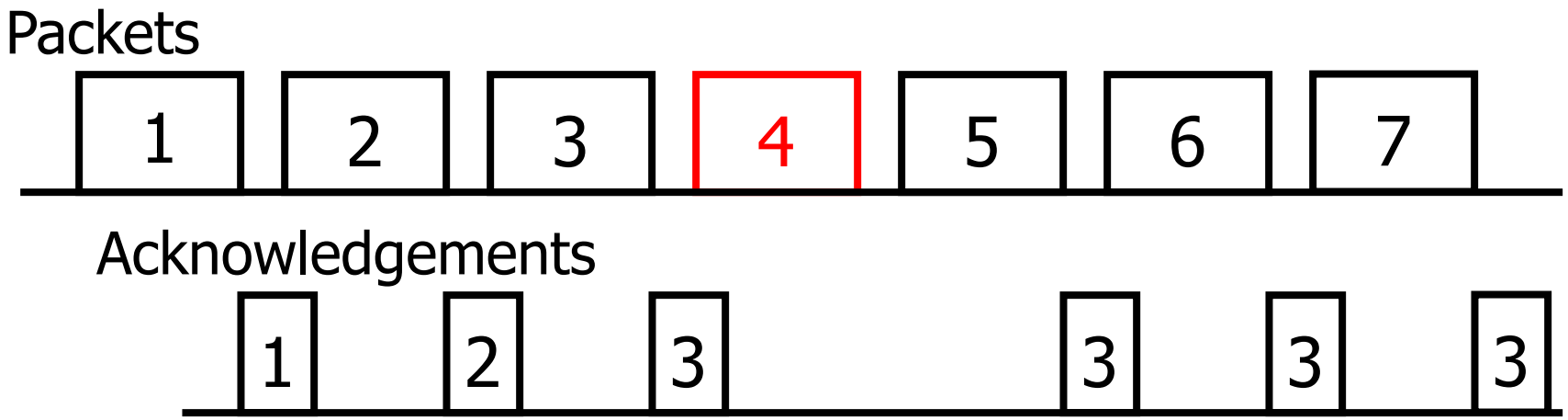


Fig. 6-32. An example of the Internet congestion algorithm.

检测丢包



- **检测丢包**
  - 计时器超时
  - 收到3个重复确认ACK

快速重传算法

- 连续收到3个重复确认后，不再等待计时器超时
- 阈值设置为当前拥塞窗口大小的一半，拥塞窗口重新设置为一个最大段长
- 执行慢启动算法

TCP Tahoe

- **Slow Start，Congestion Avoidance，Fast Retransmit 最早在4.3BSD Tahoe版本实现**

```
for every ack
{   if W < ssthresh then W ++              (ss)
                    else W += 1/W         (ca)  }
for every loss
{   ssthresh := W/2
          W  := 1     }
```

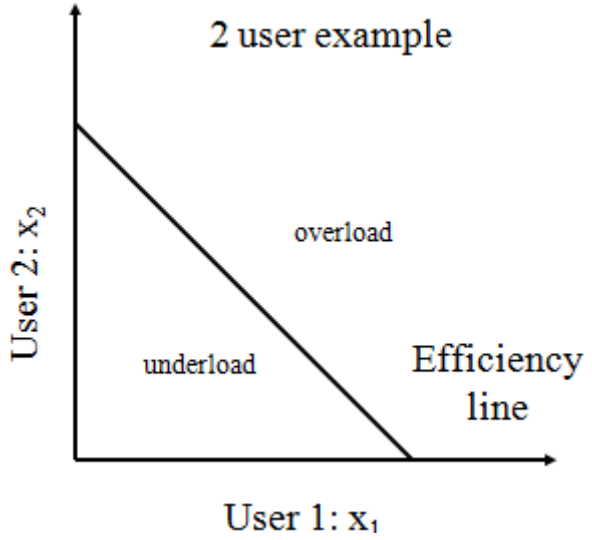
TCP拥塞控制分析

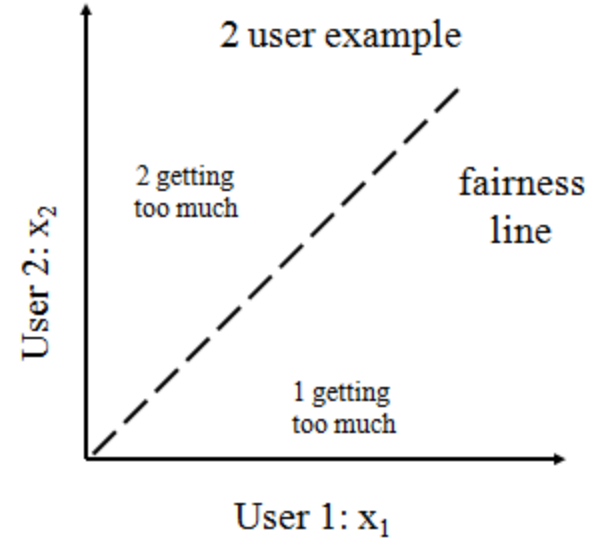
- **有效分配**
  - 太慢浪费带宽，太快导致拥塞
  - 优化目标:

$$\Sigma x_i = X_{goal}$$

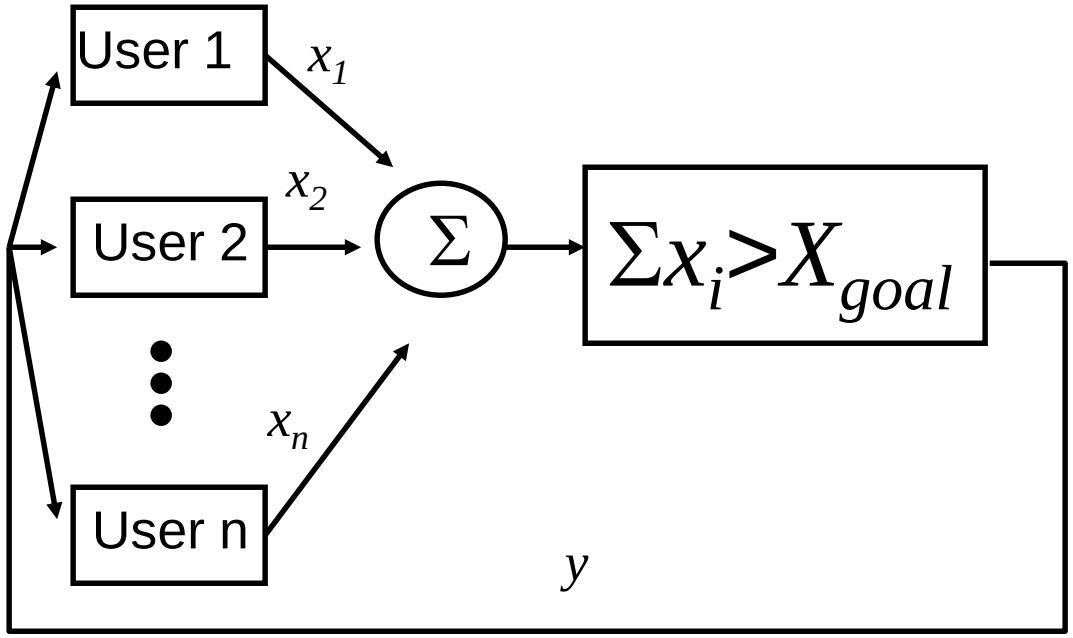
- **公平分配**
  - 共享相同瓶颈链路的流获得相同带宽

$$F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)}$$





TCP拥塞控制系统模型



如何选择拥塞控制函数？

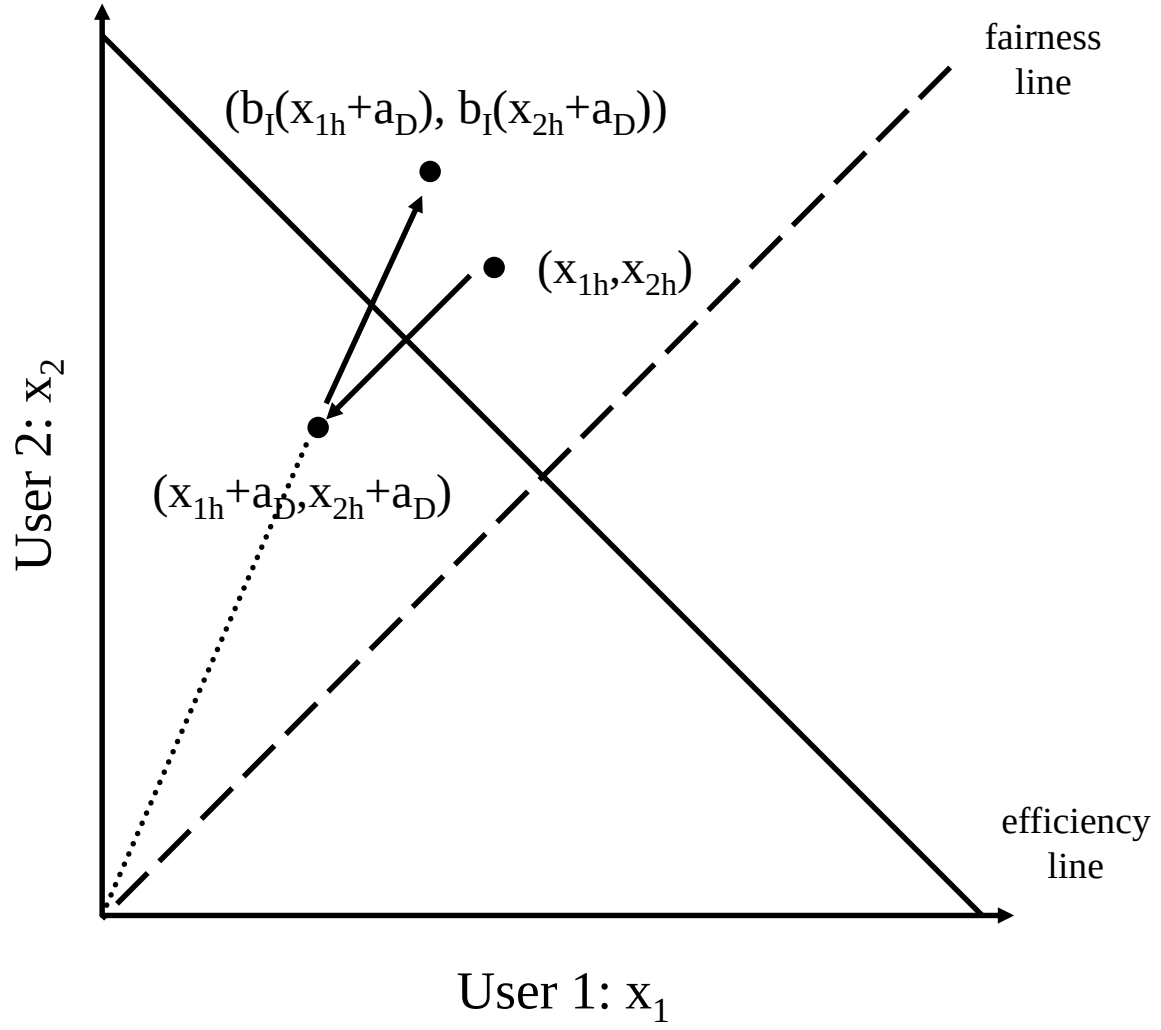
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & increase \\ a_D + b_D x_i(t) & decrease \end{cases}$$

- **Multiplicative increase, additive decrease**
  - $a_I=0$, $b_I>1$, $a_D<0$, $b_D=1$
- **Additive increase, additive decrease**
  - $a_I>0$, $b_I=1$, $a_D<0$, $b_D=1$
- **Multiplicative increase, multiplicative decrease**
  - $a_I=0$, $b_I>1$, $a_D=0$, $0<b_D<1$
- **Additive increase, multiplicative decrease**
  - $a_I>0$, $b_I=1$, $a_D=0$, $0<b_D<1$
- **Which one?**

Multiplicative Increase,Additive Decrease

- **公平性不收敛**
  - 也不稳定
- **有效性不收敛**
  - 稳定的条件：

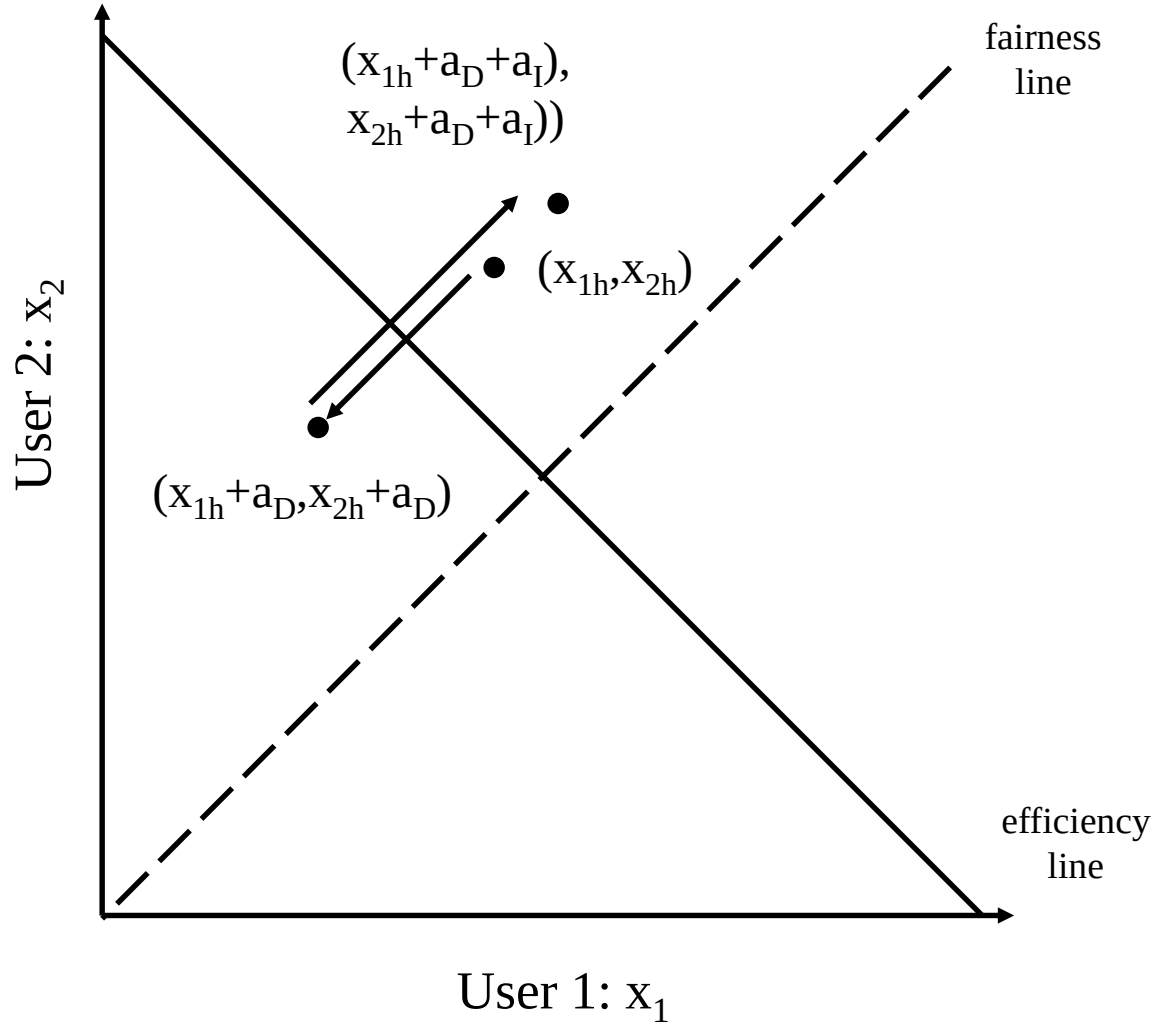
$$x_{1h} = x_{2h} = \frac{b_I a_D}{1 - b_I}$$



Additive Increase, Additive Decrease

- **公平性不收敛**
  - 但是稳定
- **有效性不收敛**
  - 稳定的条件

$$a_D = a_I$$

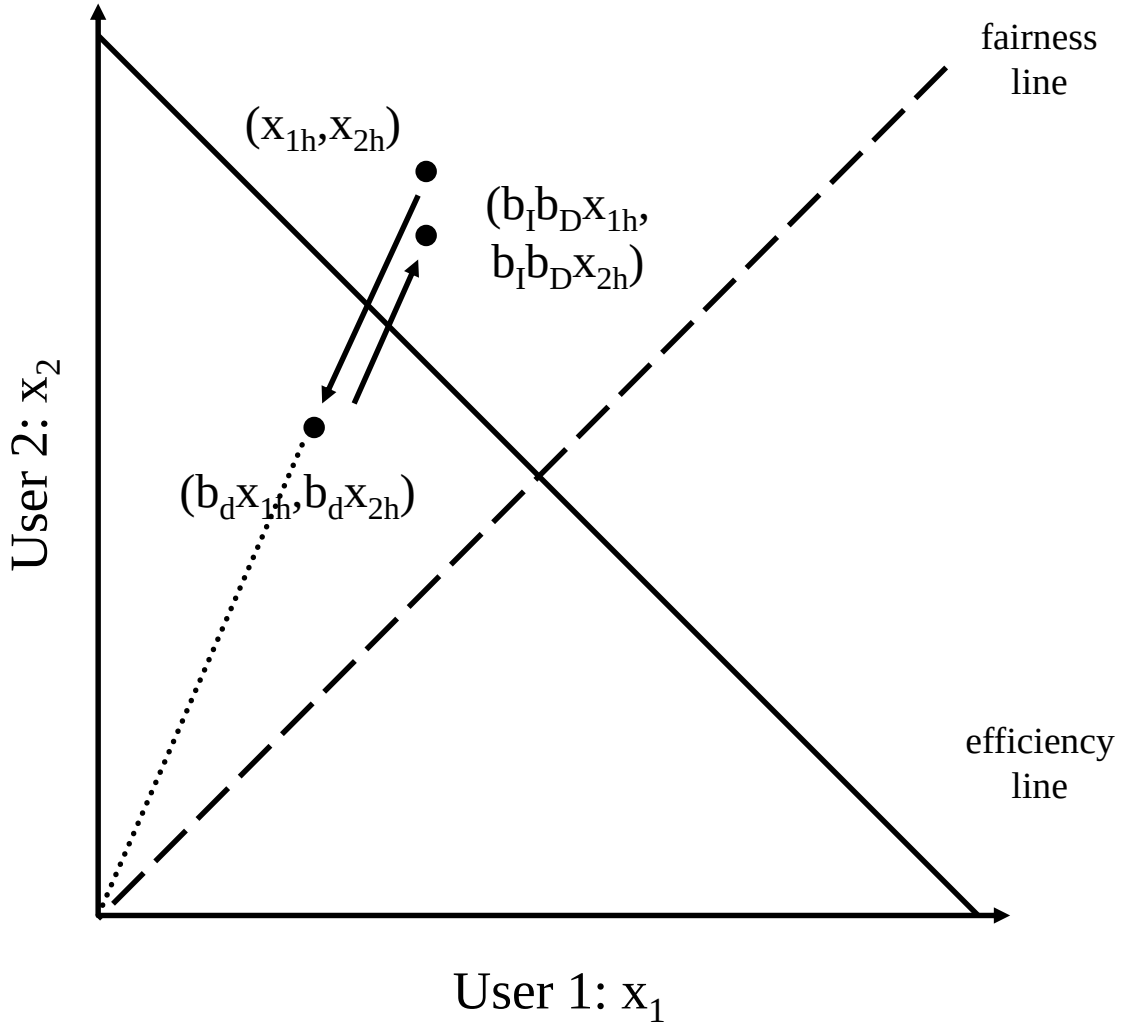


Multiplicative Increase, Multiplicative Decrease

- **公平性不收敛**
  - 但是稳定
- **有效性收敛的条件**

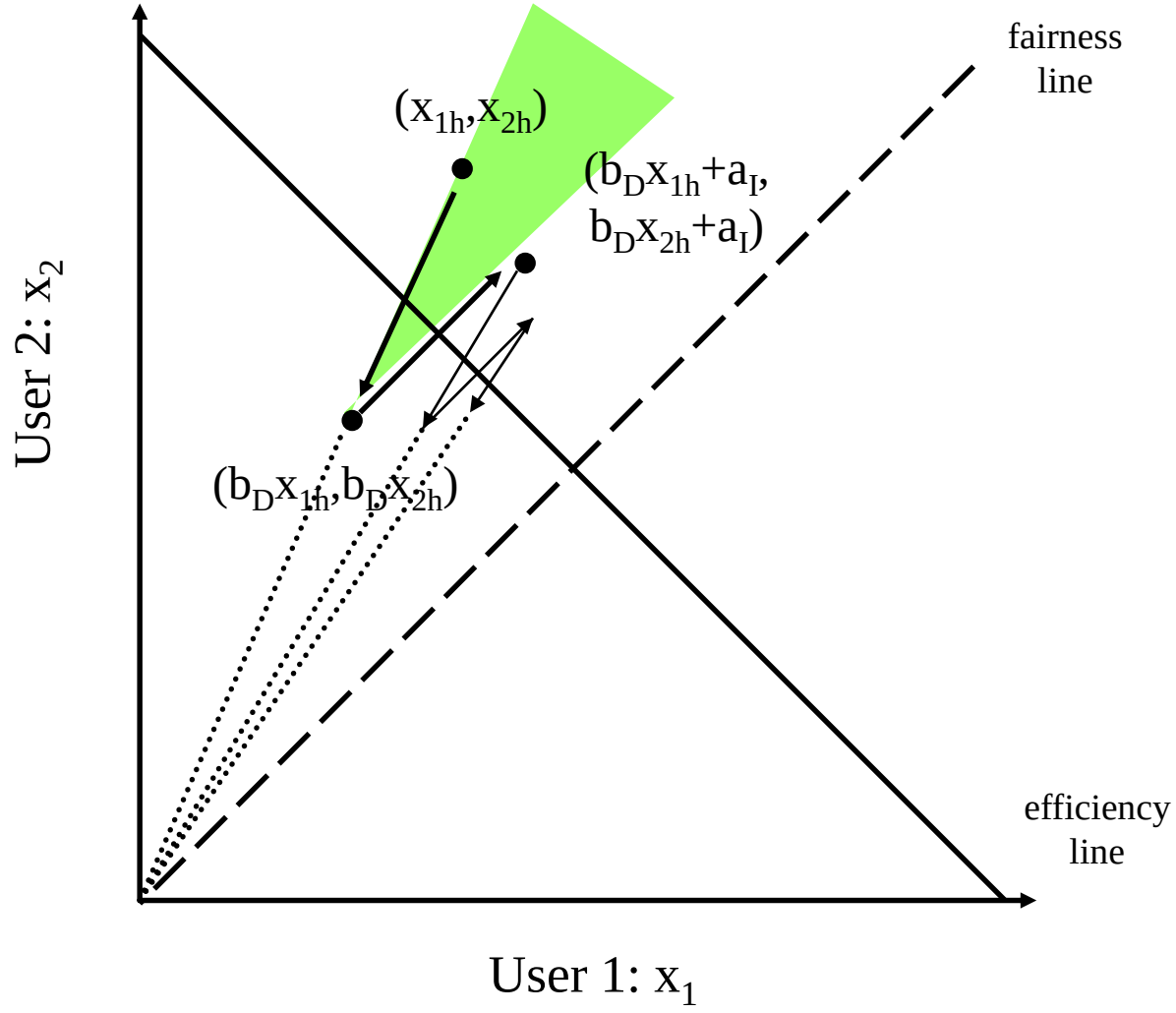
$$b_I \geq 1$$

$$0 \leq b_D < 1$$



Additive Increase, Multiplicative Decrease

- **公平性收敛**
- **有效性收敛**



例题

- **A、B双方已经建立了TCP连接，初始阈值为32K字节(1K = 1024)，最大发送段长MSS为1K字节。发送方向为A->B，B没有数据要发送，B每收到一个数据段都会发出一个应答段。在整个过程中上层一直有数据要发送，并且都以MSS大小的段发送。A的发送序列号从0开始。**
  - 在传送过程中，A收到1个ACK为10240的数据段，收到这个应答段后，A处拥塞窗口的大小是多少?
  - 当收到ACK为32768的数据段后，A处拥塞窗口的大小是多少?
  - 当阈值为32K字节、拥塞窗口为40K字节时，发送方发生了超时，求超时发生后拥塞窗口的大小和阈值的大小。

例题（续）

1.收到的是对第10个段的应答，变化后拥塞窗口的大小为11

2.收到的是对第32个段的应答，这时拥塞窗口已经超过阈值，应该使用线性增长，变化后的拥塞窗口大小为 32 K字节。

3.拥塞窗口 = 1 MSS = 1KB，阈值 = 40 / 2 = 20 KB

UDP 协议

- **UDP：User Datagram Protocol**
  - RFC 768
- **为什么会有 UDP**
  - 不需要建立连接，延迟小
  - 简单，没有连接状态
  - 报文头小
  - 没有拥塞控制，可以尽快的发送

UDP 协议（续）

- **UDP 协议的特点**
  - 简单的传送协议
  - “best effort”服务，UDP 报文可能会丢失、乱序
  - 无连接
    - 收发双方不需要握手
    - 每个UDP报文的处理都独立于其他报文

UDP协议（续）

- **经常用于流媒体应用**
  - 可以容忍丢包
  - 速率敏感
- **UDP的其他应用范围**
  - RIP：路由信息周期发送
  - DNS：避免 TCP 连接建立延迟
  - SNMP：当网络拥塞时，网管也要运行。网管信息带内（in-band）传送，用UDP比用可靠的、具有拥塞控制的TCP效果要好
  - 基于 UDP 的可靠传送：应用程序自己定义错误恢复

UDP头格式

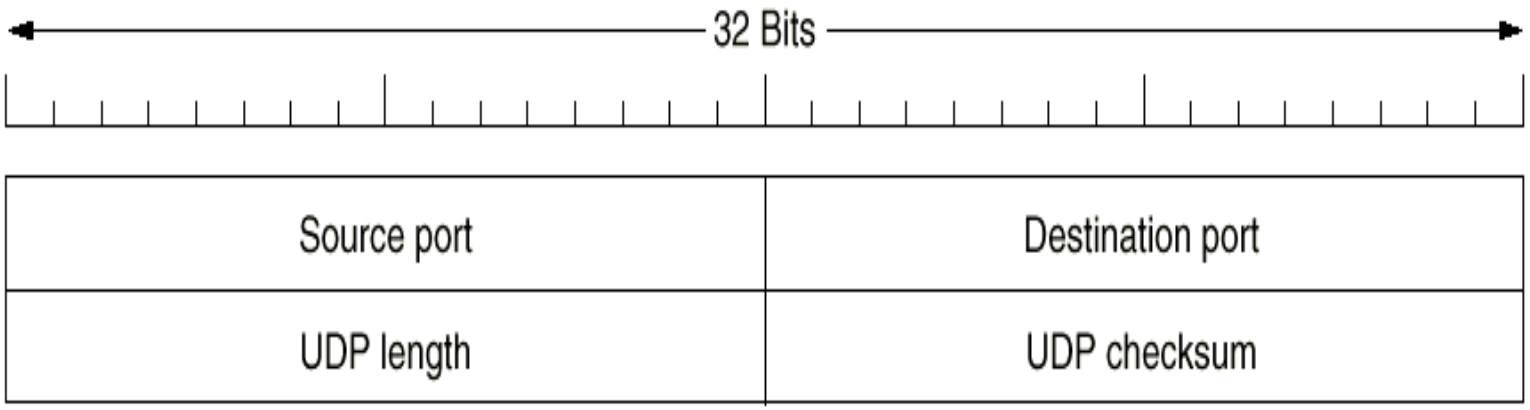


Fig. 6-34. The UDP header.

总结

- **Berkeley Sockets**
- **传送层编址：（IP address, local port)**
- **TCP**
  - 连接管理
    - 建立连接：三次握手
    - 释放连接：三次握手 + 定时器
  - 可靠传送：可变滑动窗口
  - 流控制和拥塞控制
    - 可变滑动窗口
    - 慢启动，拥塞避免，快速重传
- **UDP**