第八章

传输层服务和控制

主要内容

8.1 传输服务

8.2 传输协议

8.3 Internet传输协议

8.1 传输服务（1）

- 引入传输层的原因
  - 消除网络层的不可靠性；
  - 提供从源端主机到目的端主机的可靠的、与实际使用的网络无关的信息传输。
- 传输服务
  - 传输实体（transport entity）：完成传输层功能的硬软件；
  - 传输层实体利用网络层提供的服务向高层提供有效、可靠的服务；

8.1 传输服务（2）

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

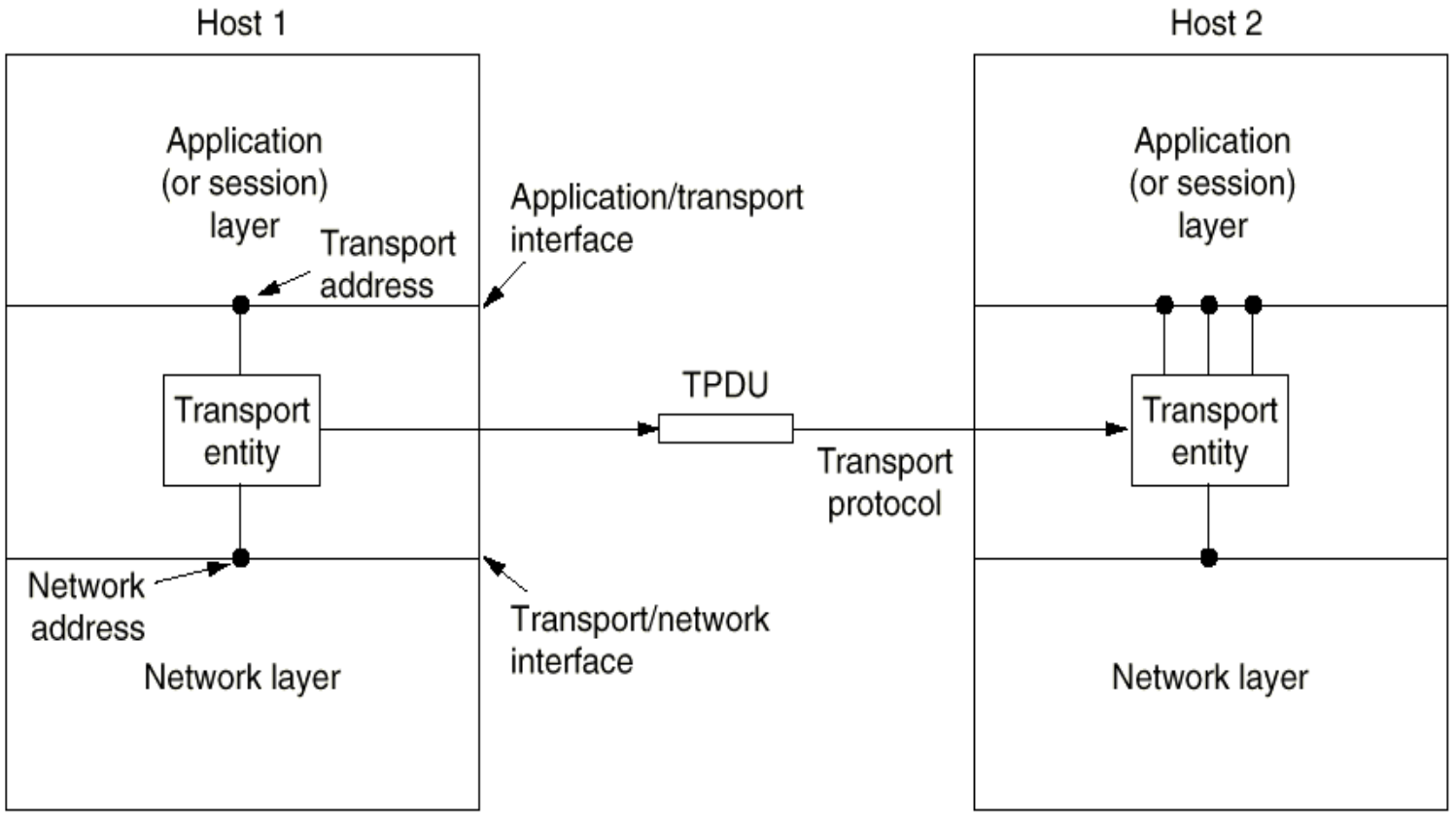


Fig. 6-1. The network, transport, and application layers.

8.1 传输服务（3）

- 传输服务原语（Transport Service Primitives）
  - 传输用户（应用程序）通过传输服务原语访问传输服务
  - 一个简单传输服务的原语
  - TPDU

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.

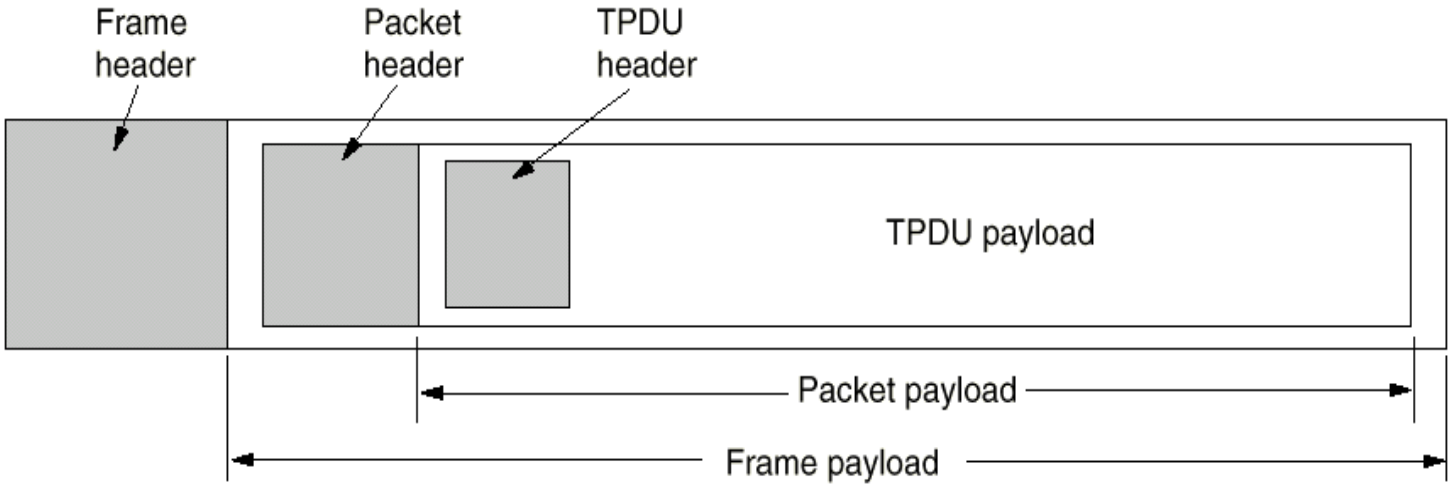


Fig. 6-4. Nesting of TPDUs, packets, and frames.

8.1 传输服务（4）

- 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

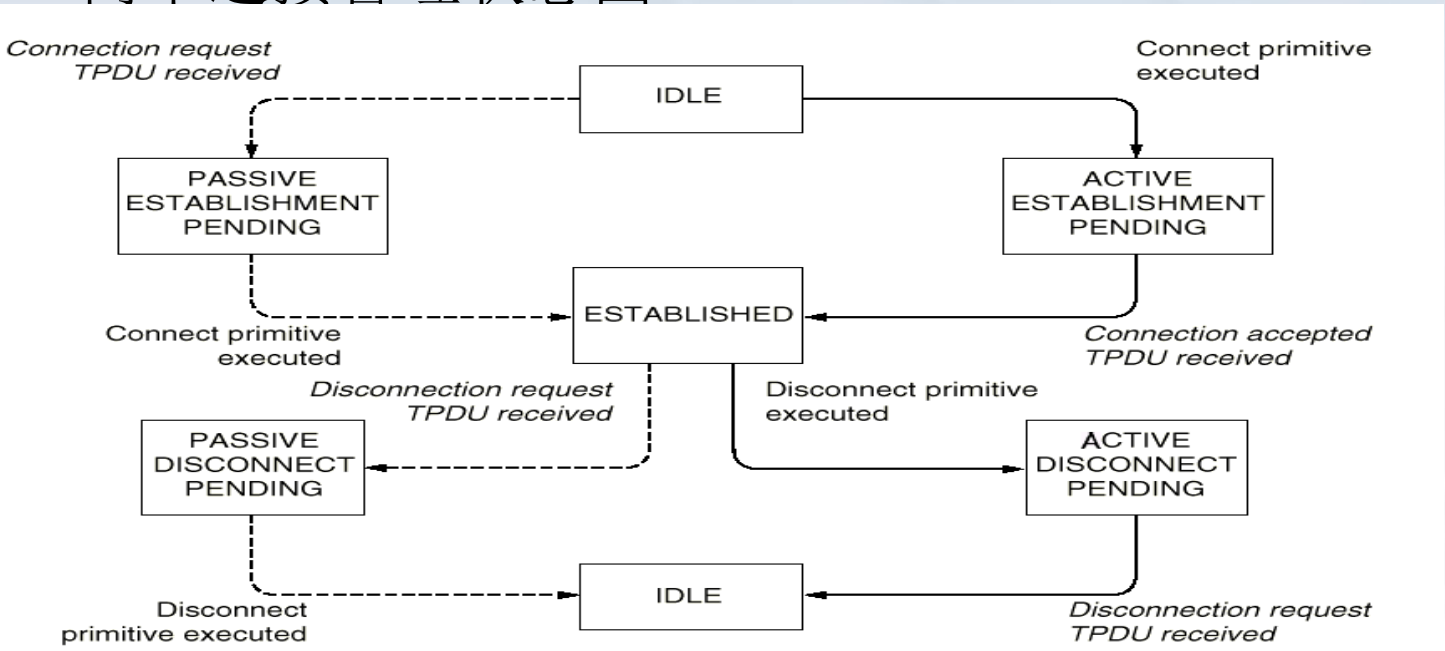


Fig. 6-5. A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

8.1 传输服务（5）

- 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.

8.1 传输服务（6）

- 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输层服务完成通信。
  - 建立连接
    - 服务程序
      - 调用socket创建一个新的套接字，并在传输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；

8.1 传输服务（7）

- 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。当传输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。

• 客户程序

- 调用socket创建一个新的套接字，并在传输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；

8.1 传输服务（8）

- 调用connect阻塞客户程序，传输层实体开始建立连接，当连接建立完成时，取消阻塞；

- 数据传输
  - 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。

8.2 传输协议（1）

- 寻址（Addressing）
  - 方法：定义传输服务访问点TSAP（Transport Service Access Point），将应用进程与这些TSAP相连。在Internet中，TSAP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，象telnet是（IP地址，端口23）；
    - 方法2：从名字服务器（name server）或目录服务器（directory server）获得TSAP

8.2 传输协议（2）

- 一个特殊的进程称为**名字服务器**或**目录服务器**（TSAP众所周知）；
- 用户与名字服务器建立连接，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；
- 与服务进程建立连接。

- 当服务程序很多时，使用初始连接协议（initial connection protocol）
  - 一个称为**进程服务器（process server）**的进程(inetd)同时在多个端口上监听；
  - 远方客户程序向它实际想访问的服务程序的TSAP发出连接建立请求；

8.2 传输协议（3）

- 如果没有服务程序在此TSAP上监听，则远方客户和进程服务器建立连接；
- 进程服务器产生所请求的服务进程，并使该进程继承和远程客户的连接；
- 进程服务器返回继续监听；
- 远方客户程序与所希望的服务程序进行数据传输。

8.2 传输协议（4）

- 建立连接
  - 网络可能丢失、重复包，特别是延迟重复包（delayed duplicates）的存在，导致传输层建立连接的复杂性；
  - 解决延迟重复包的关键是丢弃过时的包；
  - 两次握手方案
    - 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、存储和重复包。

8.2 传输协议（5）

- 三次握手方案（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。

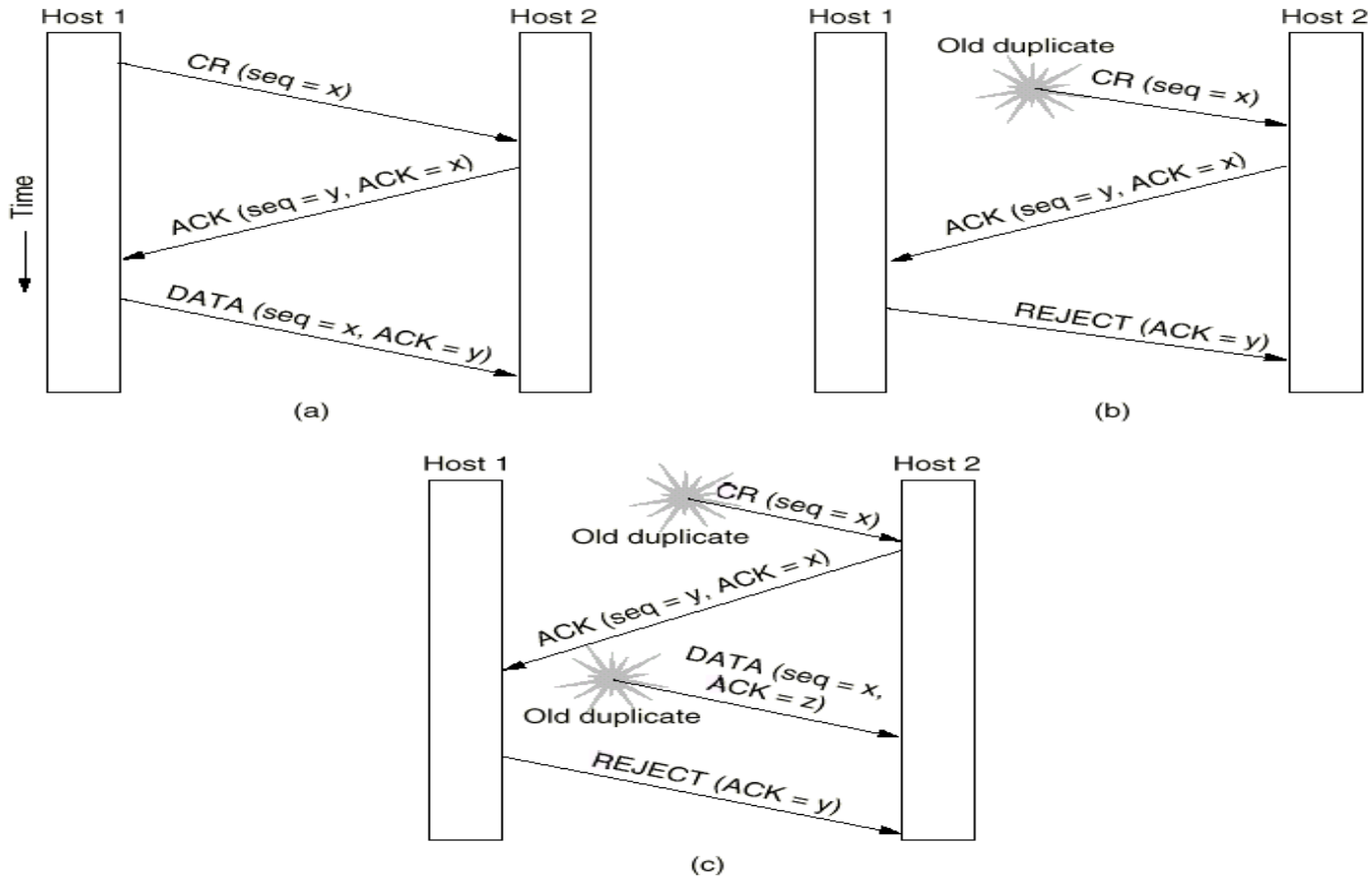


Fig. 6-11. Three protocol scenarios for establishing a connection using a three-way handshake. CR and ACC denote CONNECTION REQUEST and CONNECTION ACCEPTED, respectively. (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate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

8.2 传输协议（6）

- 释放连接
  - 两种连接释放方法
    - 非对称式：一方释放连接，整个连接断开，存在丢失数据的危险；

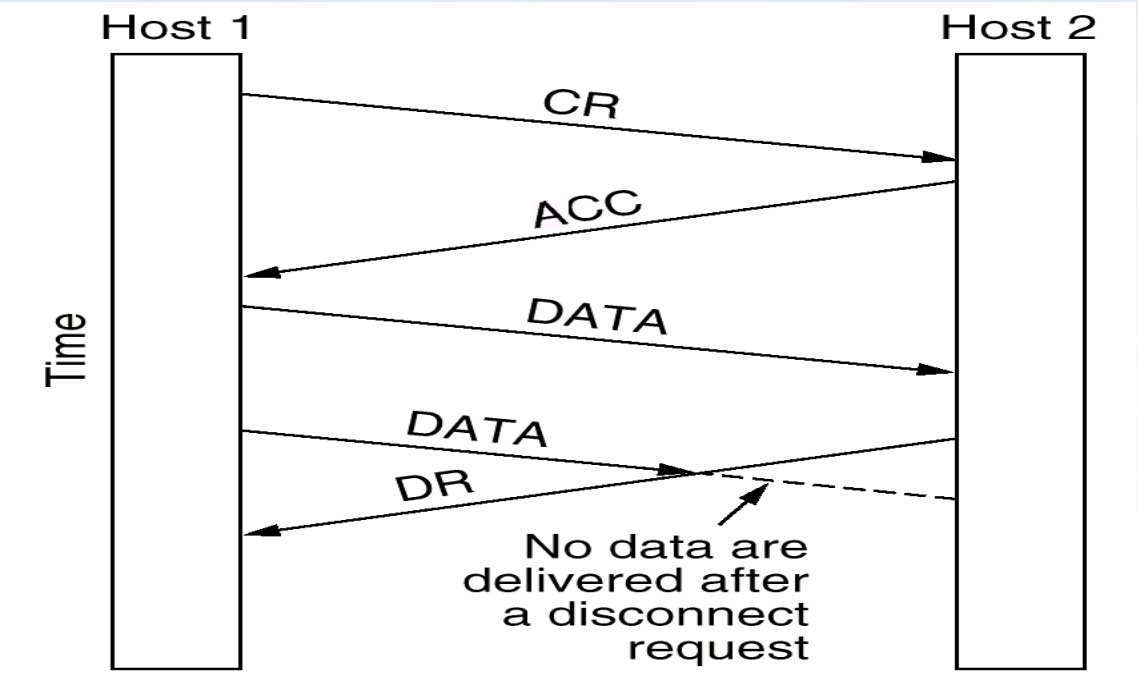


Fig. 6-12. Abrupt disconnection with loss of data.

8.2 传输协议（7）

- 对称式：由于**两军问题（two-army problem）**的存在，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；

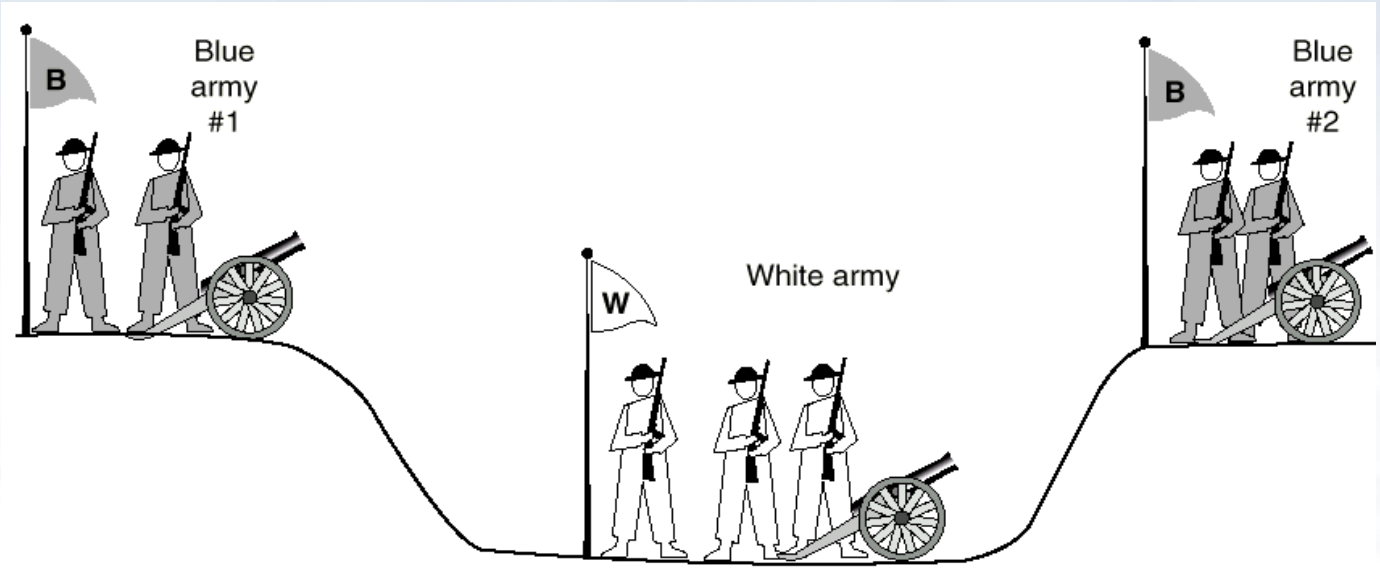


Fig. 6-13. The two-army problem.

8.2 传输协议（8）

- 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。

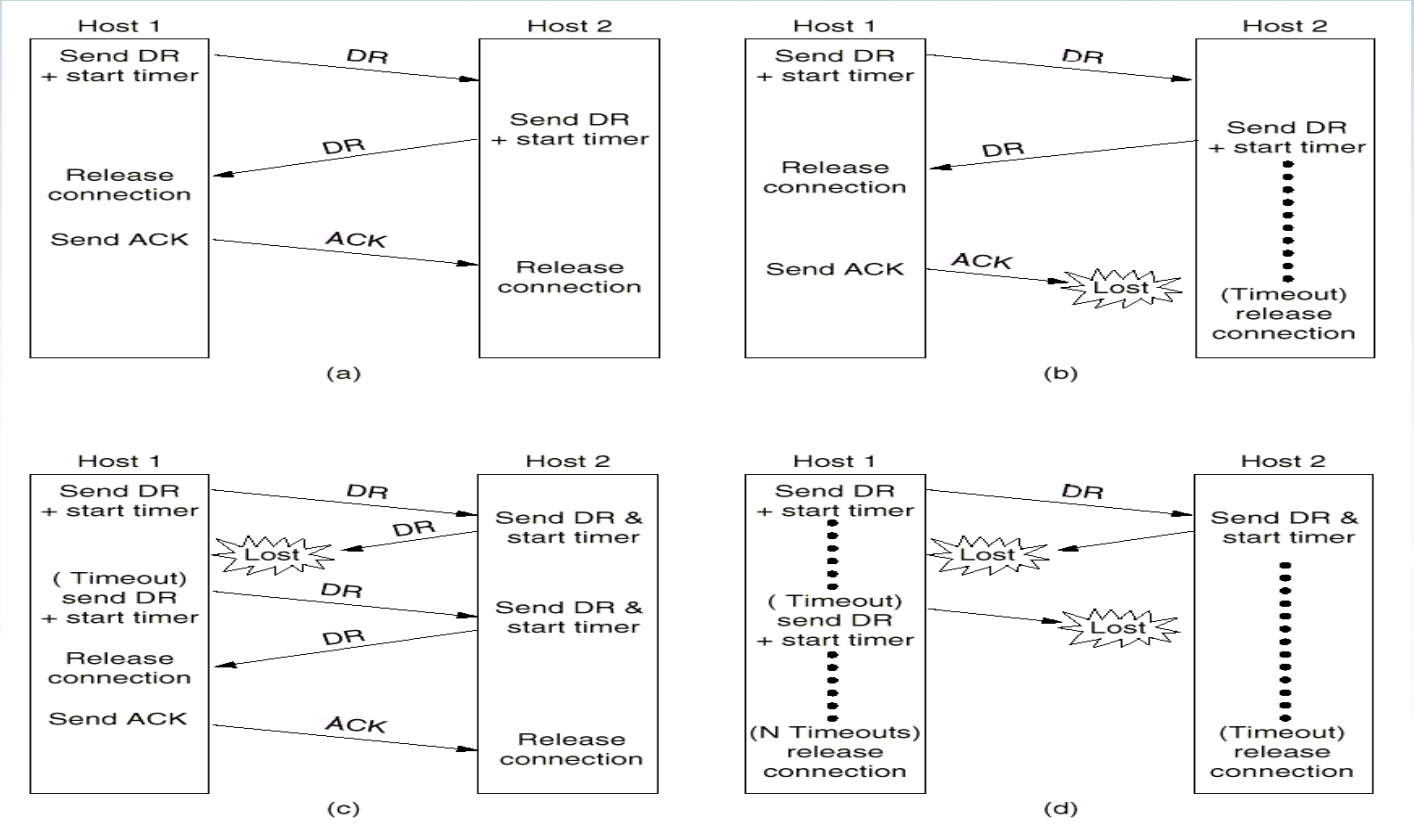


Fig. 6-14. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.

8.2 传输协议（9）

- 流控和缓存（Flow Control and Buffering）
  - 缓存：由于网络层服务是不可靠的，传输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。接收方的传输层实体既可以做也可以不做缓存。缓存区的设计有三种。
  - 流控：传输层利用可变滑动窗口协议来实现流控。所谓可变滑动窗口协议，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，主机应该周期性的发送TPDU。

8.3 Internet传输协议（1）

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

8.3 Internet传输协议（2）

8.3.1 TCP 协议

- 服务模型
  - 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；
  - 套接字的地址是用（IP地址，主机端口号）来表示的。256以下的端口号被标准服务保留，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不支持多播（multicast）和广播（broadcast）；

8.3 Internet传输协议（3）

- 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输中不能得到保留；
- 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(urgent data)，可以使用URGENT标记。

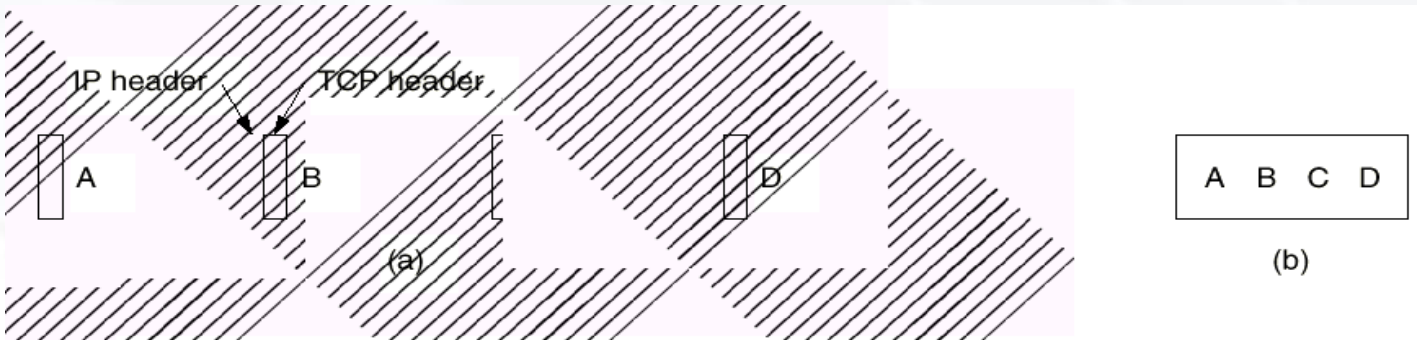


Fig. 6-23. (a) Four 512-byte segments sent as separate IP datagrams. (b) The 2048 bytes of data delivered to the application in a single READ call.

8.3 Internet传输协议（4）

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

8.3 Internet传输协议（5）

- TCP 协议
  - 按字节流分配序号，每个字节流有一个32位的序号；
  - 传输实体之间使用**段(segment)**(TPDU)交换数据
  - 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。段的大小必须首先满足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输单元（MTU）的限制，比如以太网的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序号。

8.3 Internet传输协议（6）

- TCP段头

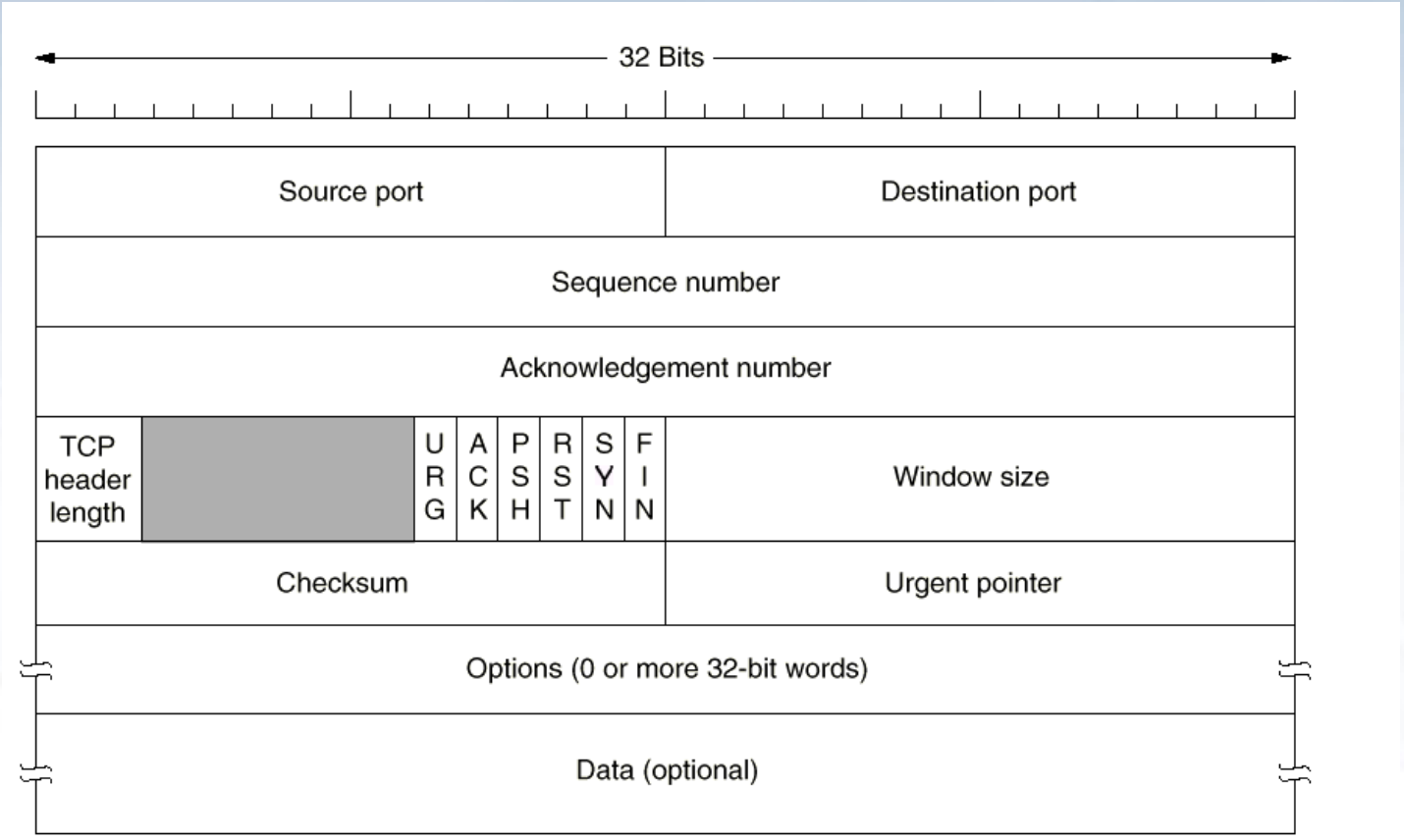


Fig. 6-24. The TCP header.

8.3 Internet传输协议（7）

- 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，包含可选项域；
- 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

8.3 Internet传输协议（8）

- 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；
- 可选项域。

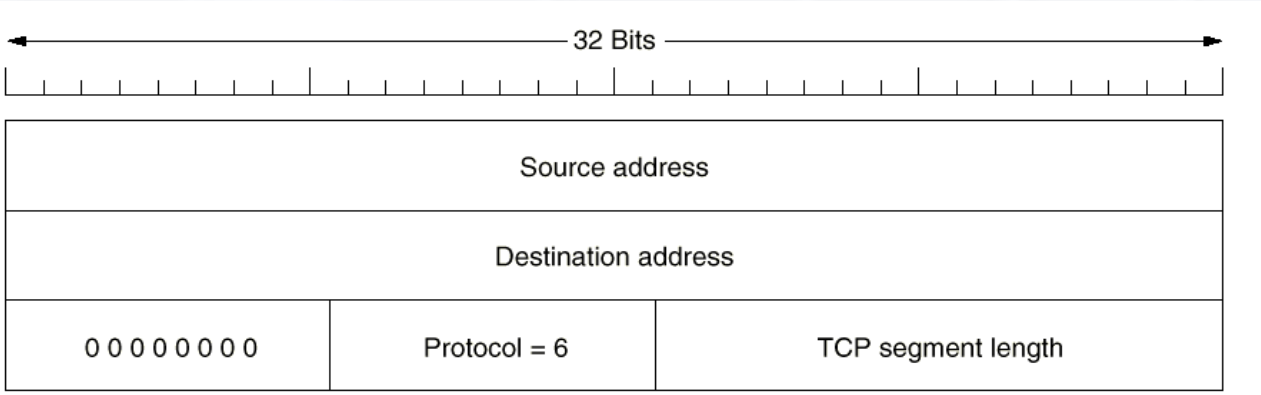


Fig. 6-25. The pseudoheader included in the TCP checksum.

8.3 Internet传输协议（9）

- 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；
    - 服务器方的传输实体接收到这个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；

8.3 Internet传输协议（10）

- 发起方收到确认后，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。

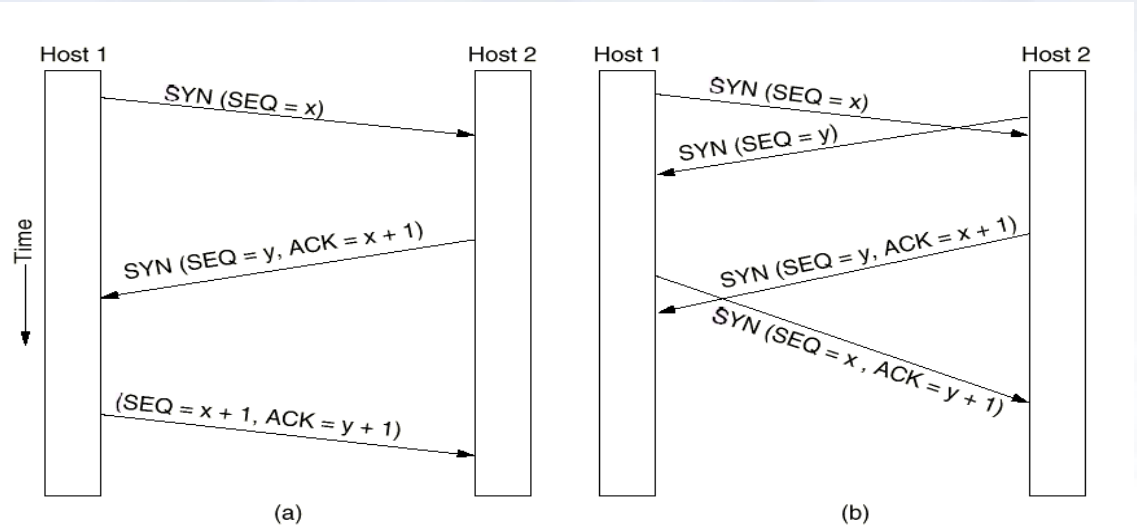


Fig. 6-26. (a) TCP connection establishment in the normal case. (b) Call collision.

8.3 Internet传输协议（11）

- 单向的连接释放
  - 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。

8.3 Internet传输协议（12）

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!).

8.3 Internet传输协议（13）

- TCP连接管理的有限状态机

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.

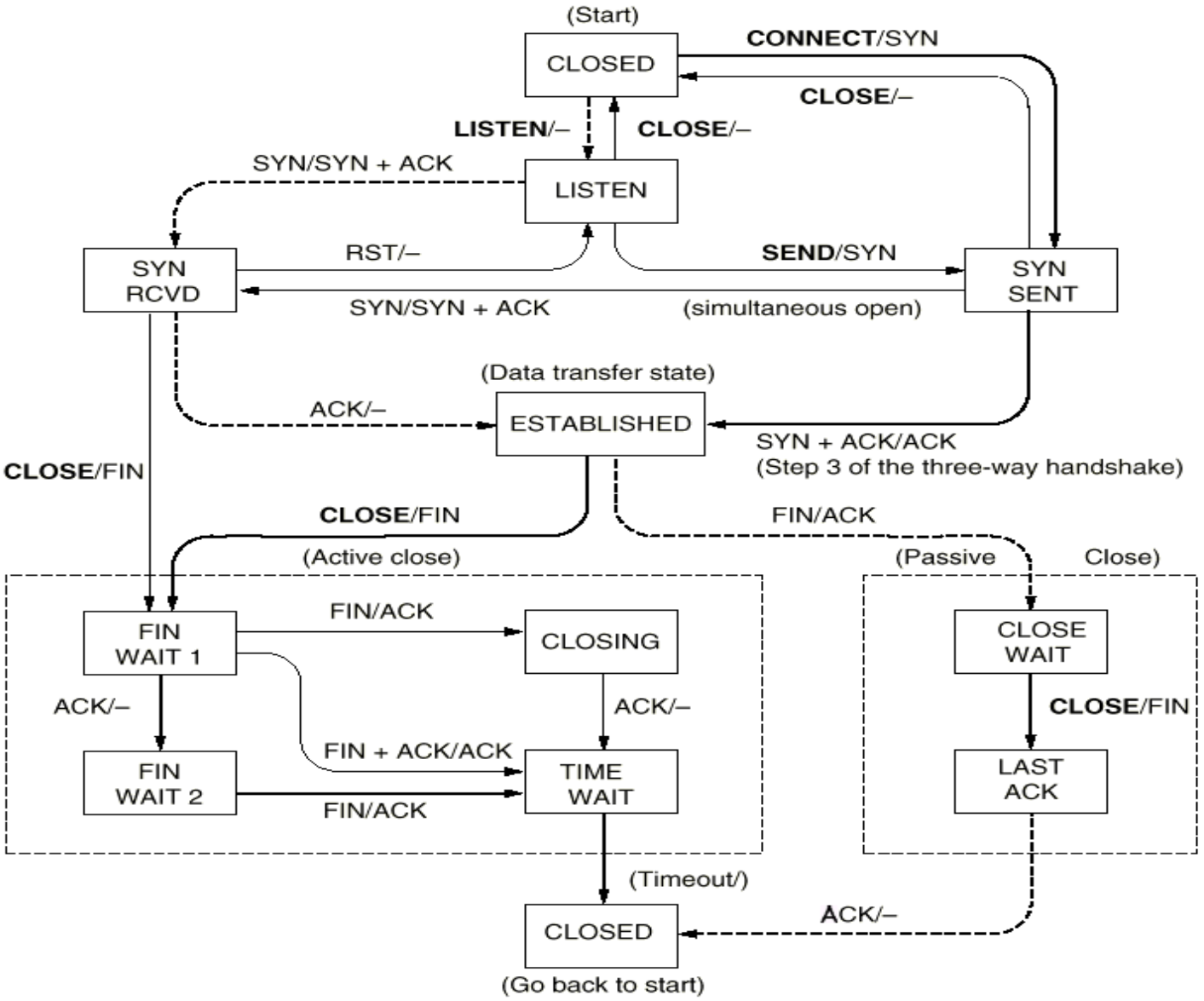


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

8.3 Internet传输协议（14）

- TCP传输策略
  - 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。

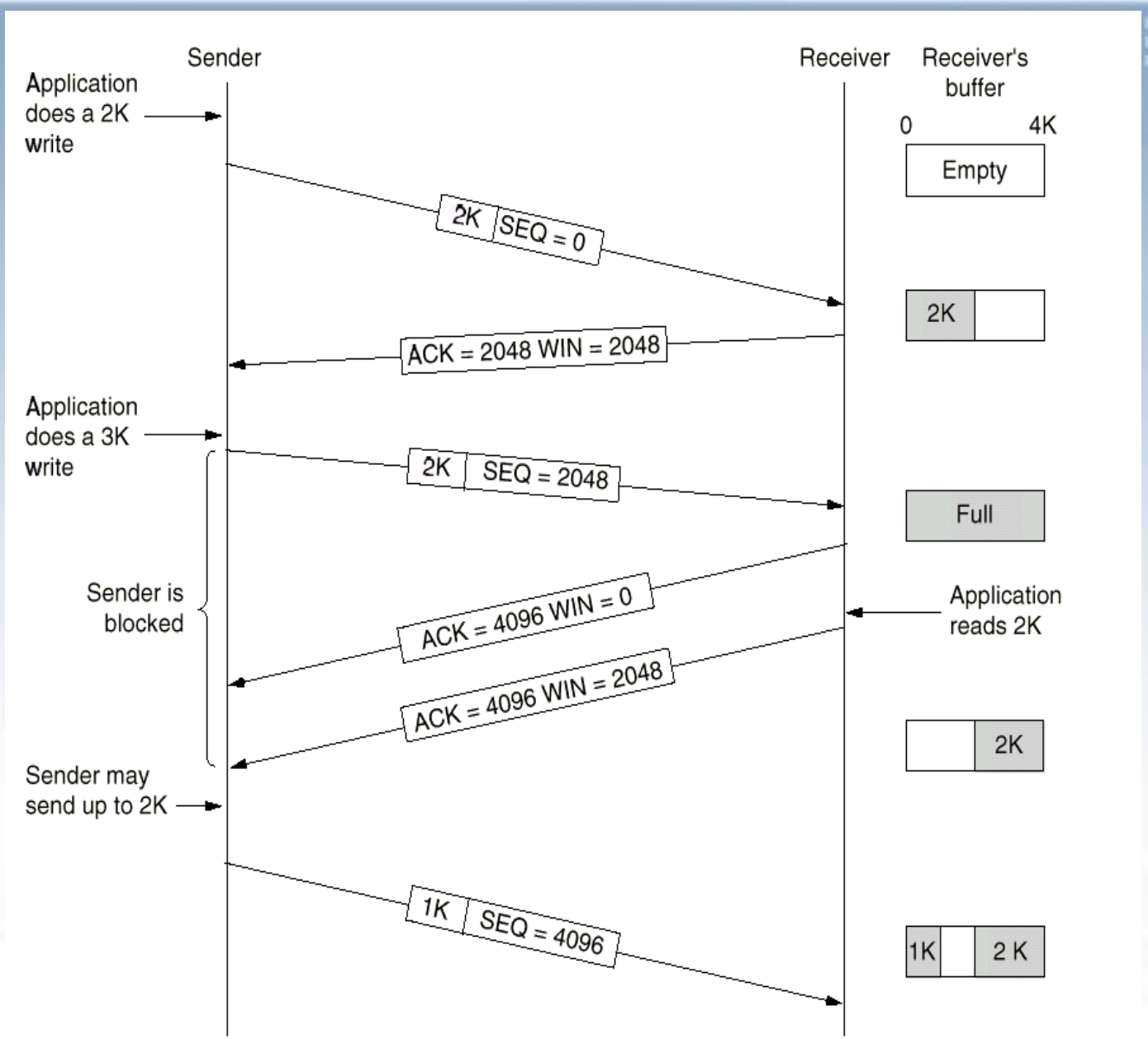


Fig. 6-29. Window management in TCP.

8.3 Internet传输协议（15）

- 如何改进传输层的性能？
  - 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输实体发出一个字节时，传输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；

8.3 Internet传输协议（16）

- 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输层实体读出一个字节时，传输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。

8.3 Internet传输协议（17）

- TCP拥塞控制
  - 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
    - 导致网络拥塞的两个潜在因素是：网络能力和接收能力。

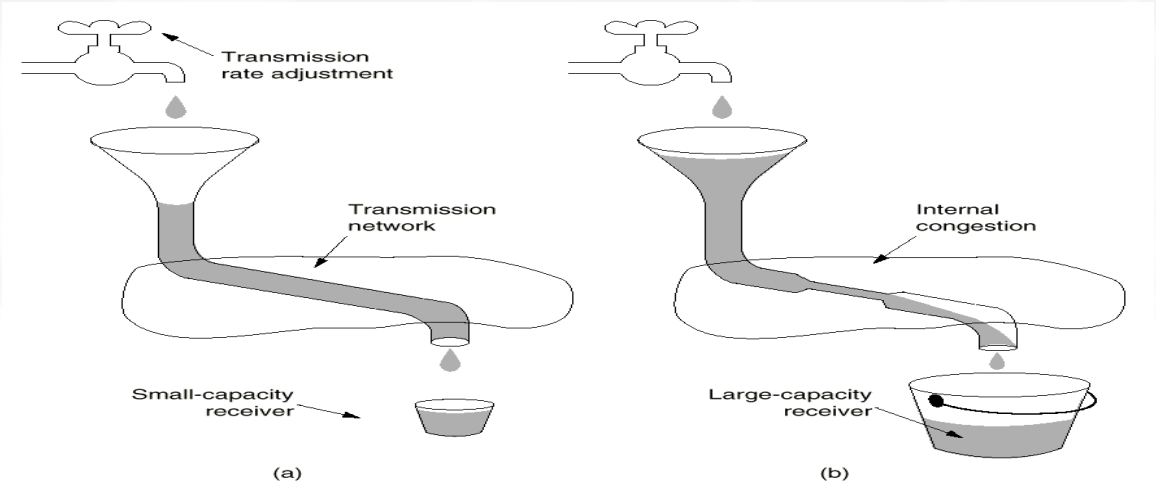


Fig. 6-31. (a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

8.3 Internet传输协议（18）

- TCP处理第一种拥塞的措施
  - 在连接建立时声明最大可接受段长度；
  - 利用可变滑动窗口协议防止出现拥塞；
- TCP处理第二种拥塞的措施
  - 发送方维护两个窗口：可变发送窗口和拥塞窗口，按两个窗口的最小值发送；
  - 拥塞窗口依照**慢启动（slow start）**算法和**拥塞避免（congestion avoidance）**算法变化。

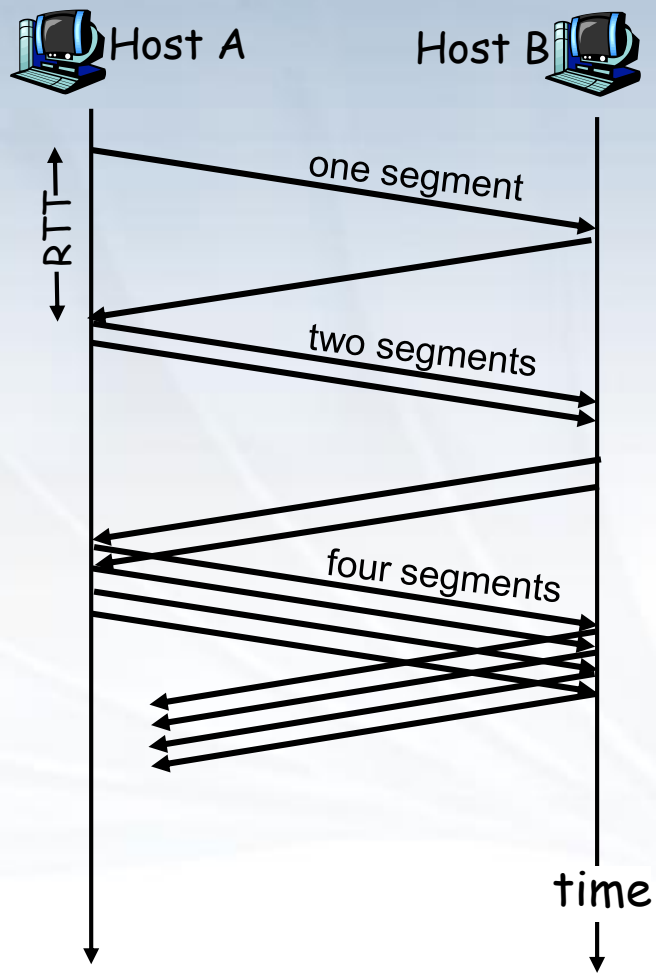
8.3 Internet传输协议（19）

- 慢启动（slow start）算法
  - 连接建立时拥塞窗口（congwin）初始值为该连接允许的最大段长，阈值（threshold）为64K；
  - 发出一个最大段长的TCP段，若正确确认，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，直至发生丢包超时事件，或拥塞窗口大于阈值。

8.3 Internet传输协议（20）

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

8.3 Internet传输协议（21）

- 拥塞避免（congestion avoidance）算法
  - 若拥塞窗口大于阈值，从此时开始，拥塞窗口线性增长，一个RTT周期增加一个最大段长，直至发生丢包超时事件；
  - 当超时事件发生后，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。

8.3 Internet传输协议（22）

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.

问题

A、B双方已经建立了TCP连接，采用Slow Start算法进行流控，初始的阈值为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每收到一个数据报文都会发出一个应答报文。在整个过程中上层一直有数据要发送，并且都以MSS大小的报文发送。A的发送序列号从0开始。

1.在传输过程中，A收到1个ACK为10240的报文，收到这个应答报文后，A处拥塞窗口的大小是多少？

2.当收到ACK = 32768的报文后，A处拥塞窗口的大小是多少？

3.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。

问题

1.收到的为第10个报文的应答,变化后拥塞窗口的大小为11

2.收到的为第32个报文的应答，这时拥塞窗口已经超过阈值，应当使用线性增长，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

8.3 Internet传输协议（23）

8.3.2 UDP（User Datagram Protocol）协议

- 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特点
  - RFC 768
  - 简单的传输协议
  - “best effort”服务, UDP 报文可能会:
    - 丢失
    - 乱序

8.3 Internet传输协议（24）

- 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
- 经常用于流媒体应用
  - 可以容忍丢包
  - 速率敏感
- UDP的其他应用领域：
  - 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）传输，用UDP比用可靠的、具有拥塞控制的TCP效果要好。

8.3 Internet传输协议（25）

- 基于UDP的可靠传输：应用程序自己定义错误恢复

- UDP头格式

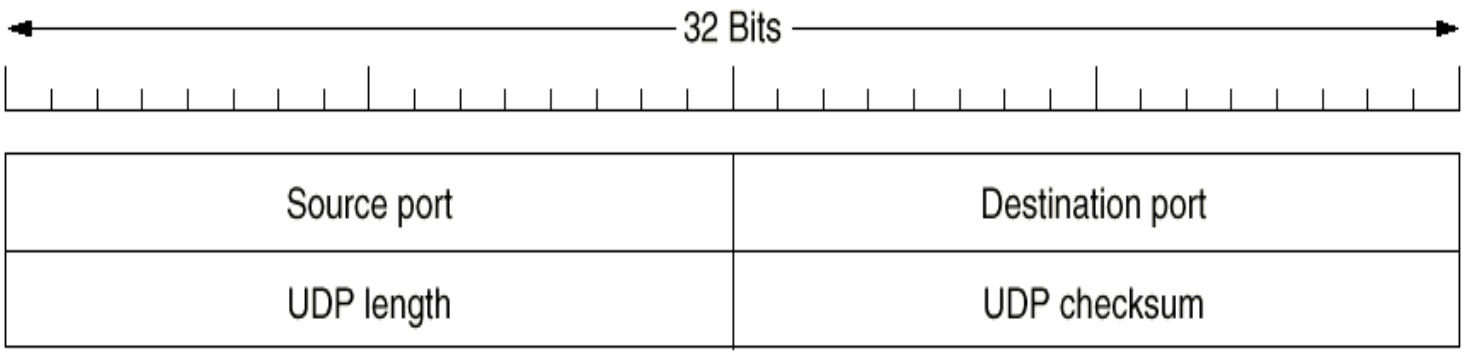


Fig. 6-34. The UDP header.

END !