计算机网络

14. TCP AND UDP

厦门大学软件学院

黄炜 助理教授

PART III Internetworking

Ch 19 Binding Protocol Addresses (ARP)

(地址解析协议)

19.3 Address Resolution 地址解析

- 地址解析协议（Address Resolution Protocol）
  - IP是虚拟的，但数据链路层需要物理地址，最终要换的
  - Translation from a computer's protocol address to an equivalent hardware address.
  - Address resolution is local to a network.
  - A computer never resolves the address of a computer on a remote network.

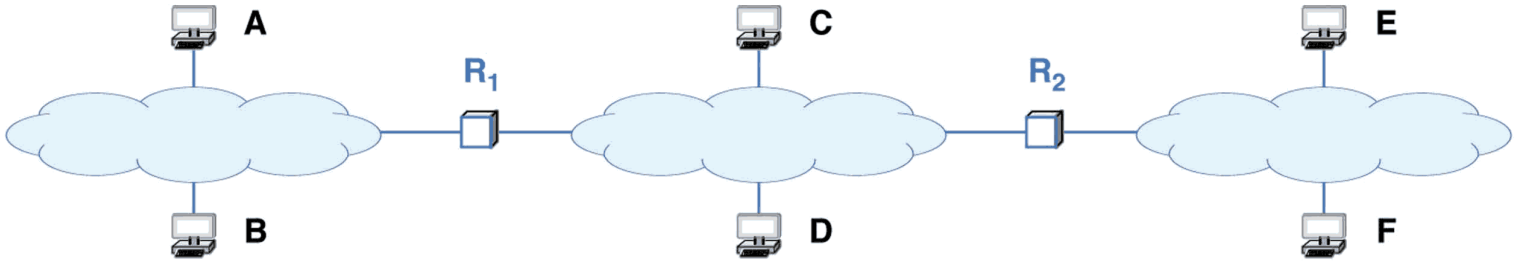


Figure 23.1 An example internet of three networks and computers connected to each.

地址解析（AR）

- 将IP地址解析为MAC地址的叫做地址解析
- 解析地址仅限同一个物理网络内
  - 不同网络内没用
- 概念地址边界
  - ARP以上的用IP地址
  - ARP以下的用物理地址
- ARP提供一种好心的服务
  - ARP欺骗

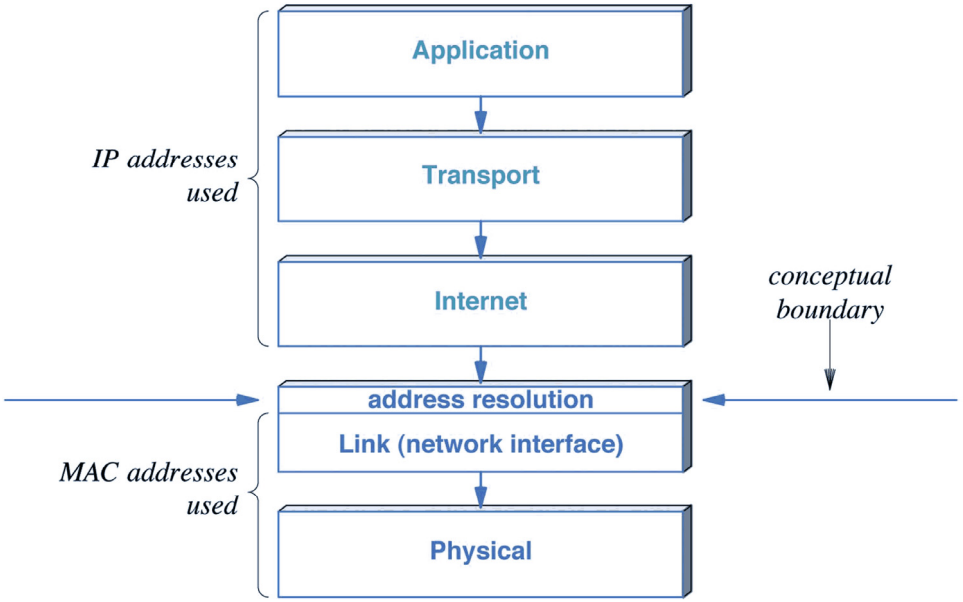


Figure 23.5 Illustration of the boundary between the use of IP addresses and MAC addresses.

19.4 Address Resolution Techniques

- 地址解析算法可分为三大基本类：
  - 查表（Table lookup）。存储在内存表。
  - 相近形式计算（Close-form computation）。配置使得硬件地址可通过简单的布尔和算术运算得出它的协议地址。
  - 消息交换（Message exchange）。计算机通过网络交换消息来解析一个地址。一台计算机发出某个地址联编的请求消息后，另一台计算机返回一个包含所需信息的应答消息。

19.4 Address Resolution Techniques

Algorithm 23.1

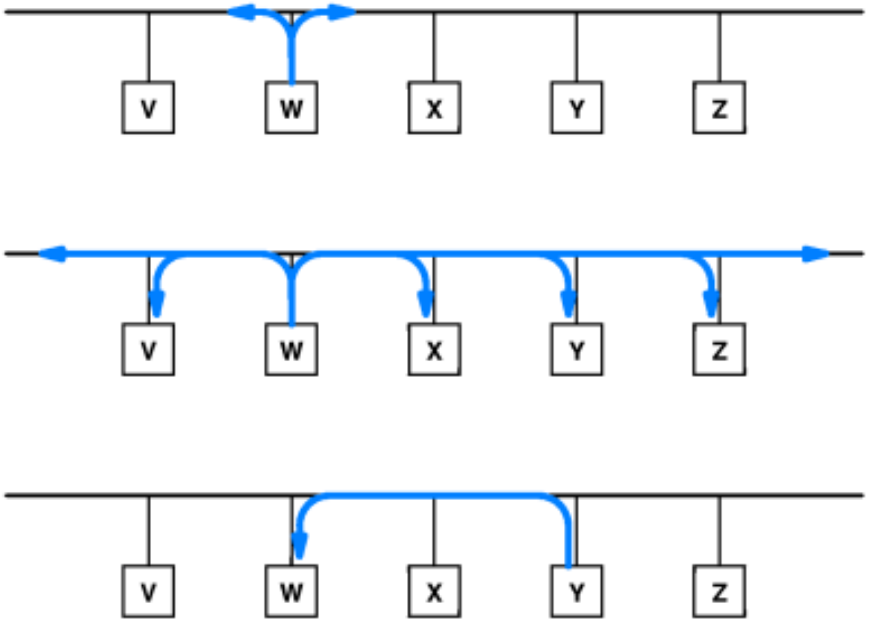
Given:
An incoming ARP message (either a request or a response)
Perform:
Process the message and update the ARP cache
Method:

```
Extract the sender's IP address, I, and MAC address, M
If (address I is already in the ARP cache) {
    Replace the MAC address in the cache with M
}
if (message is a request and target is "me") {
    Add an entry to the ARP cache for the sender
        provided no entry exists;
    Generate and send a response;
}
```

Algorithm 23.1 The steps ARP takes when processing an incoming message.

19.8 Address Resolution Protocol

- TCP/IP protocol suite includes an ARP
- The ARP standard defines a request and a response.
- Caching ARP Responses（缓存ARP相应）



19.10 ARP Message Format

- **Frame Format**

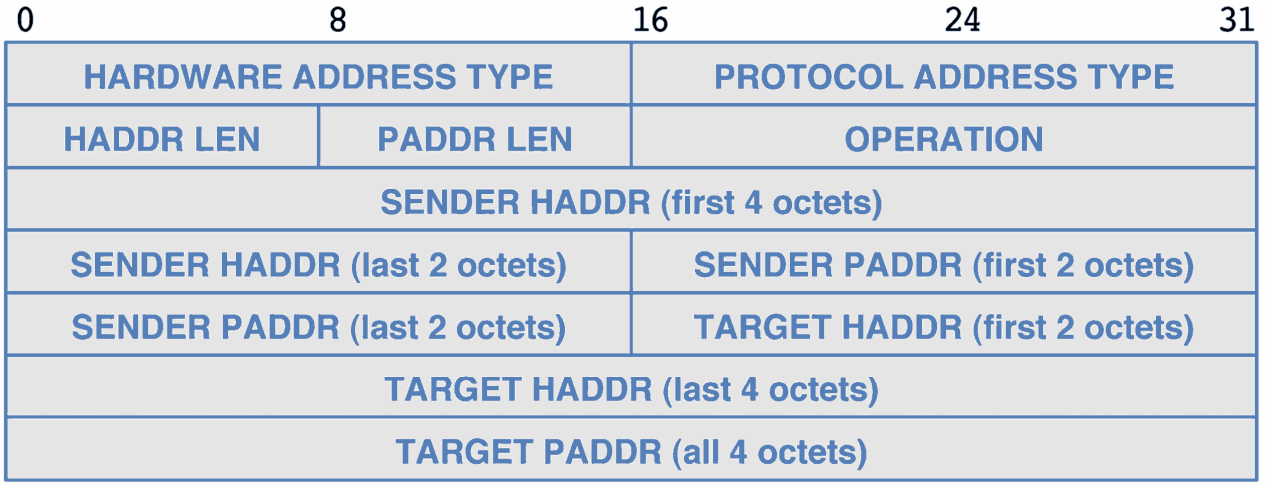


Figure 23.3 The format for an ARP message when binding an IPv4 address to an Ethernet address.

- **ARP Encapsulation**
  - **Frame Type: `0x0806`**

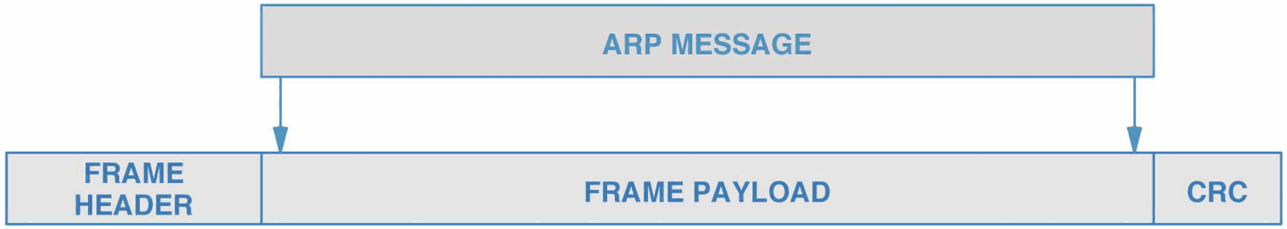


Figure 23.4 Illustration of ARP encapsulation in an Ethernet frame.

ARP路由表

```
C:\Windows\system32>arp -a

接口: 192.168.33.3 --- 0xd
  Internet 地址          物理地址                类型
  192.168.33.6           f8-b1-56-b5-39-bc     动态
  192.168.33.14          9c-21-6a-f6-82-6d     动态
  224.0.0.22             01-00-5e-00-00-16     静态

接口: 192.168.1.1 --- 0x12
  Internet 地址          物理地址                类型
  224.0.0.22             01-00-5e-00-00-16     静态

接口: 169.254.0.1 --- 0x13
  Internet 地址          物理地址                类型
  224.0.0.22             01-00-5e-00-00-16     静态
```

19.10 ARP Message Format

```
Packet Info
  Packet Number:          1
  Flags:                  0x00000000
  Status:                 0x00000000
  Packet Length:          64
  Timestamp:              14:17:23.430079000 04/11/2014
Ethernet Type 2
  Destination:            FF:FF:FF:FF:FF:FF  Ethernet Broadcast  [0-5]
  Source:                 00:0C:29:17:29:CA  VMware:17:29:CA  [6-11]
  Protocol Type:          0x0806  IP ARP  [12-13]
ARP - Address Resolution Protocol
  Hardware:               1  Ethernet (10Mb)  [14-15]
  Protocol:               0x0800  IP  [16-17]
  Hardware Addr Length:   6  [18]
  Protocol Addr Length:   4  [19]
  Operation:              1  ARP Request  [20-21]
  Sender Hardware Addr:   00:0C:29:17:29:CA  VMware:17:29:CA  [22-27]
  Sender Internet Addr:   192.168.7.4  [28-31]
  Target Hardware Addr:   00:00:00:00:00:00  Xerox:00:00:00  (ignored)  [32-37]
  Target Internet Addr:   192.168.7.2  [38-41]
Extra bytes
  Number of bytes:
  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  [42-57]
  ..                00 00  [58-59]
```

19.10 ARP Message Format

```
Packet Info
  Packet Number:          2
  Flags:                  0x00000000
  Status:                 0x00000000
  Packet Length:          64
  Timestamp:              14:17:23.516605000 04/11/2014
Ethernet Type 2
  Destination:            00:0C:29:17:29:CA  VMware:17:29:CA  [0-5]
  Source:                 00:50:56:FC:52:95  VMware:FC:52:95  [6-11]
  Protocol Type:          0x0806  IP ARP  [12-13]
ARP - Address Resolution Protocol
  Hardware:               1  Ethernet (10Mb)  [14-15]
  Protocol:               0x0800  IP  [16-17]
  Hardware Addr Length:   6  [18]
  Protocol Addr Length:   4  [19]
  Operation:              2  ARP Response  [20-21]
  Sender Hardware Addr:   00:50:56:FC:52:95  VMware:FC:52:95  [22-27]
  Sender Internet Addr:   192.168.7.2  [28-31]
  Target Hardware Addr:   00:0C:29:17:29:CA  VMware:17:29:CA  [32-37]
  Target Internet Addr:   192.168.7.4  [38-41]
Extra bytes
  Number of bytes:
  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  [42-57]
  ..                00 00  [58-59]
```

19.14 Processing An Incoming ARP Message

- To replace the previously stored binding.
- To examine the `OPERATION` field.
  - If the message is a request? a response?
- Address resolution software hides the details of physical addressing, allowing software in higher layers to use protocol addressing.

PART III Internetworking

Ch 24 TCP:

Reliable Transport Service

TCP：可靠传输服务

两类应用：如果出错

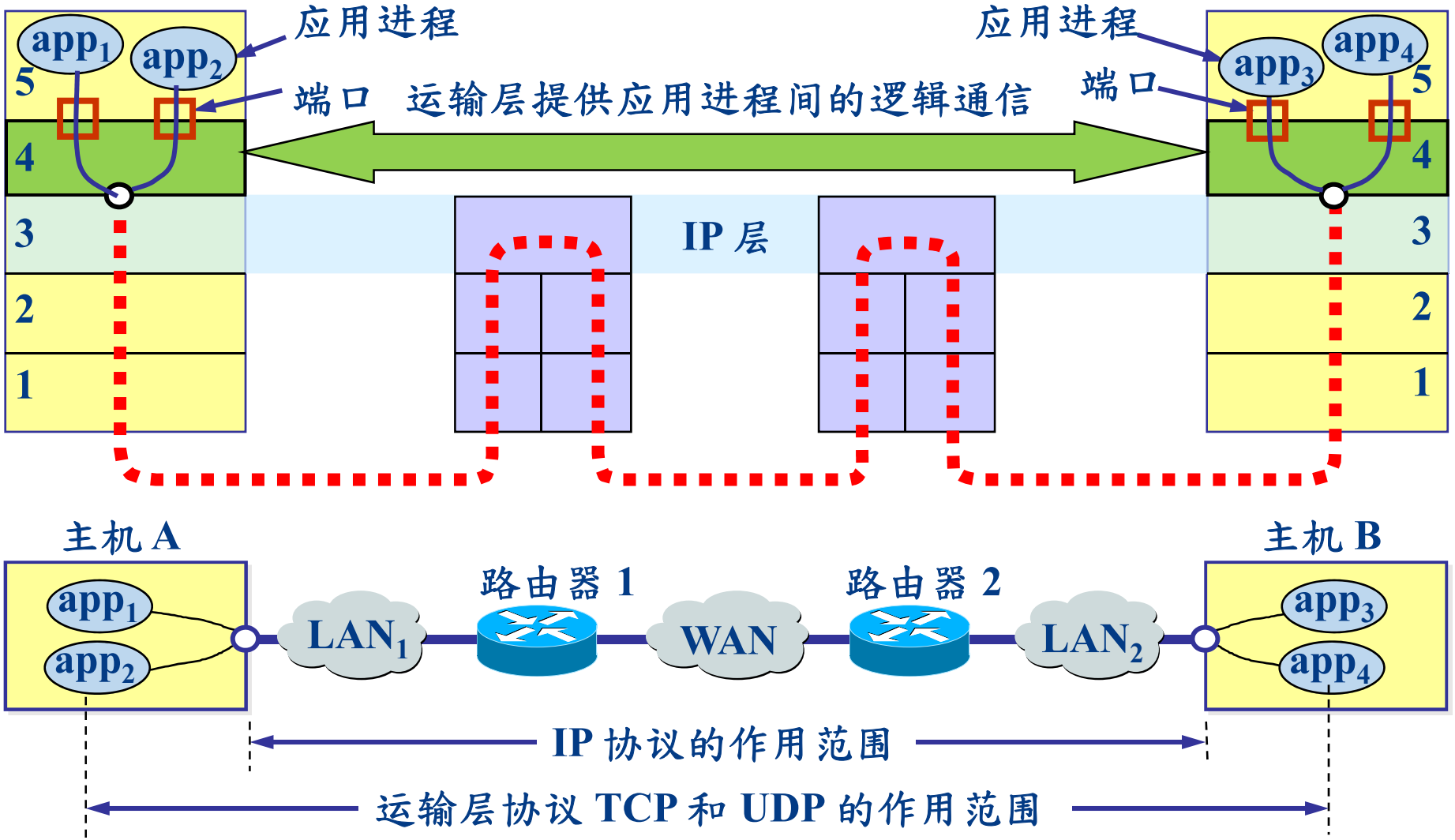
WWW服务器

球赛直播

传输层的必要性

- 网络通信本质上是两个进程的通信，不是主机间通信
- IP地址唯一标识主机；端口号唯一标识进程
- 传输层提供了进程间的复用和解复用
  - 传输层隐藏了硬件拓扑、路由细节等，使应用程序直接调用其接口，建立一条虚拟的端到端的通信信道

网络进程通信


应用进程
app1
app2
端口
运输层提供应用进程间的逻辑通信
应用进程
app3
app4
端口
5
4
3
2
1
IP 层
主机 A
主机 B
路由器 1
路由器 2
LAN1
WAN
LAN2
IP 协议的作用范围
运输层协议 TCP 和 UDP 的作用范围

运输层的两个主要协议

协议	UDP [RFC 768]	TCP [RFC 793]
全称	User Datagram Protocol，用户数据报协议	Transmission Control Protocol，传输控制协议
数据单位	TCP 报文段 (segment)	UDP 数据报 (Datagram)
作用	端到端	端到端、字节流
连接	无连接	面向连接
收到确认	接收方不给出确认	收到确认
多方	一对一、一对多、多对多	一对一
长度	任意	每报文不超过64KB
优点	高效	安全
比喻	发电报（短信）	打电话

协议端口号(Protocol Port Number)

- 端口（Port）
  - 软件端口，有别于交换机上的硬件端口
  - 端口号范围：0～65535，16bits
  - 作用：用于标识本机的不同进程
  - 分类：
    - 服务器用：熟知端口号：0～1023；登记端口号：1024～49151
    - 客户端用：49152（0xC000）～65535
    - 参考：www.iana.org

TCP与UDP端口（部分）

端口	描述	状态
0/TCP,UDP	保留端口；不使用（若发送过程不准备接受回复消息，则可以作为源端口）	官方
1/TCP,UDP	TCPMUX（传输控制协议端口服务多路开关选择器）	官方
5/TCP,UDP	RJE（远程作业登录）	官方
7/TCP,UDP	ECHO（回显）协议	官方
9/TCP,UDP	DISCARD（丢弃）协议	官方
11/TCP,UDP	SYSTAT协议	官方
13/TCP,UDP	DAYTIME协议	官方
15/TCP,UDP	NETSTAT协议	官方
17/TCP,UDP	QOTD（Quote of the Day，每日引用）协议	官方
18/TCP,UDP	消息发送协议	官方
19/TCP,UDP	CHARGEN（字符发生器）协议	官方
20/TCP,UDP	文件传输协议 - 默认数据端口	官方
21/TCP,UDP	文件传输协议 - 控制端口	官方
22/TCP,UDP	SSH (Secure Shell) - 远程登录协议，用于安全登录文件传输（SCP，SFTP）及端口重新定向	官方
23/TCP,UDP	Telnet 终端仿真协议 - 未加密文本通信	官方
25/TCP,UDP	SMTP（简单邮件传输协议）- 用于邮件服务器间的电子邮件传递	官方
更多请上网查询		

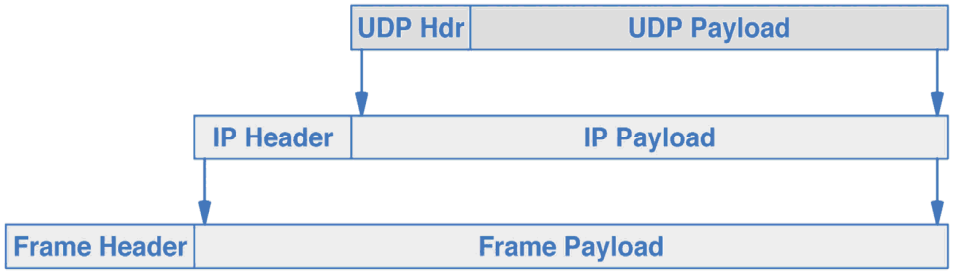
PART III Internetworking

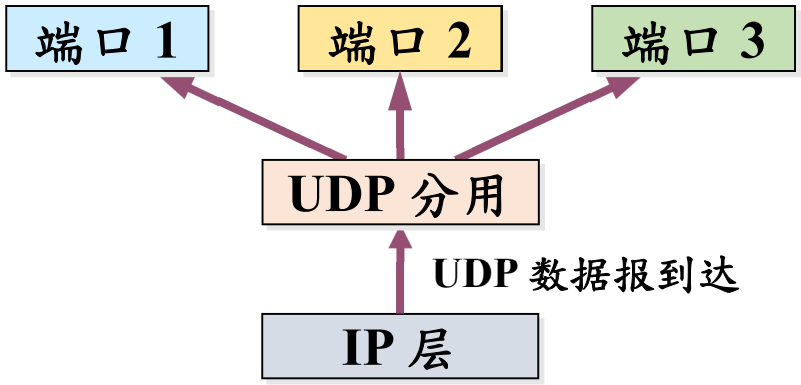
用户数据报协议

（UDP）

UDP概述

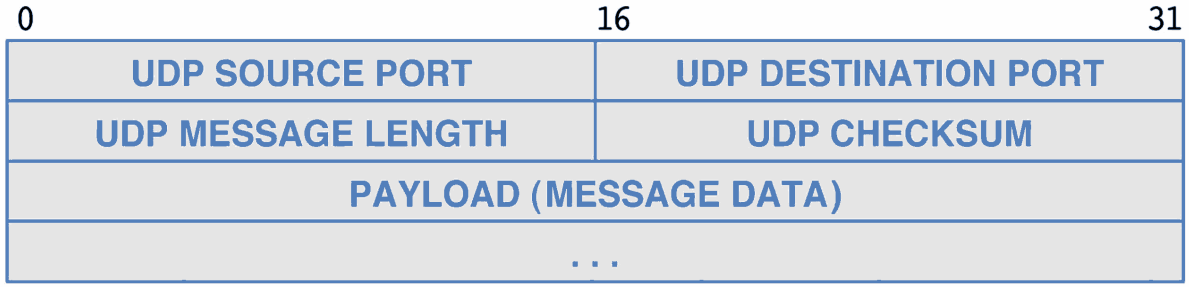
- 无连接：发送前不需要建立连接，发送后无可释放。
- 尽力交付：不保证可靠交付，同时也不使用拥塞控制。
- 面向报文的
  - 对应用层交下来的报文不合并、不拆分
  - 应用程序必须选择合适大小的报文，避免拆分



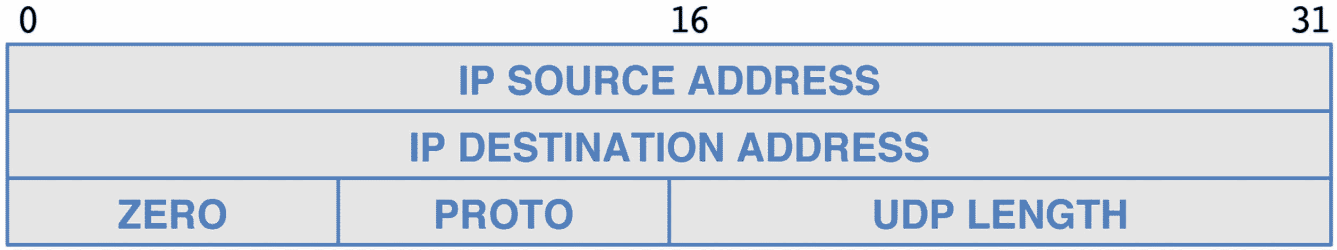


UDP分组结构

- 报文格式



  - 源端口：16 bits
  - 目的端口：16 bits
  - 数据报文长度（单位：字节）：最小为8，即只有头部
  - 校验和：不校正，错误即丢
    - 组成：伪首部（12B）、UDP首部（8B）、数据
    - 伪报文头



使用UDP的场合

- **UDP offers best-effort delivery semantics as IP**
  - **A UDP message can be lost, duplicated, delayed, delivered out-of-order or bits can be corrupted in transit.**
- 场合
  - 图像缺几个像素
  - 音频缺几个帧（MP3 44.1kHz，每帧26ms）
  - 音视频缺几个画面
  - 远程桌面连接丢失几个包
  - 丢包损失不大，应用层可以控制丢包

应用

- **Transaction-oriented**
  - **Simple query-response protocols**
    - **Domain name system or network time protocol.**
- **Provides datagrams**
  - **Modeling other protocols**
    - **IP tunneling, remote procedure call, the network file system.**
- **Simple**
  - **Bootstrapping or other purposes without full protocol stack**
    - **DHCP and trivial file transfer protocol**

应用（续）

- **Stateless**
  - **Very large numbers of clients**
    - **Streaming media applications for example IPTV**
- **Lack of retransmission delays**
  - **Real-time applications**
    - **Voice over IP, online games, and many protocols built on top of the real time streaming protocol.**
- **Good at unidirectional communication**
  - **Broadcast information in many kinds of service discovery and shared information**
    - **Broadcast time or routing information protocol**

PART III Internetworking

传输控制协议

（TCP）

24.2 The Need for Reliable Transport

- IP协议面临的问题：丢包、重复、乱序、数据损坏等。
- 部分程序需要可靠的传输
  - 传输信道不发生差错，如果出错要“抛出异常”
  - 不管多快的速度发送数据，接收端都来得及处理
- 传输控制协议（TCP），RFC 793
  - TCP provides a completely reliable (完全可靠) (no data duplication or loss), connection-oriented (面向连接), full-duplex (全双工) stream (流) transport service.

24.4 The Service TCP Provides to Applications

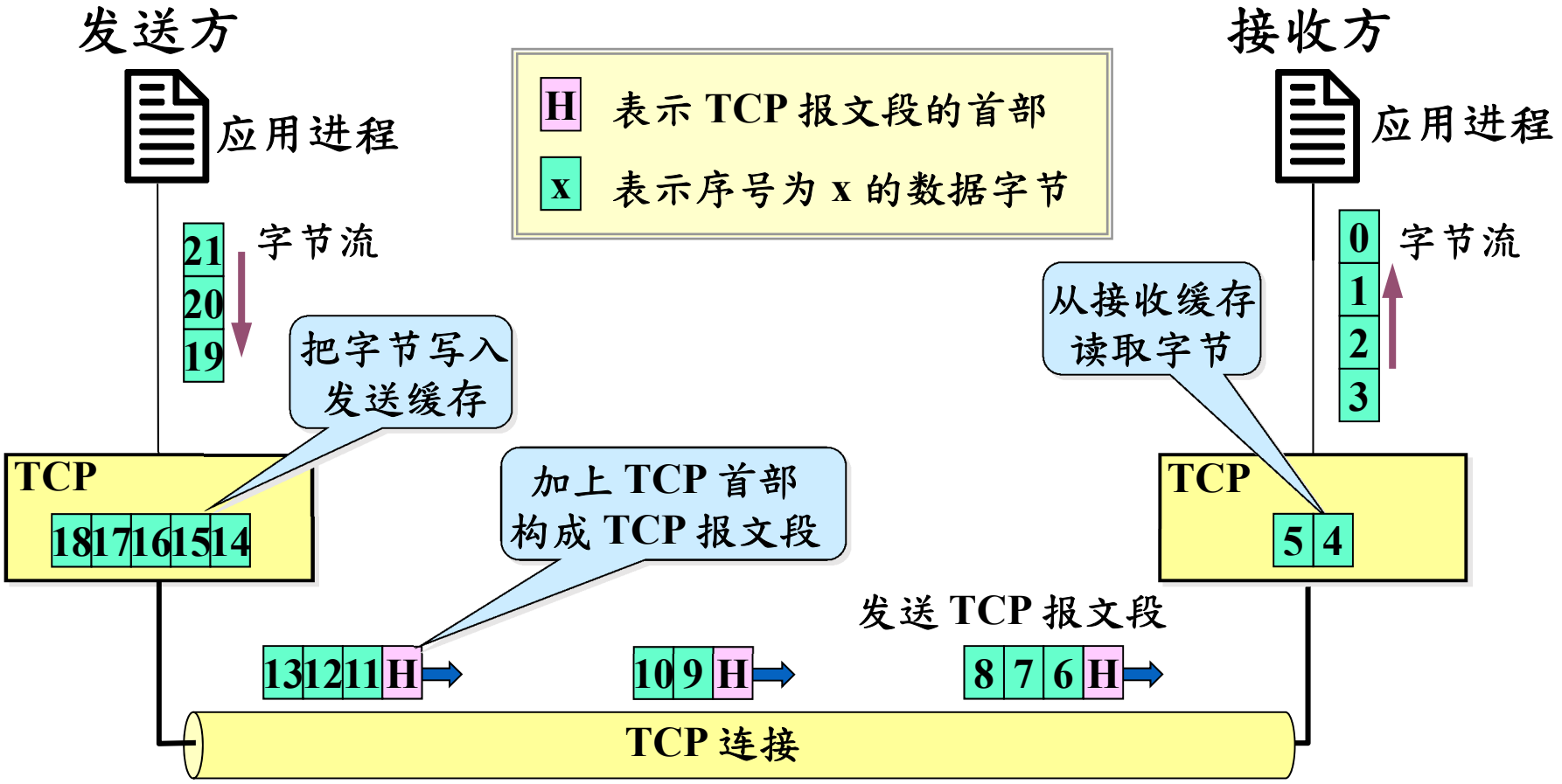
- Connection orientation（面向连接）
- Point-To-point communication（点对点通信）
- Complete reliability（完整可靠性）
- Full duplex communication（全双工通信）
- Stream interface（流接口）
  - 流读出、写入，信息不丢失
- Reliable connection startup（可靠的连接建立）
- Graceful connection shutdown（友好的连接关闭）

TCP最主要的特点

- TCP 是面向连接的运输层协议
- TCP 连接点对点，只能有两个端点（endpoint）
- TCP 提供可靠交付的服务
  - 无差错、不丢失、不重复、按序到达
- TCP 提供全双工通信（收发两端有缓存）
- 面向字节流
  - 流（Stream）：流入到进程或从进程流出的字节序列
  - 收发包数据块大小一致，数量不一致

TCP最主要的特点

- TCP 连接是一条虚连接而不是一条真正的物理连接。

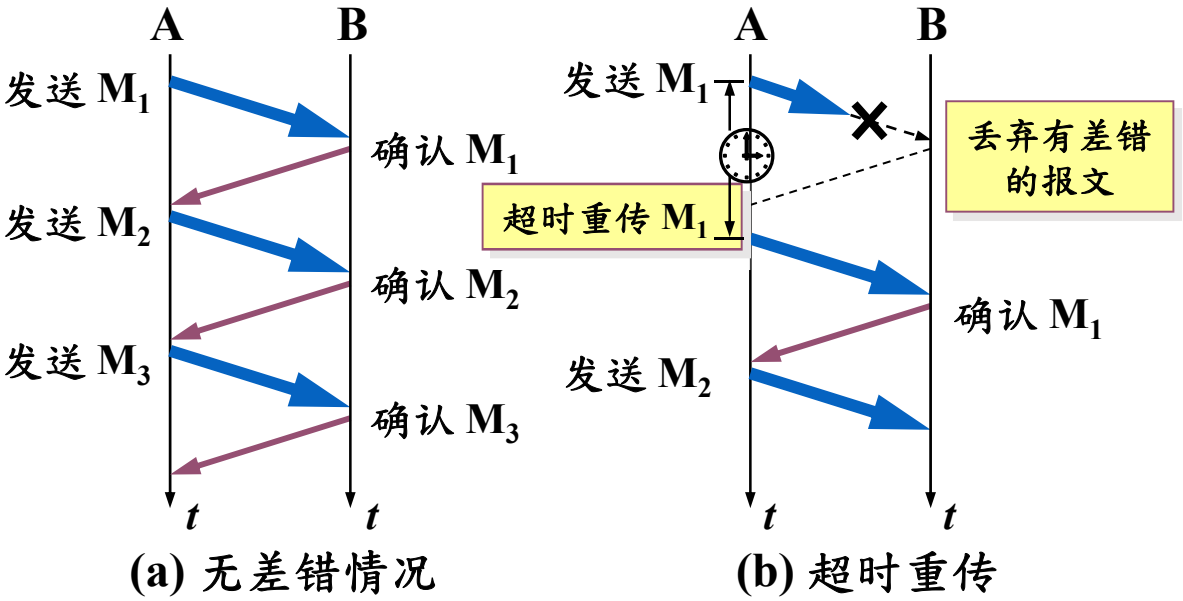


虚连接

- TCP 连接是一条虚连接而不是一条真正的物理连接。
  - TCP 对应用进程一次把多长的报文发送到TCP 的缓存中是不关心的。
  - TCP 根据对方给出的窗口值和当前网络拥塞的程度来决定一个报文段应包含多少个字节（UDP 发送的报文长度是应用进程给出的）。
  - TCP 可把太长的数据块划分短一些再传送。TCP 也可等待积累有足够多的字节后再构成报文段发送出去。

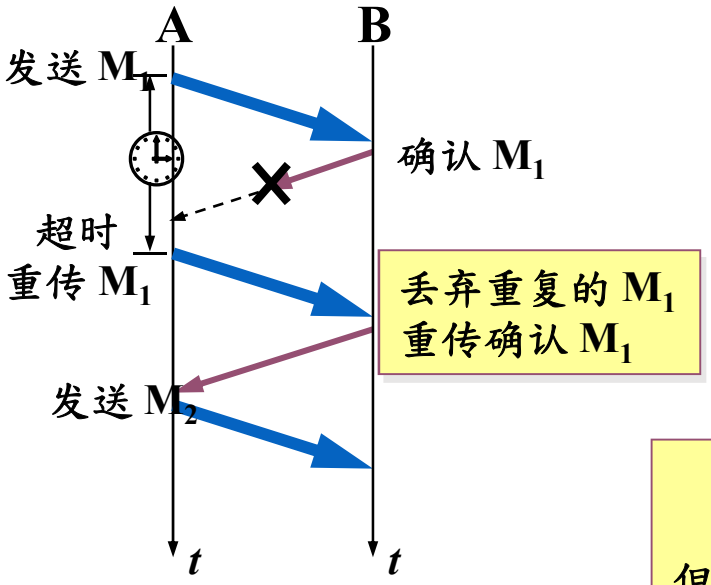
停止-等待协议的工作原理

- 无差错情况：发送、停止、等待
- 有差错情况：设置超时计时器
  - 发送以后，保留副本（万一需要重传）
  - 分组编号
  - 重传时间比平均往返时间长一些
    - 不确定因素：拥塞、路径

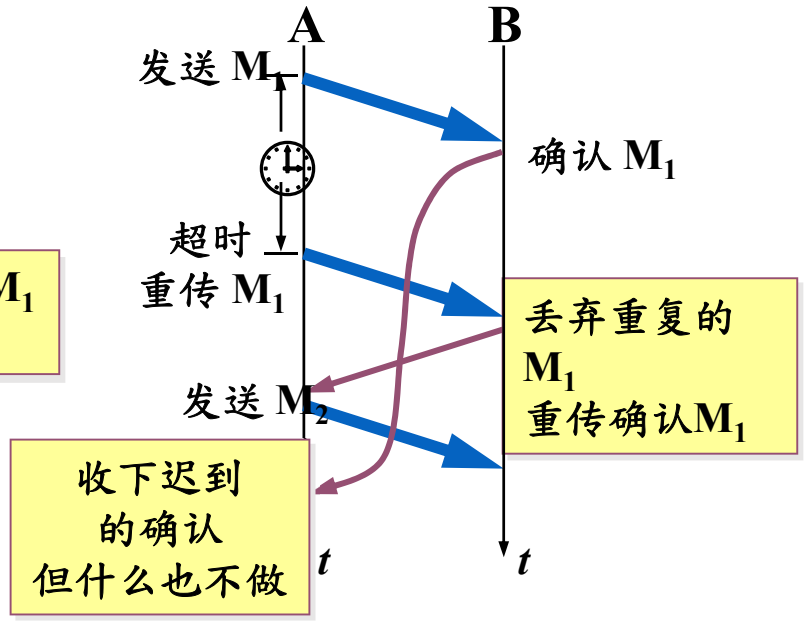


停止-等待协议的工作原理

- 确认丢失和确认收到
  - 通过自动重传请求（Automatic Repeat reQuest，ARQ），可以在不可靠的传输网络上实现可靠的通信



(a) 确认丢失



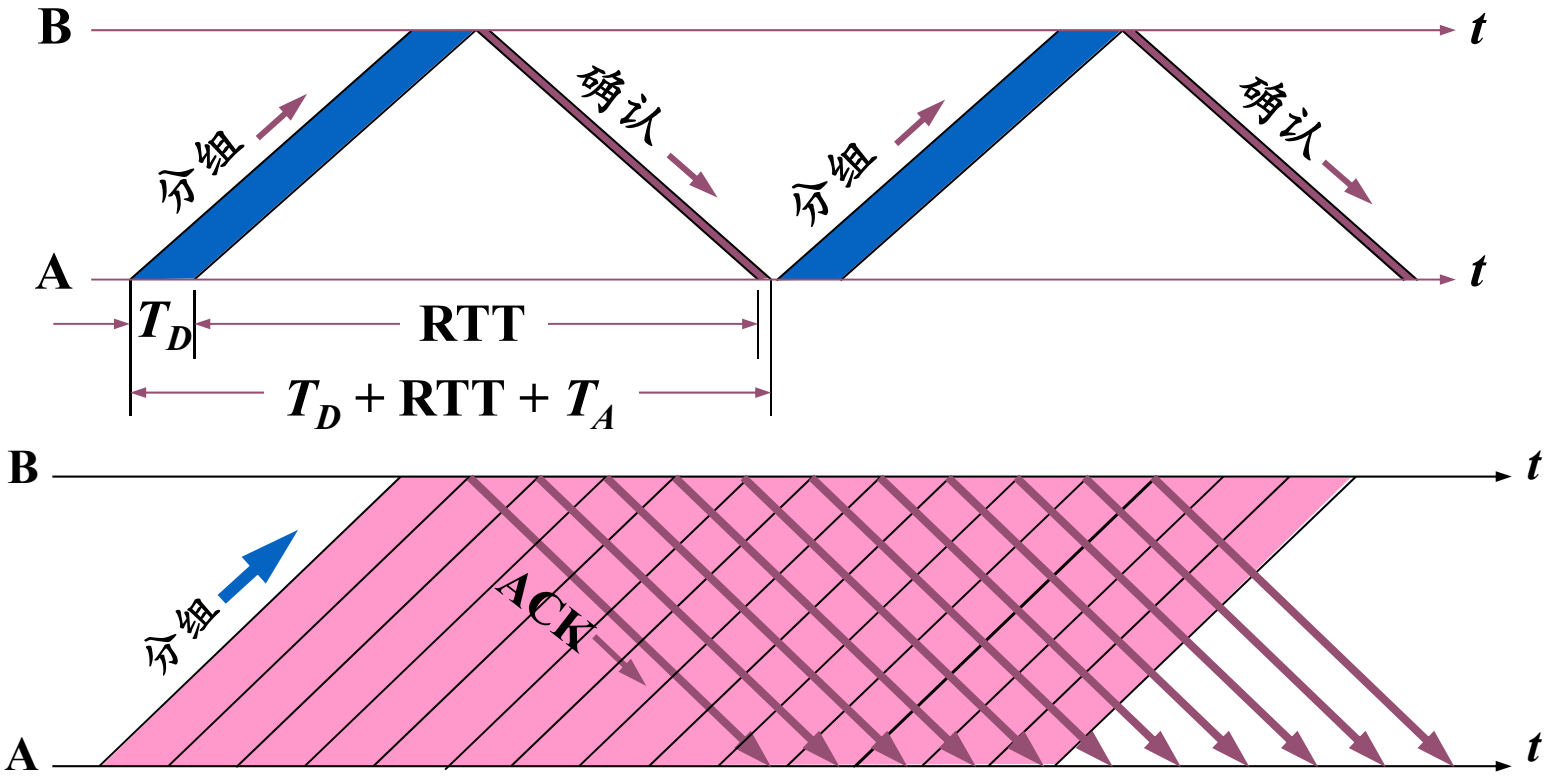
(b) 确认迟到

停止-等待协议的工作原理

- 确认丢失和确认收到
  - 超时未收到确认：分组丢失？确认丢失？
  - 丢弃重复的分组，重新发送收到
  - 一直未收到确认，则线路太差

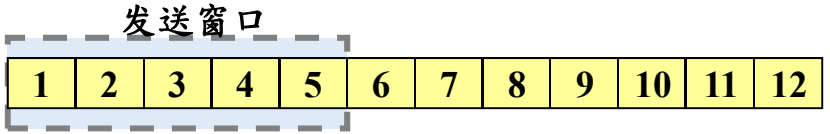
停止-等待协议的工作原理

- 停止等待协议：优点是简单，缺点是信道利用率太低
- 流水线传输提高了利用率

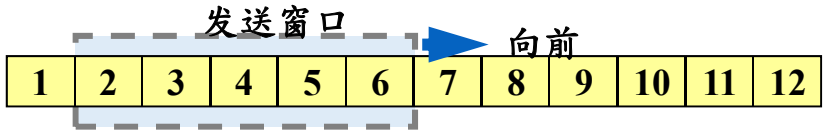


连续ARQ协议

- 发送窗口
  - 逐一确认
    - 窗口内的分组都连续发送出去，不需要等待确认
    - 每收到一个确认，窗口就继续往前推进一格
  - 累积确认
    - 接收方收到几个分组后，对最后一个分组发送确认
    - 例如窗口为5，第3分组丢失，只确认前2个，则3-5需重发



(a) 发送方维持发送窗口（发送窗口是5）



(b) 收到一个确认后发送窗口向前滑动

TCP报文段的首部格式

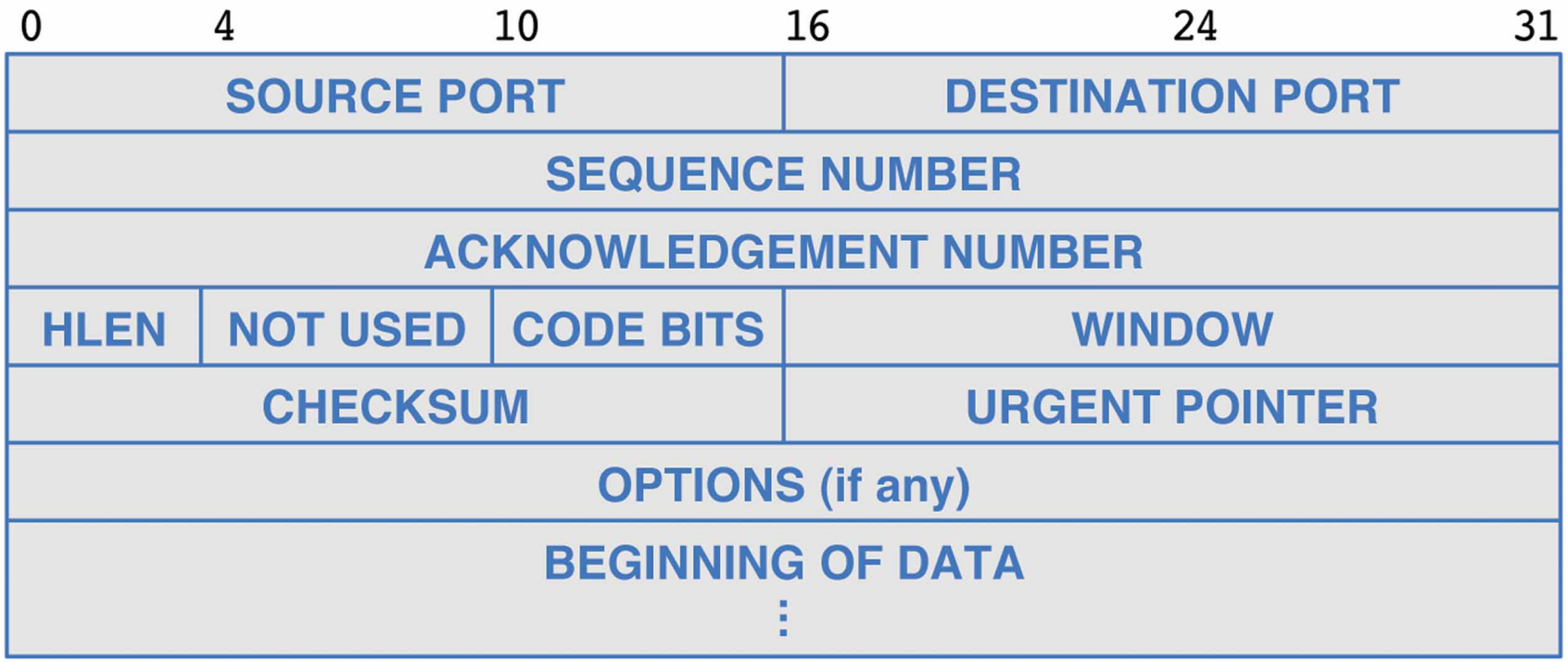


Figure 26.10 The TCP segment format used for both data and control messages.

TCP报文格式

- TCP报文头数据项：基本信息20B
  - 源端口号：16bits；目标端口号：16bits
  - 发送数据序列号（报文段序号）：32bits
    - TCP将每一个字节按顺序编号（与IP不同！）
    - 指的是本报文段第一个字节的的序号
  - 确认序列号：32bits
    - 期望收到下一个字节的序号（而不是已收到的序号），表明之前的字节都已经收到

TCP报文格式

- TCP报文头数据项：基本信息20B
  - 报头长度：4bits，单位4Bytes；
  - 保留位：4bits
  - 标识符：8bits
    - 拥塞窗口下降、ECN回显
    - 紧急指针：优先传输；ACK字段：确认号有效
    - 数据前推：不等待缓存满了再一起推送
    - 连接复位：拒绝连接；序号同步：建立连接时用来同步信号
    - 终止连接：数据已发送完毕，释放信号

TCP报文格式

- TCP报文头数据项：基本信息20B
  - 滑动窗口缓冲区大小：16bits
  - 校验和：16bits；紧急指针：32bits
  - 选项字段：变长
    - 最大报文段长度
    - 窗口扩大选项
    - 时间戳

TCP报文头部

```
  Timestamp:            22:28:39.376666700 04/21/2015
Ethernet Type 2
  Destination:          F8:B1:56:**:**:**  [0-5]
  Source:               9C:21:6A:**:**:**  [6-11]
  Protocol Type:        0x0800  IP  [12-13]
IP Version 4 Header - Internet Protocol Datagram
  Version:              4  [14 Mask 0xF0]
  Header Length:        5  (20 bytes)  [14 Mask 0x0F]
  Diff. Services:       %00000000  [15]
                        0000 00.. Default
                        .... ..00 Not-ECT
  Total Length:         141  [16-17]
  Identifier:           186  [18-19]
  Fragmentation Flags:  %010  [20 Mask 0xE0]
                        0.. Reserved
                        .1. Do Not Fragment
                        ..0 Last Fragment
  Fragment Offset:      0  (0 bytes)  [20-21 Mask 0x1FFF]
  Time To Live:         54  [22]
  Protocol:             6  TCP - Transmission Control Protocol  [23]
  Header Checksum:      0x28E6  [24-25]
  Source IP Address:    111.187.**.**  [26-29]
  Dest. IP Address:     192.168.**.**  [30-33]
```

TCP报文头部（续）

```
TCP - Transport Control Protocol
  Source Port:            20919  [34-35]
  Destination Port:       3389  ms-wbt-server  [36-37]
  Sequence Number:        538319976  [38-41]
  Ack Number:             3805334299  [42-45]
  TCP Offset:             5  (20 bytes)  [46 Mask 0xF0]
  Reserved:               %0000  [46 Mask 0x0F]
  TCP Flags:              %00011000  ...AP...  [47]
                          0... .... (No Congestion Window Reduction)
                          .0.. .... (No ECN-Echo)
                          ..0. .... (No Urgent pointer)
                          ...1 .... Ack
                          .... 1... Push
                          .... .0.. (No Reset)
                          .... ..0. (No SYN)
                          .... ...0 (No FIN)
  Window:                 1353  [48-49]
  TCP Checksum:           0x3B1A  [50-51]
  Urgent Pointer:         0  [52-53]
  No TCP Options
Application Layer
  Data Area:
  ....
```

TCP报文头部

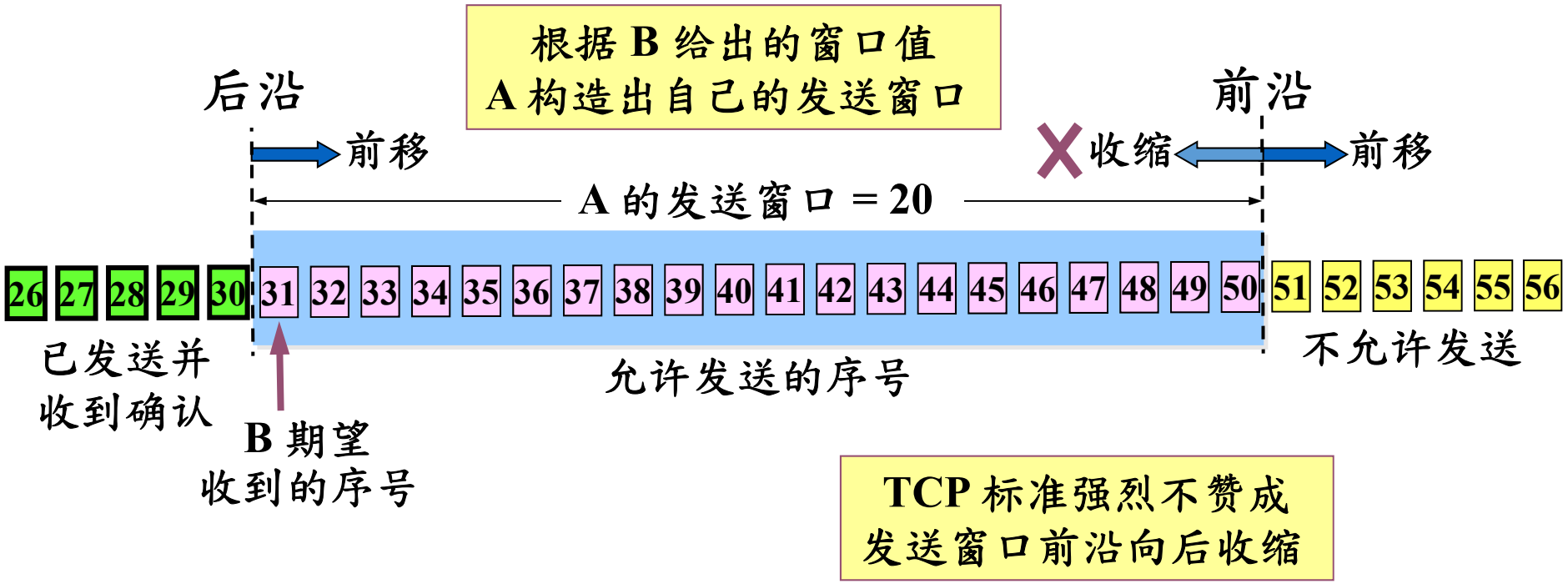
```
  Timestamp:            22:28:39.376666700 04/21/2015
Ethernet Type 2
  Destination:          9C:21:6A:**:**:**  [0-5]
  Source:               F8:B1:56:**:**:**  [6-11]
  Protocol Type:        0x0800  IP  [12-13]
IP Version 4 Header - Internet Protocol Datagram
  Version:              4  [14 Mask 0xF0]
  Header Length:        5  (20 bytes)  [14 Mask 0x0F]
  Diff. Services:       %00000000  [15]
                        0000 00.. Default
                        .... ..00 Not-ECT
  Total Length:         40  [16-17]
  Identifier:           25622  [18-19]
  Fragmentation Flags:  %010  [20 Mask 0xE0]
                        0.. Reserved
                        .1. Do Not Fragment
                        ..0 Last Fragment
  Fragment Offset:      0  (0 bytes)  [20-21 Mask 0x1FFF]
  Time To Live:         128  [22]
  Protocol:             6  TCP - Transmission Control Protocol  [23]
  Header Checksum:      0x7BEE  [24-25]
  Source IP Address:    192.168.**.**  [26-29]
  Dest. IP Address:     111.187.**.**  [30-33]
```

TCP报文头部（续）

```
TCP - Transport Control Protocol
  Source Port:            3389  ms-wbt-server  [34-35]
  Destination Port:       20919  [36-37]
  Sequence Number:        3805334299  [38-41]
  Ack Number:             538320077  [42-45]
  TCP Offset:             5  (20 bytes)  [46 Mask 0xF0]
  Reserved:               %0000  [46 Mask 0x0F]
  TCP Flags:              %00010000  ...A....  [47]
                          0... .... (No Congestion Window Reduction)
                          .0.. .... (No ECN-Echo)
                          ..0. .... (No Urgent pointer)
                          ...1 .... Ack
                          .... 0... (No Push)
                          .... .0.. (No Reset)
                          .... ..0. (No SYN)
                          .... ...0 (No FIN)
  Window:                 62735  [48-49]
  TCP Checksum:           0x5635  [50-51]
  Urgent Pointer:         0  [52-53]  No TCP Options
Extra bytes
  Number of bytes:  ......              00 00 00 00 00 00  [54-59]
FCS - Frame Check Sequence
  FCS:                    0x7D0900B8  Calculated
```

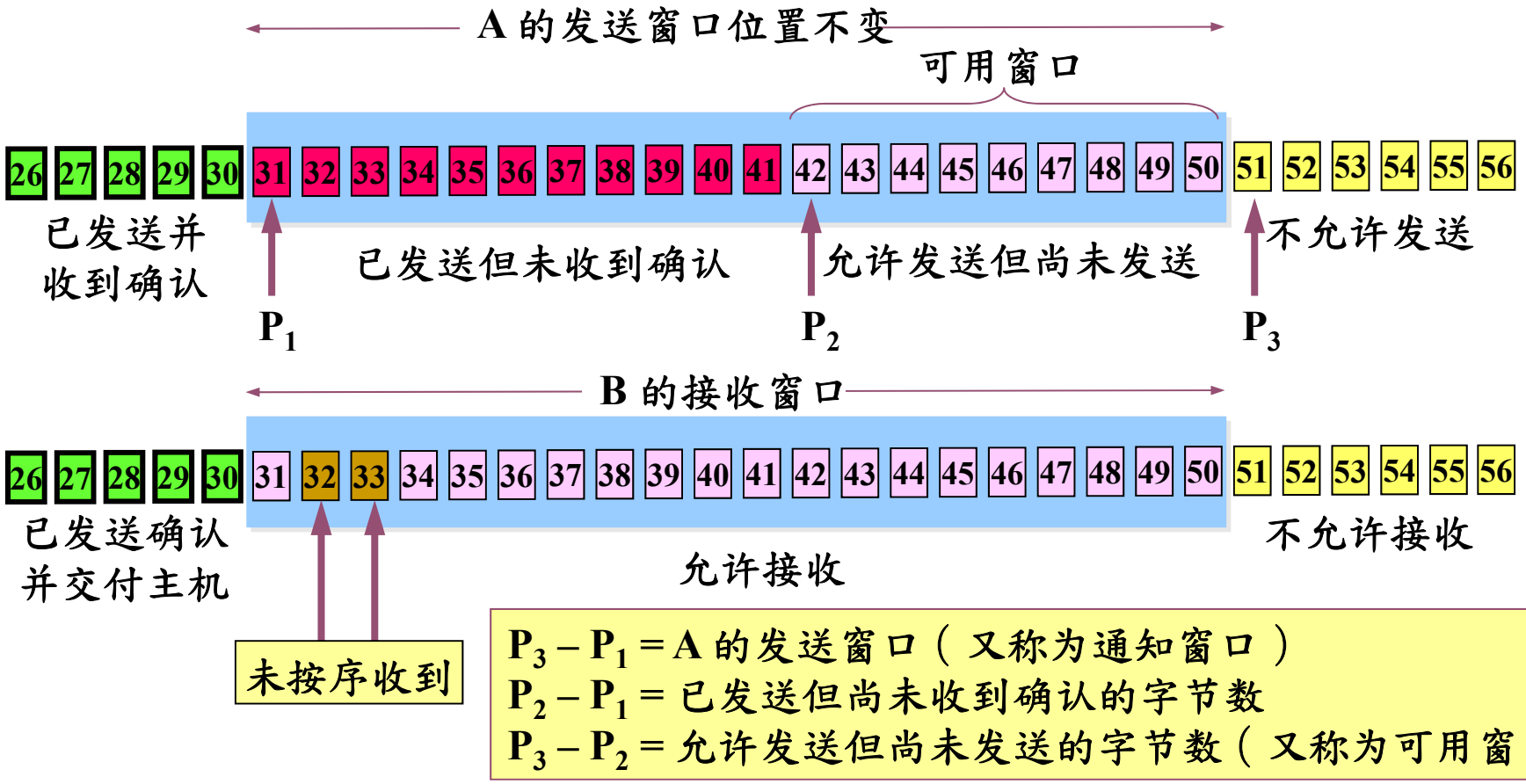
TCP机制：字节为单位的滑动窗口

- 以字节为单位的滑动窗口



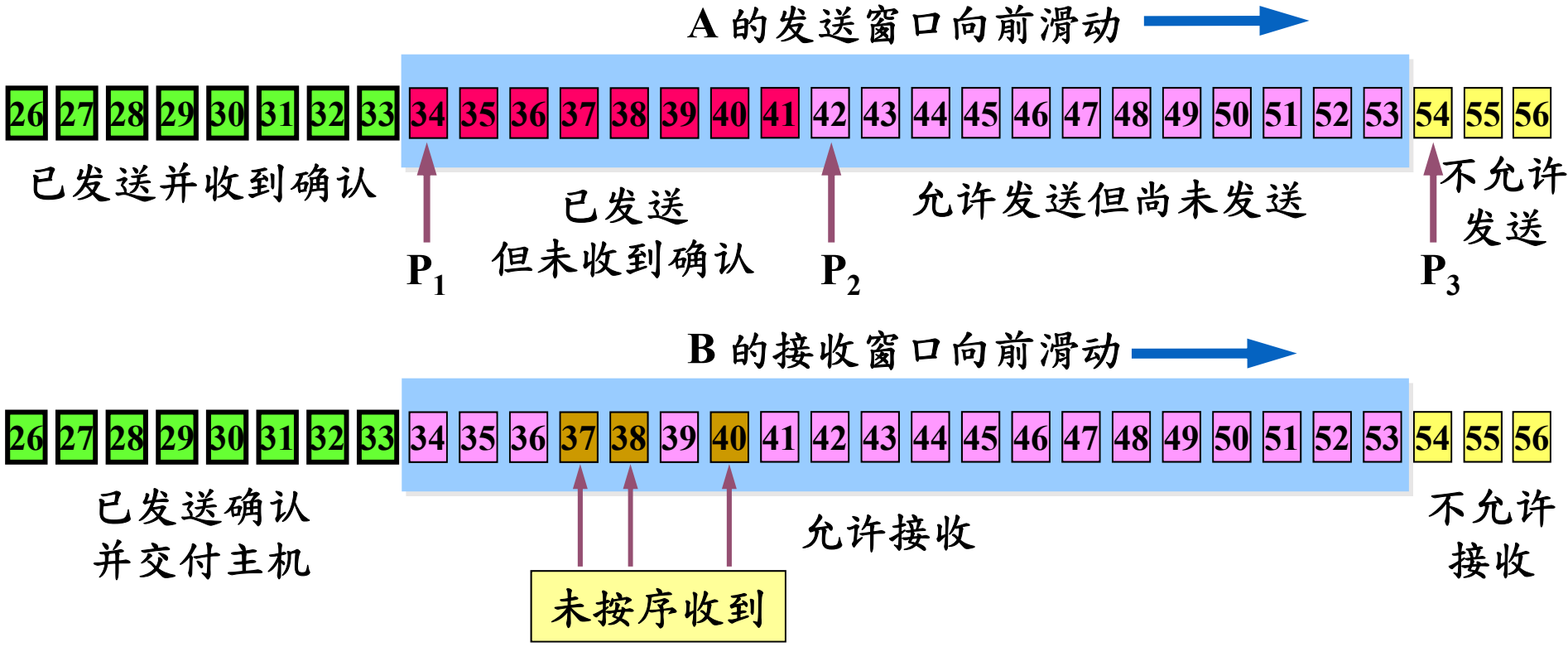
TCP机制：字节为单位的滑动窗口

- A 发送了 11 个字节的数据



TCP机制：字节为单位的滑动窗口

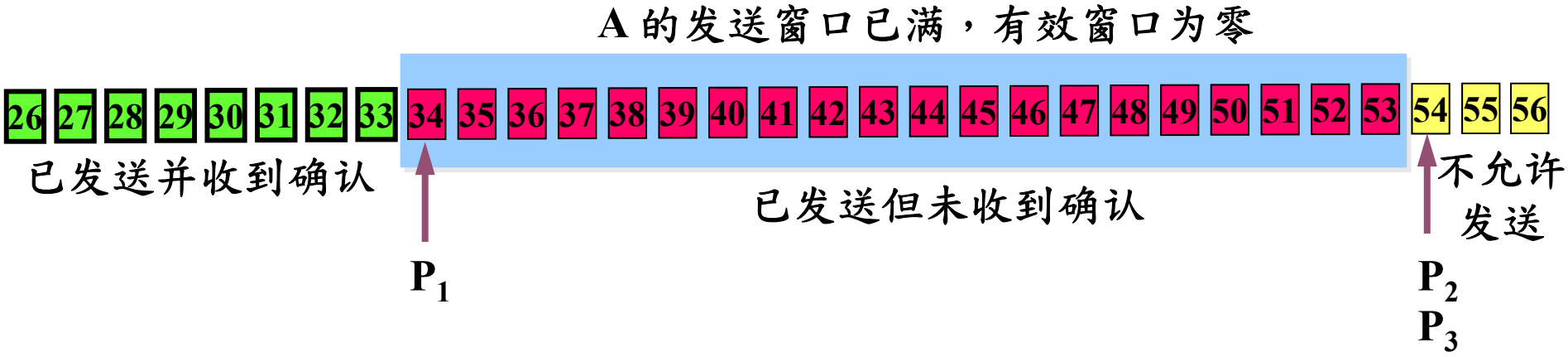
- A 收到新的确认号，发送窗口向前滑动



先存下，等待缺少的数据的到达

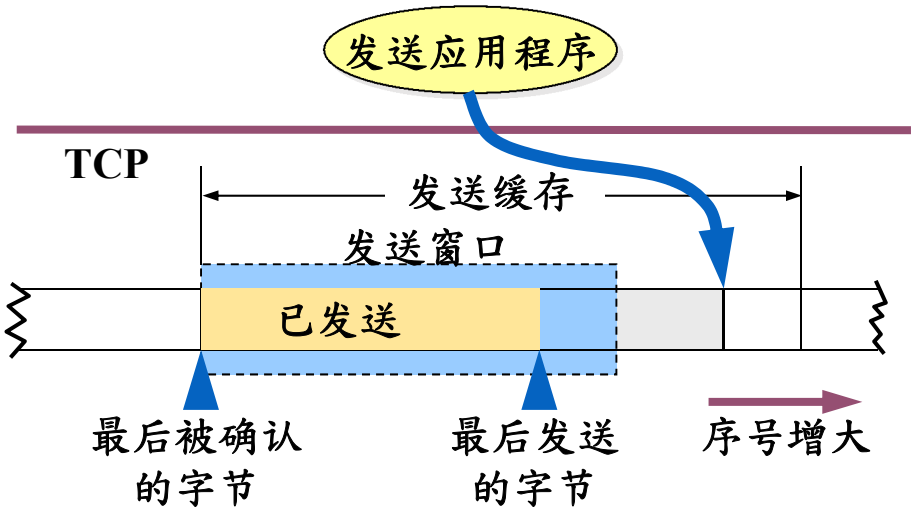
TCP机制：字节为单位的滑动窗口

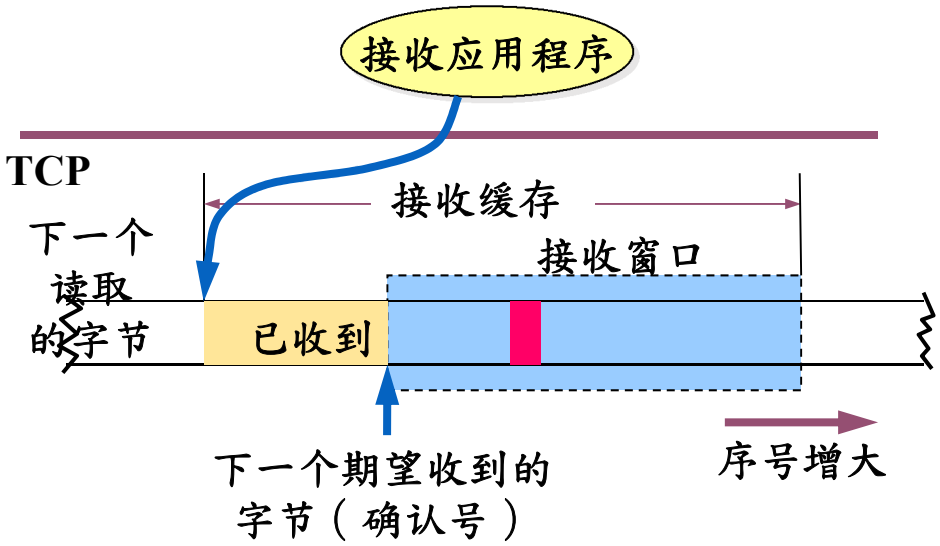
- A 的发送窗口内的序号都已用完，但还没有再收到确认，必须停止发送。



TCP机制：字节为单位的滑动窗口

- A 的发送窗口内的序号都已用完，但还没有再收到确认，必须停止发送。





TCP机制：字节为单位的滑动窗口

- 发送缓存与接收缓存的作用
  - 发送应用程序传送给发送方 TCP 准备发送的数据；
  - TCP 已发送出但尚未收到确认的数据。
- 接收缓存用来暂时存放：
  - 按序到达的、但尚未被接收应用程序读取的数据；
  - 不按序到达的数据。

TCP机制：字节为单位的滑动窗口

- A 的发送窗口并不总是和 B 的接收窗口一样大（因为有一定的时间滞后）。
- TCP 标准没有规定对不按序到达的数据应如何处理。通常是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程。
- TCP 要求接收方必须有累积确认的功能，这样可以减小传输开销。

TCP机制：超时重传时间的选择

- TCP 每发送一个报文段，就对这个报文段设置一次计时器。只要计时器设置的重传时间到但还没有收到确认，就要重传这一报文段。
- 由于 TCP 的下层是一个互联网环境，IP 数据报所选择的路由变化很大。因而运输层的往返时间的方差也很大。
- TCP 保留了 RTT 的一个加权平均往返时间 RTT_S（这又称为平滑的往返时间）。

TCP机制：超时重传时间的选择

- 第一次测量到 RTT 样本时，RTT_S 值就取为所测量到的 RTT 样本值。以后每测量到一个新的 RTT 样本，就按下式重新计算一次 RTT_S：

  新的 $RTT_S = (1 - \alpha) \times$ (旧的 RTT_S) $+ \alpha \times$ (新的 RTT 样本)

  – 式中，$0 \le \alpha < 1$。若 α 很接近于零，表示 RTT 值更新较慢。若选择 α 接近于 1，则表示 RTT 值更新较快。

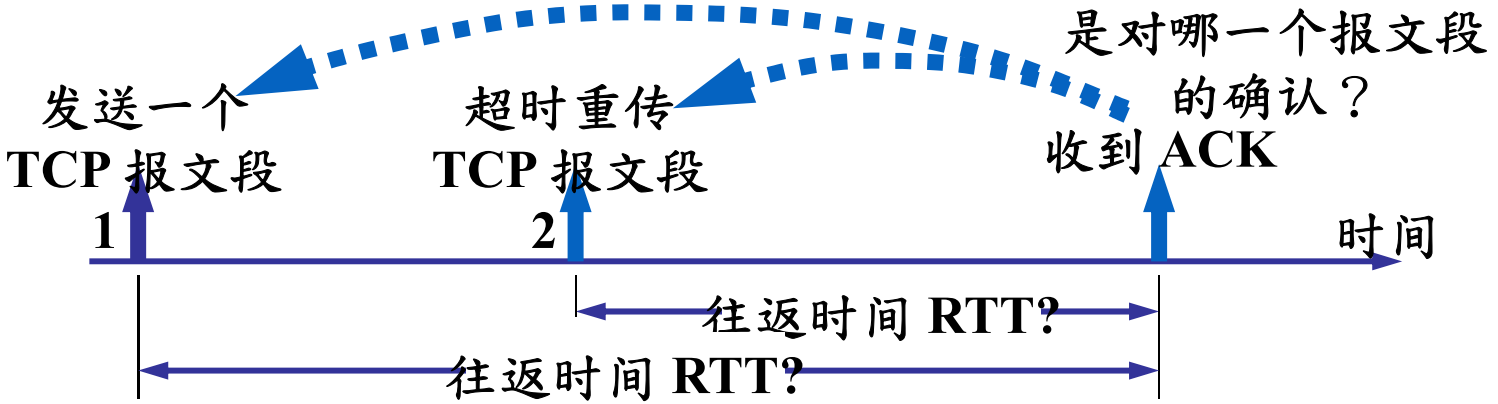
  – RFC 2988 推荐的 α 值为 1/8，即 0.125。

TCP机制：超时重传时间的选择

- 超时重传时间（Retransmission Time-Out，RTO）应略大于上面得出的加权平均往返时间 RTT_S。
  - $RTO = RTT_S + 4 \times RTT_D$ (RFC 2988)
  - RTT_D 是 RTT 的偏差的加权平均值。
  - 第一次测量时，RTT_D 值取为测量到的 RTT 样本值的一半。在以后的测量中，则使用下式计算加权平均的 RTT_D：
  - 新 $RTT_D = (1 - \beta) \times (\text{旧}RTT_D) + \beta \times |RTT_S - \text{新 RTT样本}|$
    - β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

TCP机制：超时重传时间的选择

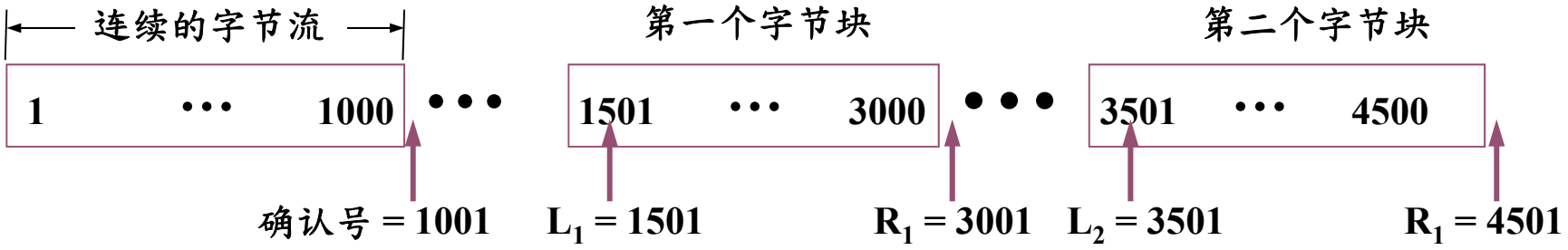
- 误判重传确认和原报文的确认



- Karn算法
  - 只要报文段重传了，就不采用其往返时间样本。
  - 超时重传时间更新：每重传一次RTO乘以2，不重传时恢复

TCP机制：选择确认SACK

- 接收方收到了和前面的字节流不连续的两个字节块。
- 如果这些字节的序号都在接收窗口之内，那么接收方就先收下这些数据，在选项区域把这些信息准确地告诉发送方，使发送方不再重复已收到的数据。



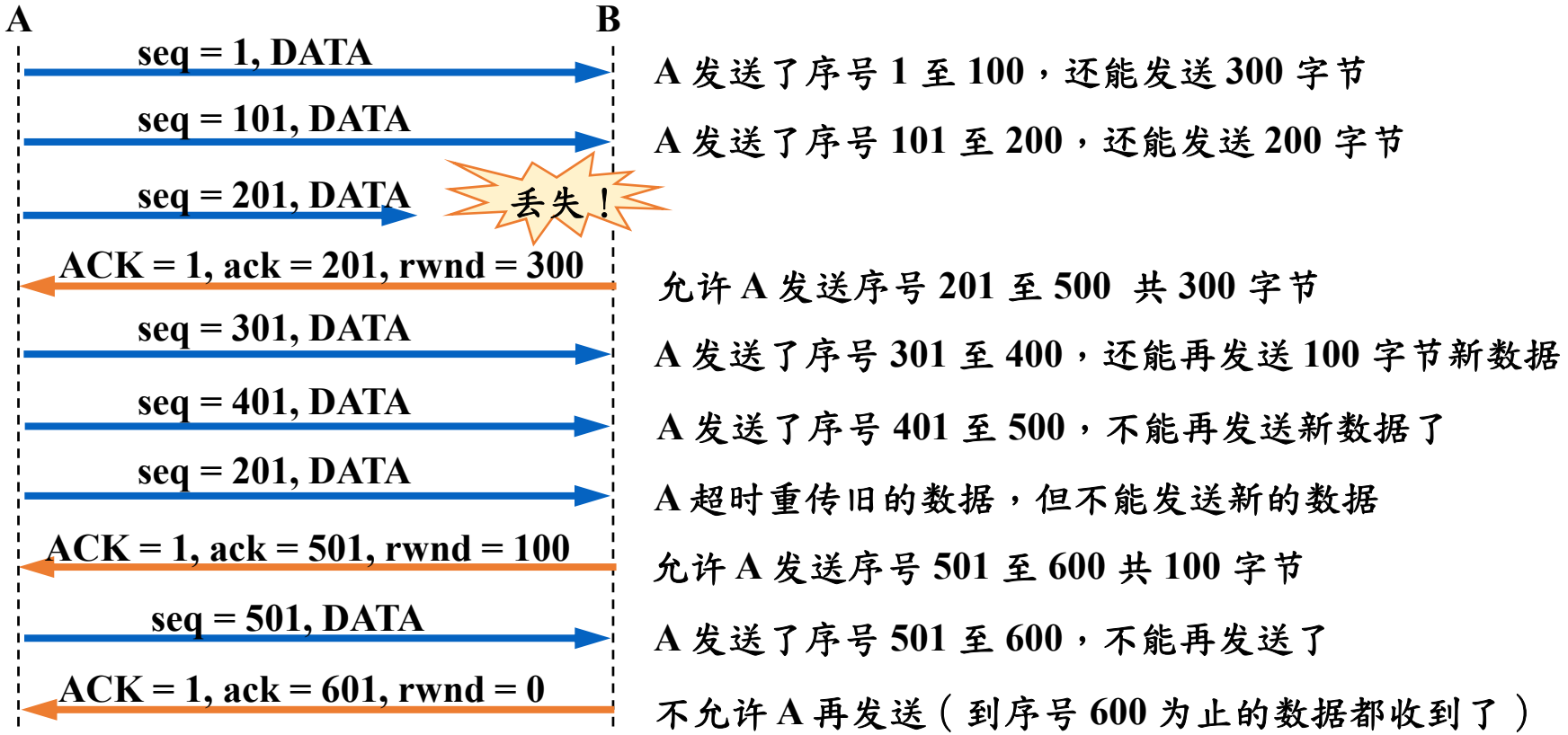
TCP流量控制：滑动窗口

- 但如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失。
- 流量控制（flow control）的目的是使发送方的发送速率，让接收方来得及接收，且不使网络发生拥塞。
- 利用滑动窗口机制可以很方便地在TCP连接上实现流量控制。
- TCP窗口单位是字节，不是报文段。
  - 发送方不能超过接收方给出的接受窗口值

TCP流量控制：滑动窗口举例

- A 向 B 发送数据。在连接建立时，

  B 告诉 A：“我的接收窗口 rwnd = 400（字节）”。



TCP流量控制：滑动窗口举例

- TCP 为每一个连接设有一个持续计时器。
  - 只要一方收到对方的零窗口通知，就启动持续计时器。
- 若持续计时器设置的时间到期，就发送一个零窗口探测报文段（仅携带 1 字节的数据），而对方就在确认这个探测报文段时给出了现在的窗口值。
  - 若窗口仍然是零，则收到这个报文段的一方就重新设置持续计时器。
  - 若窗口不是零，则死锁的僵局就可以打破了。

TCP流量控制：滑动窗口举例

- 传输效率：用不同机制控制 TCP 报文段的发送时机
  - 第一种机制是 TCP 维持一个变量，它等于最大报文段长度 MSS。只要缓存中存放的数据达到 MSS 字节时，就组装成一个 TCP 报文段发送出去。
  - 第二种机制是由发送方的应用进程指明要求发送报文段，即 TCP 支持的推送（push）操作。
  - 第三种机制是发送方的一个计时器期限到了，就把当前已有缓存数据装入报文段（但长度不能超过 MSS）发送出去。

TCP拥塞控制：原理

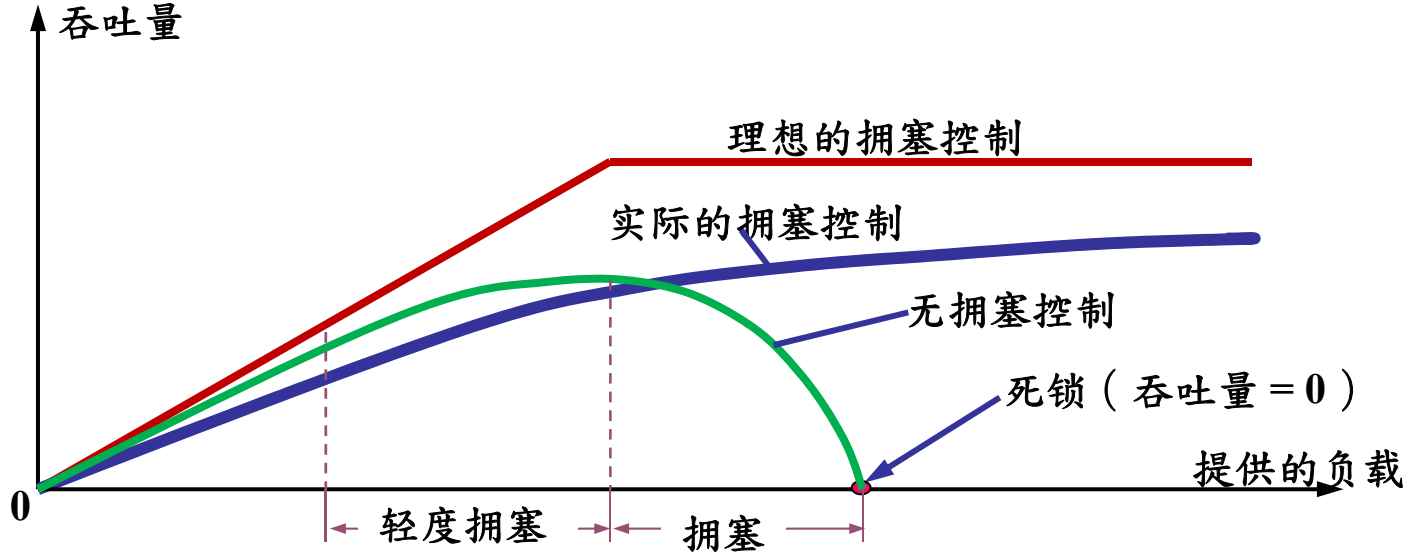
- 在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏——产生拥塞(congestion)。
- 出现资源拥塞的条件：对资源需求的总和＞可用资源
- 若网络中有许多资源同时产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降。

TCP拥塞控制：和流量控制关系

- 拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。
- 拥塞控制是一个全局性的过程，涉及到所有主机、路由器，以及与降低网络传输性能有关的所有因素。
- 流量控制往往指在给定的发送端和接收端之间的点对点通信量的控制。
- 流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

TCP拥塞控制：拥塞控制的作用

- 拥塞控制是一个动态的（而不是静态的）问题
  - 网络高速化，这很容易出现缓存不够大而造成分组的丢失。但分组的丢失是网络发生拥塞的征兆而不是原因。
  - 许多情况下，拥塞控制成为网络性能恶化甚至死锁的原因。



TCP拥塞控制：拥塞控制思路

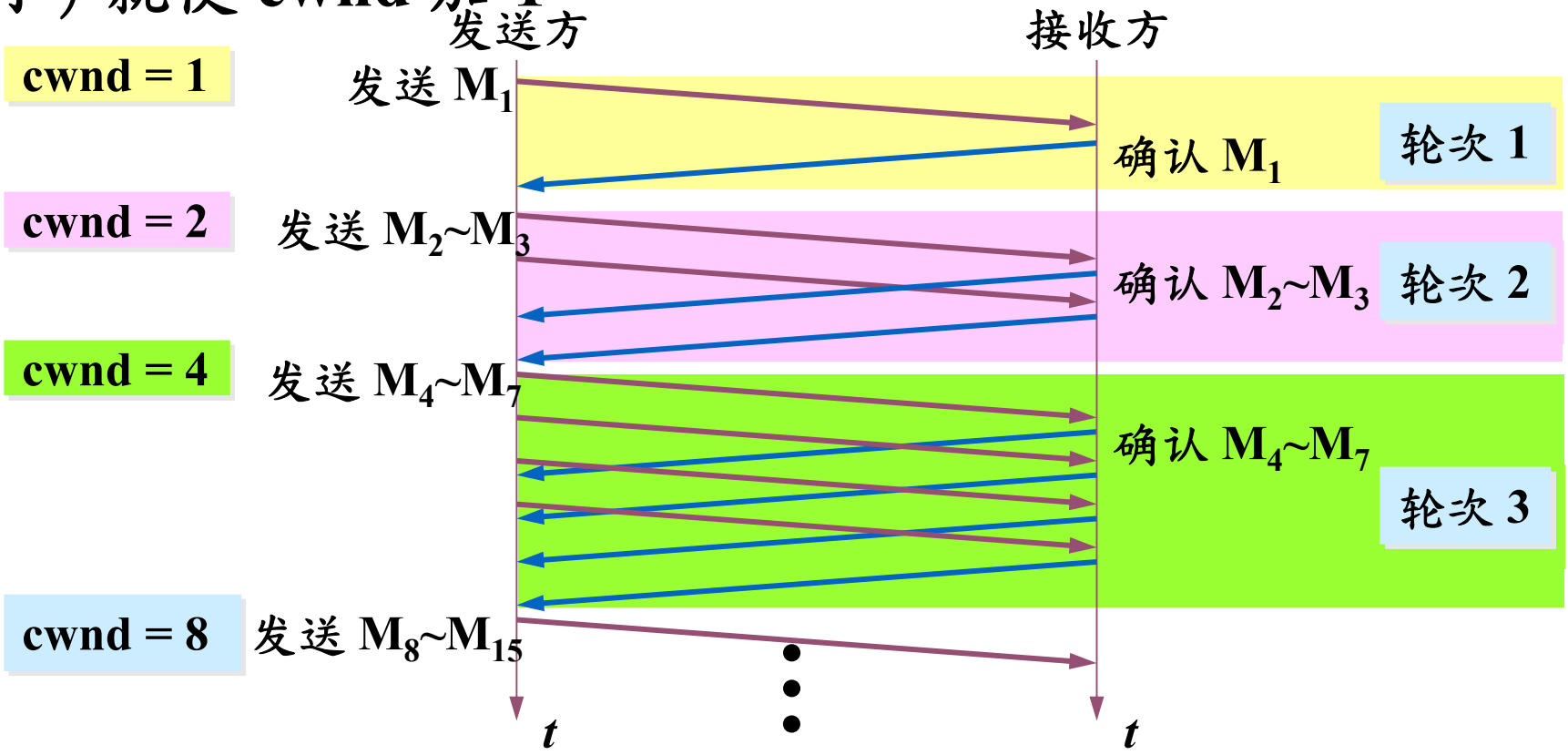
- 开环控制（Open loop）：设计网络事先将有关发生拥塞的因素考虑周到，力求网络在工作时不产生拥塞。
- 闭环控制（Close loop）：基于反馈环路的概念，属于闭环控制的有以下几种措施：
  - 监测网络系统以便检测到拥塞在何时、何处发生
  - 将拥塞发生的信息传送到可采取行动的地方
  - 调整网络系统的运行以解决出现的问题

TCP拥塞控制：拥塞窗口

- 拥塞窗口（congestion window）
  - 发送方维持的状态变量。
  - 拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口。如再考虑到接收方的接收能力，则发送窗口还可能小于拥塞窗口。
- 发送方控制拥塞窗口的原则是
  - 只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

TCP拥塞控制：慢开始

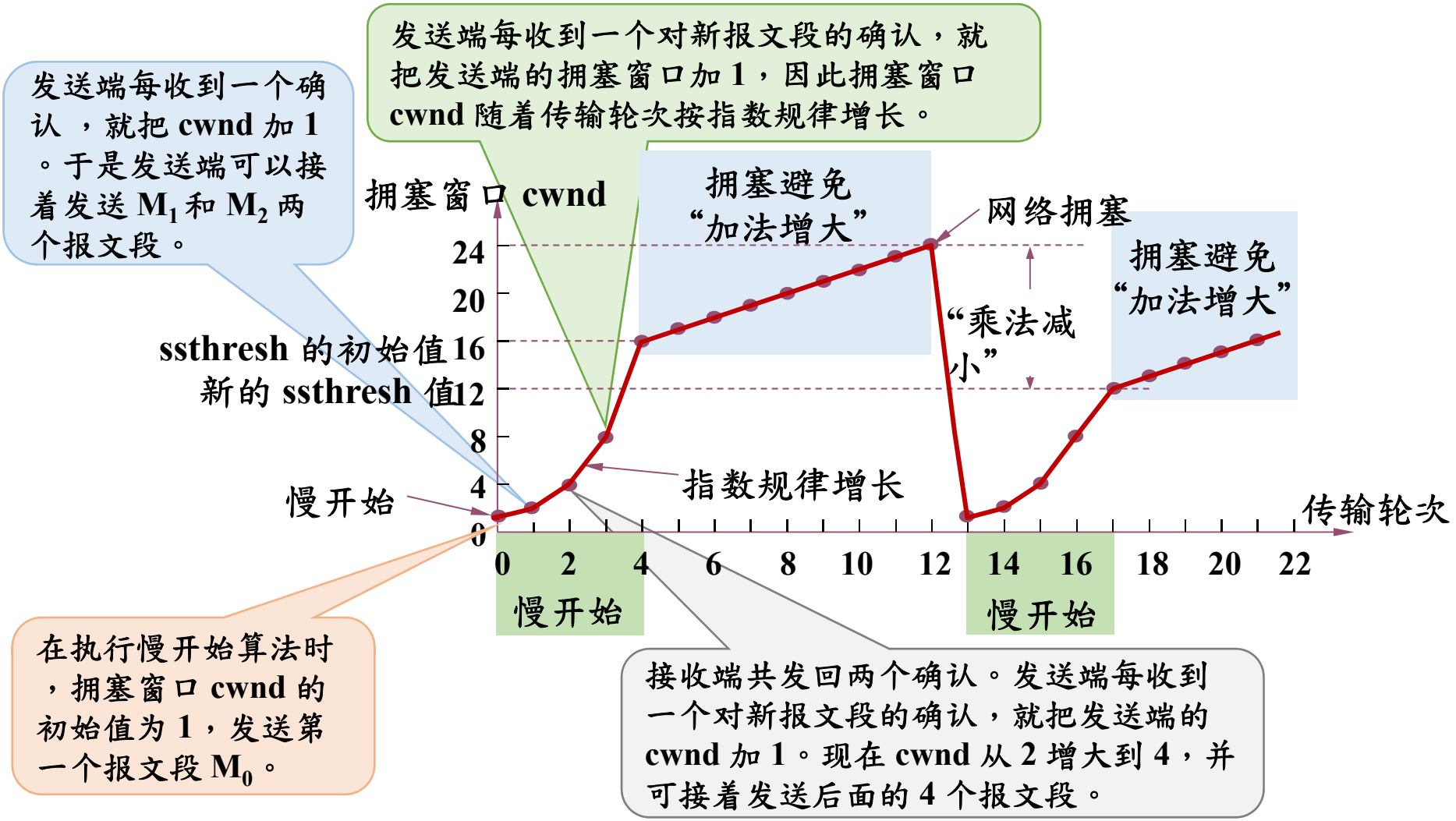
- 发送方每收到一个对新报文段的确认（重传的不算在内）就使 cwnd 加 1。



TCP拥塞控制：慢开始和拥塞避免

- 加性增大
  - TCP 初始化时，拥塞窗口置为 1，每收到确认增加1
  - 到达门限值（初始为16）改用拥塞避免算法
- 乘性减小
  - 一但拥塞，门限值改为窗口大小的一半
- “拥塞避免” 在拥塞避免阶段把拥塞窗口控制为按线性规律增长，使网络较不容易拥塞。（而非完全避免）

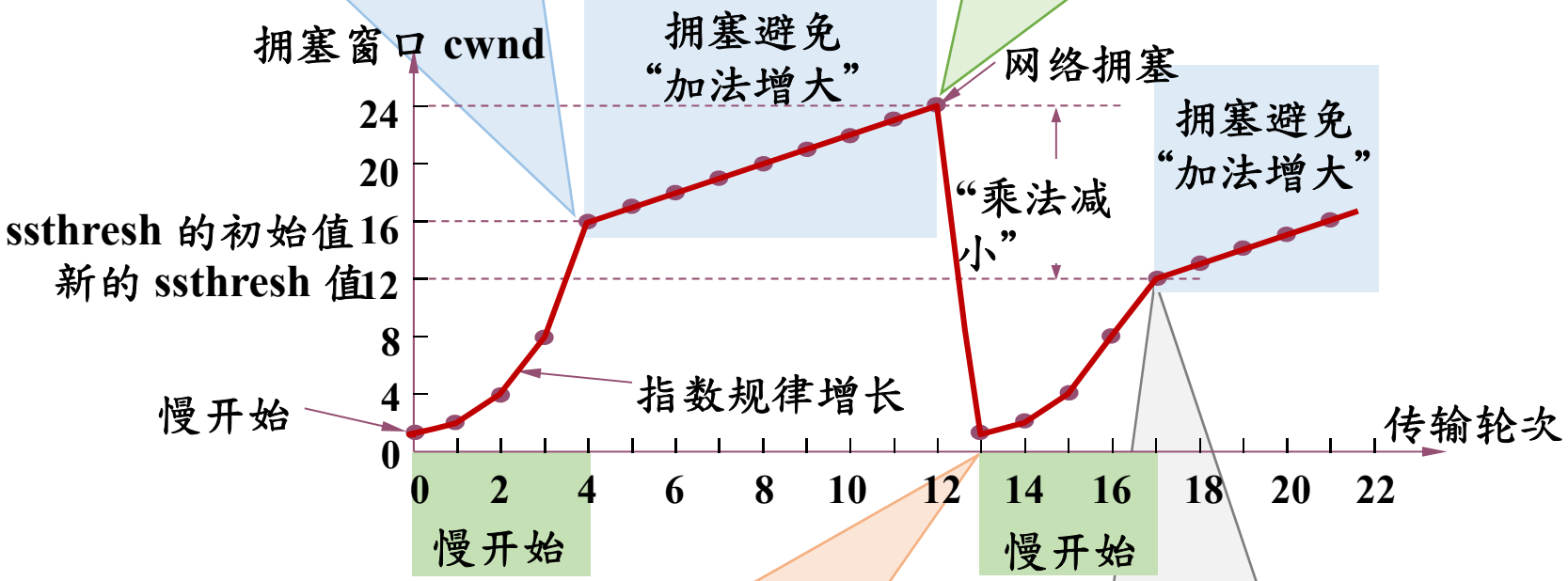
TCP拥塞控制：慢开始和拥塞避免



TCP拥塞控制：慢开始和拥塞避免

当拥塞窗口 cwnd 增长到慢开始门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络拥塞了。



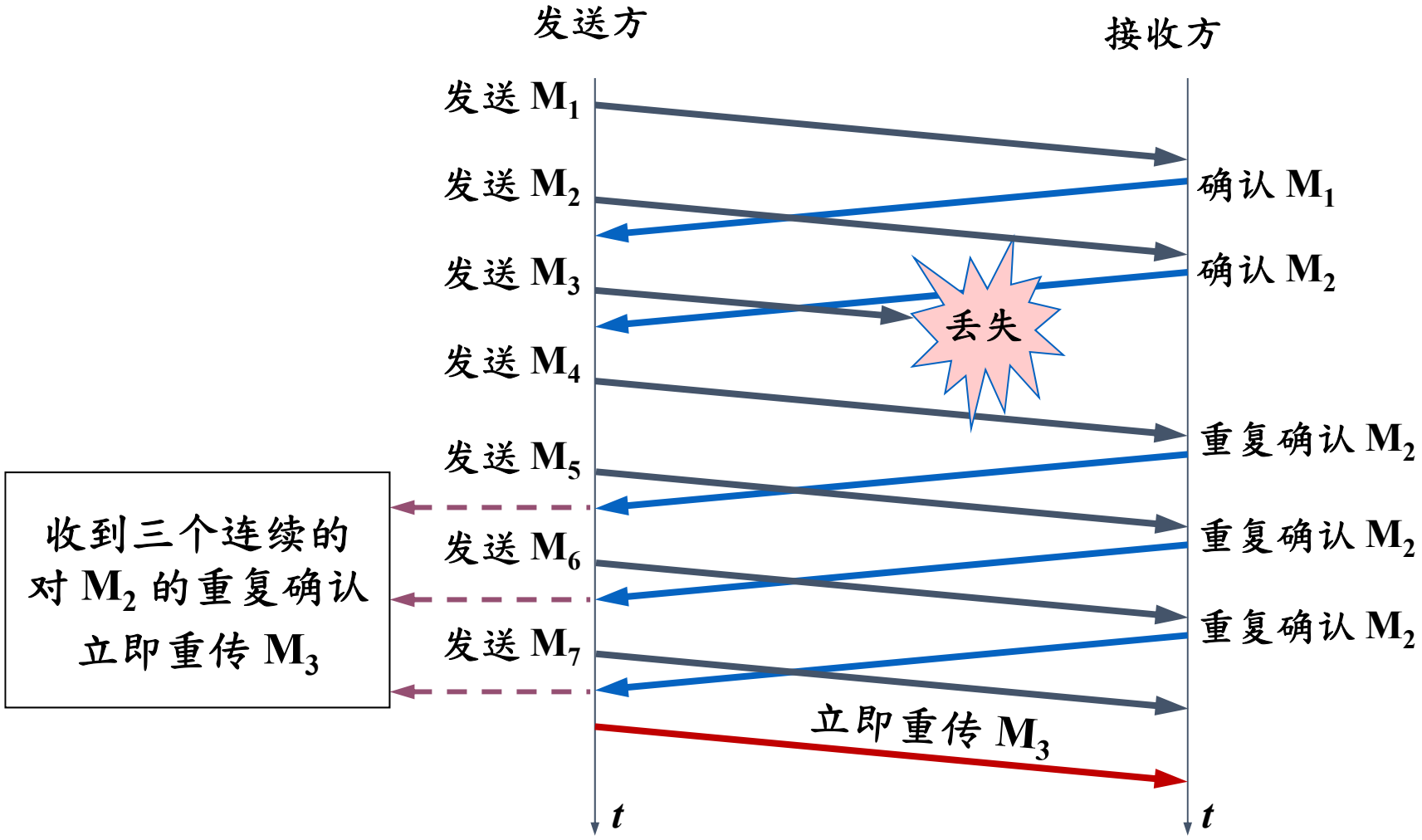
更新后的 ssthresh 值变为 12（即发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并执行慢开始算法。

当 cwnd = 12 时改为执行拥塞避免算法，拥塞窗口按按线性规律增长，每经过一个往返时延就增加一个 MSS 的大小。

TCP拥塞控制：快重传和快恢复

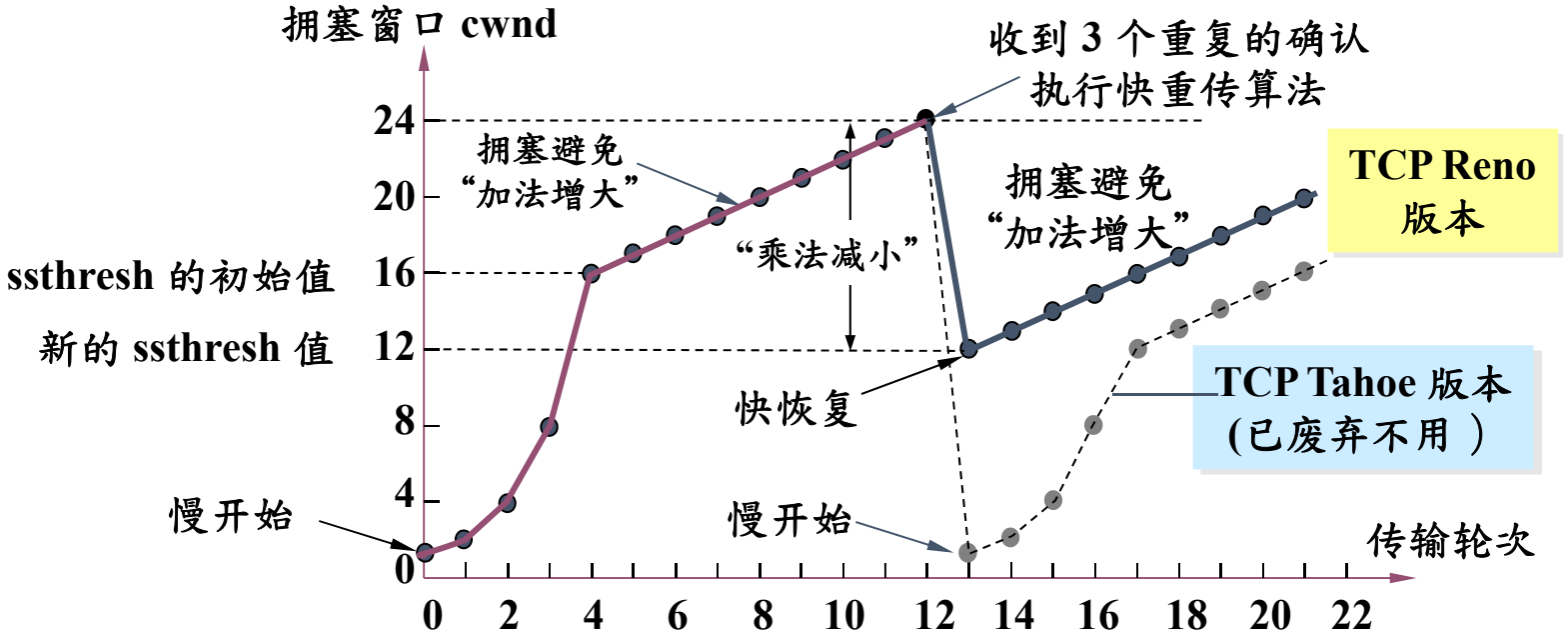
- 快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认。这样做可以让发送方及早知道有报文段没有到达接收方。
- 发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段。
- 不难看出，快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段。

TCP拥塞控制：快重传和快恢复



TCP拥塞控制：快重传和快恢复

- 当发送端收到连续三个重复的确认时，就执行“乘法减小”算法，把慢开始门限 ssthresh 减半。但接下去不执行慢开始算法。

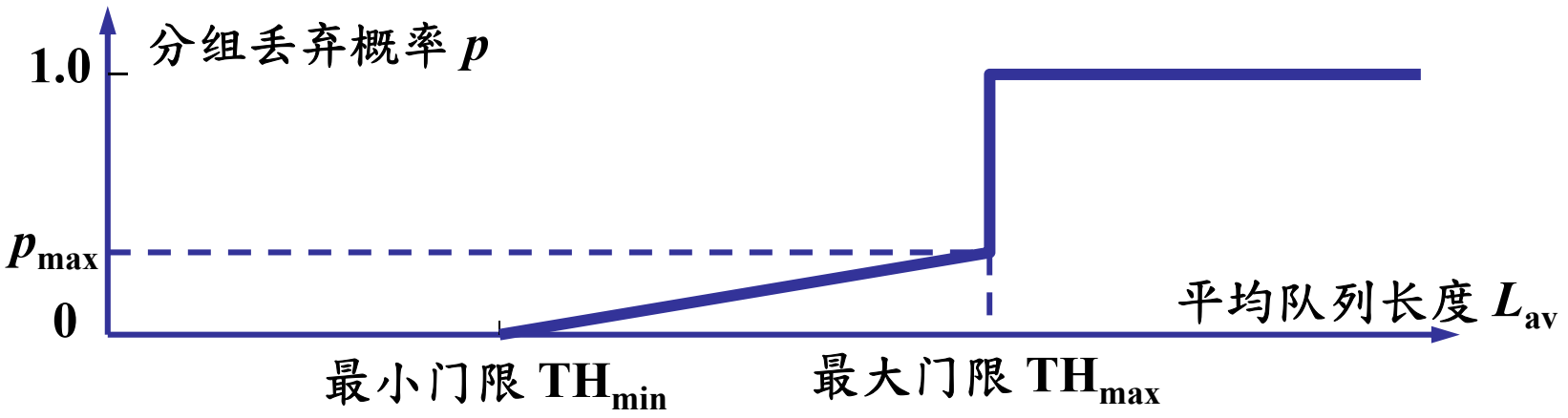


TCP拥塞控制：快重传和快恢复

- 发送方的发送窗口的上限值应当取为接收方窗口 rwnd 和拥塞窗口 cwnd 这两个变量中较小的一个，即应按以下公式确定：
  - 发送窗口的上限值=Min [rwnd, cwnd]
  - 当 rwnd < cwnd 时，是接收方的接收能力限制发送窗口的最大值。
  - 当 cwnd < rwnd 时，则是网络的拥塞限制发送窗口的最大值。

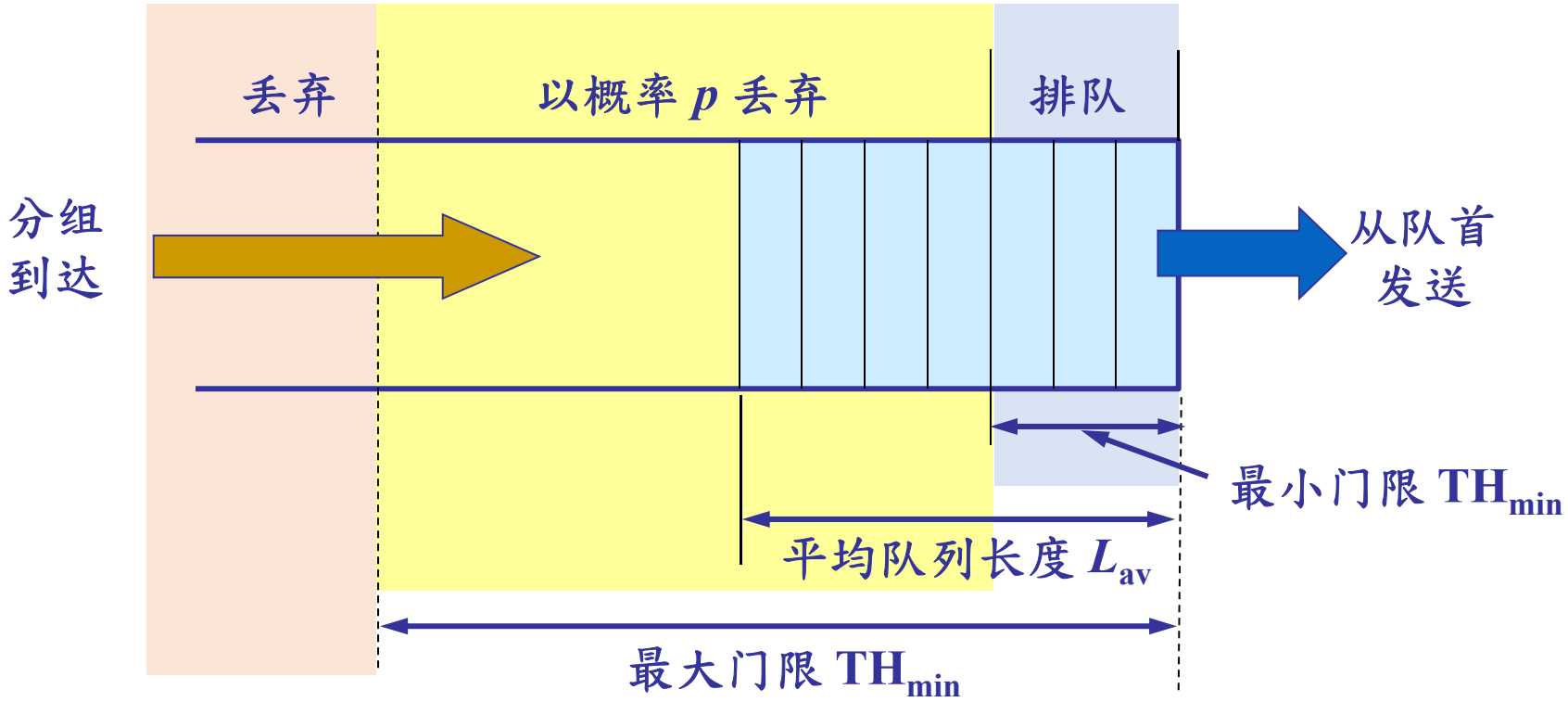
TCP拥塞控制：随机早期检测RED

- 随机早期检测 RED (Random Early Detection)
  - 使路由器的队列维持两个参数，即队列长度最小门限 TH_{min} 和最大门限 TH_{max}。
  - RED 对每一个到达的数据报都先计算平均队列长度 L_{AV}。



TCP拥塞控制：随机早期检测RED

- 不使用瞬时队列长度是因为突发数据不太可能使队列溢出

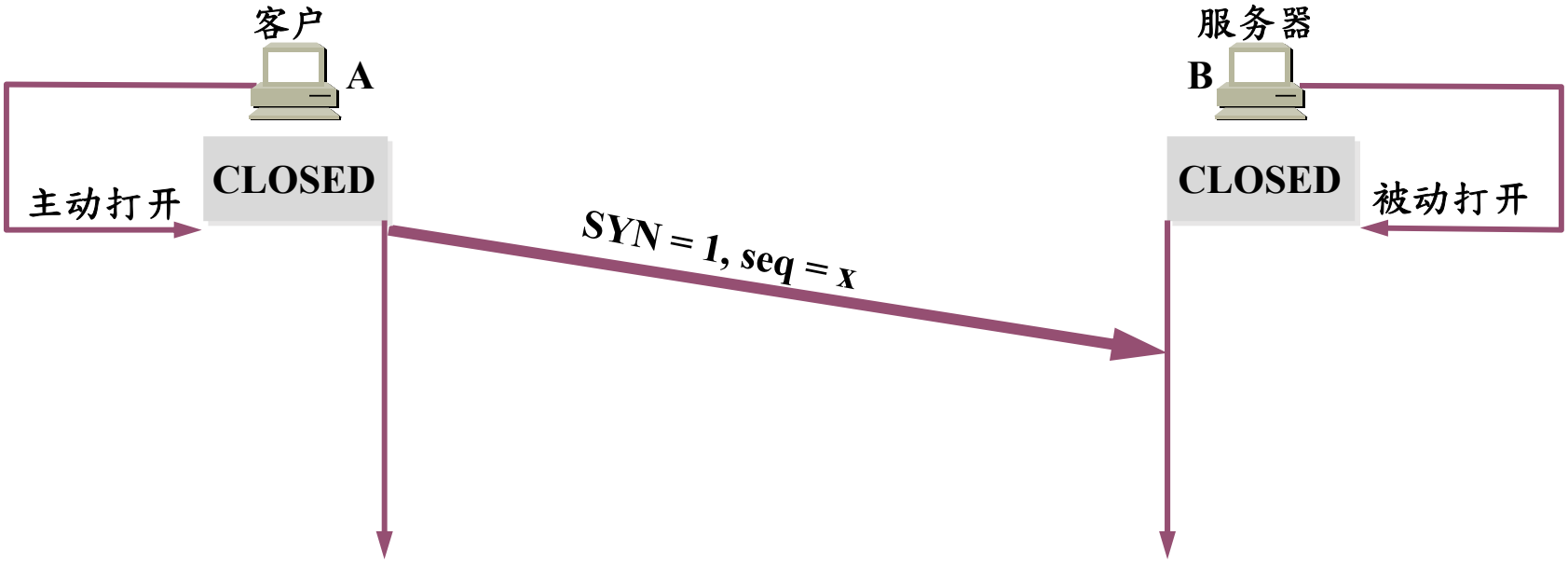


TCP传输连接管理

- 面向连接的传输有三个阶段
  - 连接建立、数据传送和连接释放。
- 连接建立过程中要解决以下三个问题：
  - 要使每一方能够确知对方的存在。
  - 要允许双方协商一些参数（如最大报文段长度，最大窗口大小，服务质量等）。
  - 能够对运输实体资源（如缓存大小等）进行分配。
- 主动发起连接的应用进程叫做客户（Client）
- 被动等待连接建立的进程叫做服务器（Server）

TCP传输连接管理：连接建立

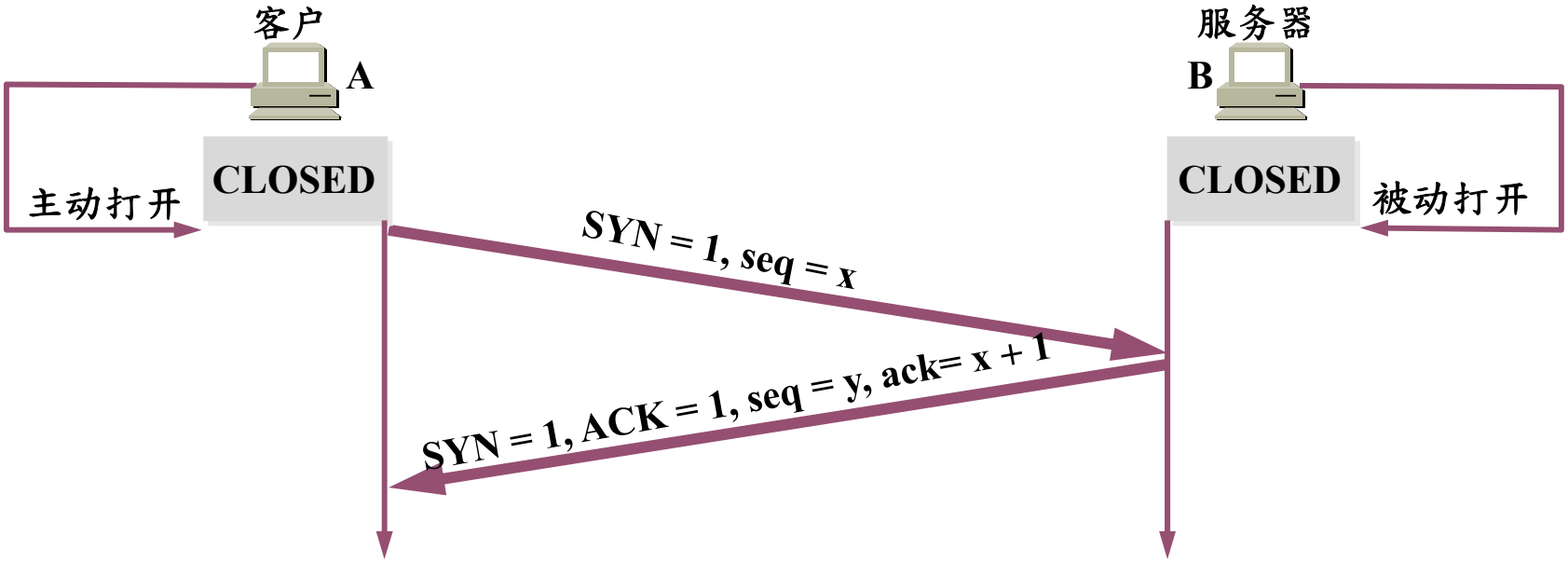
- 三次握手建立 TCP 连接



- A 的 TCP 向 B 发出连接请求报文段，其首部中的同步位 SYN = 1，并选择序号 seq = x，表明传送数据时的第一个数据字节的序号是 x。

TCP传输连接管理：连接建立

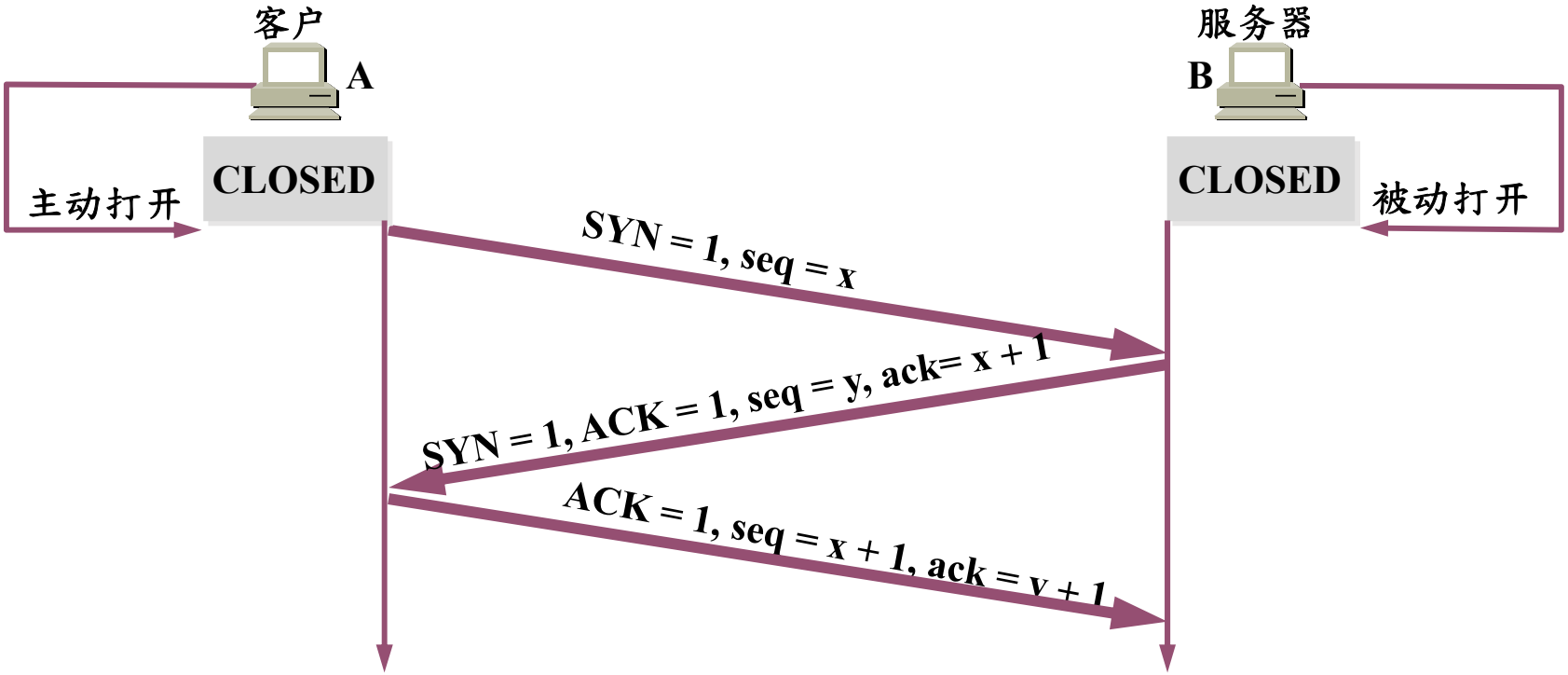
- 三次握手建立 TCP 连接



- B 的 TCP 收到连接请求报文段后，如同意，则发回确认。
- B 在确认报文段中应使 SYN＝1，使 ACK＝1， 其确认号 ack＝x＋1，自己选择的序号 seq＝y。

TCP传输连接管理：连接建立

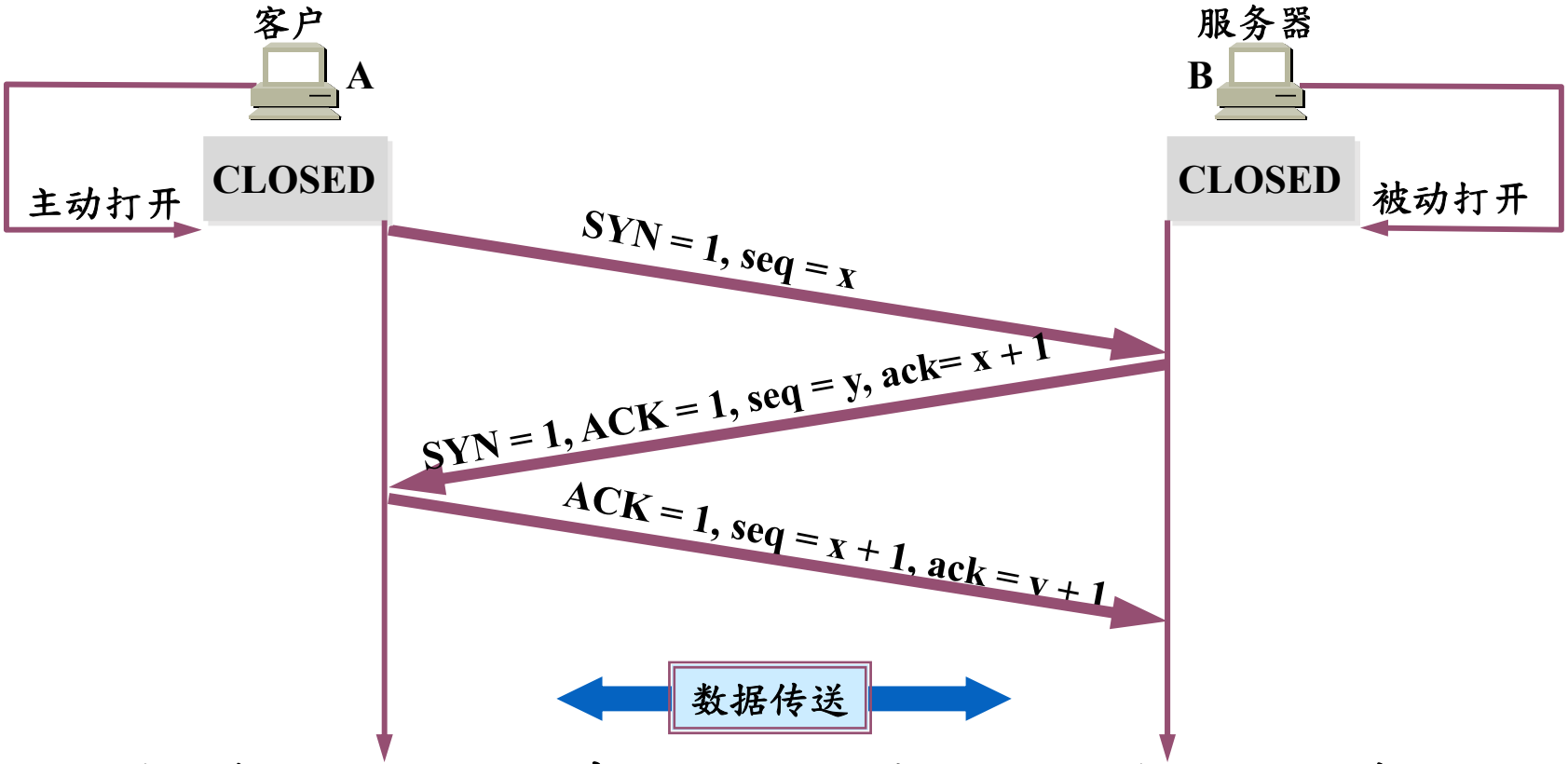
- 三次握手建立 TCP 连接



- A收到此报文段后向 B 确认，ACK=1，确认号ack = y + 1。
- A 的 TCP 通知上层应用进程，连接已经建立。

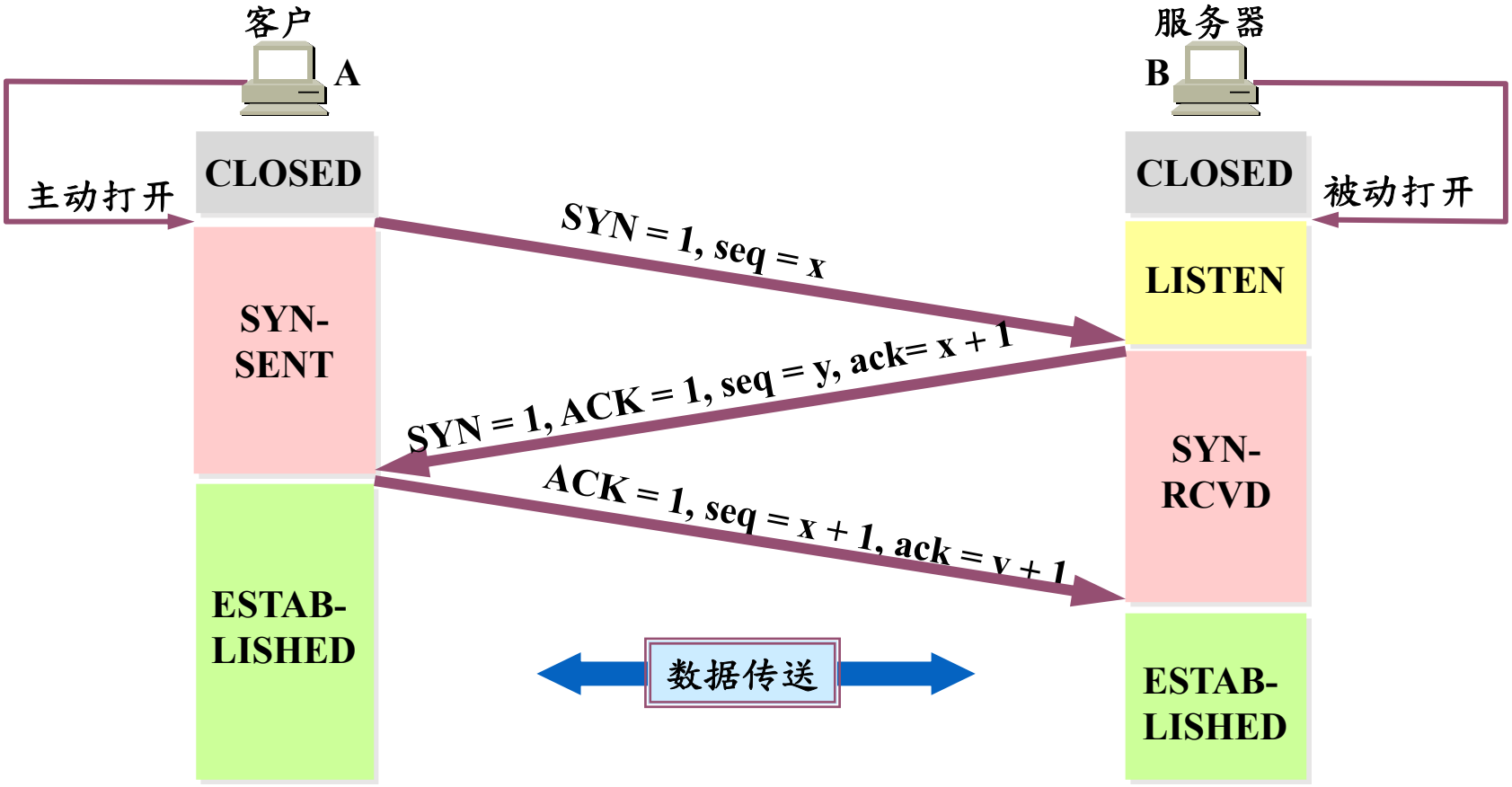
TCP传输连接管理：连接建立

- 三次握手建立 TCP 连接

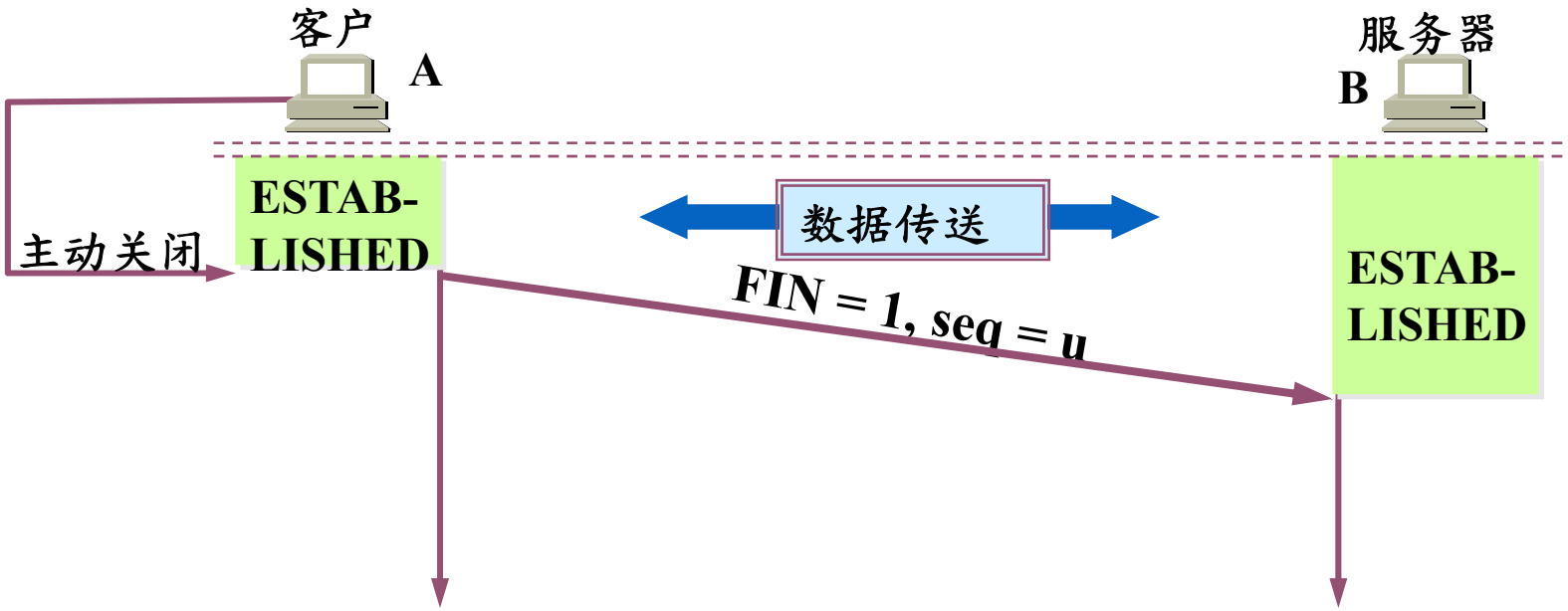


- B收到A确认后，通知其上层应用进程TCP 连接已经建立。

TCP传输连接管理：连接建立

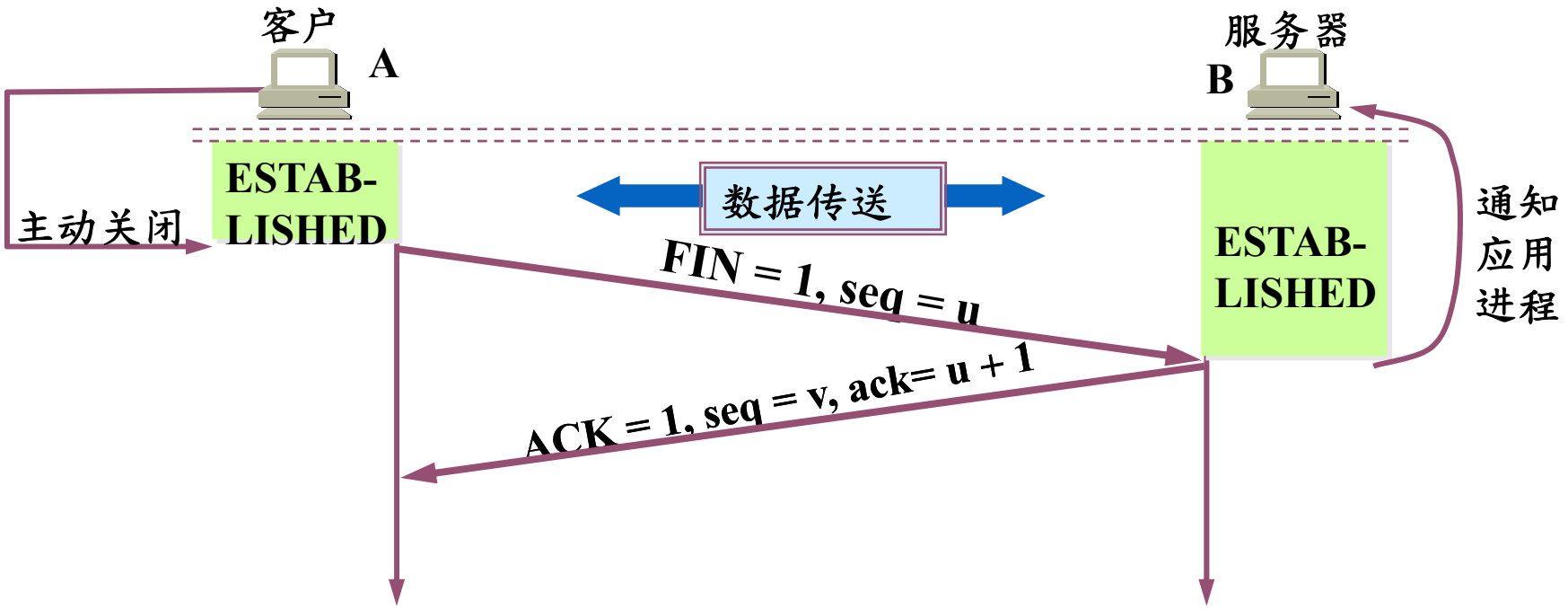


TCP传输连接管理：连接解除



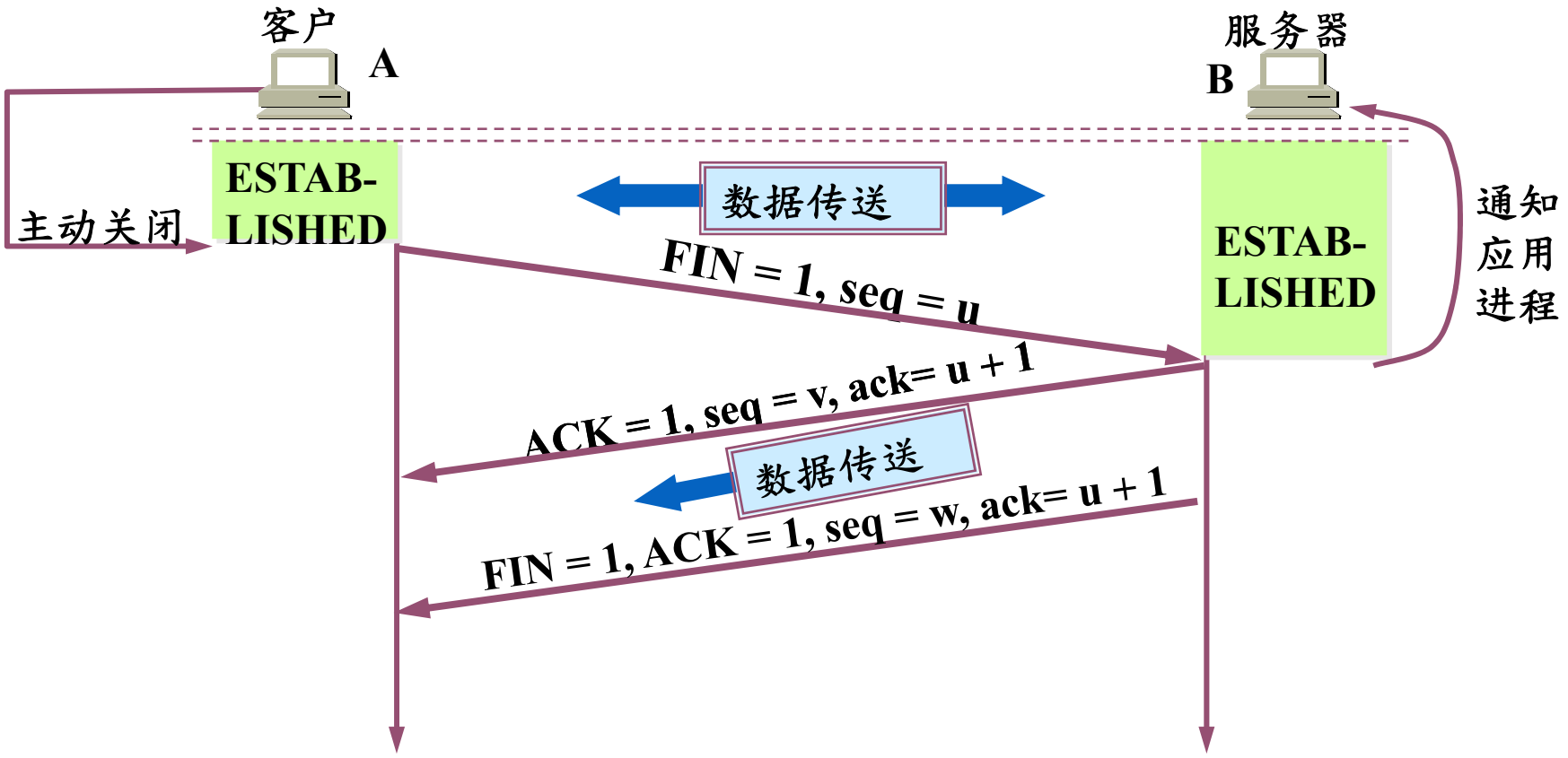
- 数据传输结束后，通信的双方都可释放连接。现在 A 的应用进程先向其 TCP 发出连接释放报文段，并停止再发送数据，主动关闭 TCP连接。
- A 把连接释放报文段首部的 FIN = 1，其序号 seq = u，等待 B 的确认。

TCP传输连接管理：连接解除



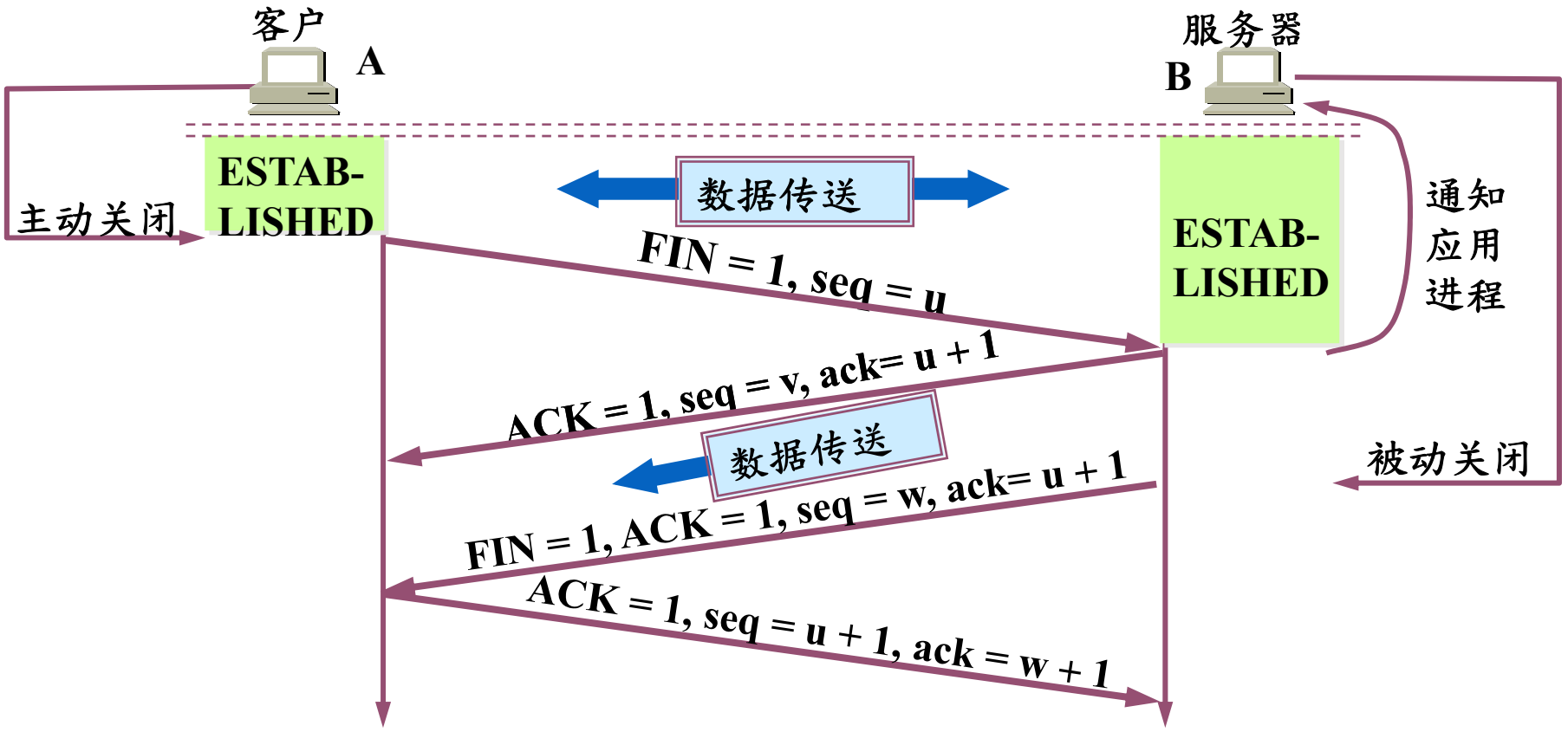
- B 发出确认，确认号ack=u+1，该报文段的序号 seq = v。
- TCP 服务器进程通知高层应用进程。
- 从 A 到 B 这个方向的连接就释放了，TCP 连接处于半关闭状态。B 若发送数据，A 仍要接收。

TCP传输连接管理：连接解除



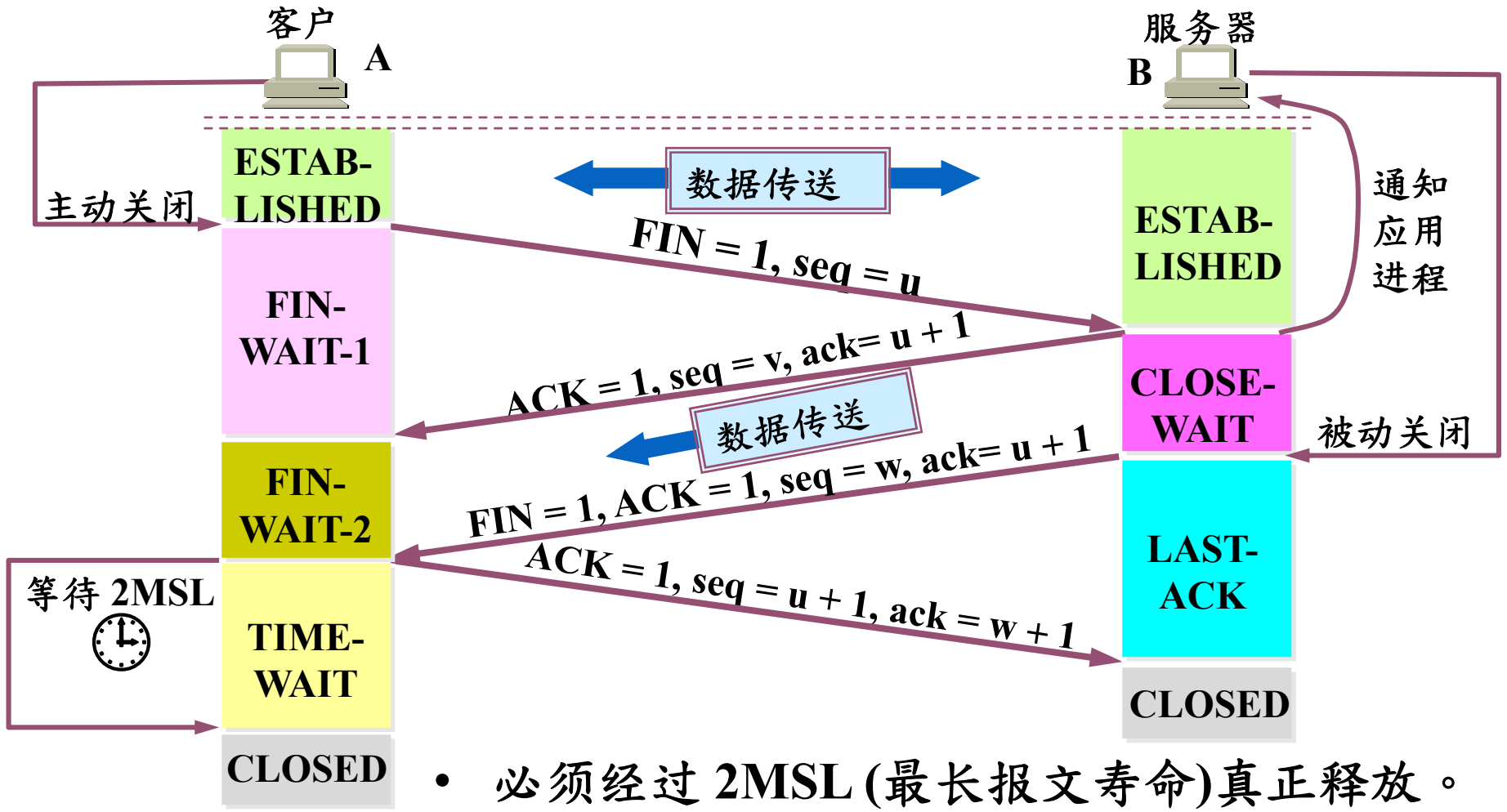
- 若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。

TCP传输连接管理：连接解除



- A 收到连接释放报文段后，必须发出确认。
- 在确认报文段中 ACK=1，确认号 ack=w+1，序号seq=u+ 1

TCP传输连接管理：连接解除



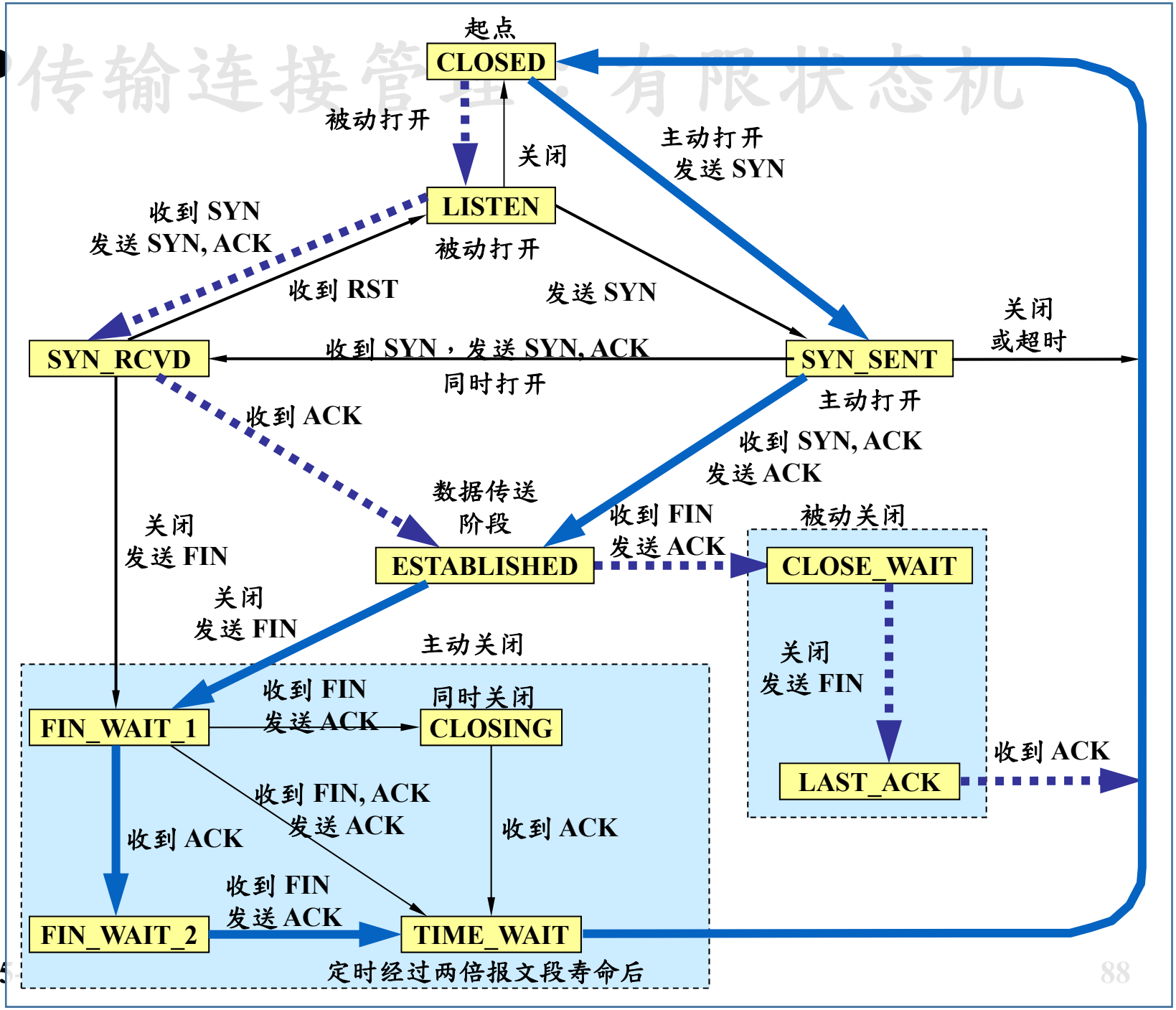
- 必须经过 2MSL (最长报文寿命)真正释放。

TCP传输连接管理：有限状态机

- 图中每一个方框都是 TCP 可能具有的状态。
- 方框中的大写英文字符串是 TCP 标准使用的 TCP 连接状态名。状态之间的箭头表示可能发生的状态变迁。
- 图中有三种不同的箭头。
  - 粗实线箭头表示对客户进程的正常变迁。
  - 粗虚线箭头表示对服务器进程的正常变迁。
  - 另一种细线箭头表示异常变迁。
- 箭头旁边的字，表明引起这种变迁的原因，或表明发生状态变迁后又出现什么动作。

TCP传输连接管理有限状态机

- 有限状态机



TCP与UDP编程实验

连接时间服务器的TCP端口

```
using System;
using System.Net.Sockets;
using System.Text;
public class TcpTimeClient {
    private const int portNum = 13;
    private const string hostName = "time.nist.gov";
    public static int Main(String[] args) {
        try {
            TcpClient client = new TcpClient(hostName,
portNum);
            NetworkStream ns = client.GetStream();
            byte[] bytes = new byte[1024];
            int bytesRead = ns.Read(bytes, 0, bytes.Length);
            Console.WriteLine(Encoding.ASCII.GetString(bytes,
0, bytesRead));
            client.Close();
```

连接时间服务器的TCP端口（续）

```
        }
        catch (Exception e) {
            Console.WriteLine(e.ToString());
        }
        return 0;
    }
}
```

Output on success:

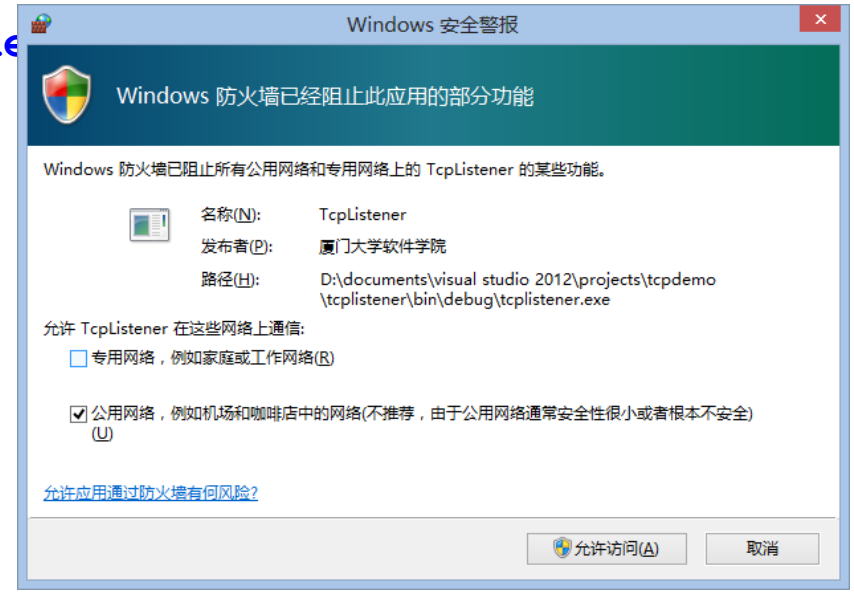
```
56399 13-04-17 06:07:47 50 0 0 658.3 UTC(NIST) *
```

Output on failure:

```
System.Net.Sockets.SocketException (0x80004005): 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。 65.55.21.14:13
   在 System.Net.Sockets.TcpClient..ctor(String hostname, Int32 port)
```

监视端口13以提供时间服务

```
using System;
using System.Net.Sockets;
using System.Text;
public class TcpTimeServer {
    private const int portNum = 13;
    public static int Main(String[] args) {
        bool done = false;
        TcpListener listener = ne
        listener.Start();
        while (!done) {
      ......
        }
        listener.Stop();
        return 0;
    }
}
```



监视端口13以提供时间服务（续）

```
        while (!done) {
            Console.Write("Waiting for connection...");
            TcpClient client = listener.AcceptTcpClient();

            Console.WriteLine("Connection accepted.");
            NetworkStream ns = client.GetStream();

            byte[] byteTime =
Encoding.ASCII.GetBytes(DateTime.Now.ToString());
            try {
                ns.Write(byteTime, 0, byteTime.Length);
                ns.Close();
                client.Close();
            }
            catch (Exception e)
```

监视端口13以提供时间服务（续）

```
        {
                Console.WriteLine(e.ToString());
            }
        }
```

```
Output:
Waiting for connection...Connection accepted.
Waiting for connection...
```

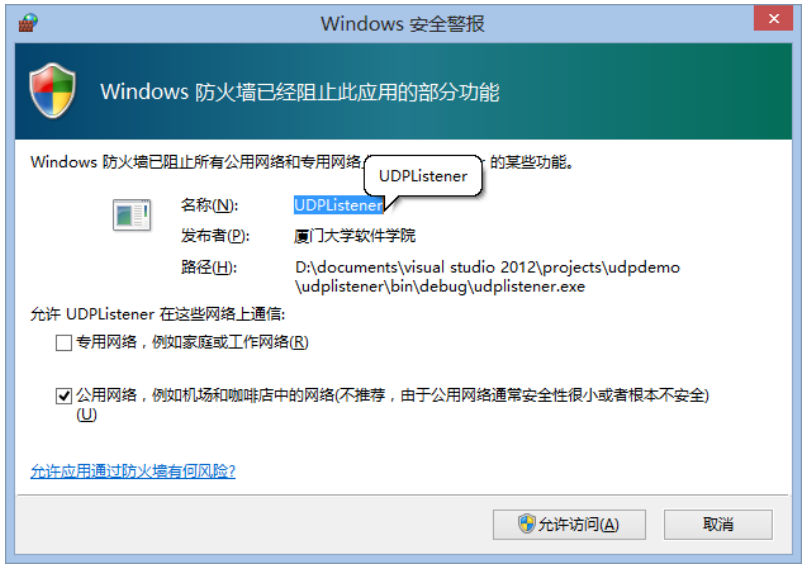
向广播地址11000端口发送UDP消息

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class Program {
    static void Main(string[] args)  {
        Socket s = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
        IPAddress broadcast = IPAddress.Parse("192.168.1.255");
        byte[] sendbuf = Encoding.ASCII.GetBytes("HELLO
NETWORK");
        IPEndPoint ep = new IPEndPoint(broadcast, 11000);
        s.SendTo(sendbuf, ep);
        Console.WriteLine("Message sent to the broadcast
address");
        }
}
Output: Message sent to the broadcast address
```

侦听广播地址11000端口的消息

```
using System.Net;
using System.Net.Sockets;
using System.Text;

public class UDPListener {
    private const int listenPort = 11000;
    private static void StartListener()  {
            ……
    }
    public static int Main()  {
        StartListener();
        return 0;
    }
}
```



侦听广播地址11000端口的消息（续）

```
 private static void StartListener()    {
        bool done = false;
        UdpClient listener = new UdpClient(listenPort);
        IPEndPoint groupEP = new IPEndPoint(IPAddress.Any,
listenPort);
        try {
            while (!done) {
                Console.WriteLine("Waiting for broadcast");
                byte[] bytes = listener.Receive(ref groupEP);
                Console.WriteLine("Received broadcast from
{0} :\n {1}\n", groupEP.ToString(),
Encoding.ASCII.GetString(bytes, 0, bytes.Length));
            }
         }
        catch (Exception e)
{ Console.WriteLine(e.ToString()); }
        finally { listener.Close(); }
    }
```

侦听广播地址11000端口的消息（续）

```
Output on success:

Waiting for broadcast

Received broadcast from 192.168.1.1:52600 :

 HELLO NETWORK


Waiting for broadcast
```

14. THANK YOU.

厦门大学软件学院

黄炜 助理教授