计算机网络
Computer Network

12

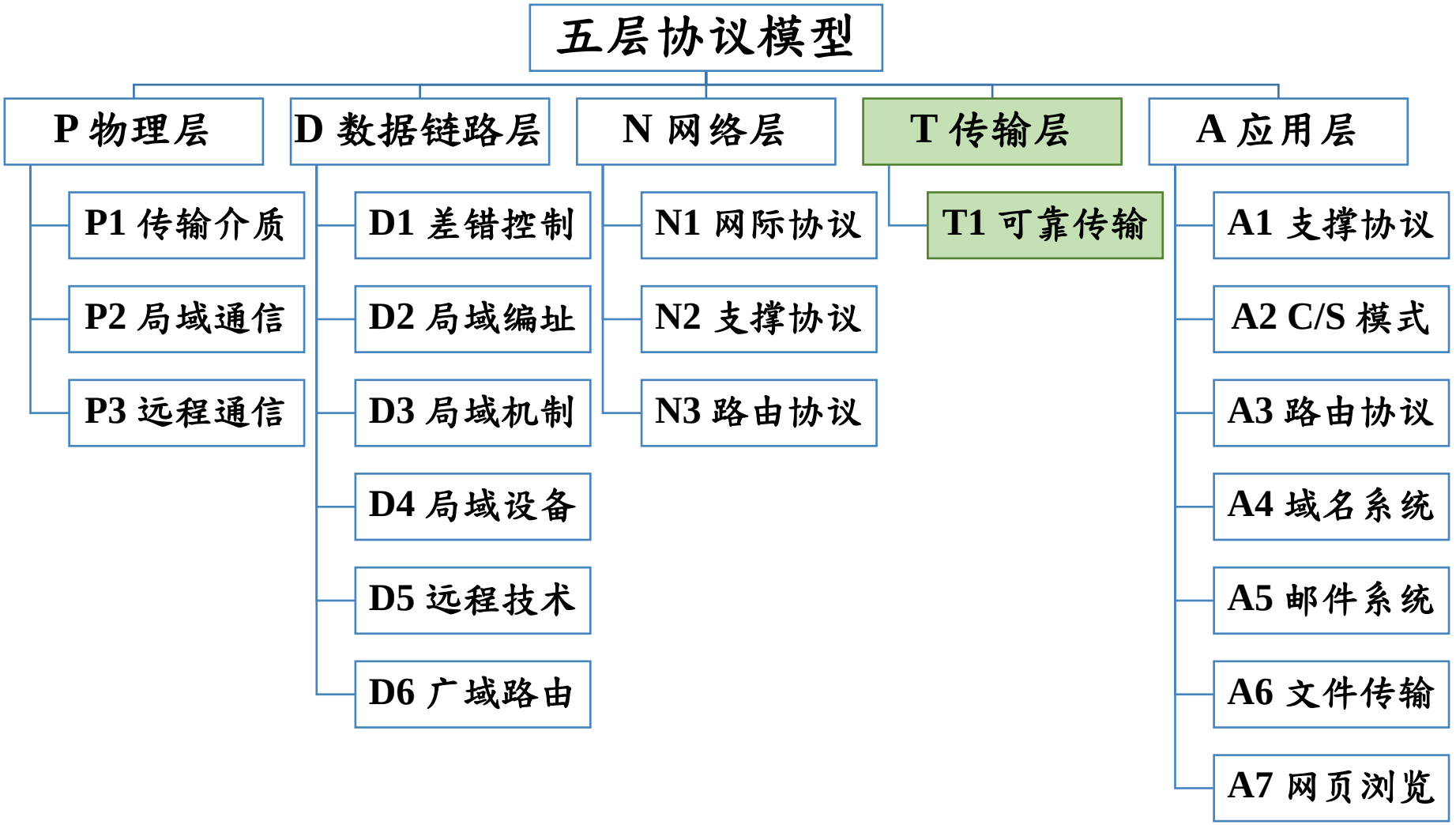
传输控制协议

理论课程

厦門大學
XIAMEN UNIVERSITY

信息学院 黄 炜
（国家示范性软件学院） 博士/副教授
School of Informatics Dr. Wei Huang

知识框架


五层协议模型
P 物理层
D 数据链路层
N 网络层
T 传输层
A 应用层
P1 传输介质
P2 局域通信
P3 远程通信
D1 差错控制
D2 局域编址
D3 局域机制
D4 局域设备
D5 远程技术
D6 广域路由
N1 网际协议
N2 支撑协议
N3 路由协议
T1 可靠传输
A1 支撑协议
A2 C/S 模式
A3 路由协议
A4 域名系统
A5 邮件系统
A6 文件传输
A7 网页浏览

主要内容

- 传输层
  - 作用
  - 端口号，端口号的分类
- UDP
  - UDP的无连接、尽力交付、面向报文、允许广播
- TCP
  - 特点：面向连接、点对点、可靠、全双工、字节流
  - TCP段格式中的各部分组成

主要内容

- **TCP**
  - 机制
    - 应答机制、超时机制、重传机制、窗口机制
    - 流量控制机制：滑动窗口
    - 拥塞控制：慢开始、拥塞避免、快重传、快恢复、随机早期检测
    - TCP的连接建立和解除（三次握手、四次挥手）
  - 传输层解决网络层的主要问题：丢包、重复、乱序

对应课本章节

- **PART IV Internetworking**
  - **Chapter 25 UDP: Datagram Transport Service**
  - **Chapter 26 TCP: Reliable Transport Service**

内容纲要

传输层的必要性

- 主机上多个进程共享主机的网络连接进行通信
  - 唯一标识主机和主机上的连接：IP地址
  - 唯一标识进程：端口号
  - 网络通信本质上是两个进程间的通信，不是主机间通信
- 传输层的作用：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

协议端口号

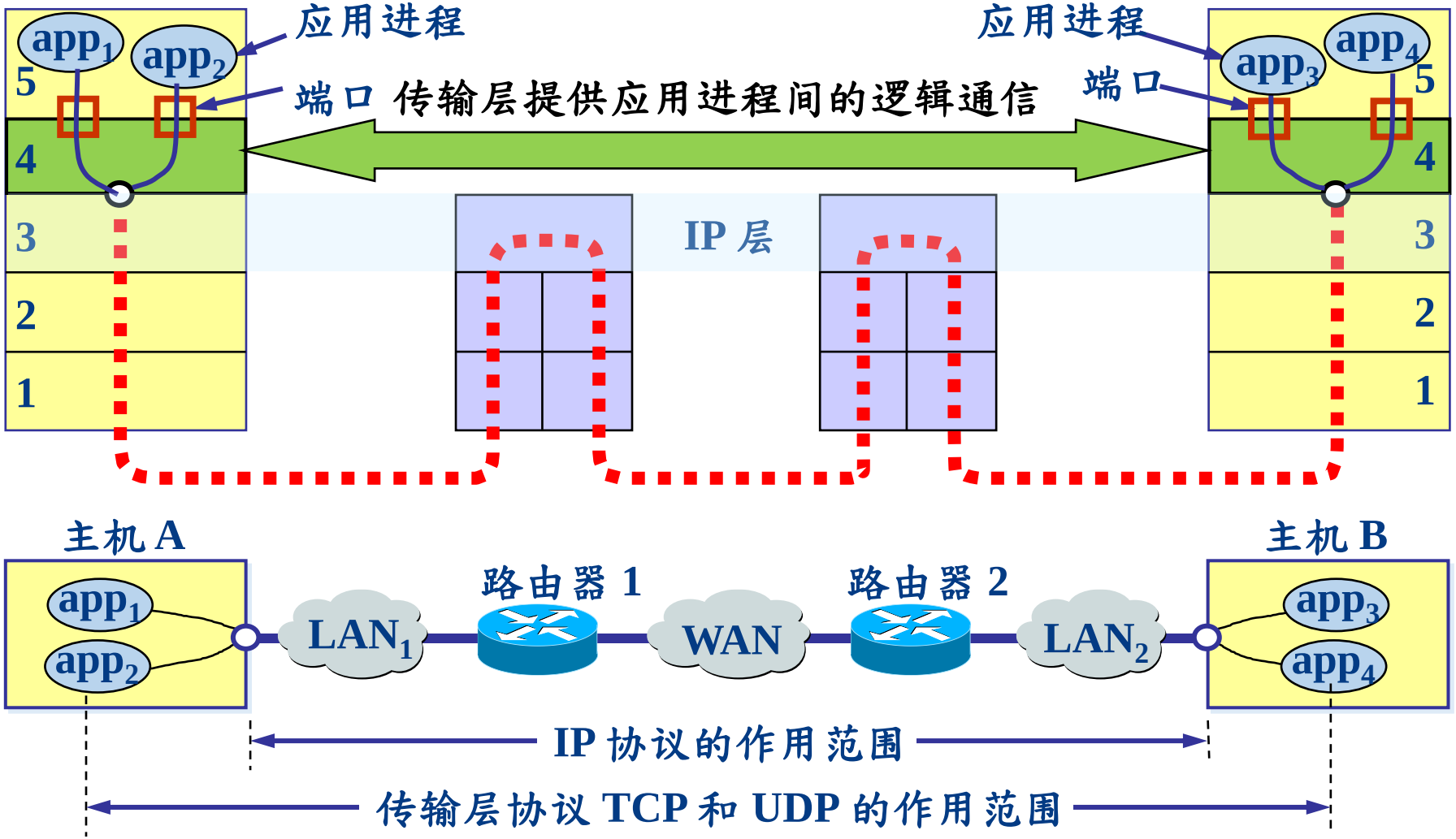
- 端口（Port）
  - 软件端口，有别于交换机上的硬件端口
  - 端口号（16bits），范围：0～65535，
  - 作用：用于标识本机的不同进程
  - 分类（参考：www.iana.org）

熟知端口号	登记端口号	客户端口号
0 ～ 1023	1024 ～ 49151	49152 ～ 65535
0 ～ 0x03FF	0x0400 ～ 0xBFFF	0xC000 ～ 0xFFFF

经常使用的端口号

协议	传输层协议	端口号	作用
FTP（数据）	TCP	20	FTP数据传输
FTP（控制）	TCP	21	FTP控制命令传输
Telnet	TCP	23	终端远程登录
SMTP	TCP	25	发送电子邮件
HTTP	TCP	80	网页服务
POP3	TCP	110	接收电子邮件
IMAP	TCP	143	同步邮箱
HTTPS	TCP	443	加密网页服务
RDP	TCP	3389	Windows 远程桌面连接
DNS	TCP/UDP	53	域名解析
DHCP（源）	UDP	68	动态IP获取的客户端端口。
DHCP（目的）	UDP	67	动态IP获取的服务器端端口。

网络进程通信


应用进程
应用进程
app1
app2
app3
app4
端口
端口
传输层提供应用进程间的逻辑通信
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
传输层协议 TCP 和 UDP 的作用范围

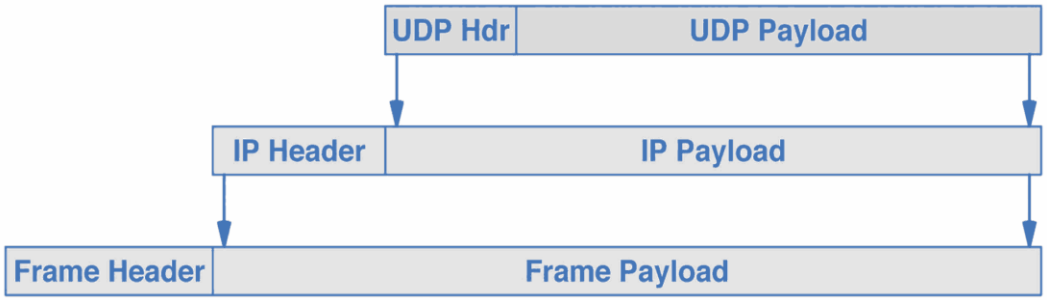
传输层的两个主要协议

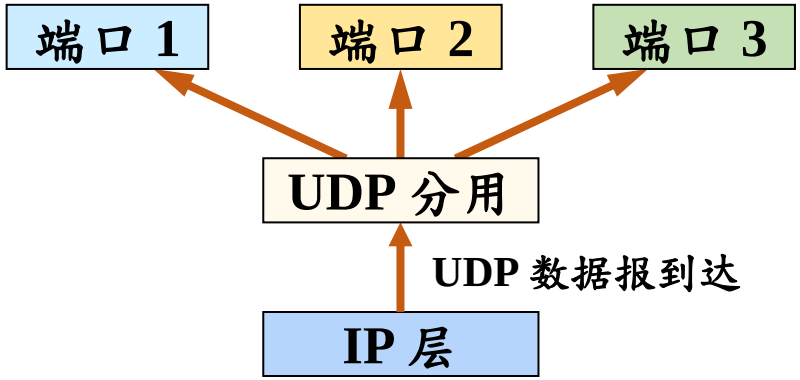
协议	UDP [RFC 768]	TCP [RFC 793]
全称	User Datagram Protocol，用户数据报协议	Transmission Control Protocol，传输控制协议
数据单位	UDP 数据报 (Datagram)	TCP 报文段 (segment)
作用	端到端	端到端、字节流
连接	无连接	面向连接
收到确认	接收方不给出确认	收到确认
多方	一对一、一对多、多对多	一对一
长度	任意	每报文不超过64KB
优点	高效	安全
比喻	发电报（短信）	打电话

内容纲要

UDP概述

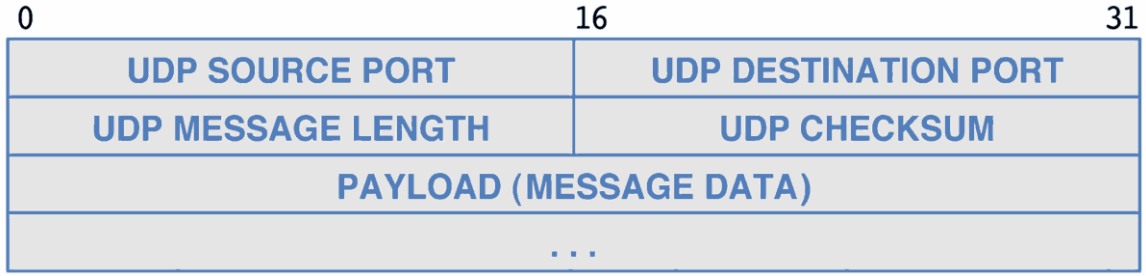
- 用户数据报协议（User Datagram Protocol，UDP）
- UDP的作用：不可靠但轻快的传输
  - 在IP服务之上，增加端口的功能和差错检测的功能
    - UDP校验和是可选的（0表示不计算）
- UDP消息可以丢失、重复、延迟、乱序、损坏。



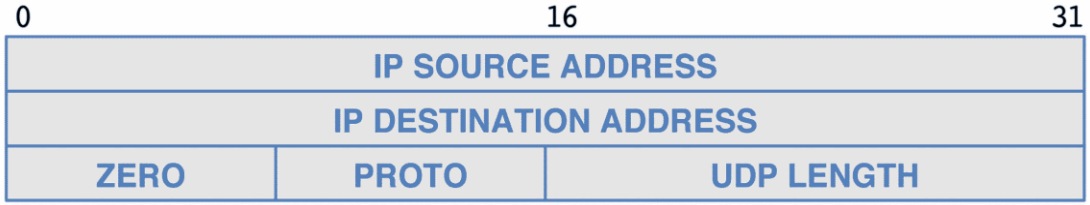


UDP分组结构

- 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



UDP的特点

- 无连接：发送数据之前不需要建立连接
  - 连接是指对状态（变量）的保持，不是对网络的保持
    - TCP协议中，通信双方保持序列号等变量，确保通信可靠
- 尽力交付：不保证可靠交付，同时也不使用拥塞控制
  - UDP发送报文前，不判断网络产生拥塞，不调节发送速率
    - 很适合多媒体通信等实时应用的要求
- 支持一对一、一对多的交互通信
- 首部开销小：只有 8 个字节

面向报文的 UDP

- 发送方
  - UDP 对应用程序交来的报文，在添加首部后向下交付 IP 层
  - 既不合并，也不拆分，而是保留这些报文的边界
- 接收方
  - UDP 对 IP 层交来的数据，去除首部后原封交付应用进程
  - 一次交付一个完整的报文
- 应用程序必须选择合适大小的报文。
  - 应用层交给 UDP 多长的报文，UDP 照样发送。

UDP的应用场景

- 常用于丢包损失不大，应用层可以控制丢包的场景。

类别	作用	示例
面向 简单事务	查询响应协议	域名系统或网络时间协议
	没有完整协议栈的引导	DHCP和TFTP
提供 数据报	构建其他协议	IP隧道，远程过程调用，网络文件系统
大量客户端	流媒体应用	IPTV
实时应用	建立在实时流的协议	IP语音，网络游戏
单向沟通	服务发现和共享信息中的广播信息	广播时间或路由信息协议

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");
    }
}
```

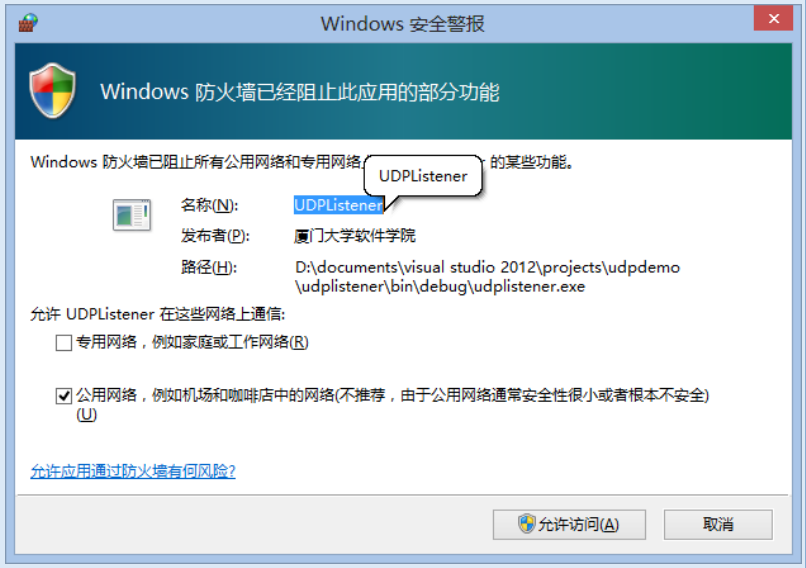
输出：

```
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(); }
    }
```

输出（连接成功）：

```
Waiting for broadcast
Received broadcast from 192.168.1.1:52600 :
 HELLO NETWORK

Waiting for broadcast
```

内容纲要

可靠传输的需求

- 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（Transmission Control Protocol，TCP）
  - TCP提供面向连接、点对点、流接口、完整可靠的全双工通信。

TCP报文段的首部格式

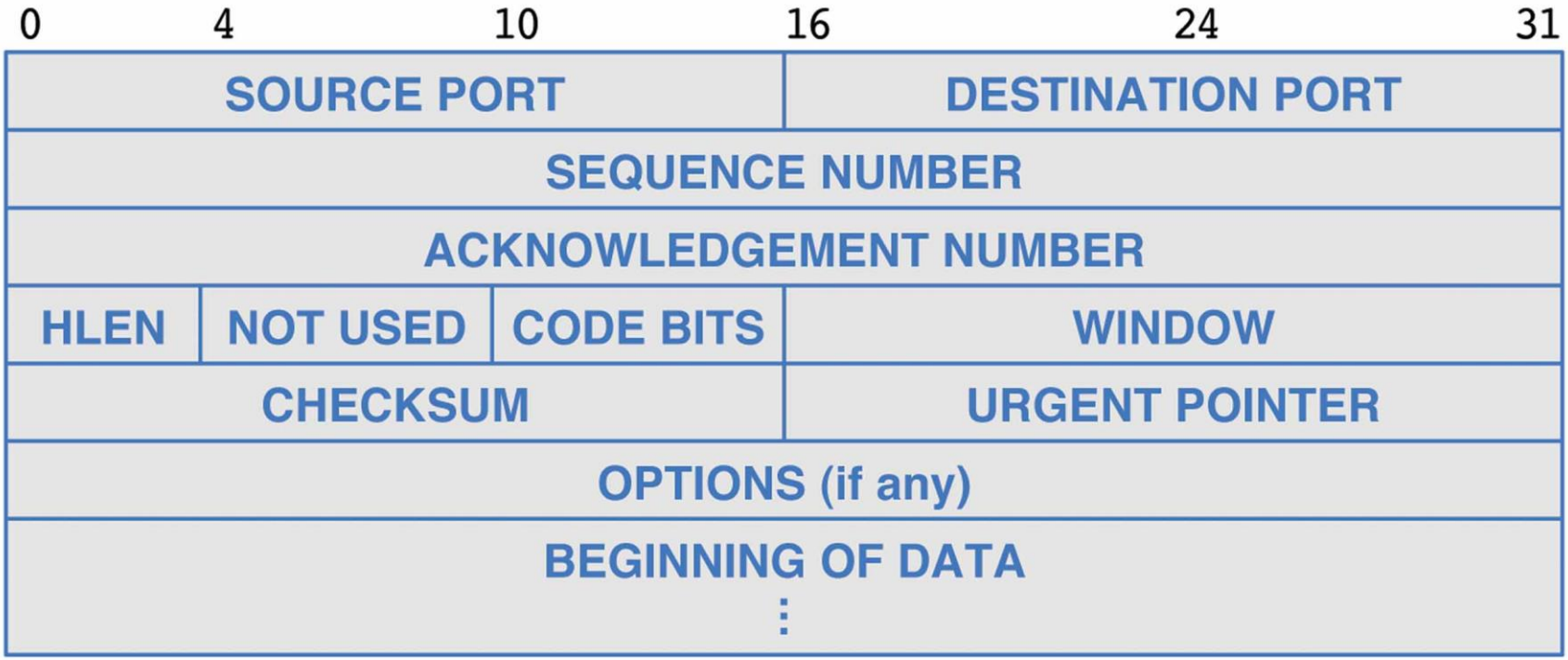


Figure 26.10 The TCP segment format used for both data and control messages.

TCP报文段的首部格式

- **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报文段的首部格式

- **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报文段的首部格式

- **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：16bits**
  - **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]
```

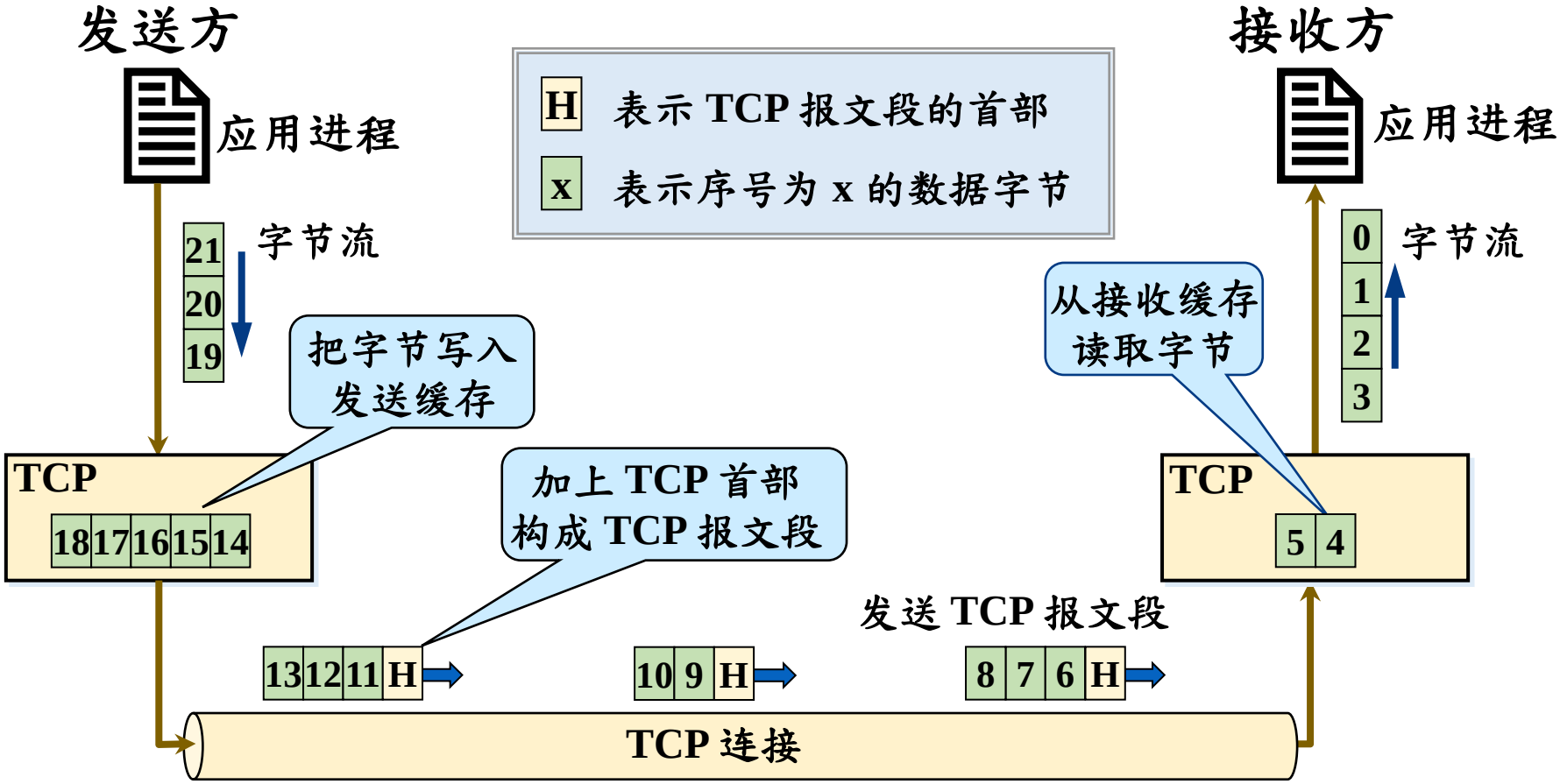
TCP报文头部（续）

```
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
```

内容纲要

TCP流接口

- TCP 的传输是以字节为单位封装在报文段中传输。

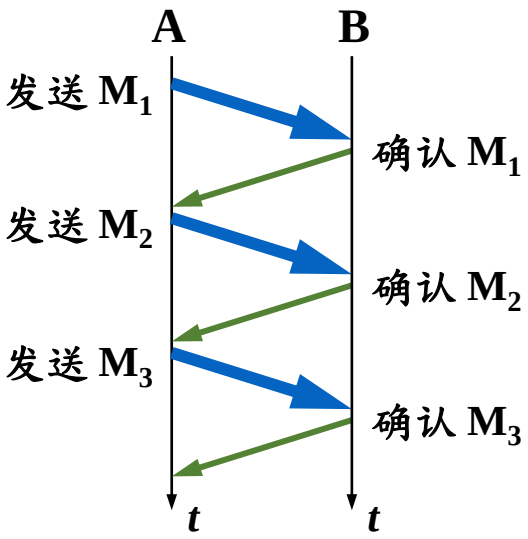


TCP虚连接

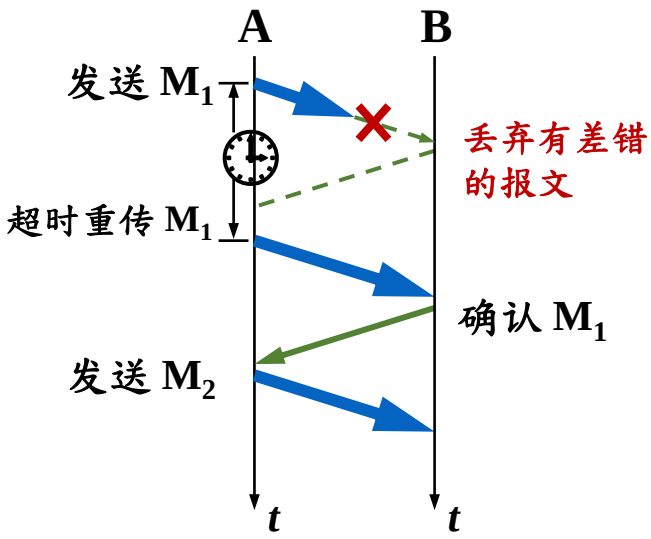
- TCP 连接是一条虚连接而不是一条真正的物理连接。
  - TCP 不关心应用进程发送到TCP 的缓存中的消息长度。
  - TCP 根据对方给出的窗口值和当前网络拥塞的程度来决定一个报文段应包含的字节数。
    - UDP 发送的报文长度是应用进程给出的。
  - TCP 将过长的数据块划分再传送，也可以等待积累有足够多的字节后再构成报文段发送出去。

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



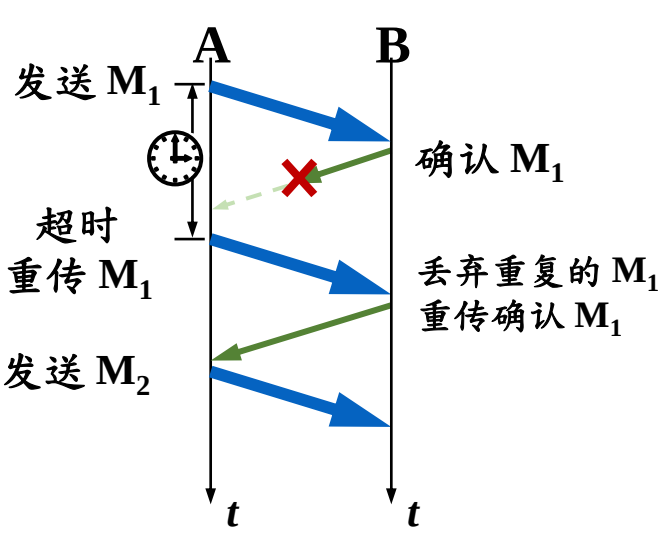
(a) 无差错情况



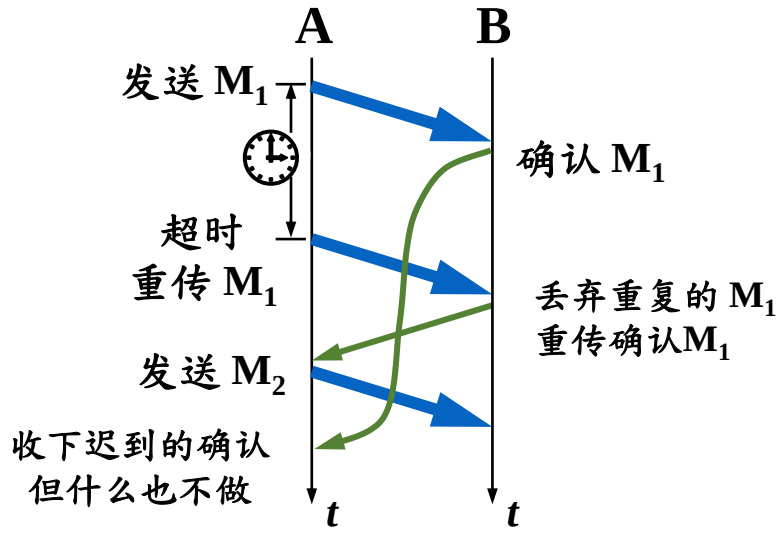
(b) 超时重传

停等协议的自动重传请求机制

- 确认丢失和确认收到
  - 通过自动重传请求，在不可靠的网络上实现可靠的通信
  - 不区分分组丢失和确认丢失，
    - 一直未收到确认，则线路太差



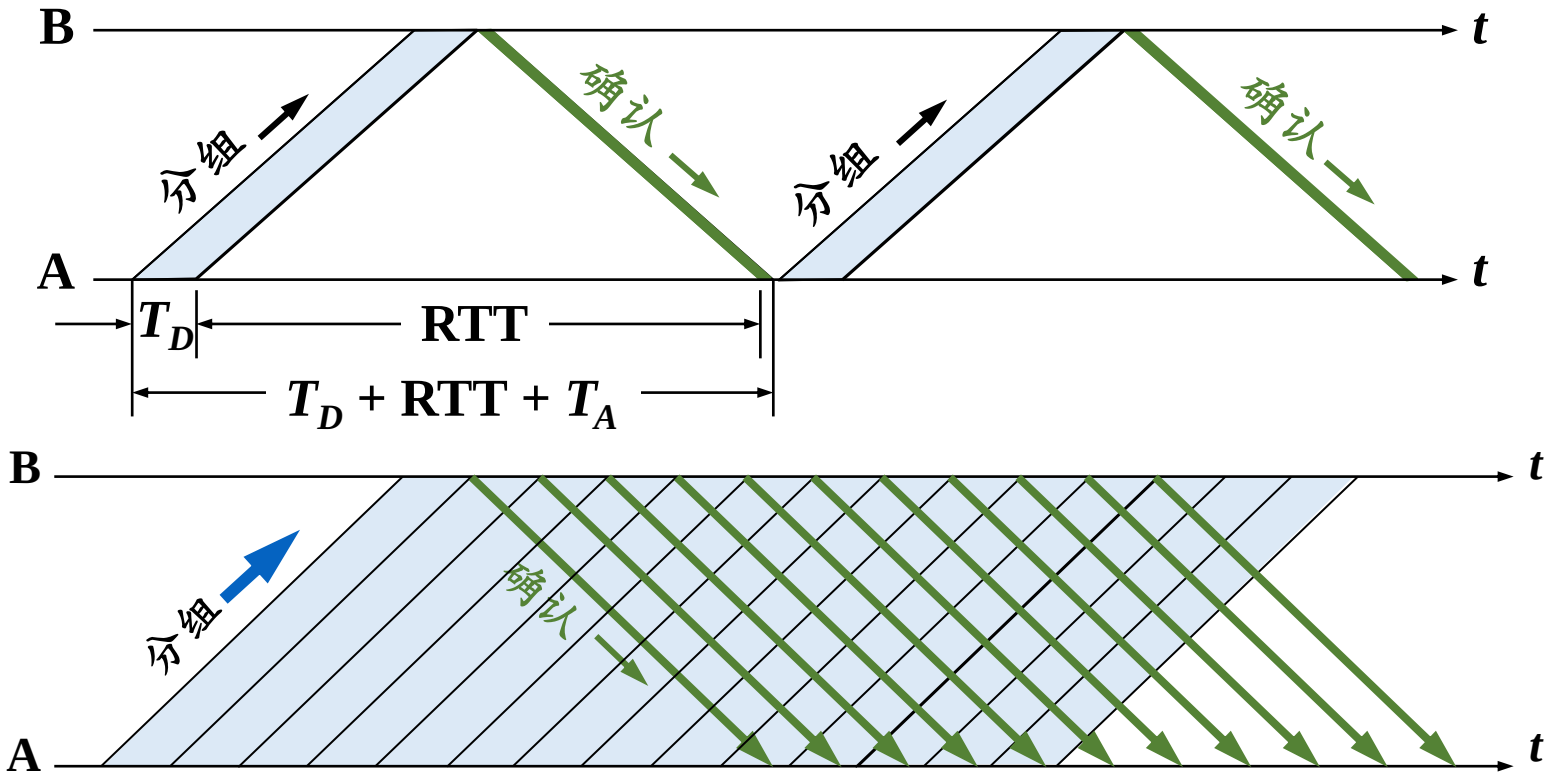
(a) 确认丢失



(b) 确认迟到

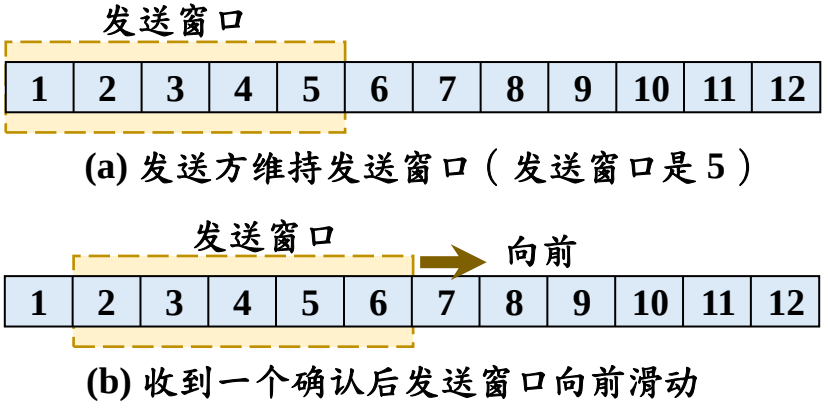
连续自动重传请求

- 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



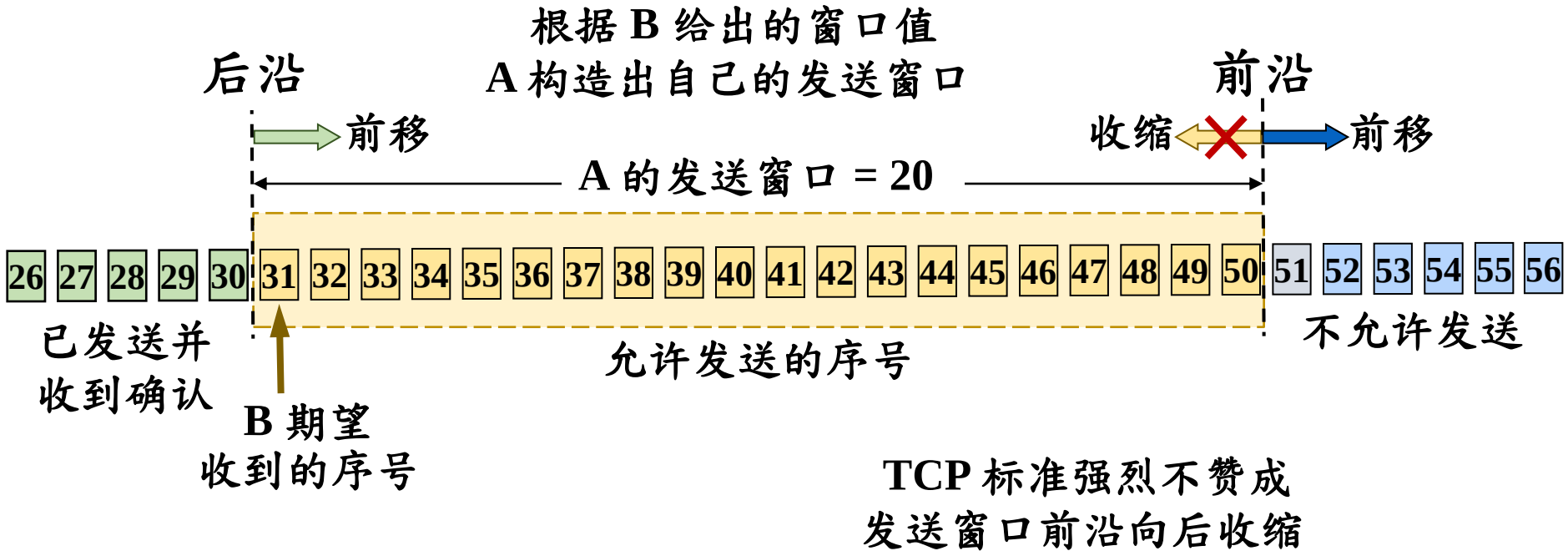
窗口机制：连续自动重传请求

- 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



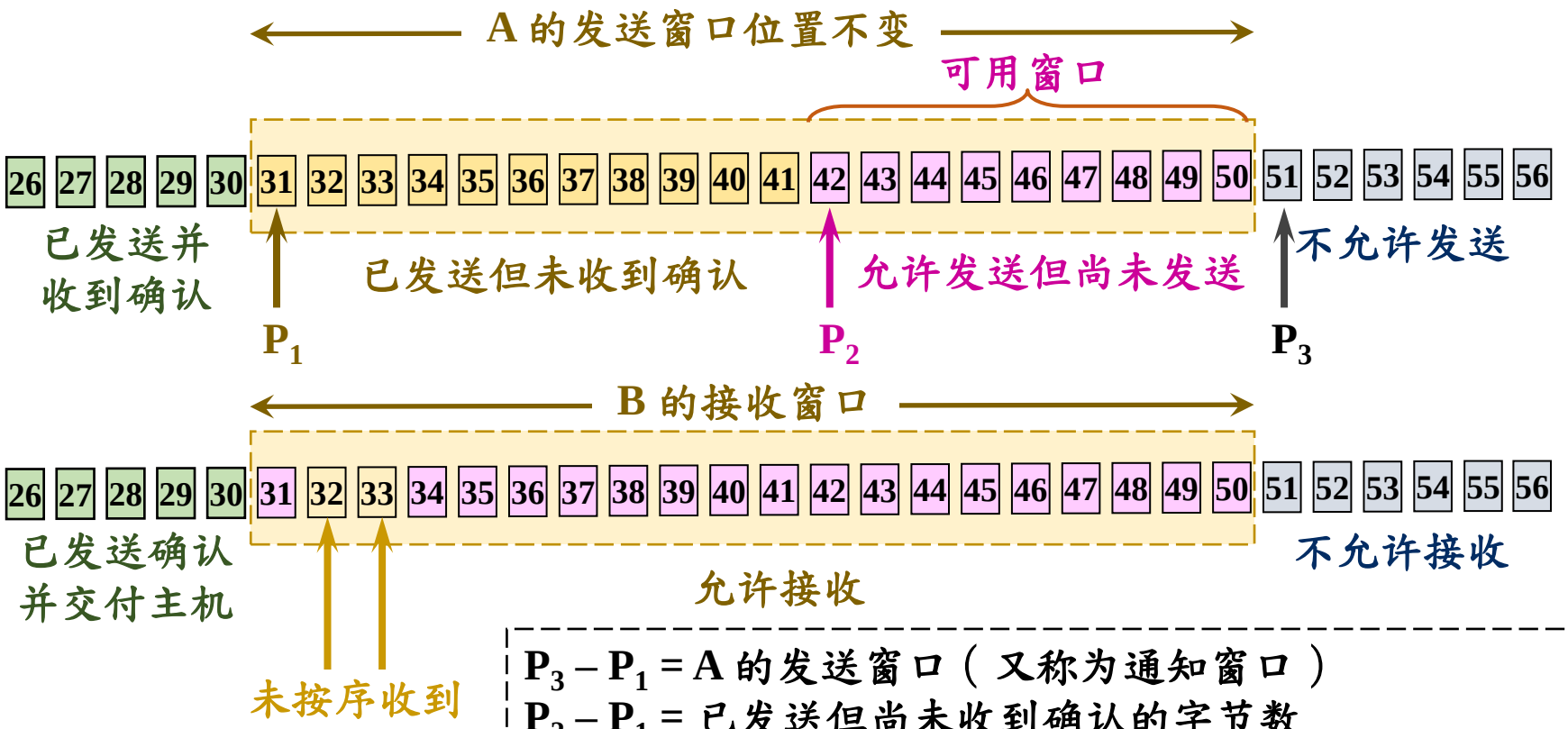
窗口机制：字节为单位的滑动窗口

- 以字节为单位的滑动窗口



窗口机制：字节为单位的滑动窗口

- A 发送了 11 个字节的数据



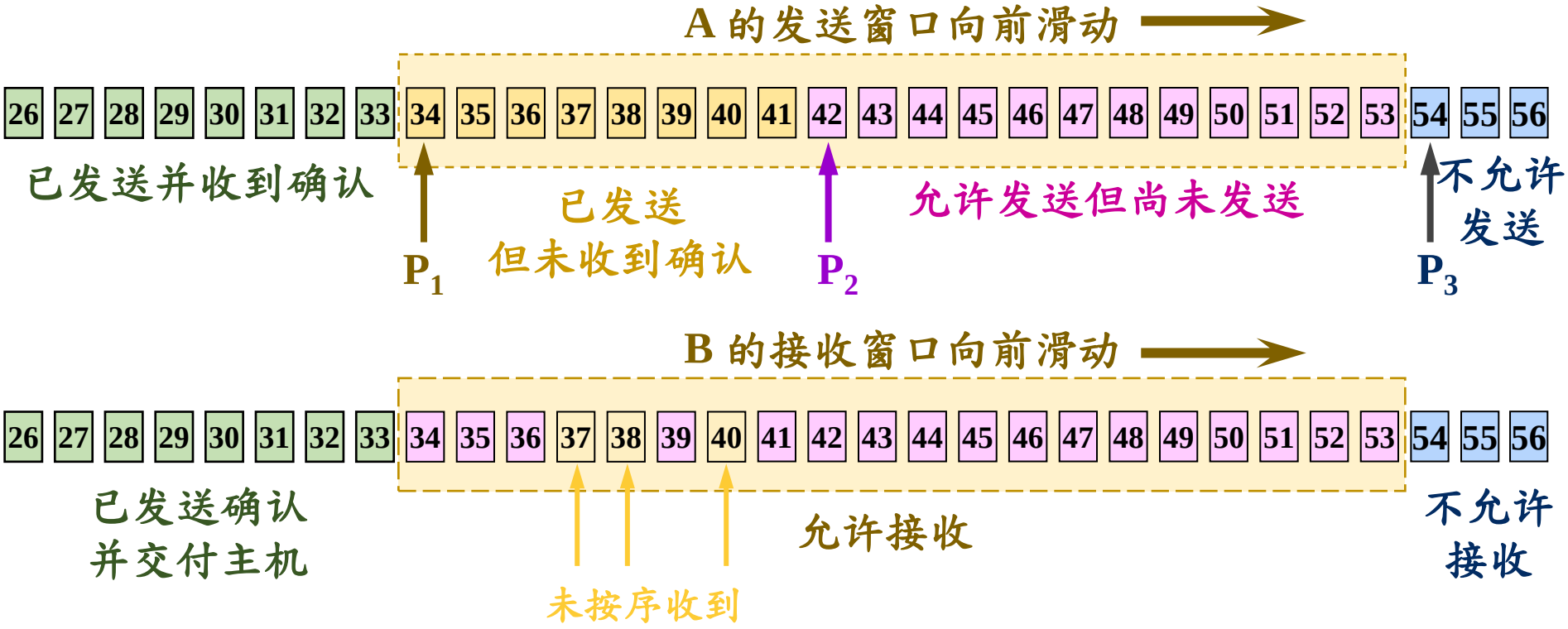
$P_3 - P_1$ = A 的发送窗口（又称为通知窗口）

$P_2 - P_1$ = 已发送但尚未收到确认的字节数

$P_3 - P_2$ = 允许发送但尚未发送的字节数（又称为可用窗口）

窗口机制：字节为单位的滑动窗口

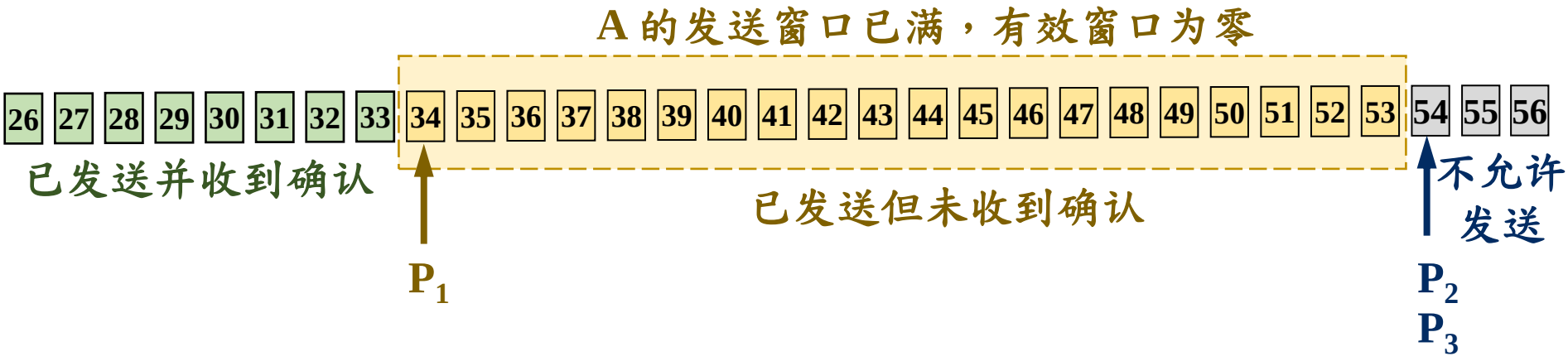
- A 收到新的确认号，发送窗口向前滑动



先存下，等待缺少的数据到达

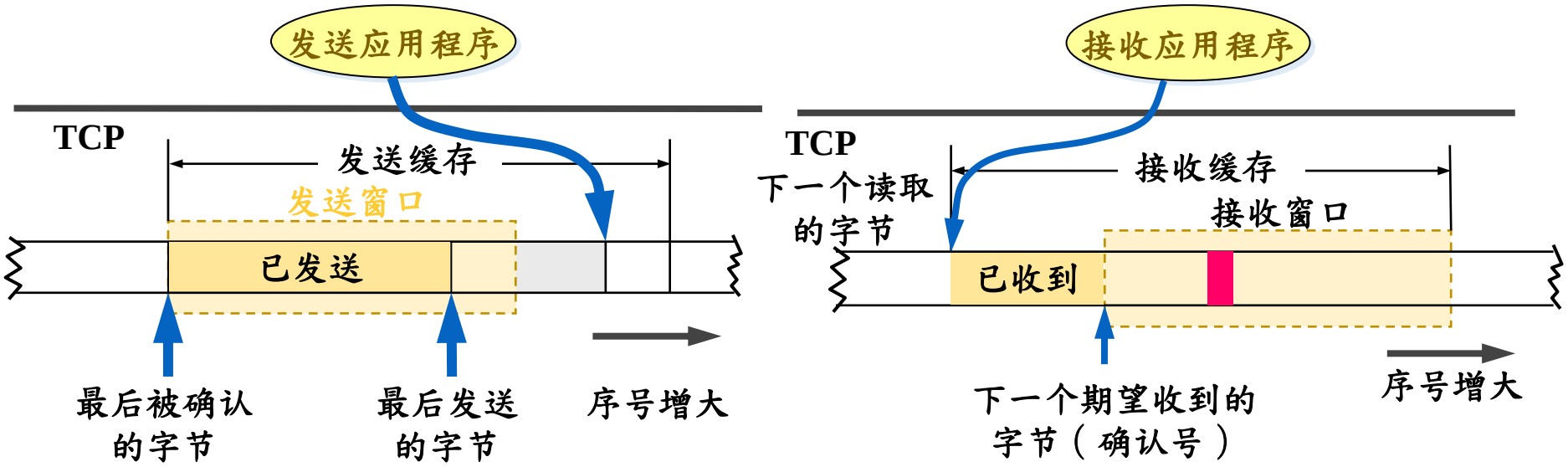
窗口机制：字节为单位的滑动窗口

- 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。



窗口机制：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。



窗口机制：发送与接收缓存

- 发送缓存的作用
  - 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。
- 接收缓存的作用
  - 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。

窗口机制

- 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。

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）。

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。

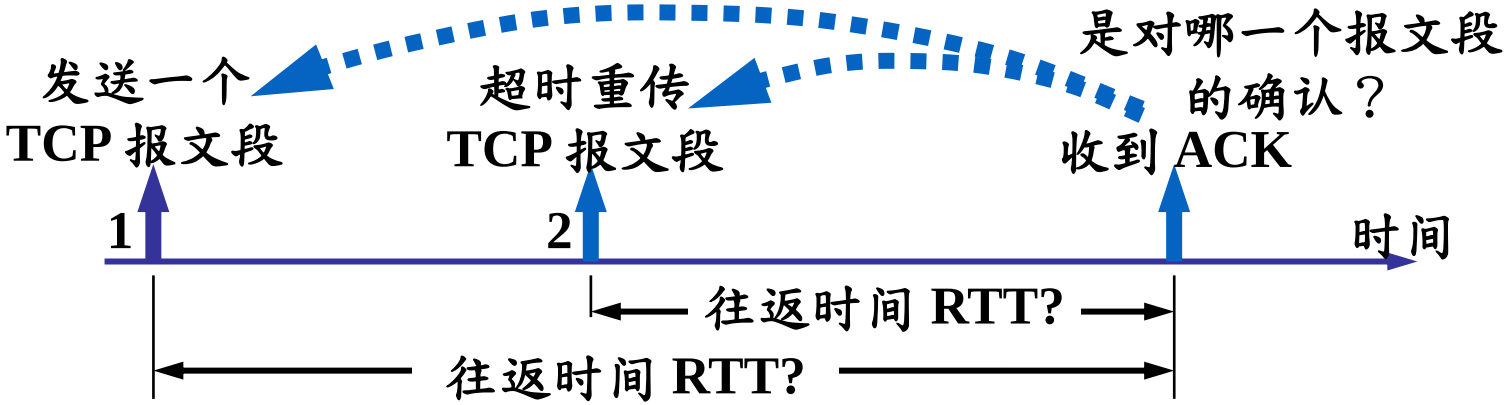
  – RFC 2988 推荐的 α 值为 0.125。

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。

超时重传：时间的选择

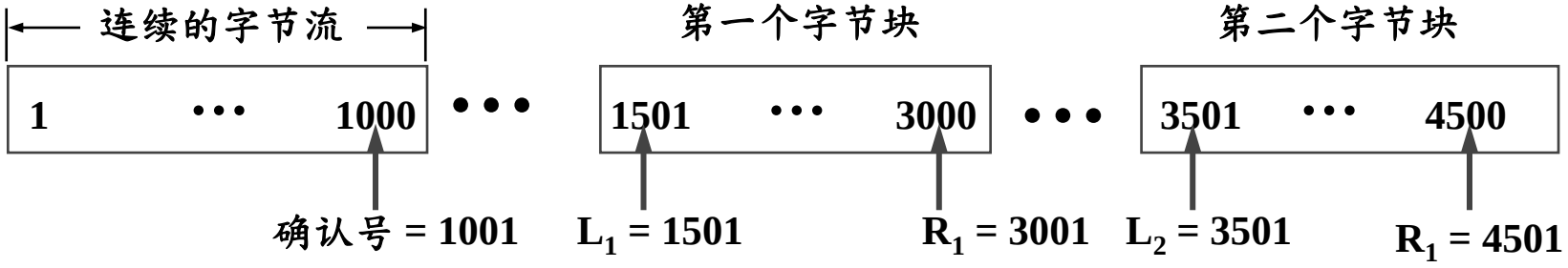
- 误判重传确认和原报文的确认



- **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

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。



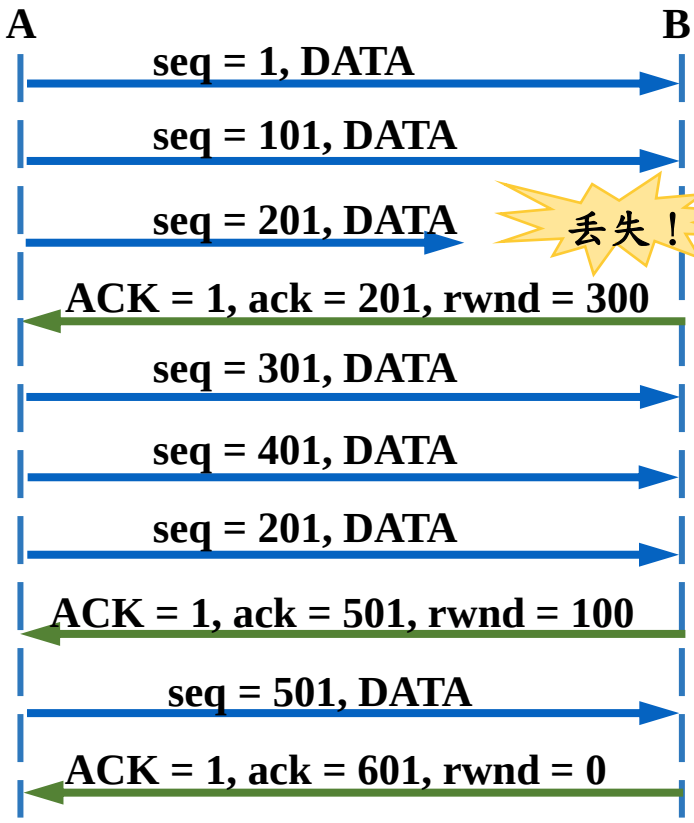
内容纲要

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

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）”。



A 发送了序号 1 至 100，还能发送 300 字节

A 发送了序号 101 至 200，还能发送 200 字节

允许 A 发送序号 201 至 500 共 300 字节

A 发送了序号 301 至 400，还能再发送 100 字节新数据

A 发送了序号 401 至 500，不能再发送新数据了

A 超时重传旧的数据，但不能发送新的数据

允许 A 发送序号 501 至 600 共 100 字节

A 发送了序号 501 至 600，不能再发送了

不允许 A 再发送（到序号 600 为止的数据都收到了）

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。

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。

内容纲要

拥塞控制：原理

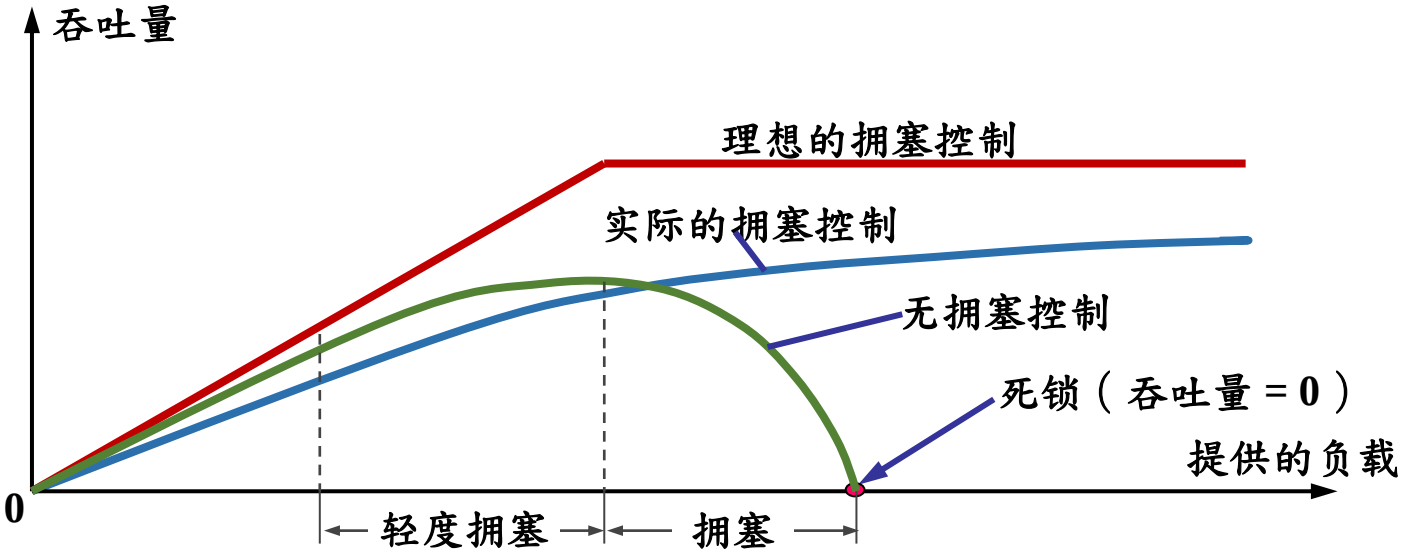
- 在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏——产生拥塞(congestion)。
- 出现资源拥塞的条件：对资源需求的总和 > 可用资源
- 若网络中有许多资源同时产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降。

拥塞控制：与流量控制关系

- 拥塞控制
  - 前提：网络能够承受现有的网络负荷。
  - 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
  - 指在给定的发送端和接收端之间的点对点通信量的控制。
  - 抑制发送端发送数据的速率，以便使接收端来得及接收。

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，拥塞成为网络性能恶化甚至死锁的原因。



拥塞控制：拥塞控制思路

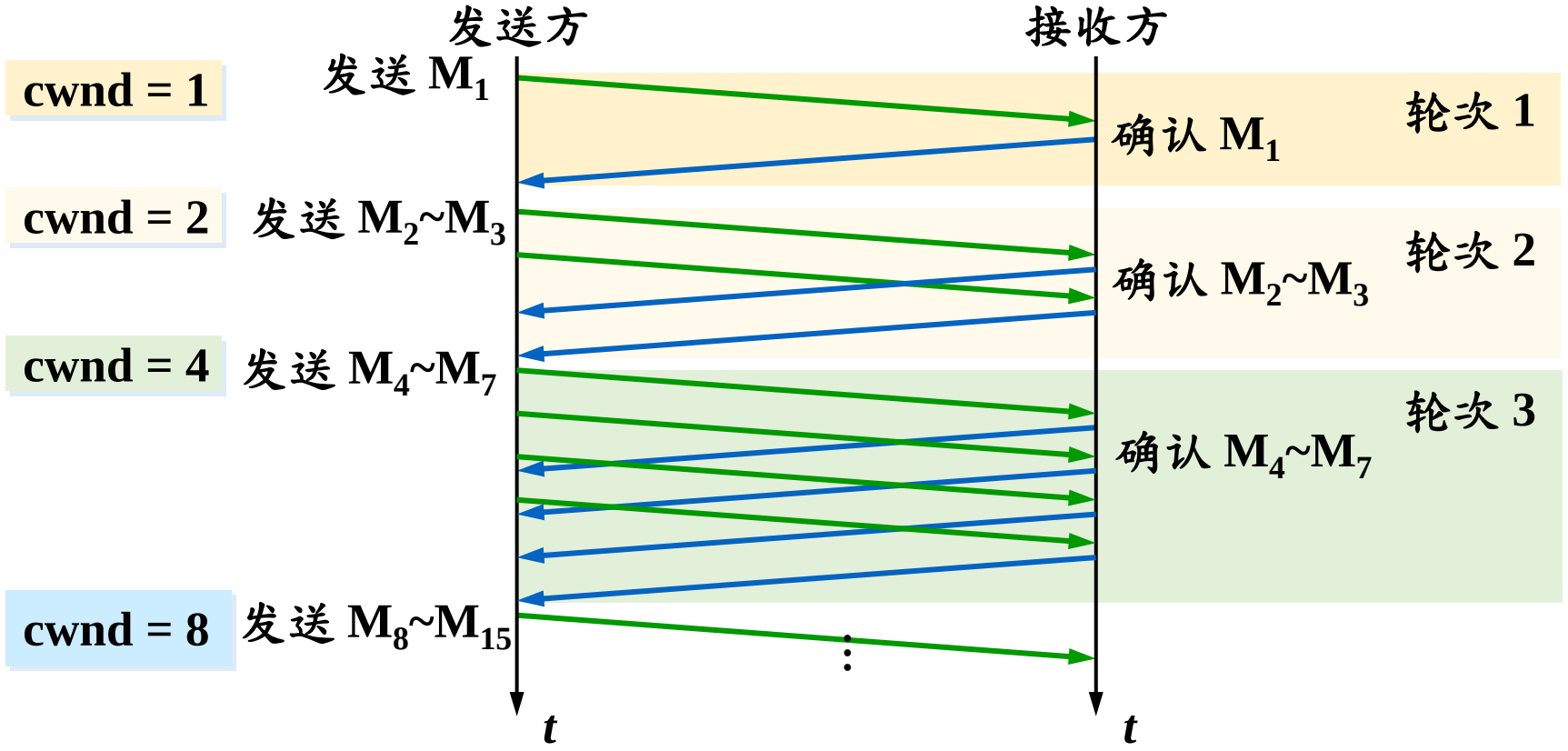
- 开环控制（Open loop）
  - 设计网络事先将有关拥塞的因素考虑周到，力求网络在工作时不产生拥塞。
- 闭环控制（Close loop）
  - 基于反馈环路的概念，属于闭环控制
  - 措施
    - 监测网络系统以便检测到拥塞在何时、何处发生
    - 将拥塞发生的信息传送到可采取行动的地方
    - 调整网络系统的运行以解决出现的问题

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。

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。



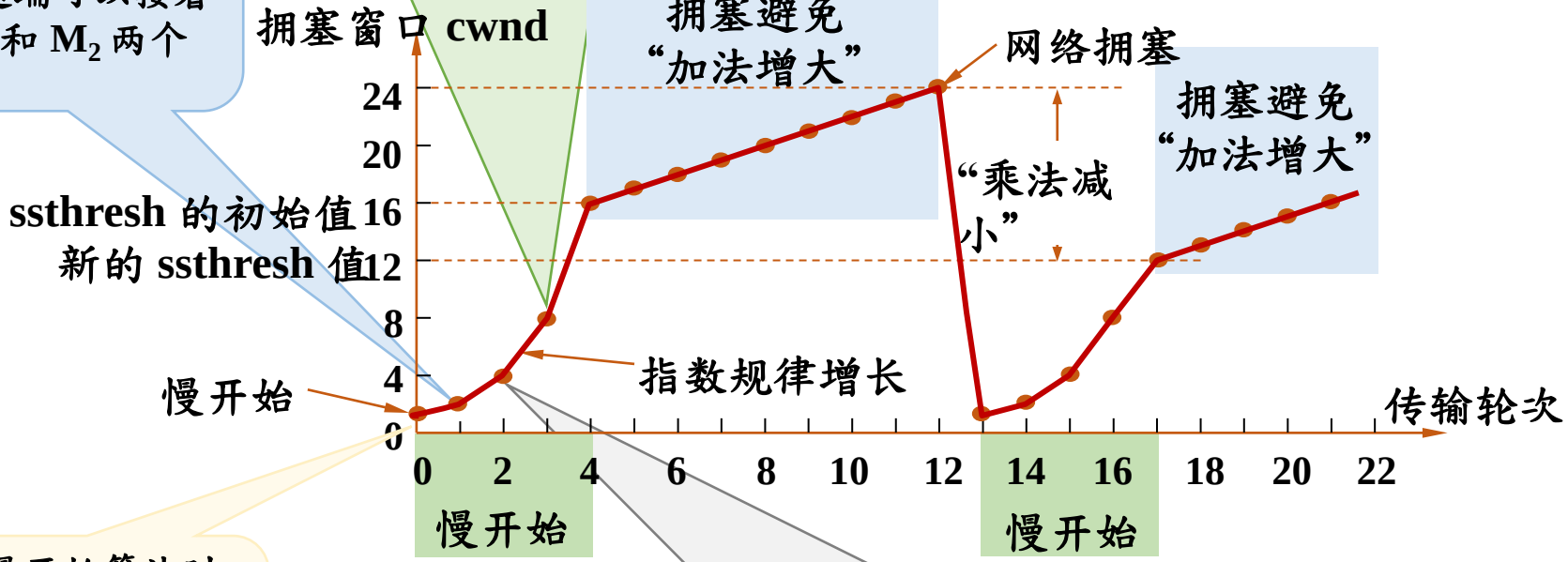
拥塞控制：慢开始和拥塞避免

- 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
  - 一旦拥塞，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）

拥塞控制：慢开始和拥塞避免

发送端每收到一个确认，就把发送端的拥塞窗口加 1，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

发送端每收到一个确认，就把 cwnd 加 1。于是发送端可以接着发送 M_1 和 M_2 两个报文段。



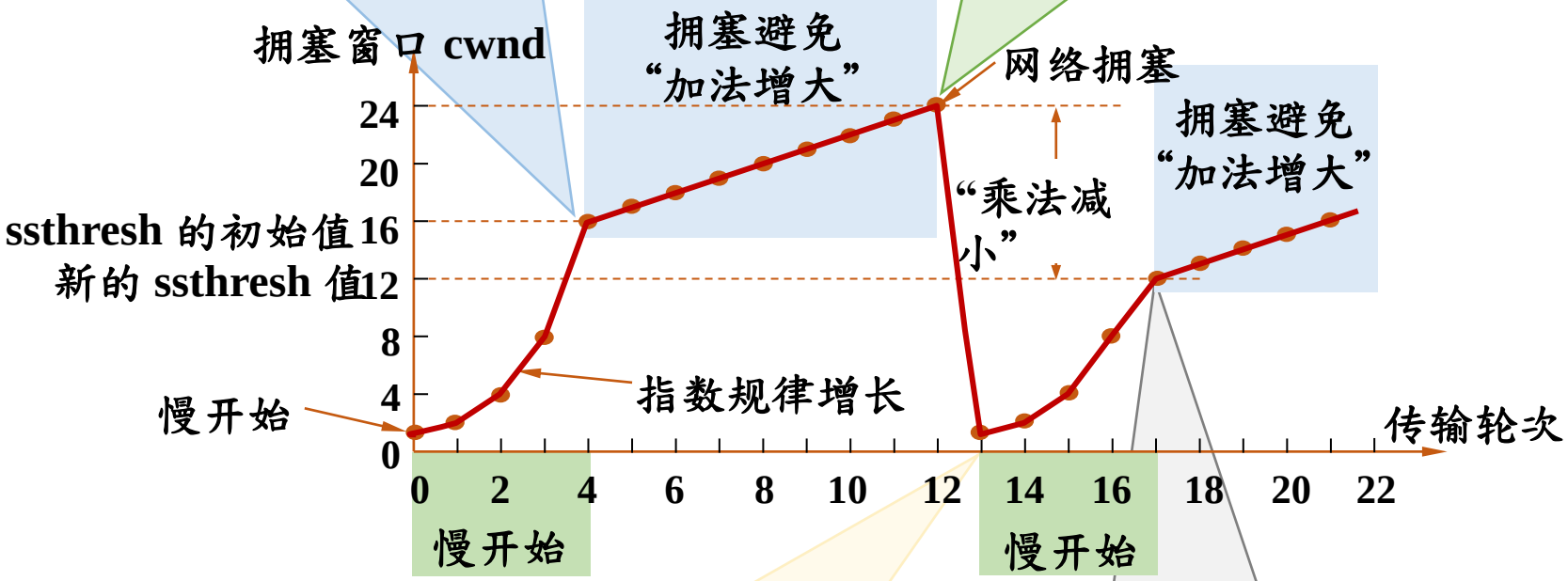
在执行慢开始算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M_0。

接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加 1。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

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。



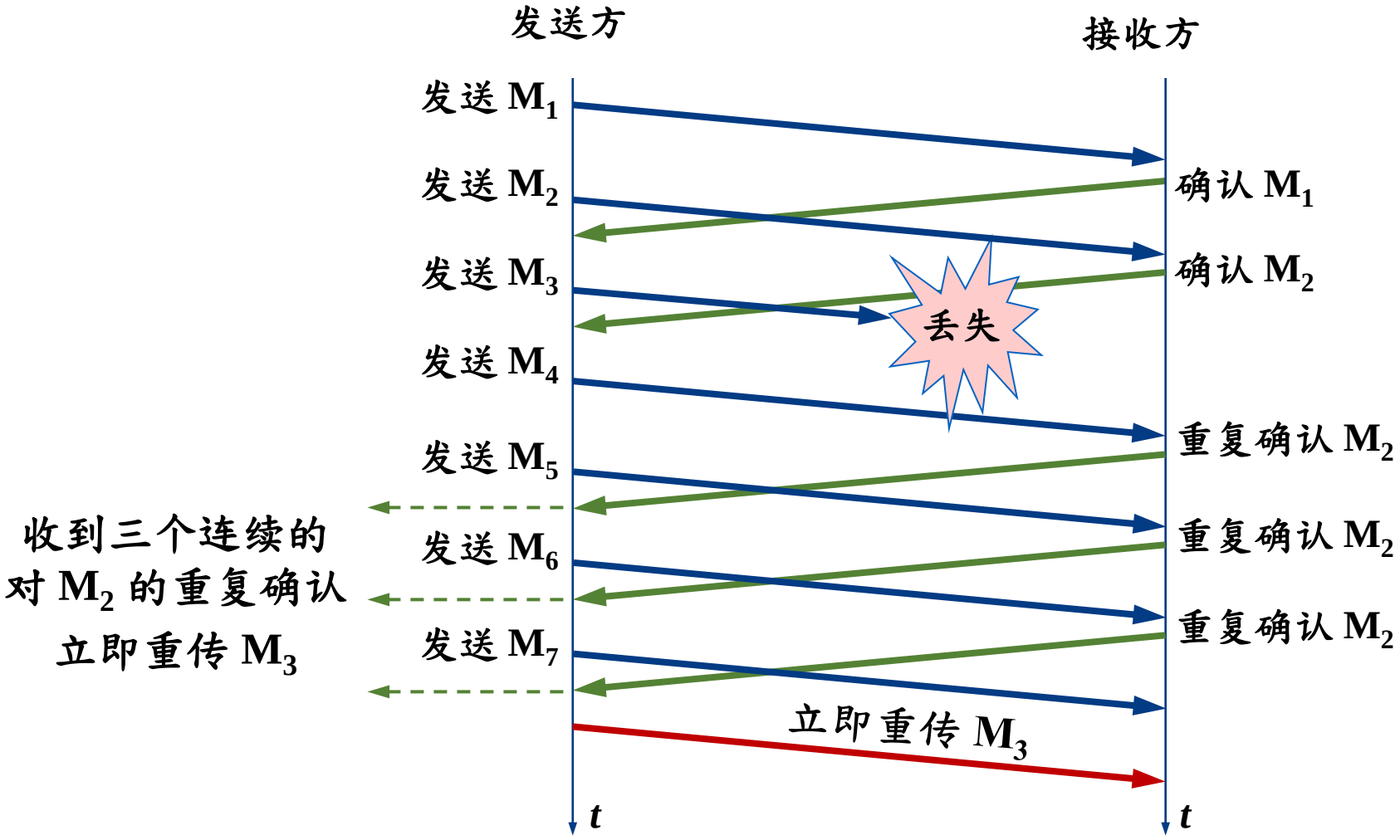
更新后的 ssthresh 值变为 12（即发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并执行慢开始算法。

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。

拥塞控制：快重传和快恢复

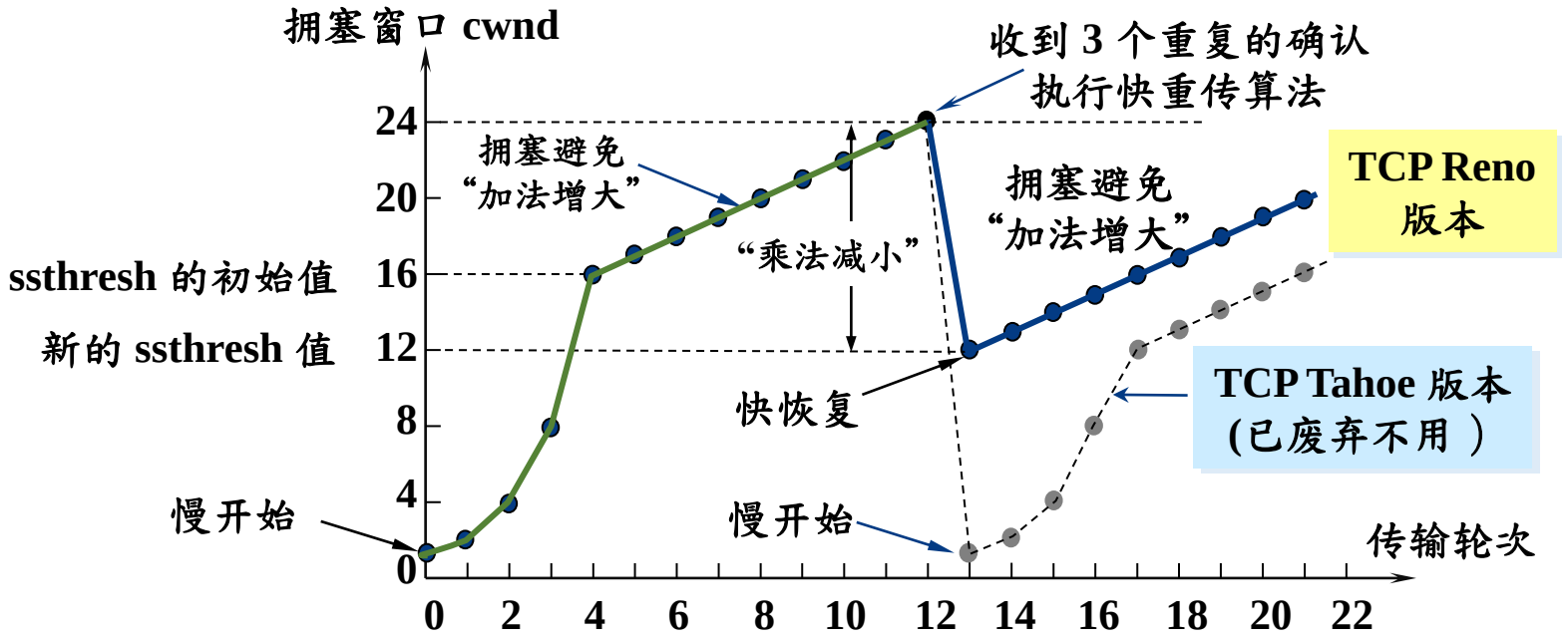
- 快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认。这样做可以让发送方及早知道有报文段没有到达接收方。
- 发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段。
- 不难看出，快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段。

拥塞控制：快重传和快恢复



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。

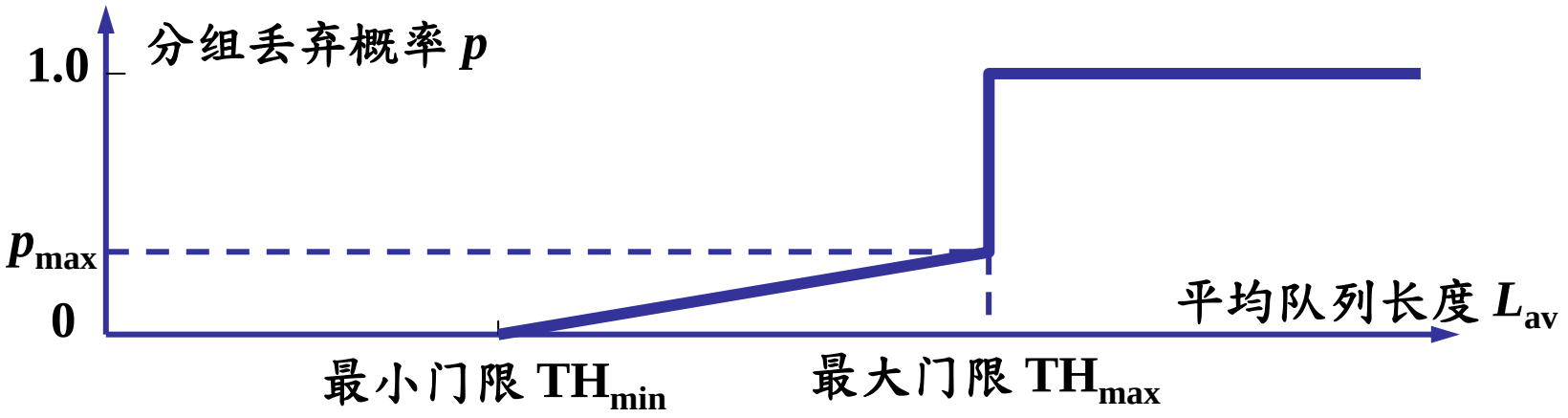


拥塞控制：快重传和快恢复

- 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，接收方的接收能力限制发送窗口的最大值。
  - 当cwnd < rwnd，网络的拥塞限制发送窗口的最大值。

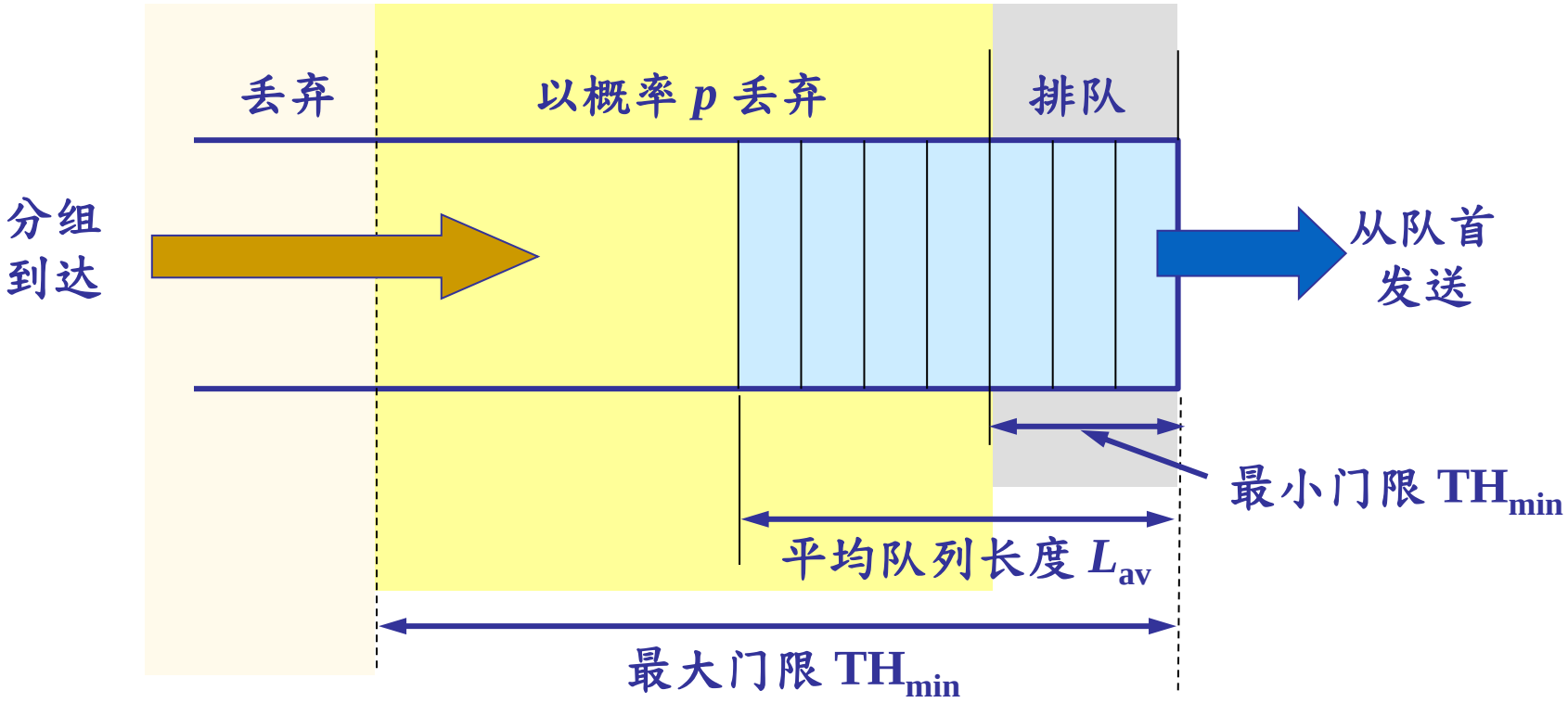
拥塞控制：随机早期检测RED

- 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}。



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



内容纲要

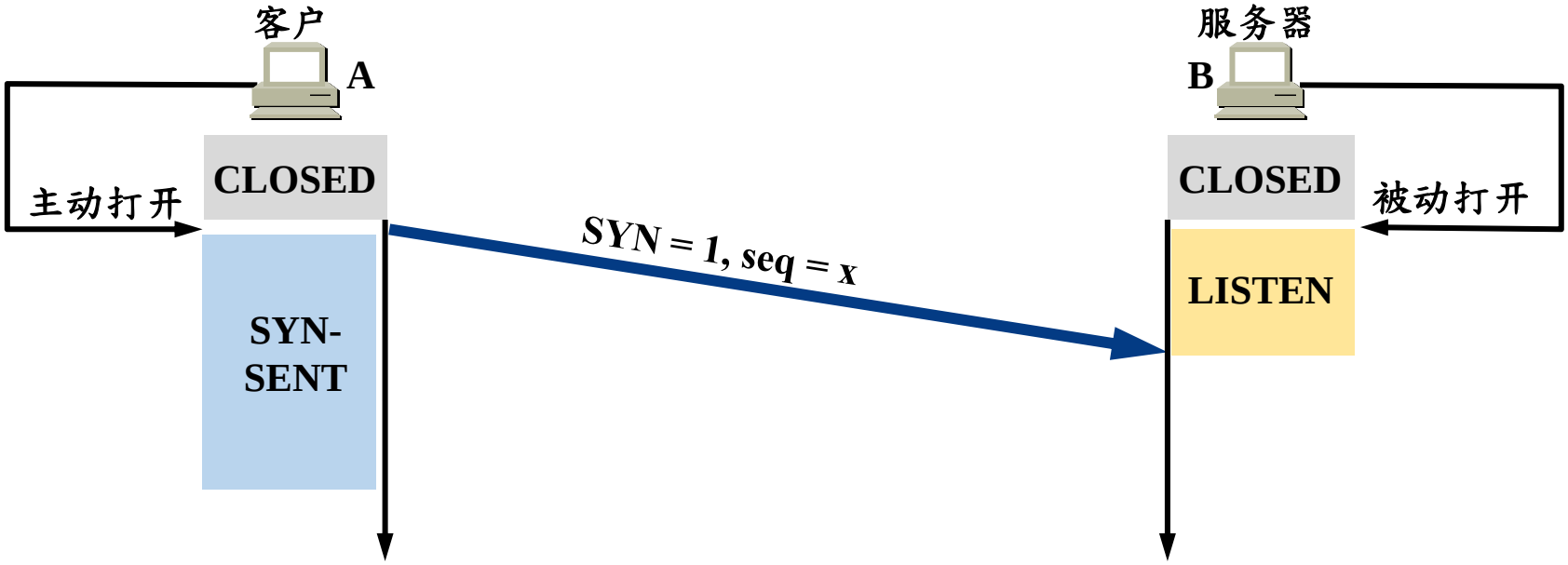
3	传输控制协议
3-2	基本机制
3-3	流量控制
3-4	拥塞控制
3-5	连接管理

传输连接管理

- 三个阶段：连接建立、数据传送和连接释放。
- 通信双方：客户端、服务器端。
  - 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）
- 要解决三个问题：
  - 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：最大窗口大小等）。
  - 能够对运输实体资源（如缓存大小等）进行分配。

传输连接管理：连接建立

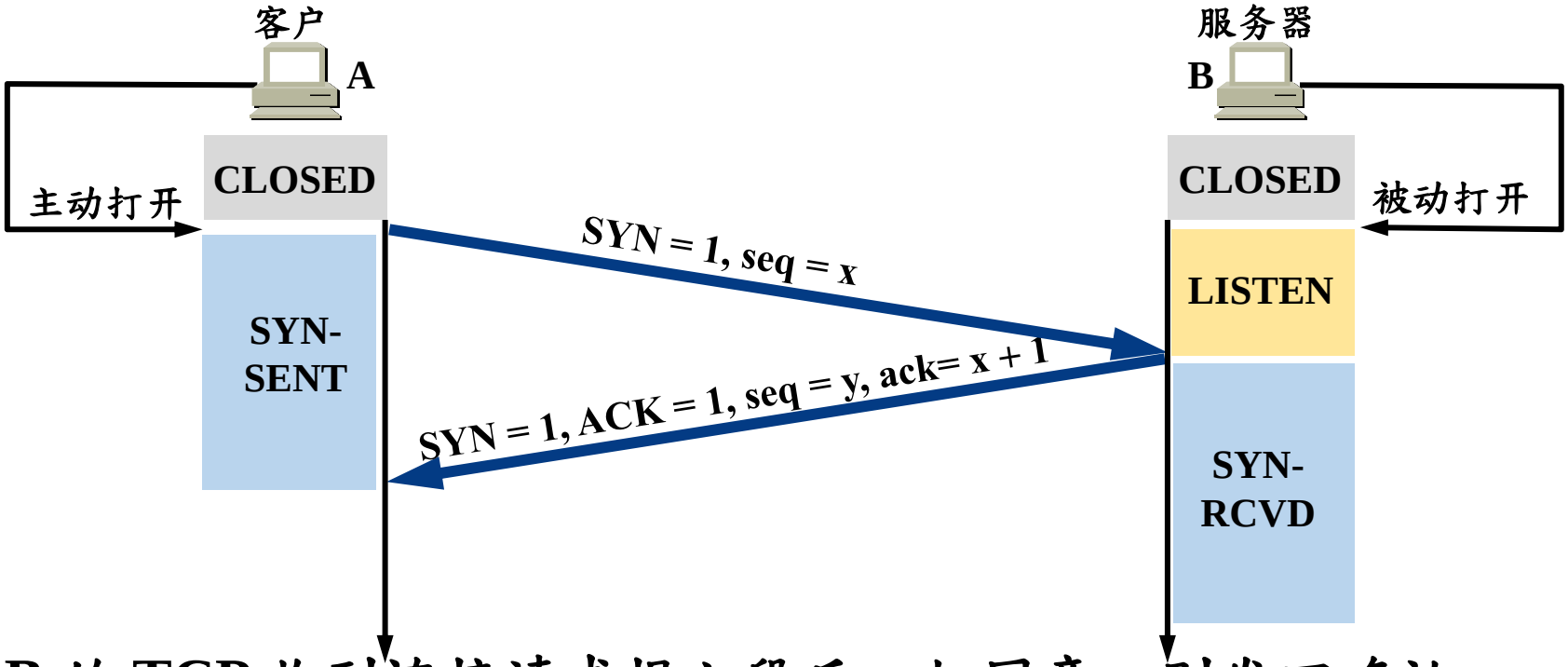
- 三次握手建立 TCP 连接



- 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。

传输连接管理：连接建立

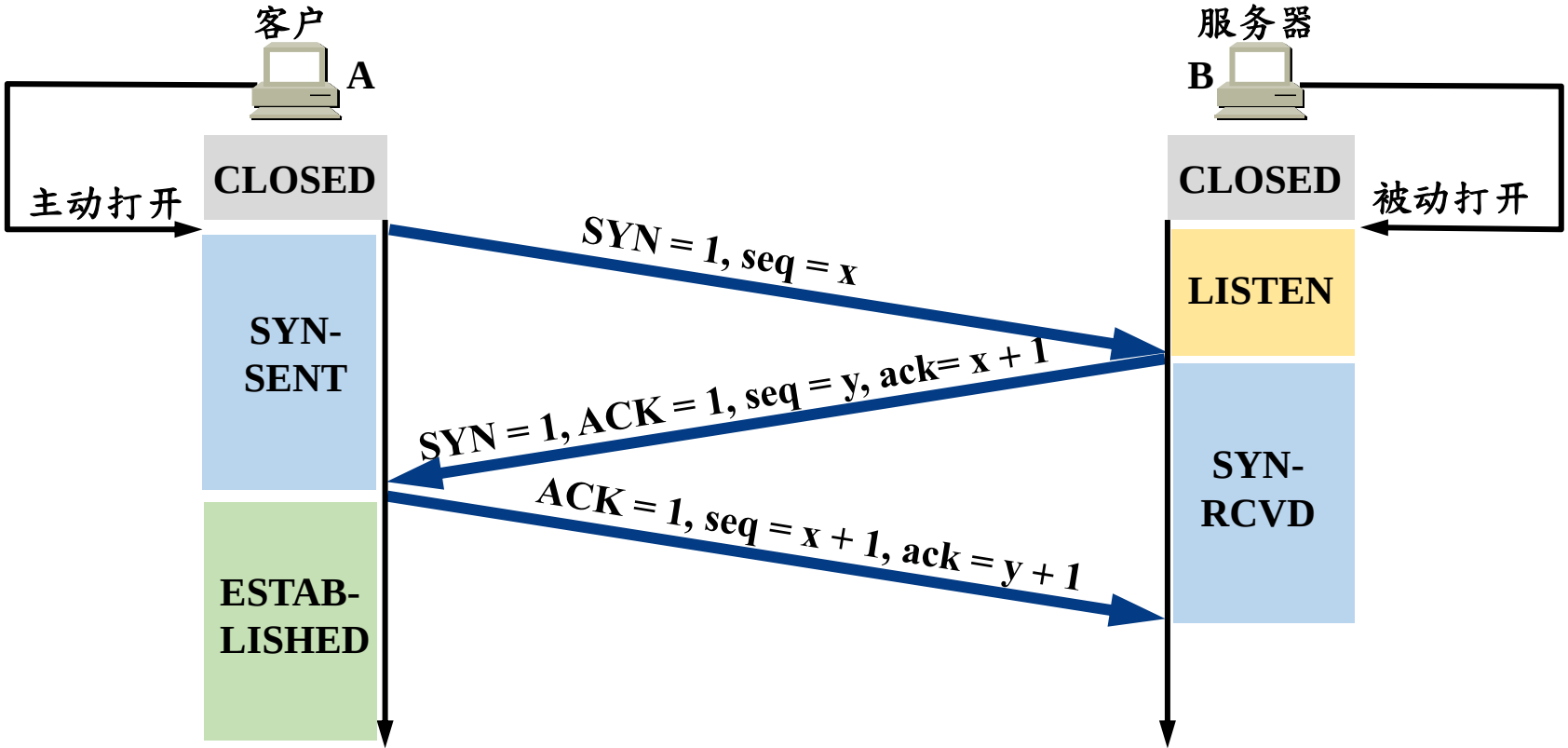
- 三次握手建立 TCP 连接



- 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。

传输连接管理：连接建立

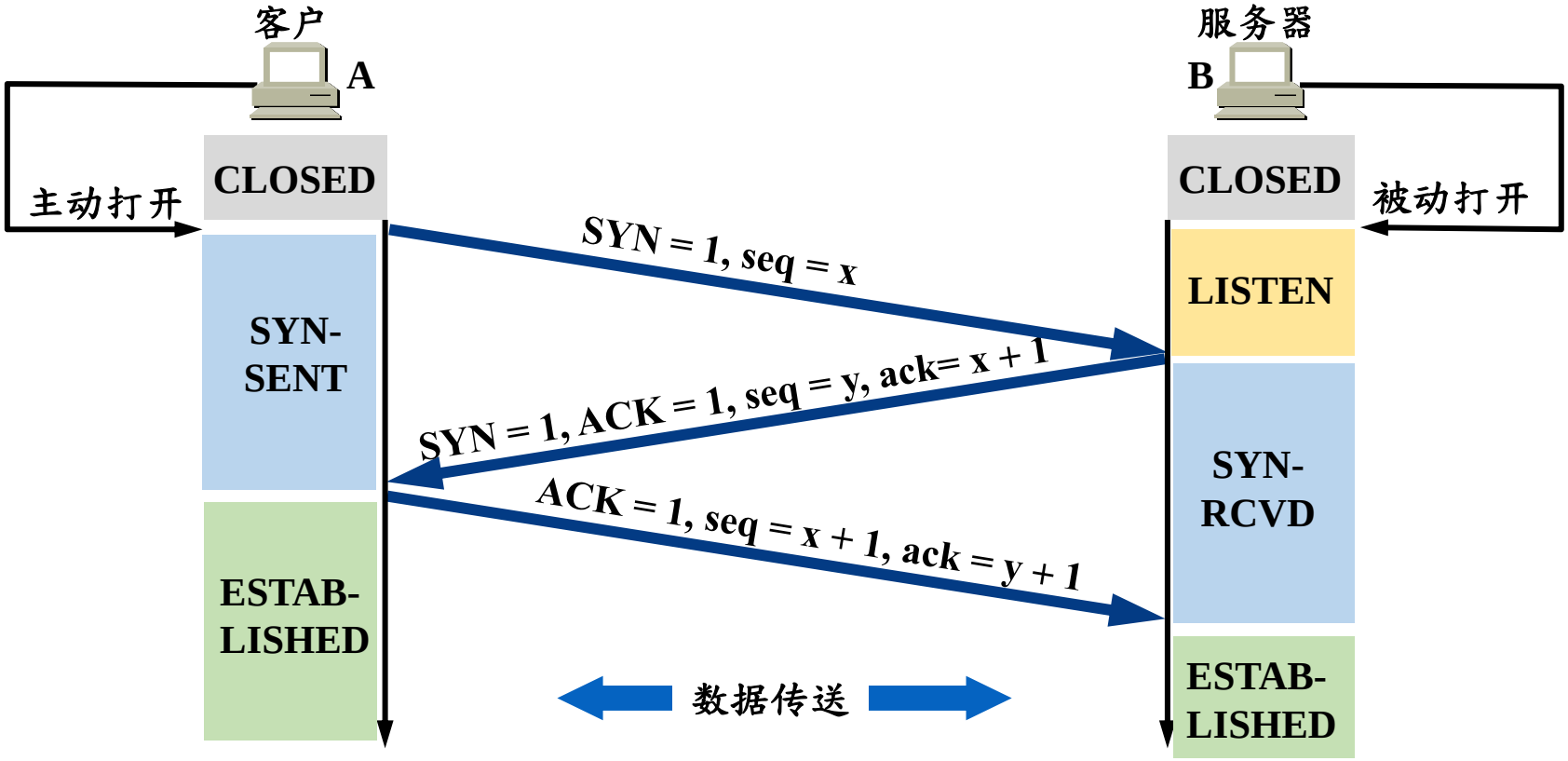
- 三次握手建立 TCP 连接



- 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。

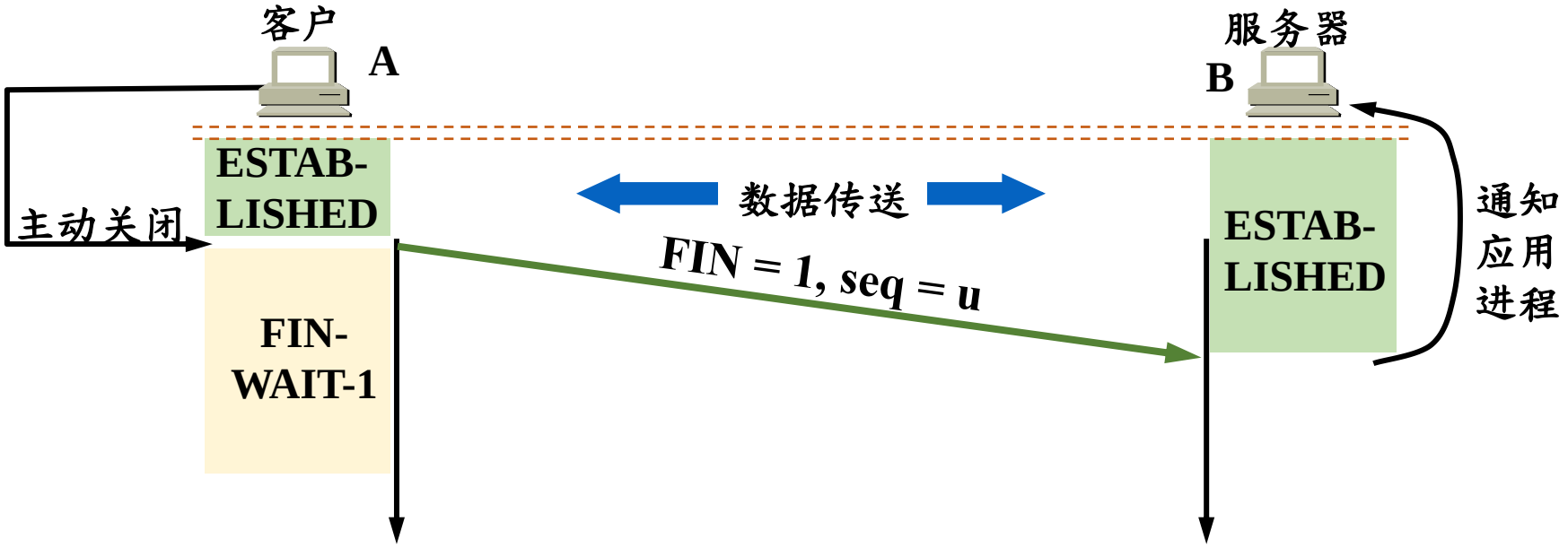
传输连接管理：连接建立

- 三次握手建立 TCP 连接



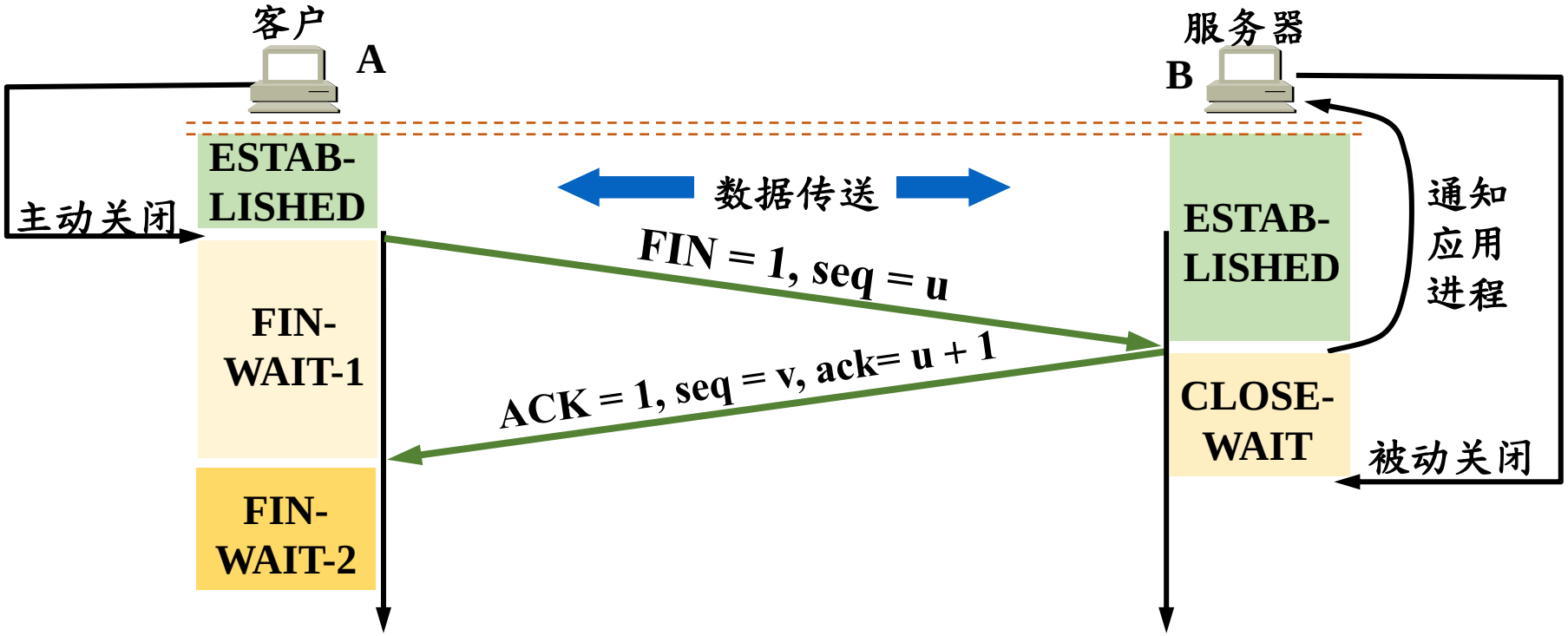
- B收到A确认后，通知其上层应用进程TCP 连接已经建立。

传输连接管理：连接解除



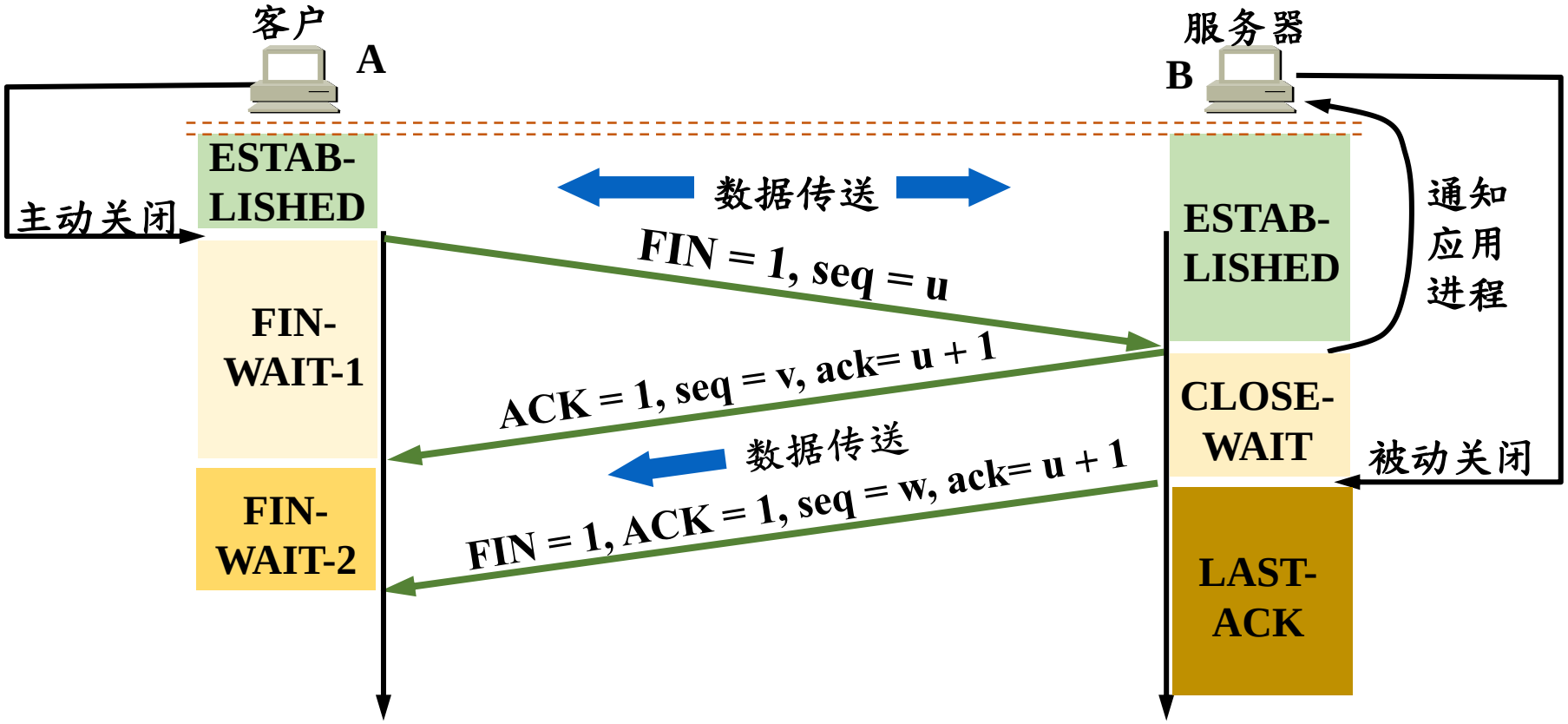
- 数据传输结束后，通信的双方都可释放连接。现在 A 的应用进程先向其 TCP 发出连接释放报文段，并停止再发送数据，主动关闭 TCP连接。
- A 把连接释放报文段首部的 FIN = 1，其序号 seq = u，等待 B 的确认。

传输连接管理：连接解除



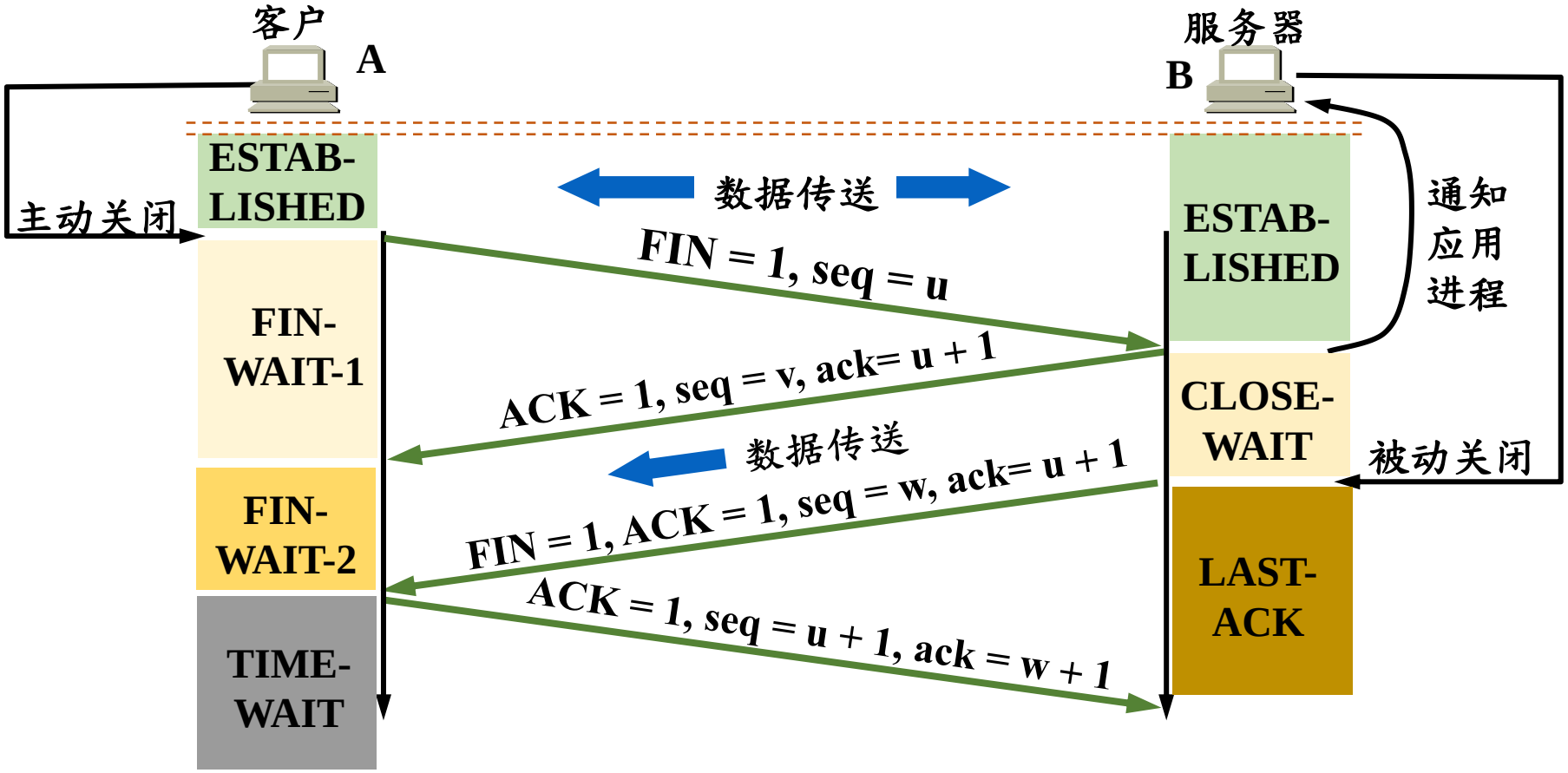
- B 发出确认，确认号ack=u+1，该报文段的序号 seq = v。
- TCP 服务器进程通知高层应用进程。
- 从 A 到 B 这个方向的连接就释放了，TCP 连接处于半关闭状态。B 若发送数据，A 仍要接收。

传输连接管理：连接解除



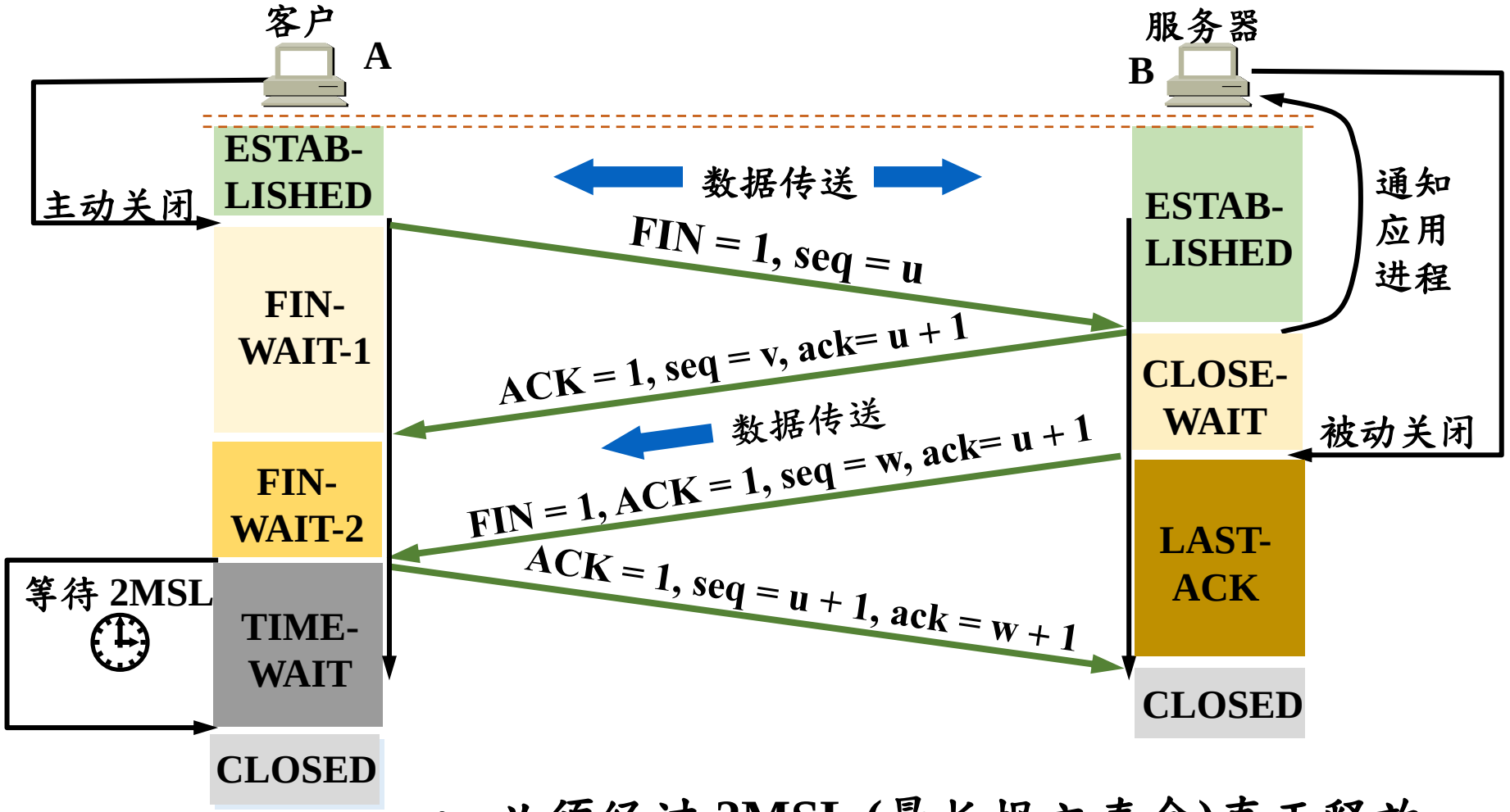
- 若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。

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

传输连接管理：连接解除



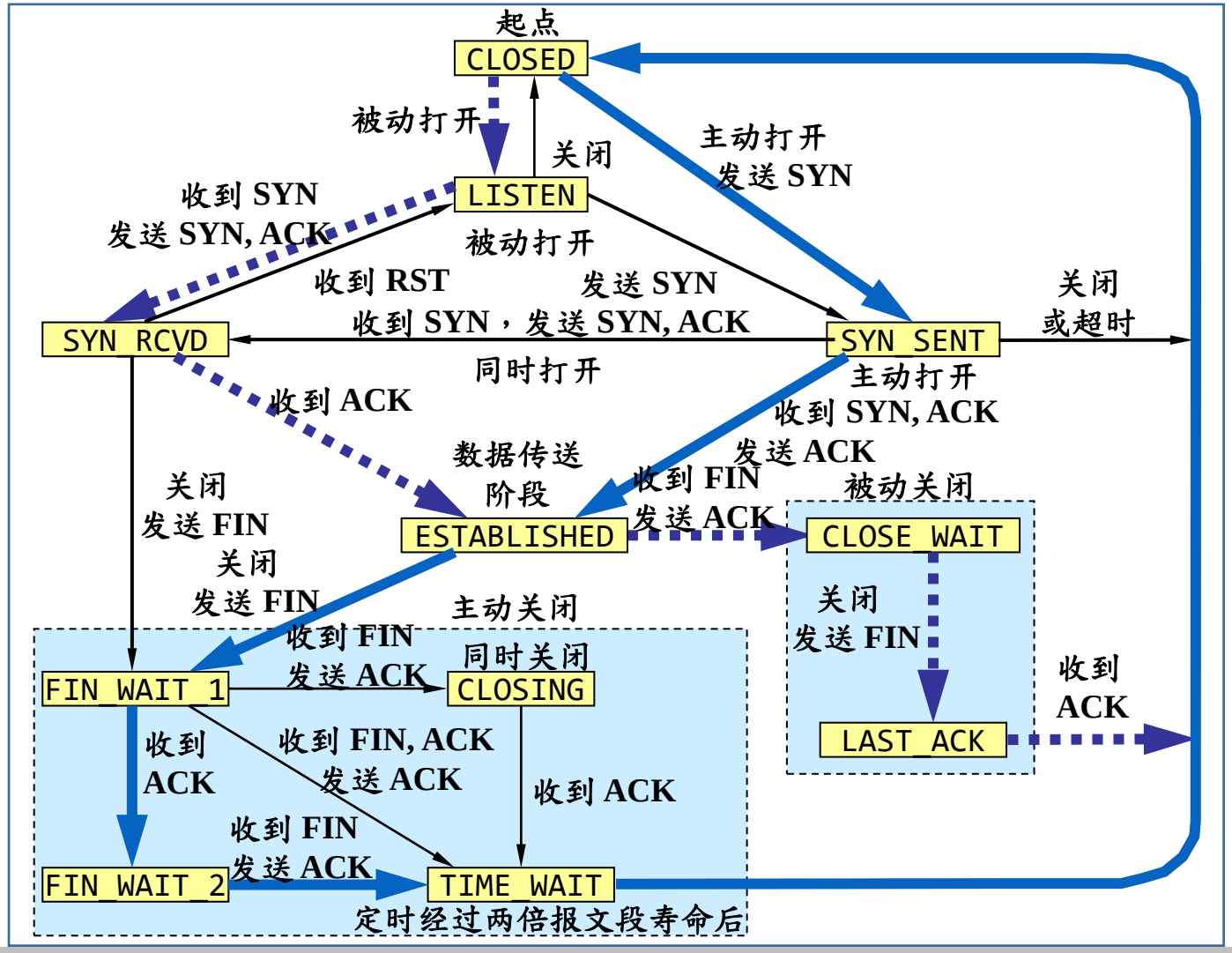
- 必须经过 2MSL (最长报文寿命)真正释放。

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标准 TCP 连接状态名。
- 状态之间的箭头表示可能发生的状态变迁。
  - 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。

传输连接管理：有限状态机

- 有限状态机



内容纲要

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) *

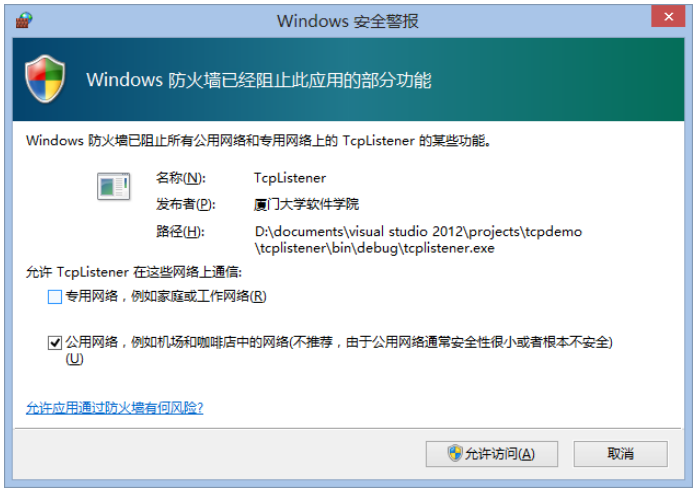
Output on failure:

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 = new TcpListener(portNum);
        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...
```

内容纲要

计 算 机 网 络
Computer Network

谢谢观看
Thank you

12

传输控制协议

理论课程

厦門大學
XIAMEN UNIVERSITY

信息学院 黄 煒
（国家示范性软件学院） 博士，副教授
School of Informatics Dr. Wei Huang