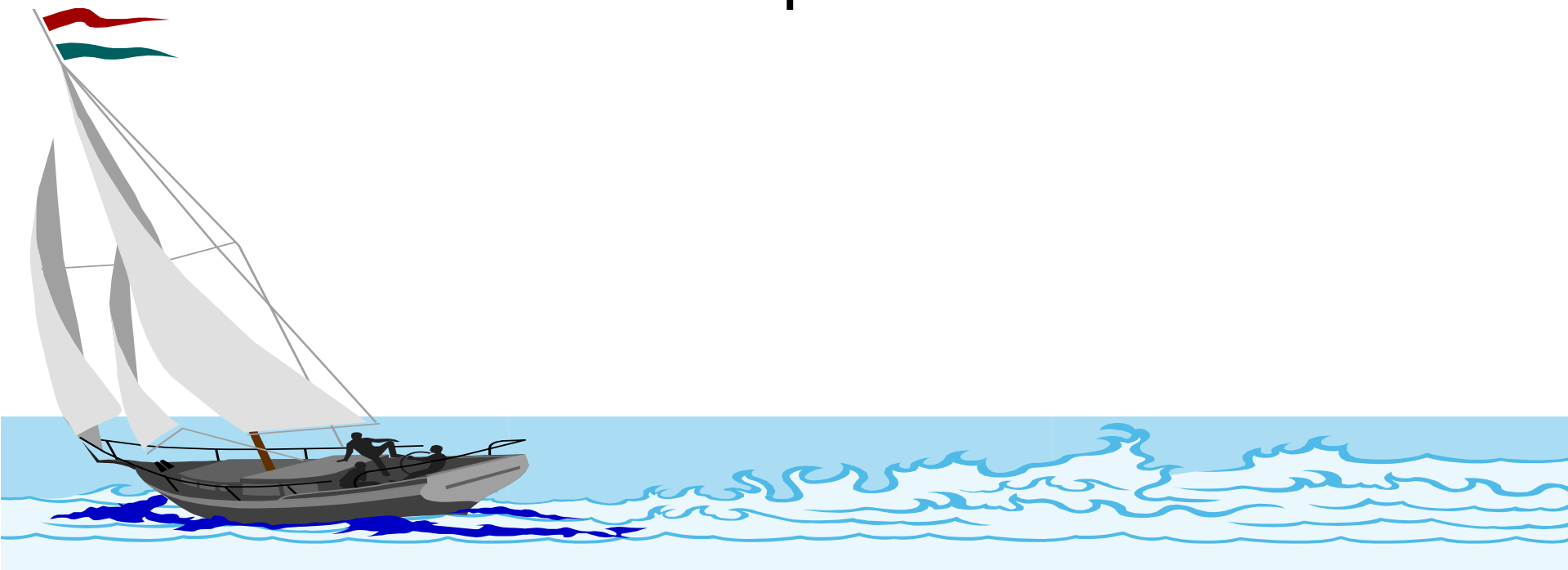
传输控制协议

TCP – Transport Control Protocol



TCP

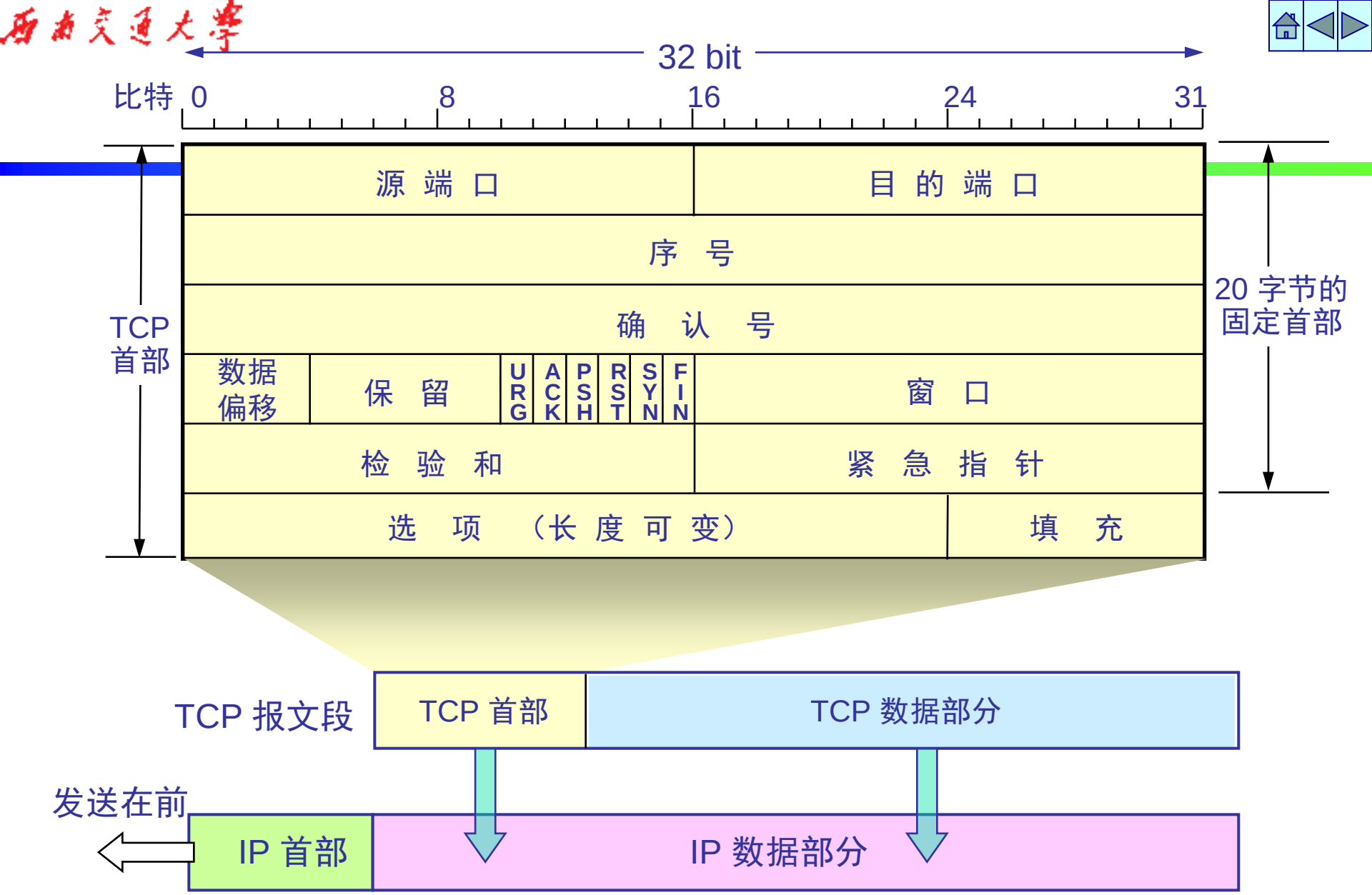
- 概述
- 报文格式
- 连接管理
- TCP中的数据传输
- 流控与阻塞控制
- 错误控制 (Timer)

TCP Overview

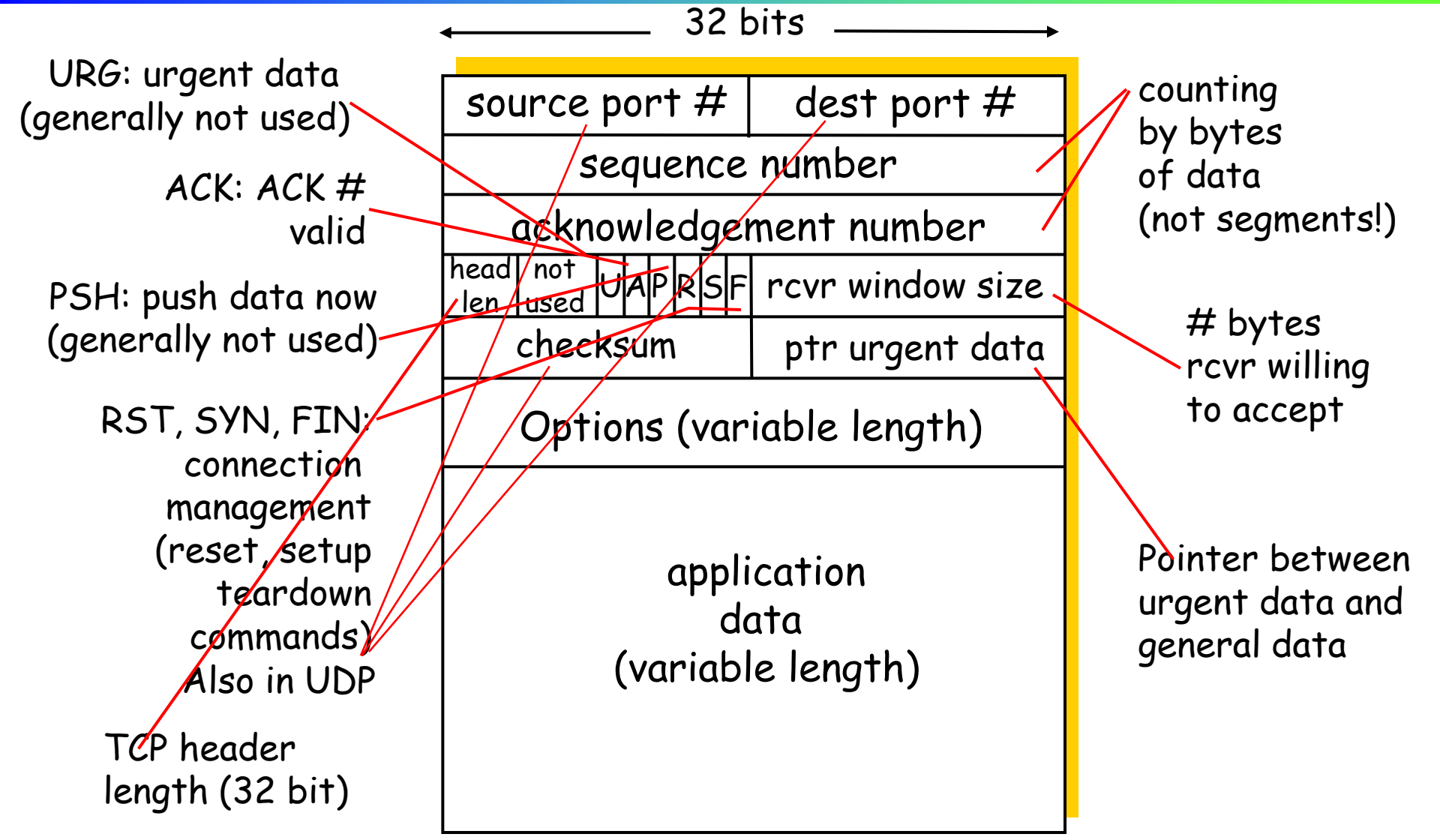
- Connection-oriented
- Reliable data transfer
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network

TCP

- 概述 √
- 报文格式
- 连接管理
- TCP中的数据传输
- 流控与阻塞控制
- 错误控制 (Timer)


32 bit
比特
0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保 留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
TCP 首部
20 字节的固定首部
TCP 报文段
TCP 首部
TCP 数据部分
发送在前
IP 首部
IP 数据部分

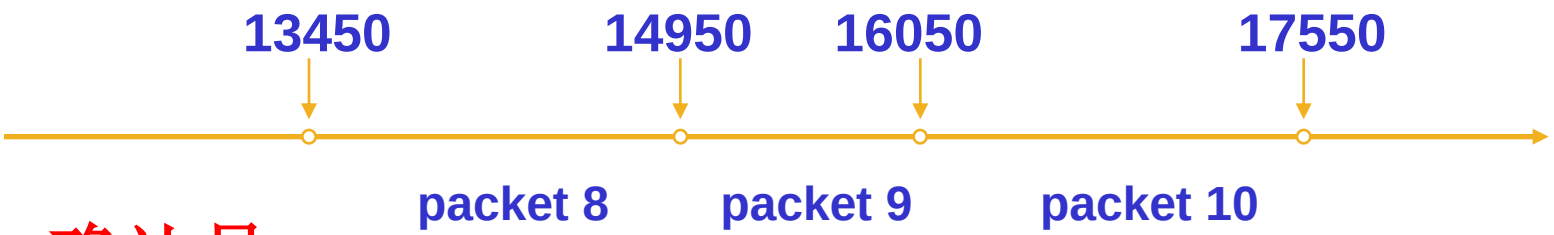
TCP Segment Structure



TCP 序列号和确认码 (I)

序列号:

- 报文段中第一个字节的序号（即数据段的首指针）



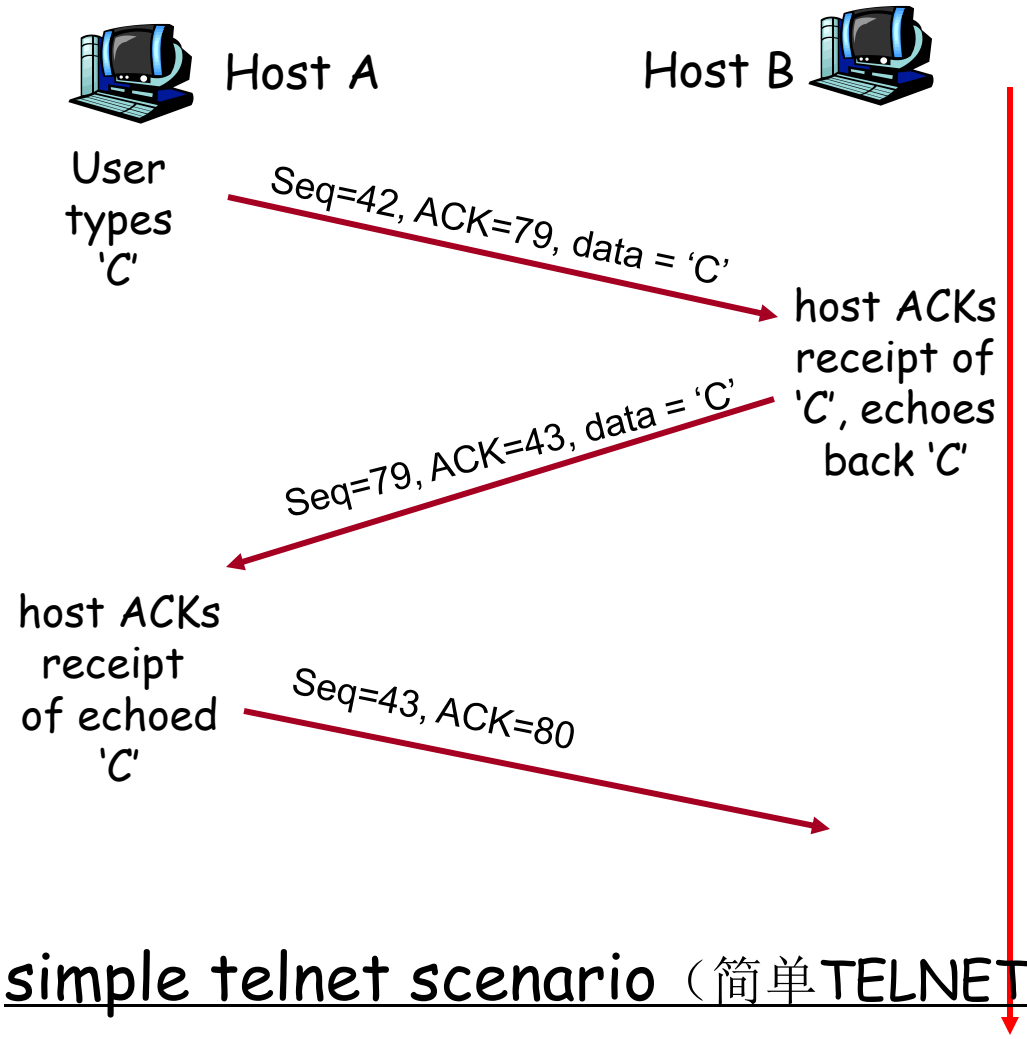
确认号:

- 另一方准备接收的流的序列号
- 累计确认（例如，上图中，如果**packet8**和**9**均正确，则**ack=16050**）

Q: 接收者怎样处理无序报文段?

- 答: TCP 没有规定, - 根据具体应用

TCP Seq. #'s and ACKs (II)



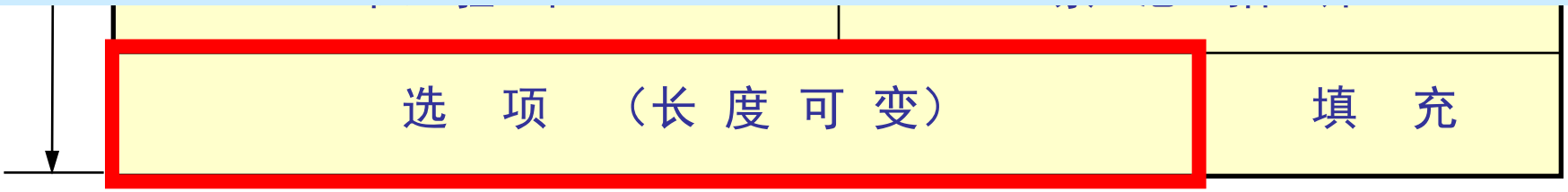
simple telnet scenario（简单TELNET情形）

检验和 —— 占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部(与UDP类似)。

TCP 伪头部

- 校验和 –下列内容以16比特为单位累加求反：
  - TCP 报头
  - TCP 数据
  - 伪头 (从IP数据报头部中提取)
    - 注意: breaks the layering!

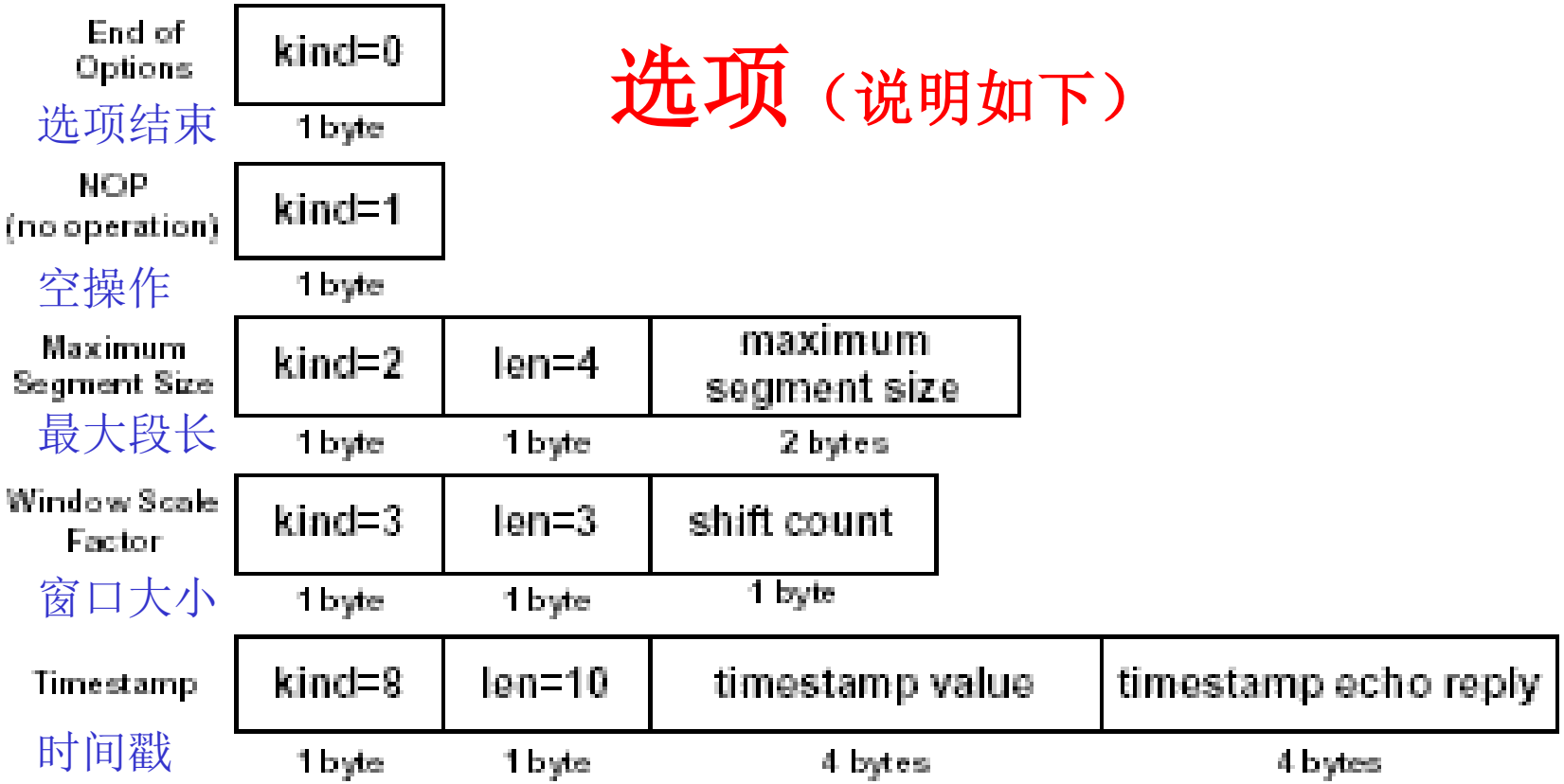
MSS 是 TCP 报文段中的数据字段的最大长度。
数据字段加上 TCP 首部
才等于整个的 TCP 报文段。



选项字段 —— 长度可变。TCP 规定了一种选项，即最大报文段长度 MSS (Maximum Segment Size)。MSS 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长度是 MSS 个字节。”

TCP 选项域

选项（说明如下）



TCP 选项域

- 选项类型及其作用
  - 选项结束
  - NOP 把 TCP 头填充成 4 字节的整数倍
  - 报文段长度的最大值MMS
  - 窗口大小选项
    - ✓ 增大 TCP 窗口，从16位到32位, 即窗口大小是可变的
    - ✓ 这一选项仅仅用于连接建立时的SYN段 (第一段)
  - 时间戳选项
    - ✓ 用于计量往返时间

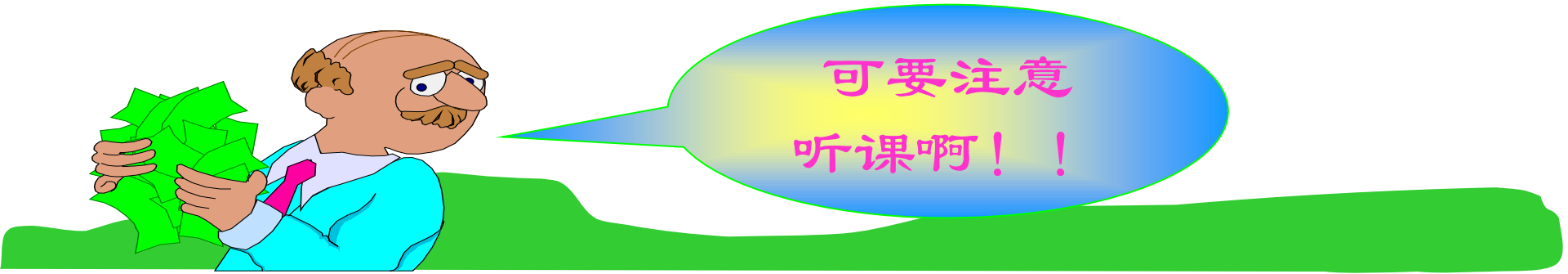
填充字段 —— 这是为了使整个首部长度是 4 字节的整数倍。

TCP

- 概述 √
- 报文格式 √
- 连接管理
- TCP的数据传输
- 流控与阻塞控制
- 错误控制 (Timer)

TCP连接管理

- 建立 一个TCP 连接
- 关闭一个 TCP 连接
- 复位连接
- TCP实现的状态转移图



TCP连接的建立——三次握手

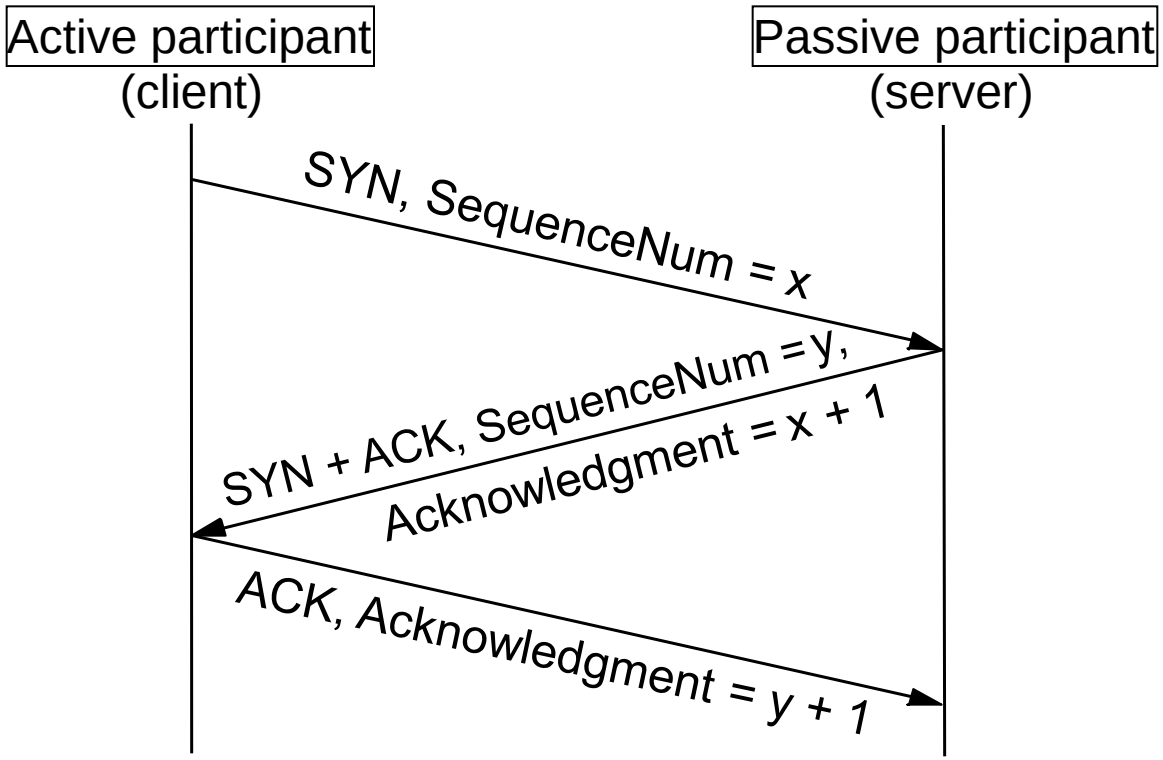
例如：A、B两个主机要建立连接

序号用于跟踪通信顺序，确保多个包传输时的顺序和无数据丢失。

通信双方在建立连接时必须互相交换各自的初始序号。

		方向	消息	含义	握手
	1.	A→B	SYN	我的初始序号是X	1
合并	2.	A←B	ACK	知道了，我准备从X+1接收	2
合并	3.	A←B	SYN	我的初始序号是Y	2
	4.	A→B	ACK	知道了，我准备从Y+1接收	3

Connection Establishment



- 通过三次握手建立一个TCP连接
  - ✓ 目的：协商参数，如双方的初始序列号、最大报文段长度、最大窗口大小、服务质量等

两军问题

两军问题：占据两个山顶的蓝军与驻扎在山谷的白军作战。力量对比是：每一个山顶上的蓝军单独都打不过白军，但两个山顶的蓝军协同作战就可战胜白军。一个山顶上的蓝军拟于次日正午向白军发起攻击。于是发送电文给另一山顶上的友军。但通信线路很不好，电文出错的可能性很大。因此要求收到电文的友军必须发送确认电文。但确认电文也可能出错。试问能否设计出一种协议，使得蓝军能实现协同作战因而一定(即100 %)取得胜利？

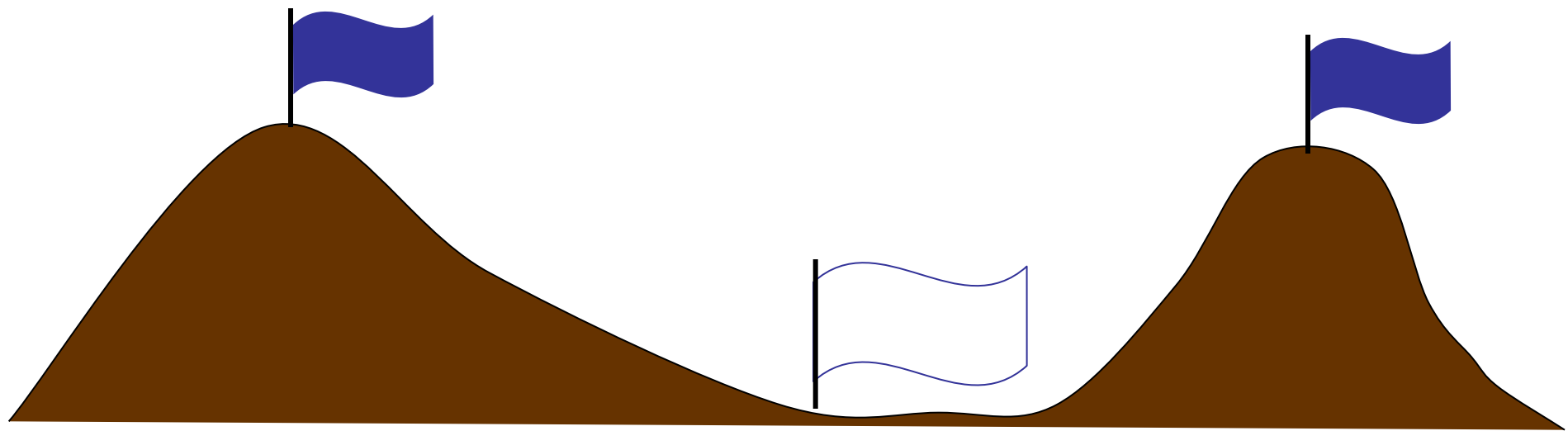
明日正午进攻，如何？

同意

收到“同

意”

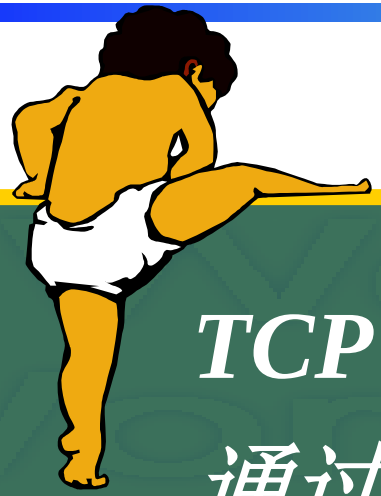
只要能成功执行前面的4步，就能保证蓝军之间协调行动。问题是中间的某一步完成不了时，就会循环。
因此，这样的协议无法实现！



结论

- 这样无限循环下去，两边的蓝军都始终无法确定自己最后发出的电文对方是否已经被对方收到。
- 因此：没有一种协议能够蓝军能 100% 获胜。

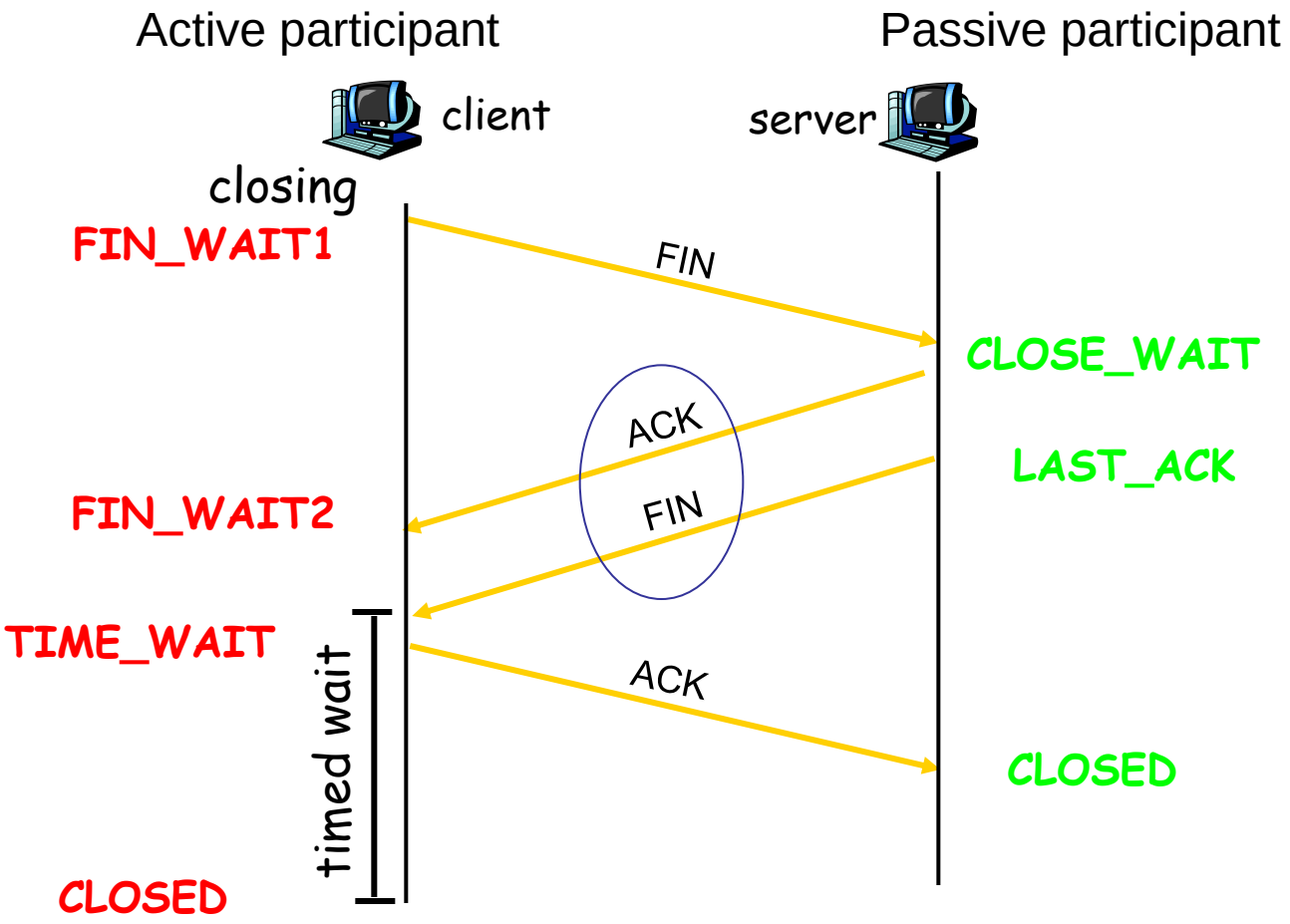
三次握手



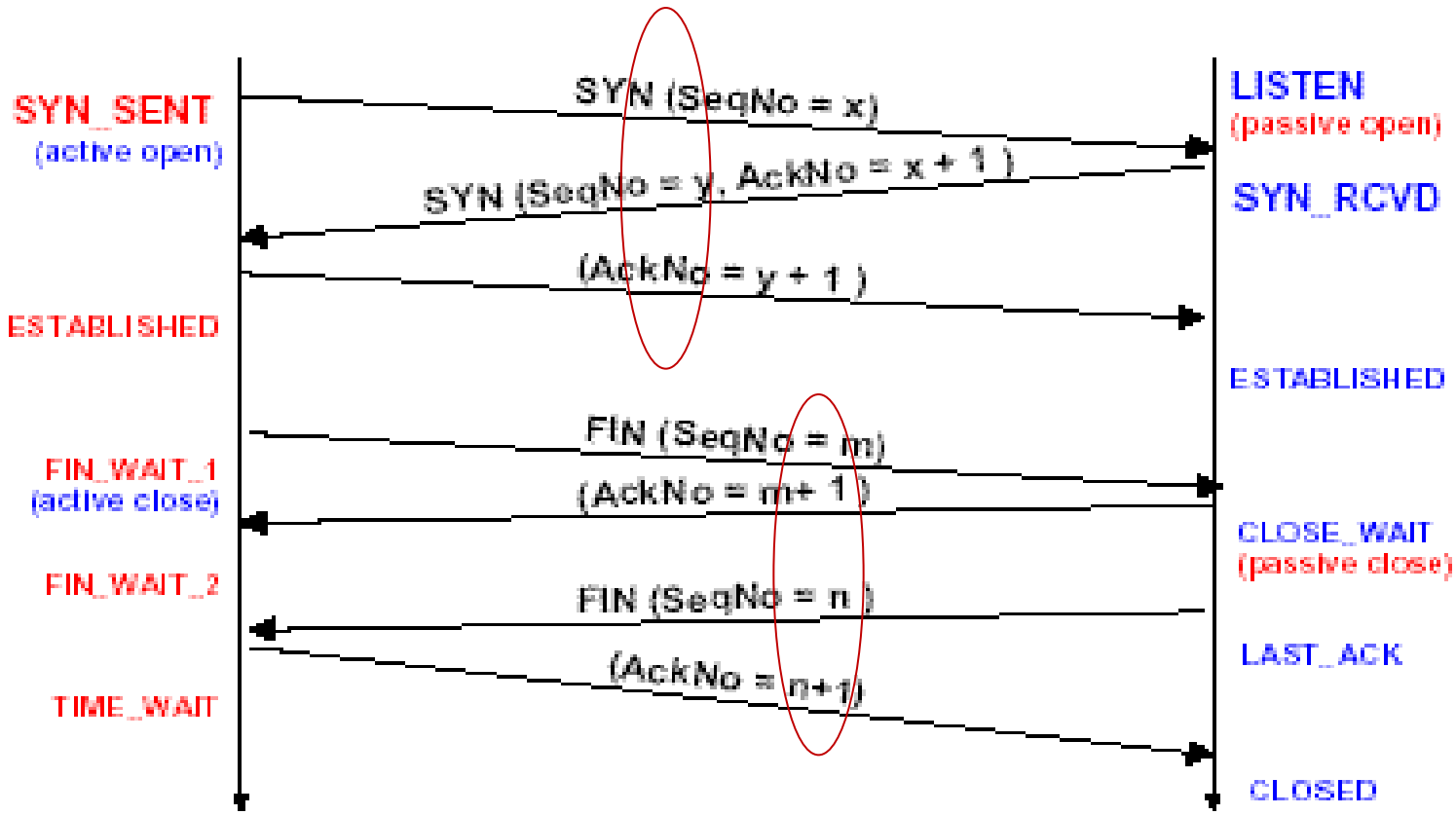
TCP 连接建立:

通过三次握手可建立一个TCP连接，避免出现半连接状态。

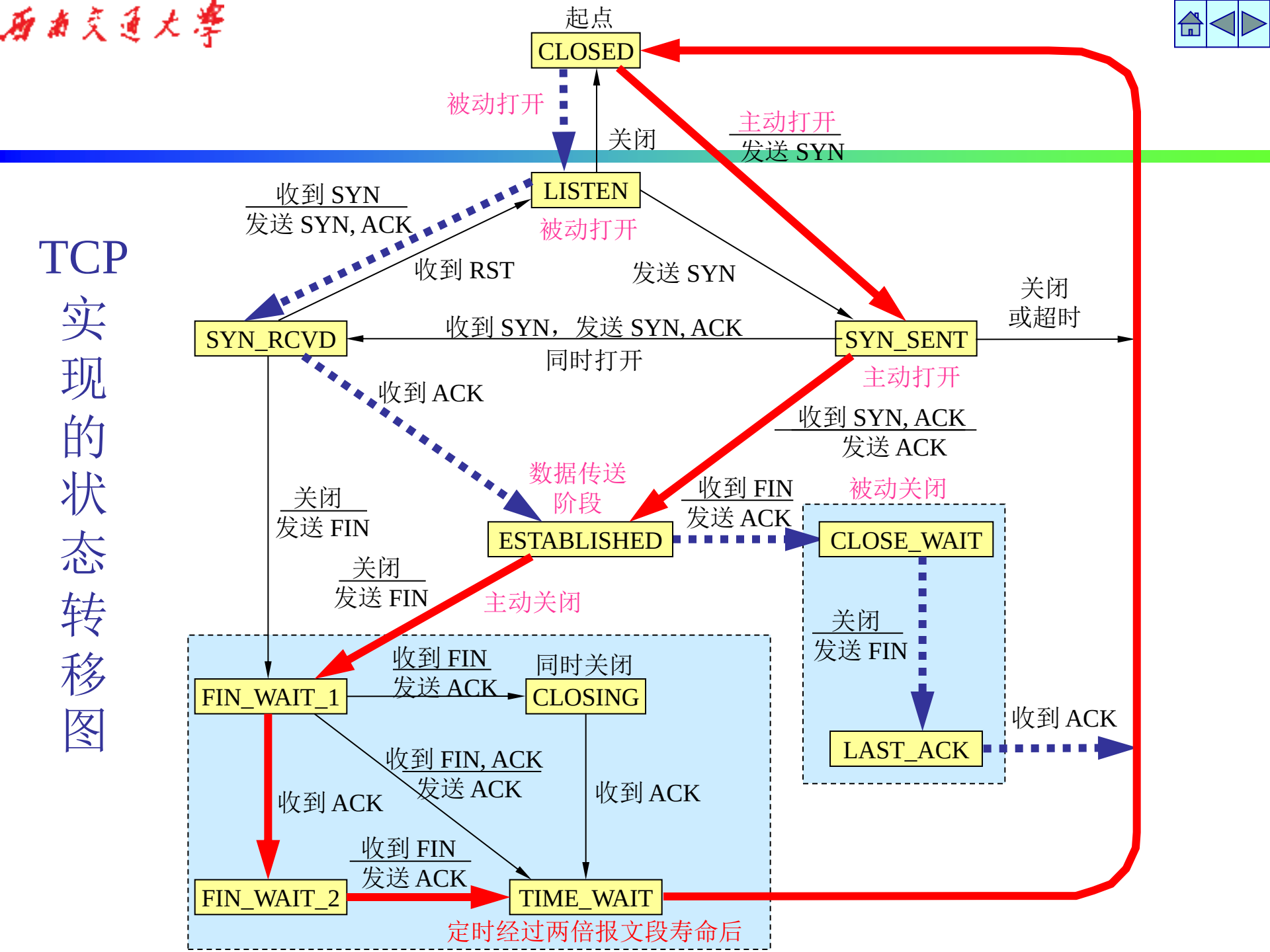
TCP Connection Termination



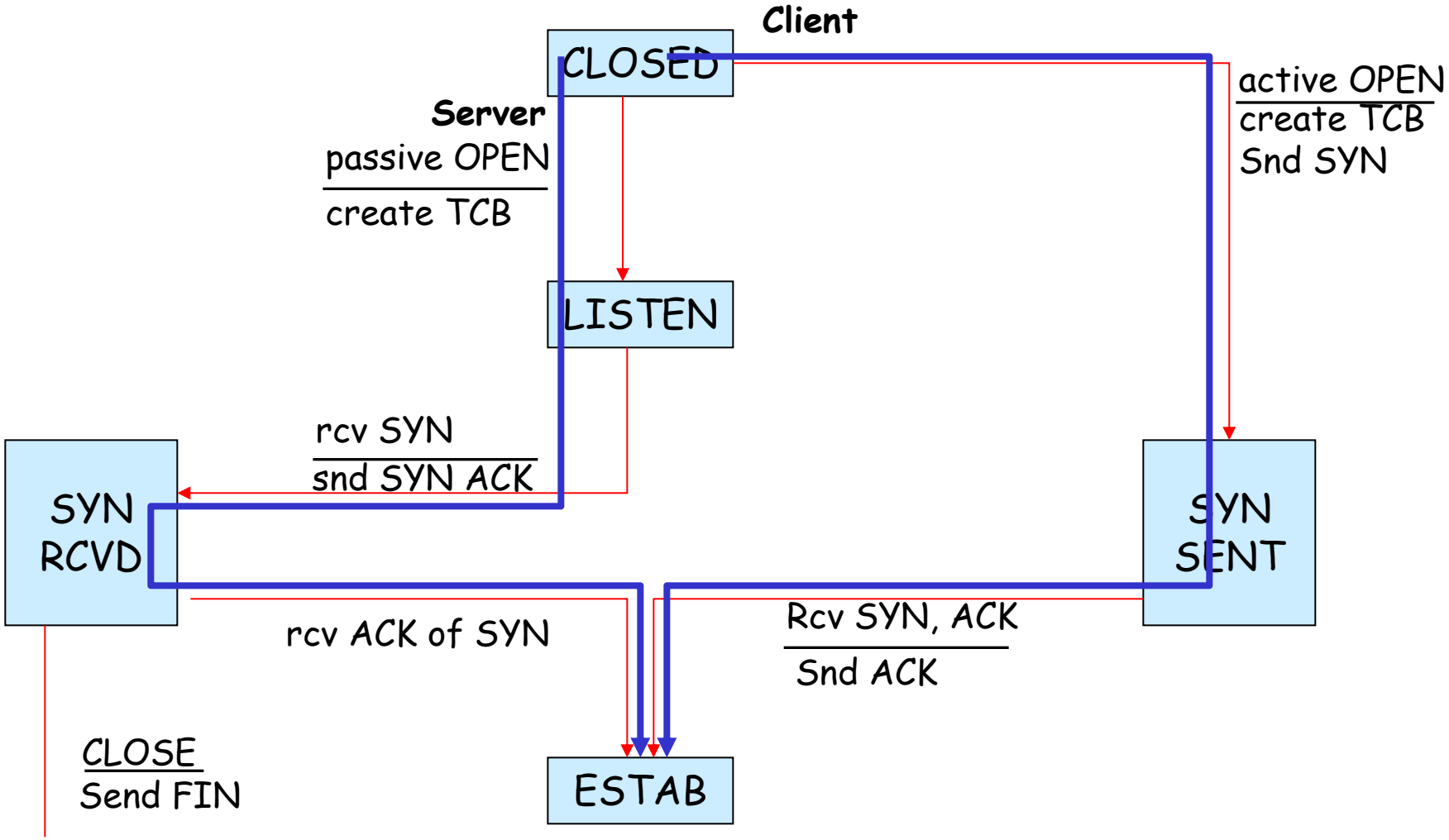
正常的TCP生命周期


SYN_SENT
(active open)
SYN (SeqNo = x)
LISTEN
(passive open)
SYN (SeqNo = y, AckNo = x + 1)
SYN_RCVD
(AckNo = y + 1)
ESTABLISHED
ESTABLISHED
FIN (SeqNo = m)
FIN_WAIT_1
(active close)
(AckNo = m+ 1)
CLOSE_WAIT
(passive close)
FIN_WAIT_2
FIN (SeqNo = n)
LAST_ACK
(AckNo = n+1)
TIME_WAIT
CLOSED

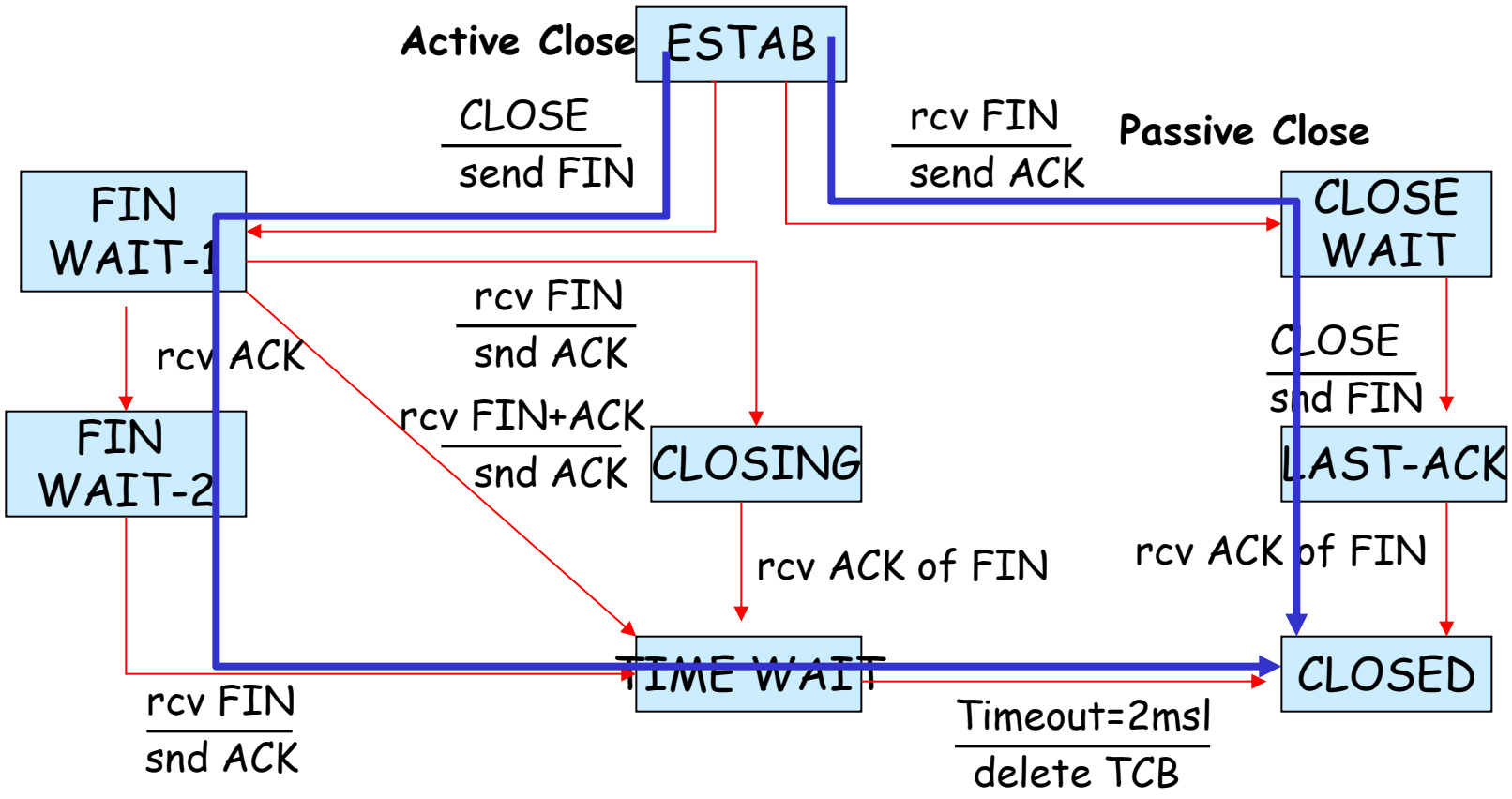
TCP 实现的状态转移图



TCP State Diagram: Connection Setup



State Diagram: Connection Tear-down



Time_Wait state is necessary in case the final ack was lost.

复位TCP连接

- 通过设置RST标志来复位TCP连接
- **设置 RST 标志的时机?**
  - 连接请求到达时，服务方端口没有相应的进程处理
  - 出现严重错误时，终止TCP连接

注：复位TCP连接是无条件的，双方必须立即执行，丢弃未完成的数据，释放TCP资源

TCP

- 概述 √
- 报文格式 √
- 连接管理 √
- TCP的数据传输
- 数据流控制和拥塞控制
- 错误控制 (Timer)

交互式与海量数据

✓TCP 可用于下列应用

- **海量数据传输** - ftp, mail, http
- **交互式数据传输** - telnet, rlogin

✓TCP试探性地处理各种应用

交互式数据传输

- 交互式数据传输 (rlogin, telnet)： 主要指不定期的、少量的数据传输
- 问题1:
  - 为了少量的数据发送很多的确认.
  - **解决方法**: **延迟确认**（尽可能采用捎带应答）
- 问题2:
  - 传输的报文段报头长（至少20字节），数据短，效率低（
  - **解决方法**: **Nagle's 算法**（发送方尽量不发送数据量小的数据段，而是缓存应用层的数据，等到形成一个比较大的段再一起发送。）

延迟确认

- 可以把确认延迟到 200ms
- 但是:
  - 连续两个分段到达时，必须发送一个确认
  - 收到副本应立即发送确认
- 延迟确认的理由:一般情况下，200ms 时延内会有用户要发送的数据准备好，这样可以通过数据分段捎带应答。

Nagle算法

- 只在下列情况下发送一个新的数据段：
  - 数据填满了最大数据段
  - 或，数据填满了接收方接收缓冲区的一半
  - 或，所有已传输的数据段都得到了确认
- 这样，Nagle'规则减少了小段的数量。
- **实现:** 发送一字节，并缓存所有后续字节直到收到确认，然后在一个报文段中发送所有缓存的字节。
- 有时候，算法可能失效。

→ **Nagle算法会造成大的延迟→解决办法：设置PUSH标志位→立即发送**

TCP 海量数据传输

- 滑动窗口协议
  - ✓ 发送方在收到确认之前连续发送多个分组
  - ✓ 接收方不必为每个分组作出确认
  - ✓ 累积确认.
  - ✓ Ack # = 收到的最大连续序列号 + 1（即Ack号之前的数据已经收妥，等待接收从Ack开始的数据）
  - ✓ 传送双方可以有不同的动态窗口
- 接收方通报的窗口域
  - ✓ 如果接收方来不及接收数据，可以减少通报窗口的大小，甚至暂停接收（通报窗口=0）

Thanks !