北京邮电大学

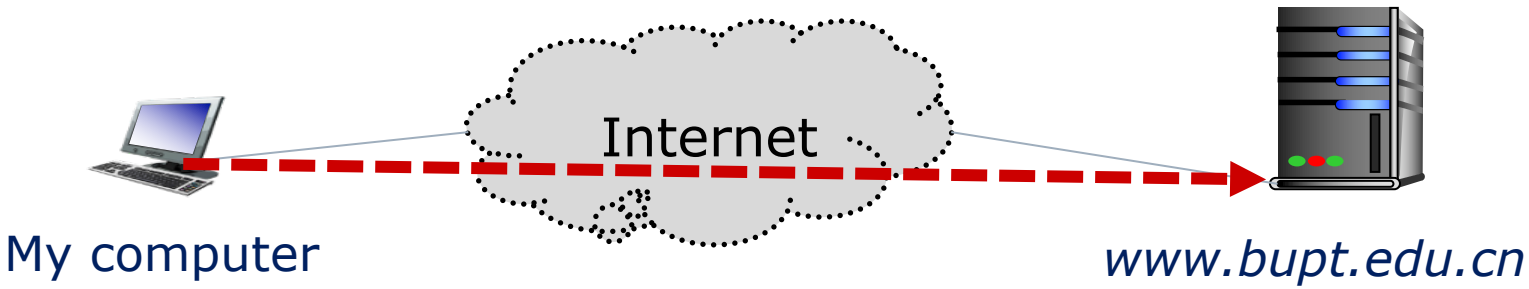
计算机网络

第三章 传输层

网络空间安全学院

2025年4月

用户的应用需求：访问北邮主页



- 这是一个WWW应用
  - 性能要求：可靠性
  - 传输层：TCP
  - 如何实现可靠传输？

本章学习重点

为什么需要传输层？

- 一些应用层协议需要消息传输的可靠性，如HTTP、SMTP…
- 网络层IP不可靠，数据可能出错、丢失或乱序

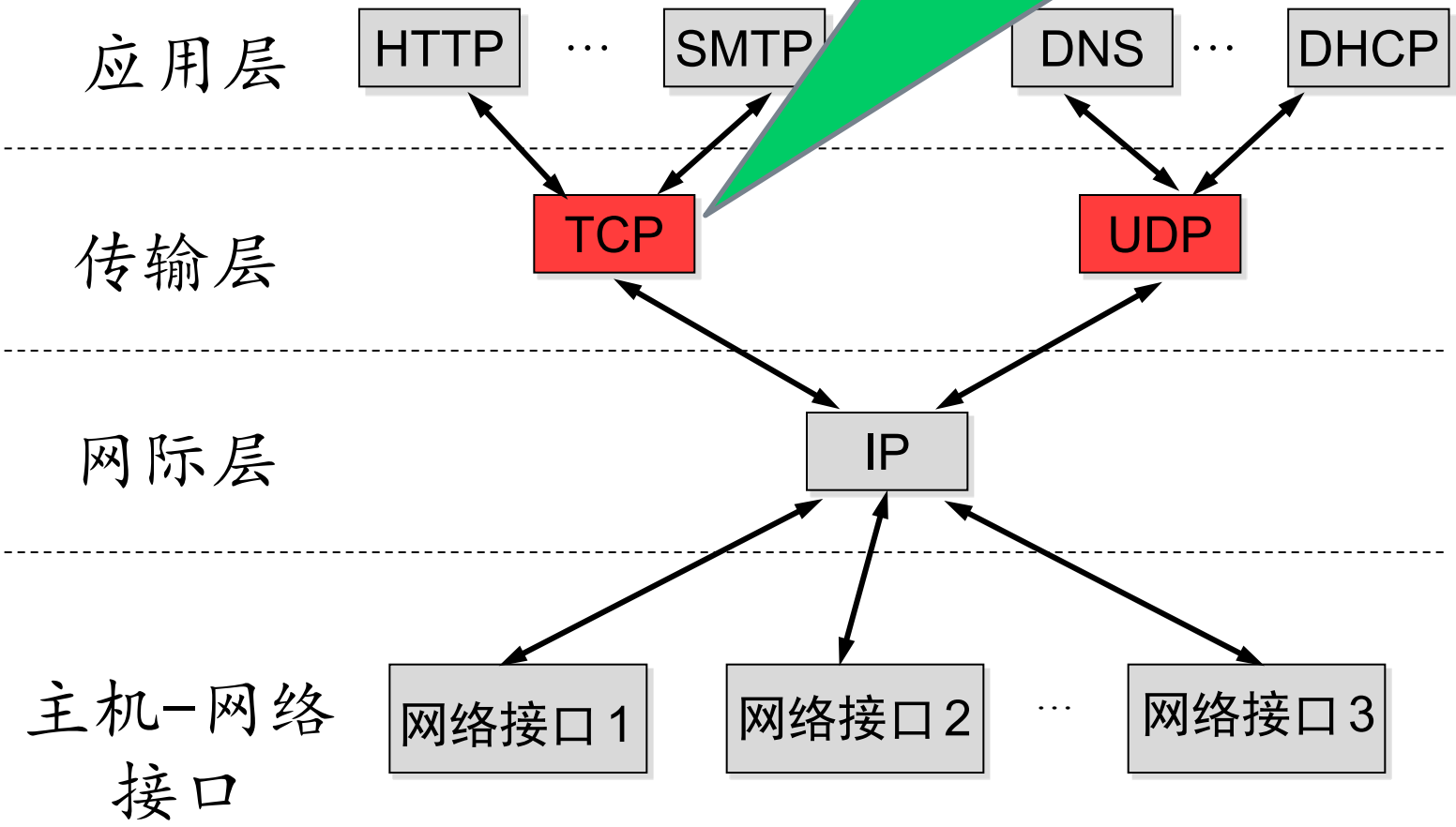
➡ *传输层实现端到端的可靠通信！*

- 根据IP地址可以将数据传送给一台主机，主机内可能同时运行多个网络应用进程，主机收到数据后，如何确定该交给哪个进程？

➡ 由传输层根据端口号确定

TCP/IP协议栈

- 传输层是整个网络体系结构中的关键层次
- 提供了面向连接的机制：可靠传输、流量控制、拥塞控制



[谢]

教学内容及要求

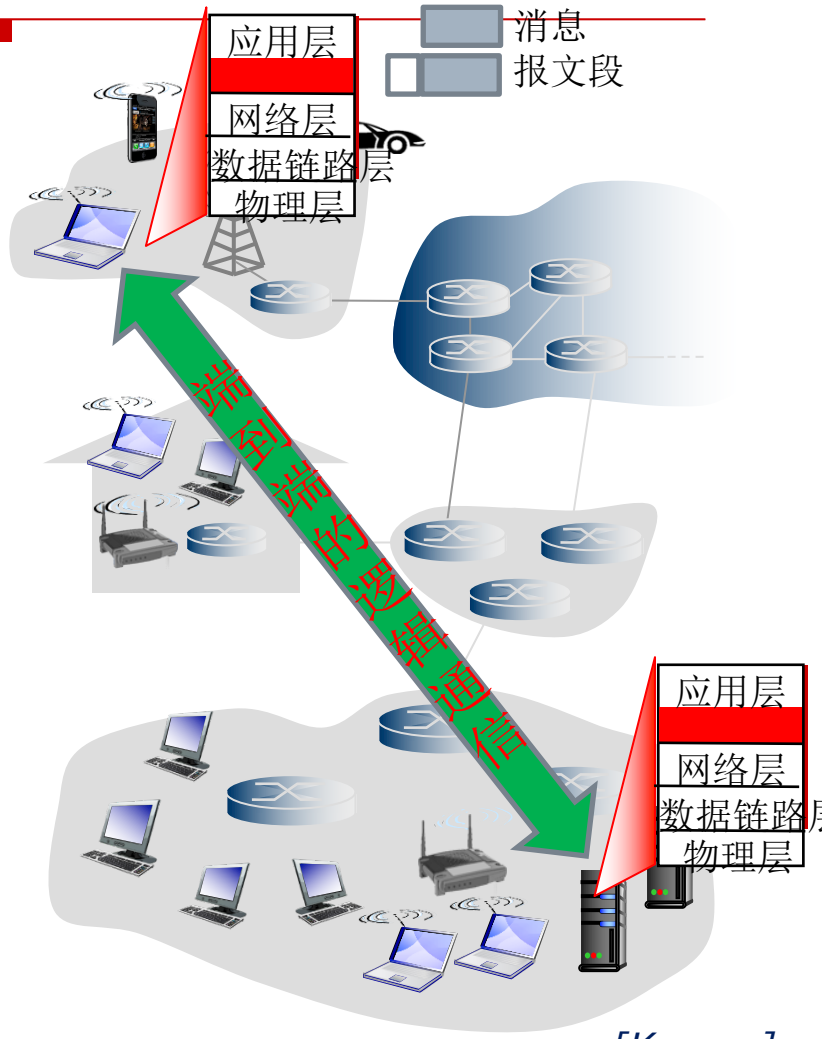
- 理解传输层服务的技术要点：
  - 多路复用
  - 可靠的数据传输
  - 流量控制
  - 拥塞控制
- 学习因特网的传输层协议：
  - UDP
  - TCP
  - TCP的拥塞控制

内容提要

- *3.1 传输层的功能及服务*
  - 进程-进程的逻辑通信
  - 复用与解复用
  - 端口号
  - 两种服务：TCP与UDP
- 3.2 可靠数据传输的原理
- 3.3 无连接传输协议：UDP
- 3.4 面向连接的传输协议：TCP
- 3.5 拥塞控制的主要原理
- 3.6 传输层的安全隐患

传输层服务和协议的要点

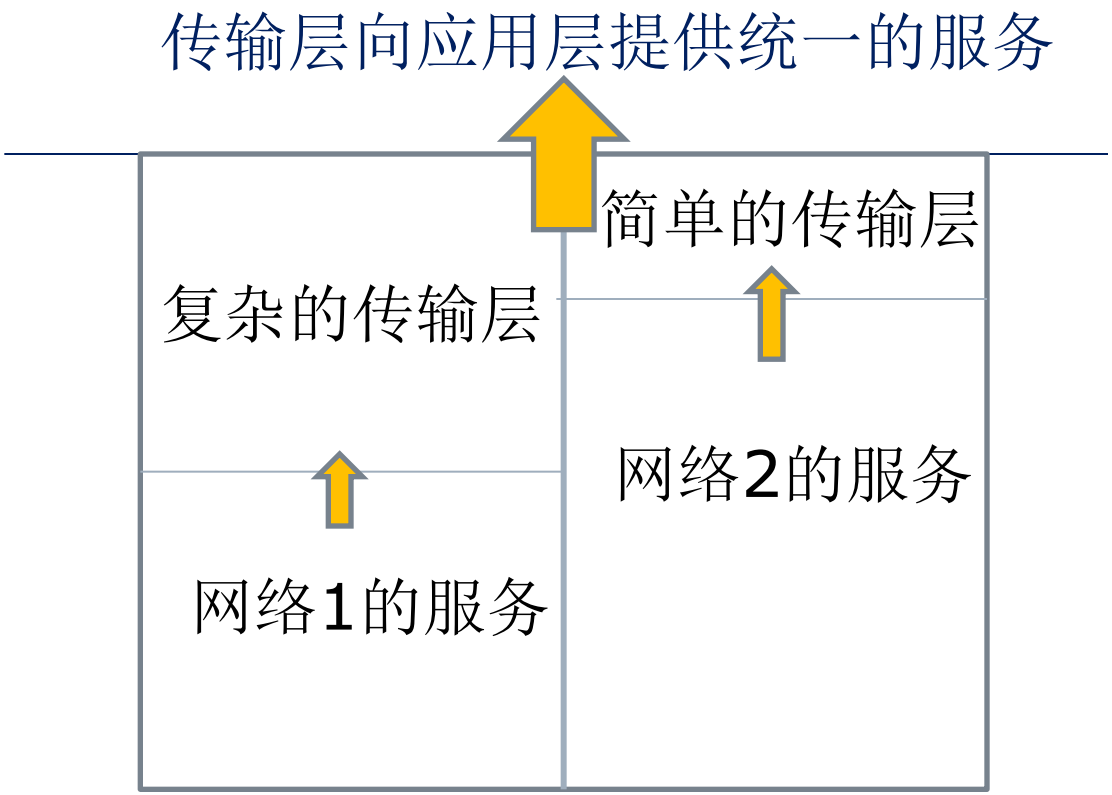
- 传输层为不同主机上的应用进程提供了**逻辑通信**功能
- 传输层协议在**端系统**中实现
  - 发送端：将应用层消息**封装**成**报文段**(segments)或数据报，然后交给网络层
  - 接收端：从报文段/数据报中取出消息，交给应用层
- 多种传输层协议
  - 因特网：TCP和UDP



[Kurose]

传输层协议的复杂度

- 取决于网络层服务的性能



传输层 vs. 网络层

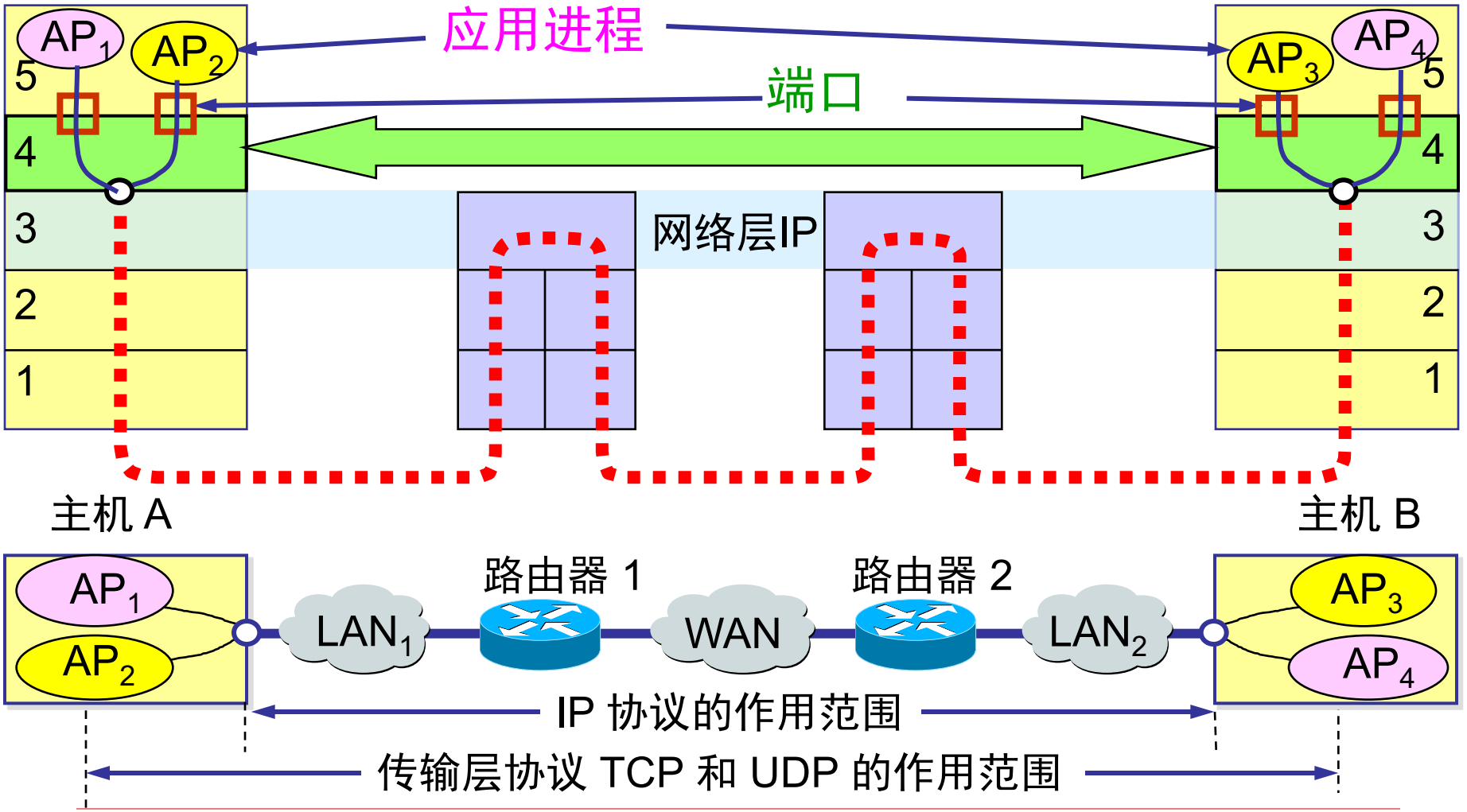
- 网络层：
  - 主机之间的逻辑通信
- 传输层：
  - 进程之间的逻辑通信
  - 使用网络层的服务，并增强后提供给应用层

生活中的实例：

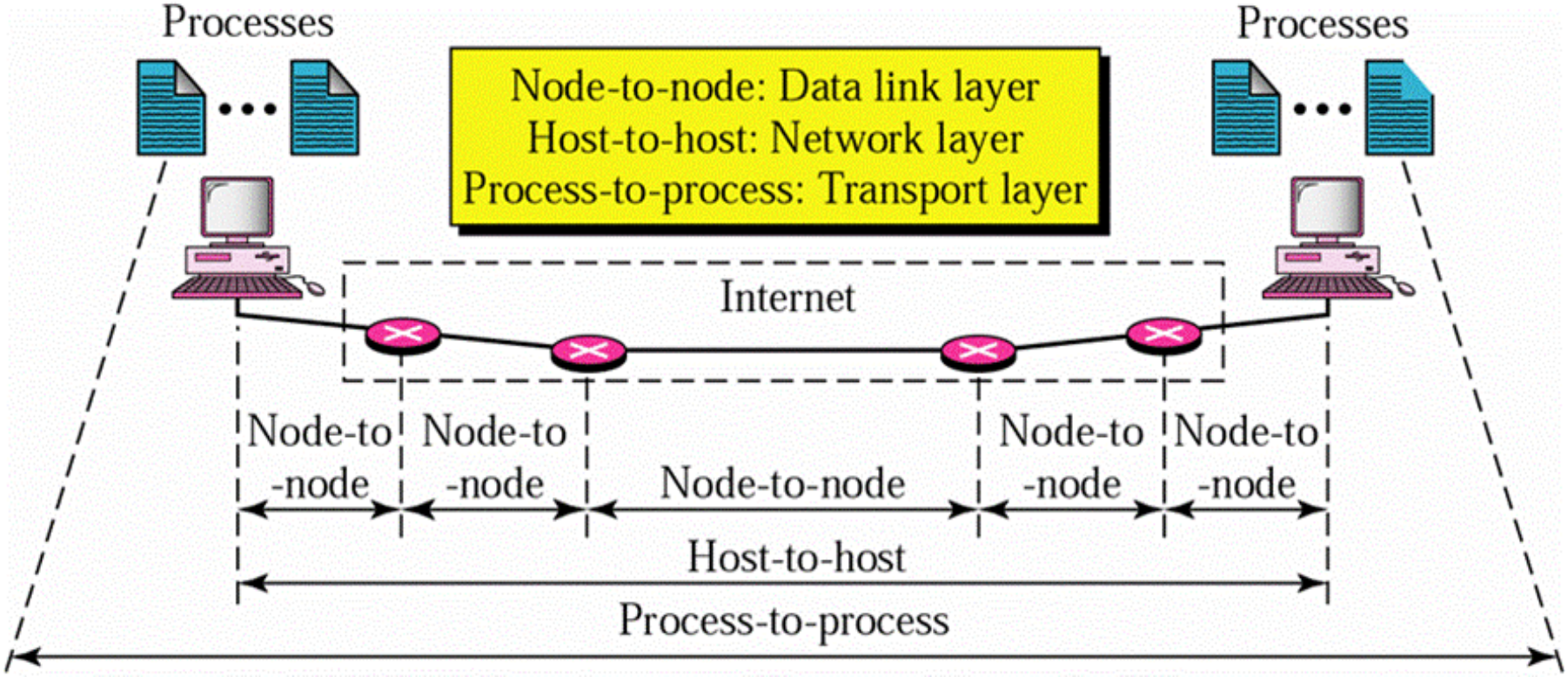
大学生和外地的同学通信

- 进程：学生
- 应用层消息：信件
- 主机：班级信箱
- 传输层协议：班级通信员
- 网络层协议：邮政系统

传输层：进程-进程通信


应用进程
端口
AP1
AP2
AP3
AP4
5
4
3
2
1
网络层IP
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

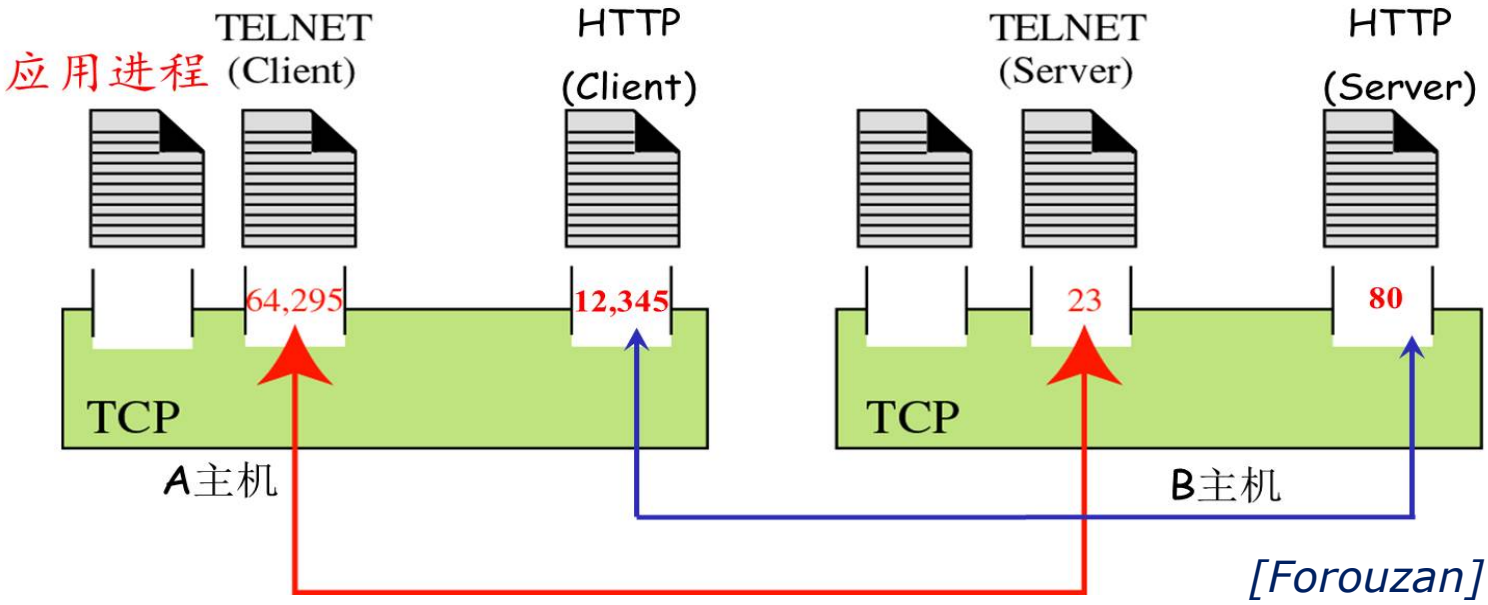
传输层：进程-进程的数据交付 (Process-to-process Delivery)



[Forouzan]

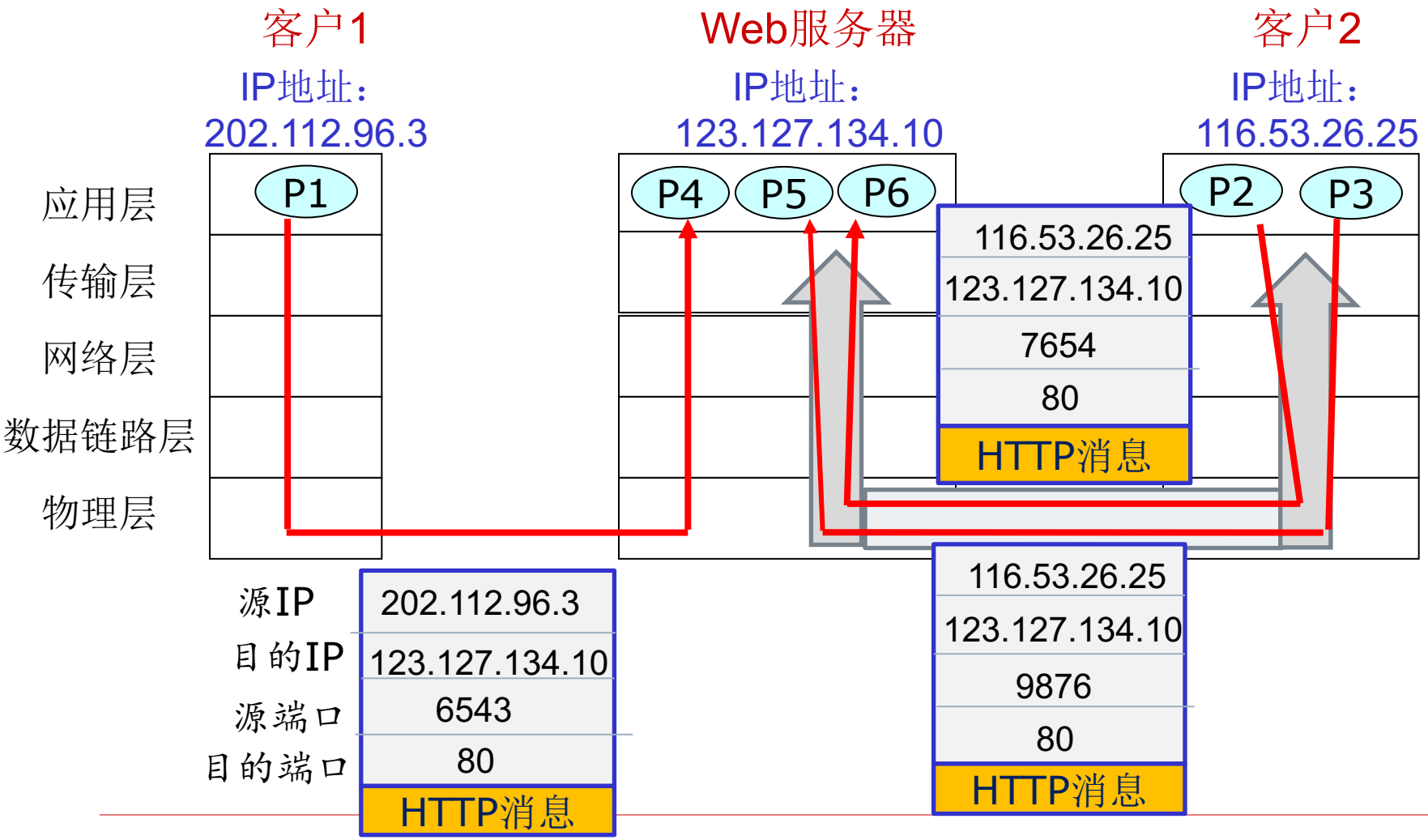
传输层的复用/解复用

- 多路复用：多个应用进程的消息在同一个网络接口上传输
- 进程标识：16位端口号
- 复用：源端传输层在应用层消息之前加上源端口号和目的端口号
- 解复用：目的端传输层根据端口号确定目的进程



[Forouzan]

TCP多路复用示例


客户1
IP地址：
202.112.96.3
Web服务器
IP地址：
123.127.134.10
客户2
IP地址：
116.53.26.25
应用层
传输层
网络层
数据链路层
物理层
P1
P4
P5
P6
P2
P3
116.53.26.25
123.127.134.10
7654
80
HTTP消息
源IP
目的IP
源端口
目的端口
202.112.96.3
123.127.134.10
6543
80
HTTP消息
116.53.26.25
123.127.134.10
9876
80
HTTP消息

端口号的规定

- 熟知端口（常用端口，Well-known port）
  - 服务器进程使用
  - 开放、固定值，例如Web服务器端口一般是80，Telnet服务器的端口号是23
  - 值一般是0～1023

- 短暂端口（Ephemeral port）
  - 客户进程使用
  - 由操作系统分配，值不固定
  - 值一般是49152～65535

因特网的传输层服务

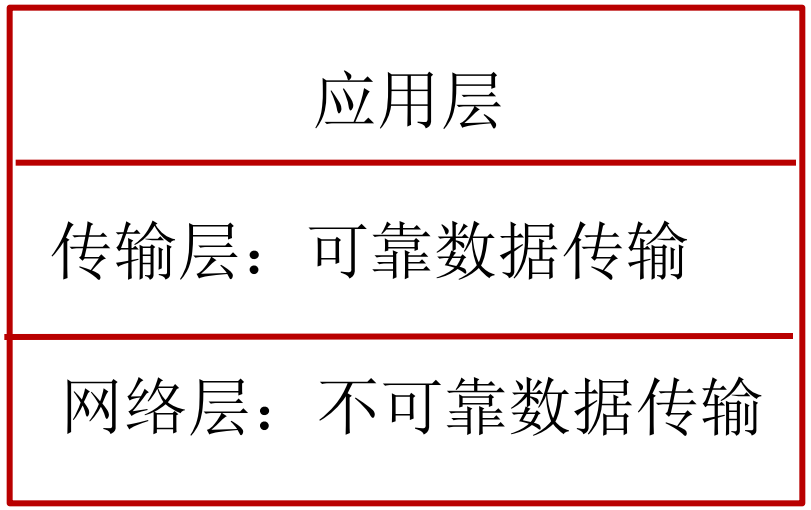
- TCP：可靠、按顺序交付
  - 建立连接
  - 流量控制
  - 拥塞控制
- UDP：不可靠、无序交付
  - 与IP相同的“尽力而为Best Effort”服务
    - 尽力传输，没有任何保障，I try my best to deliver the data, but I do not guarantee anything!
- 传输层不提供：
  - 时延保障
  - 带宽保障

内容提要

- 3.1 传输层的功能及服务
- *3.2 可靠数据传输的原理*
  - 信道可靠、无错不丢：V1.0
  - 信道有错不丢：V2.0
  - 实际信道，有错可丢：V3.0，一次只发一个包
  - 实际信道：V4.0，一次可发多个包
  - 实际信道：V5.0，减少重传
- 3.3 无连接传输协议：UDP
- 3.4 面向连接的传输协议：TCP
- 3.5 拥塞控制的主要原理
- 3.6 传输层的安全隐患

可靠数据传输

- 网络数据传输的研究要点之一
- 提供可靠传输服务的层次：传输层和数据链路层



可靠数据传输协议的复杂度取决于底层网络（信道）的可靠性!

什么是可靠的传输？

- 数据在传输过程中不会出现差错
  - 接收数据值=发送数据值
- 数据不会丢失
  - 网络不会丢失数据
  - 接收方速度够快（大于发送速度），不会因处理不及而丢失数据
- 数据不会重复
- 数据不会失序
  - 接收顺序=发送顺序

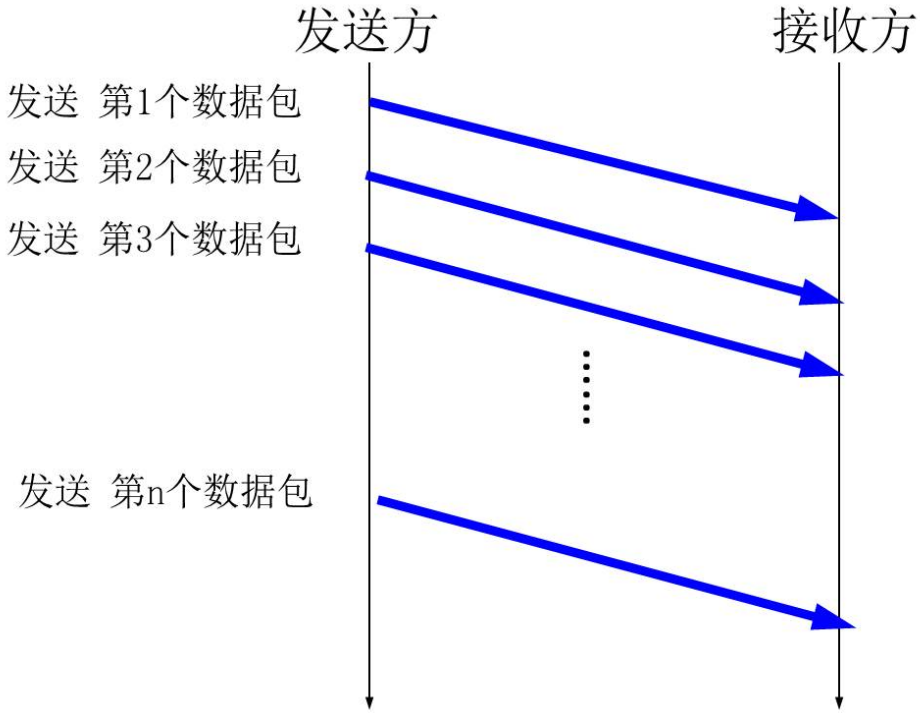
可靠数据传输（RDT）协议的演进

注：RDT并不是实际协议，只用于本课程的学习

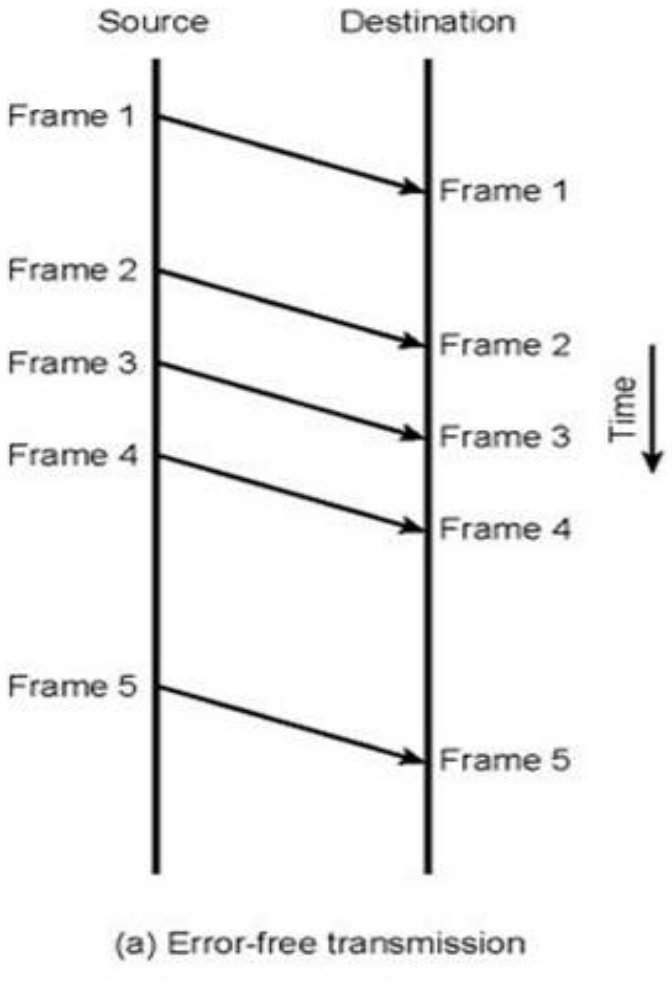
- RDT：Reliable Data Transmission
- RDT1.0：基于无差错信道（网络）
- RDT2.0：基于有传输差错的信道
  - 接收数据值≠ 发送数据值
- RDT3.0：停止等待协议，一次只发送一个数据包
- RDT4.0：Go-back-N协议，可以连续发送多个数据包，提高效率
- RDT5.0：选择重传协议，减少重传开销，进一步提高效率

RDT1.0：下层信道提供可靠传输服务

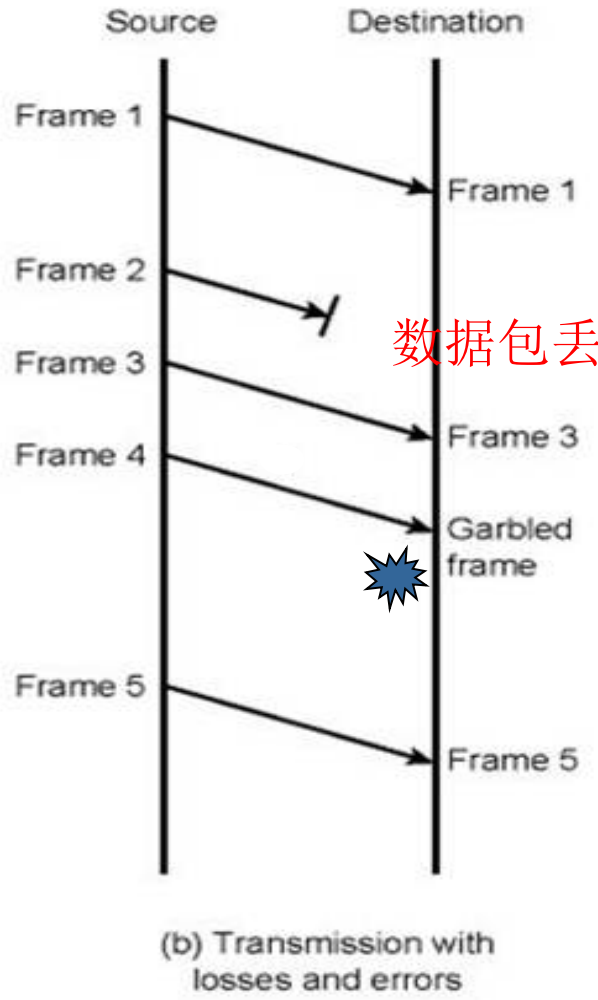
- 可靠的下层信道
  - 无传输差错
  - 无包丢失
- 协议要点：
  - 无需增加可靠性
  - 发送方将数据连续发送给下层
  - 接收方从下层连续接收数据



无差错信道 vs 有差错信道



(a) Error-free transmission



(b) Transmission with losses and errors

数据包丢失

数据包传输差错

- 信道不可能无差错
  - 噪声干扰导致数据包传输差错
  - 路由器缓存不足，丢弃数据包

[Tanenbaum]

RDT2.0：信道可能传输出错
要点（1）

- 前提：包中数据在信道上传输时某些位可能出错
- 如何发现错误？
  - 进行差错校验
  - 发送方在数据包中增加冗余的校验字段
  - 接收方进行校验检查
  - 校验字段计算方法
    - 校验和（Checksum），在传输层采用
    - 循环冗余校验（CRC），在数据链路层采用

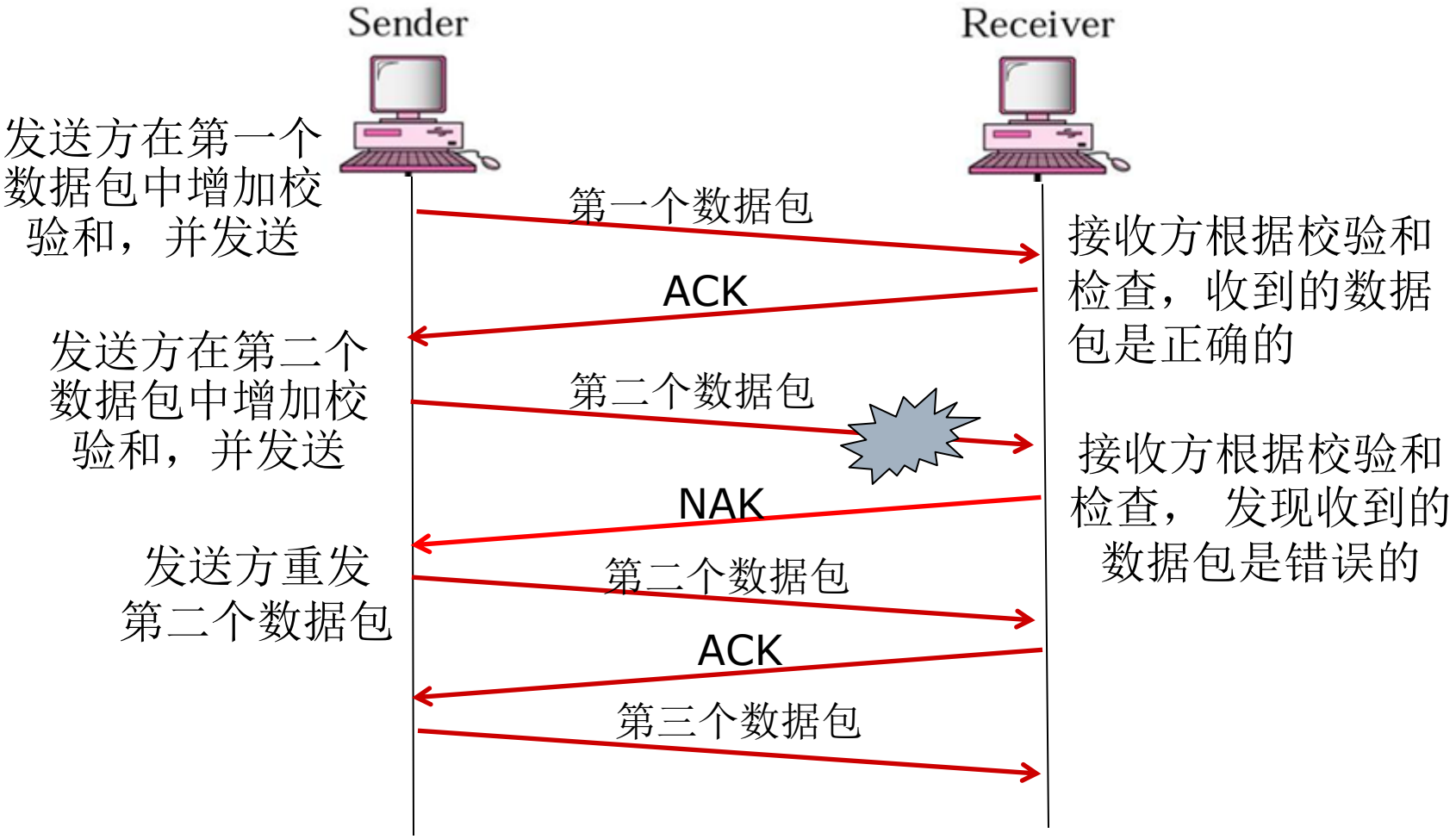
RDT2.0：要点(2)

- 发现传输错误后如何解决？
  - 接收方直接纠正错误
    - 数据链路层采用，如汉明码(Hamming Code)
  - 更通用的方法：接收方向发送方回送通知
    - 确认 (ACK)：所收到的数据是正确的
    - 否认(NAK)：所收到的数据有错
    - 收到ACK，发送方继续发送下一个数据包；
    - 收到NAK，发送方重发原数据包

→ ***ARQ：Automatic Repeat-reQuest***
自动重传请求

RDT2.0: 操作示例

- RDT2.0增加了
  - 差错检测
  - 接收方的反馈：ACK 或 NAK

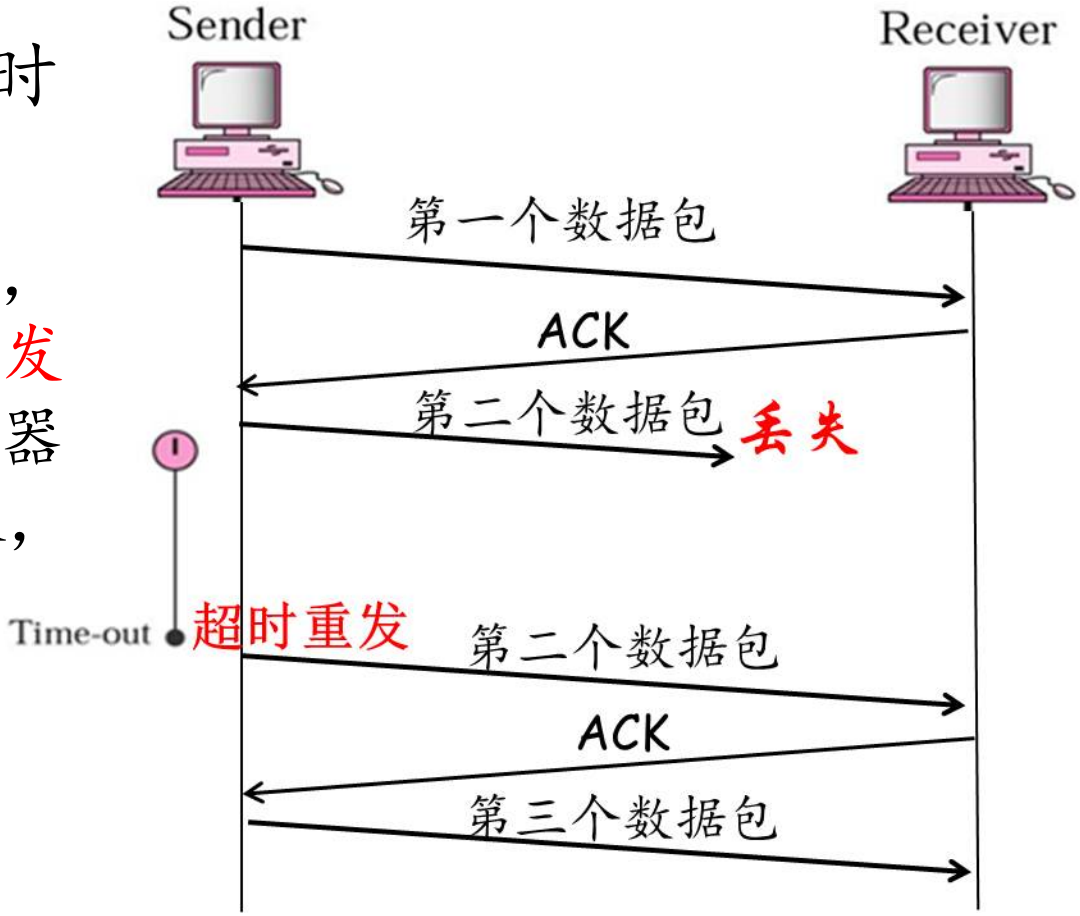


RDT2.0：要点（3）

- 停止等待协议（简称停等协议）
  - Stop-and-wait
  - 发送方发送一个数据包后停下来等待，
  - 收到接收方的ACK后再发送下一个数据包
  - 一次（在收到ACK之前）只能发送一个数据包

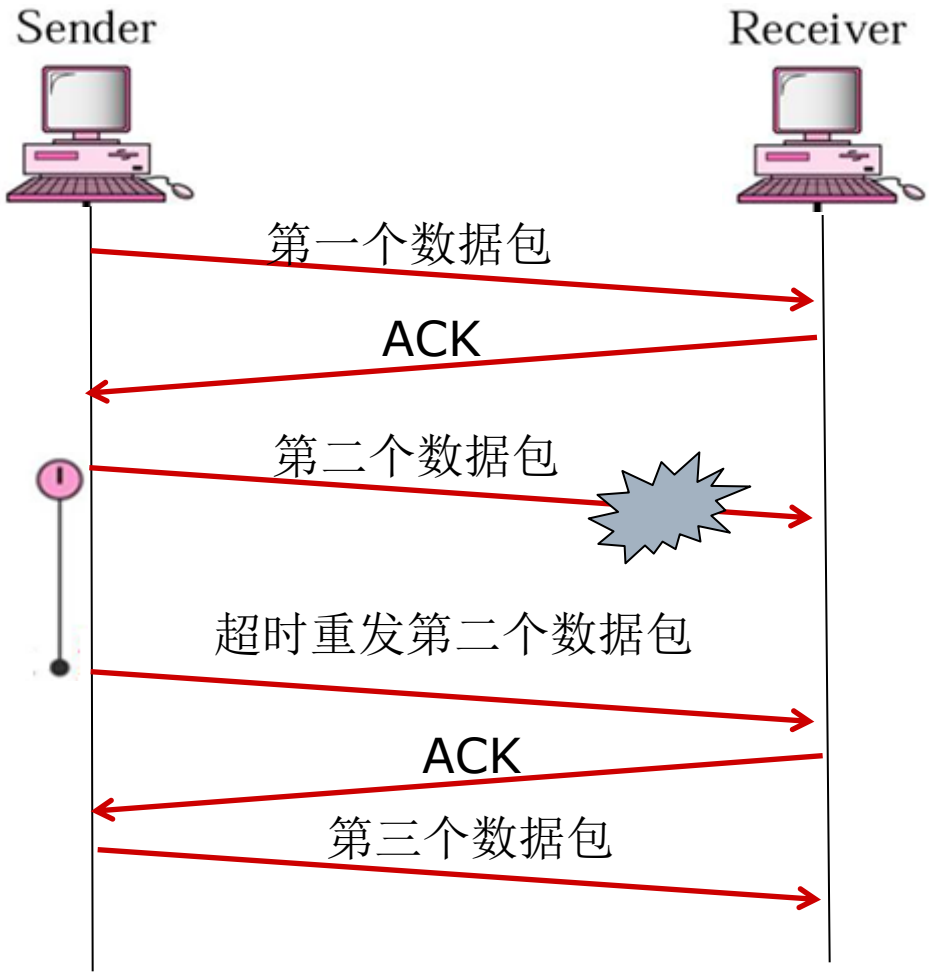
RDT2.0的问题

- 数据包丢失？
- RDT 2.1：发送方超时重发
  - 在发送完数据包后，发送方启动一个重发定时器，如果定时器超时仍未收到ACK，则重发数据包



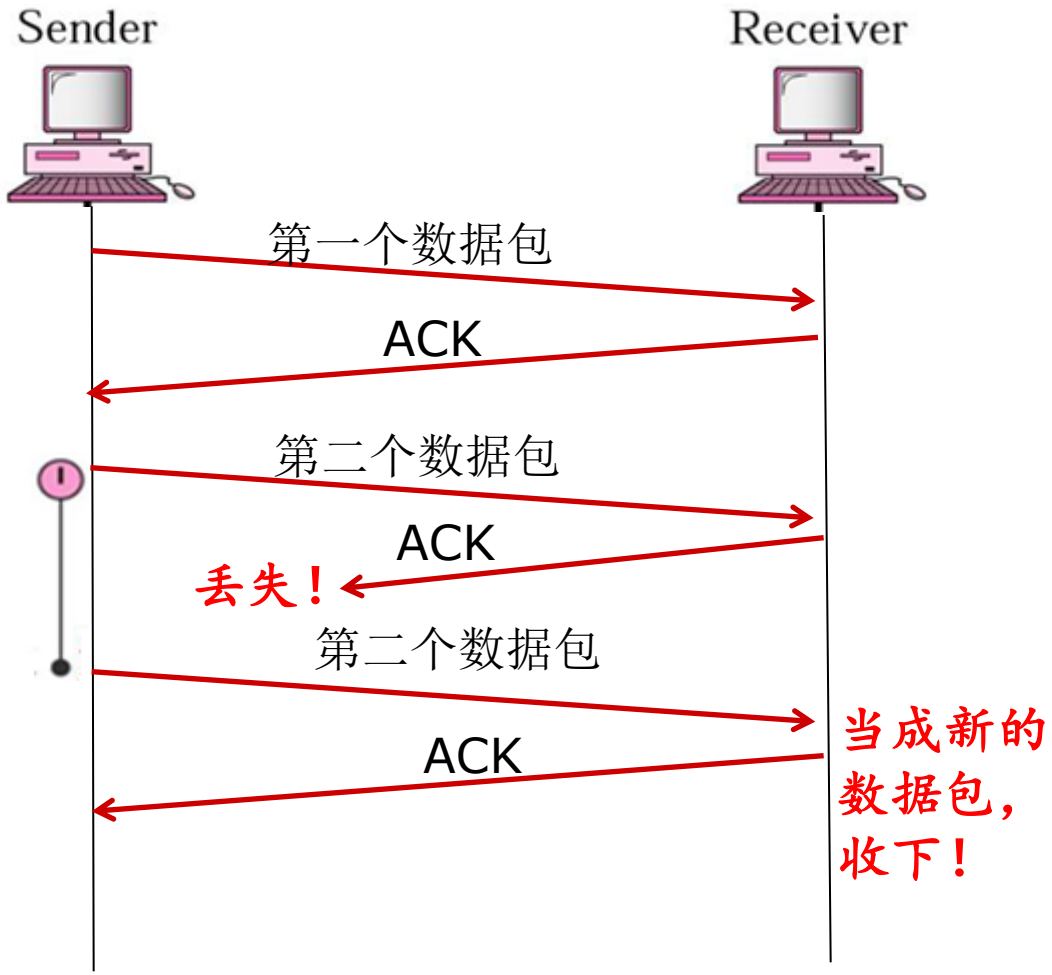
NAK是否必要？

- 超时重发同样可以解决传输差错问题
- 如果接收到有差错的数据包，接收方不回送NAK
- 发送方超时重发数据包
- NAK通常作为可选项



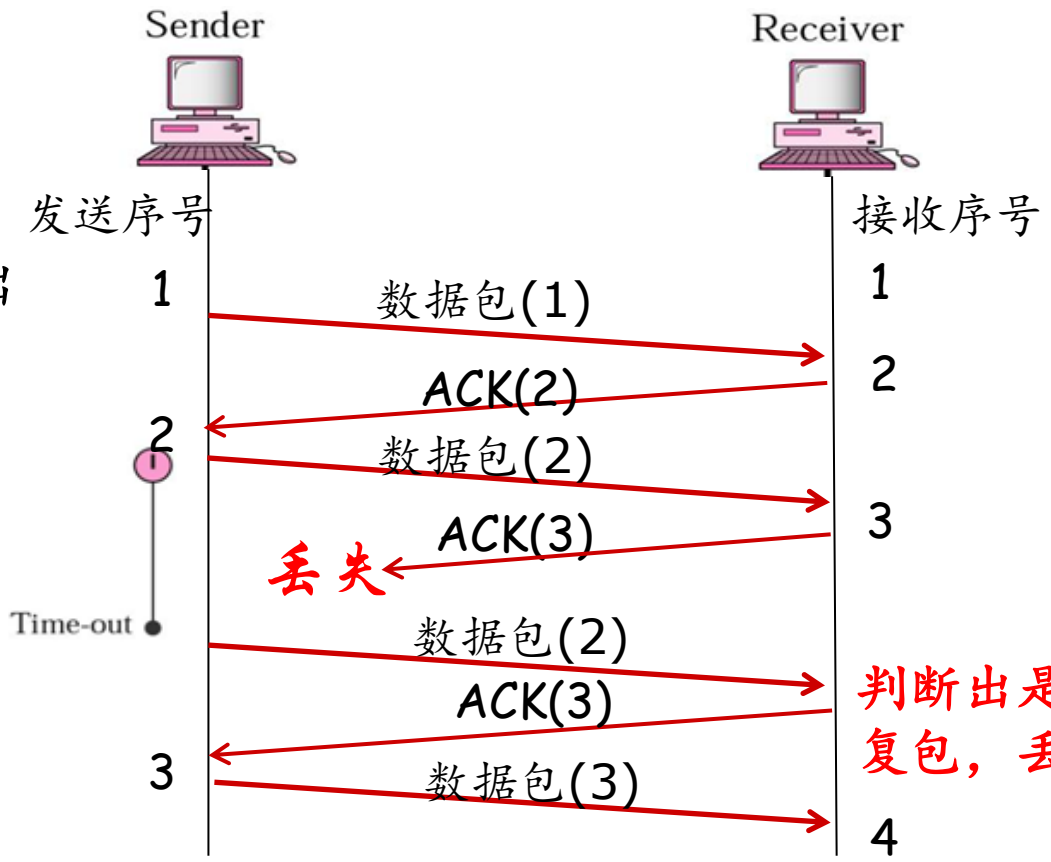
RDT2.1是完美的吗？

- 如果ACK丢失？
  - 发送方超时重传
  - 导致重复数据
  - RDT2.1出错！
- 出错原因：
  - 接收方无法判断重复包
  - 将重发数据当成新的数据



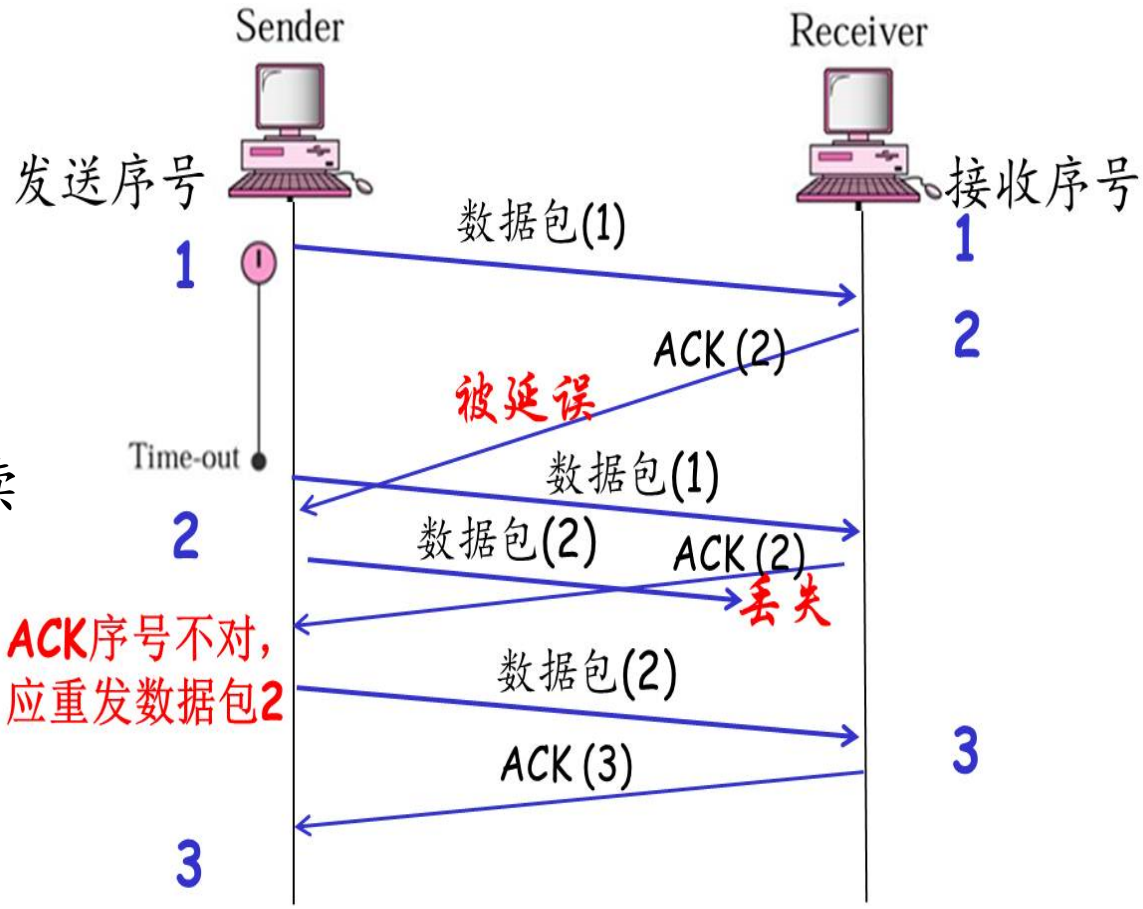
RDT2.2：增加序号

- 发送方在数据包中增加一个序号 *(sequence number)*
- 接收方根据序号判断出重复的数据包并丢弃，并重发ACK
- 数据包的序号=当前正发送包的序号
- ACK的序号=要接收的下一个数据包的序号，即已经收到的包序号+1



ACK的序号是必要的吗？

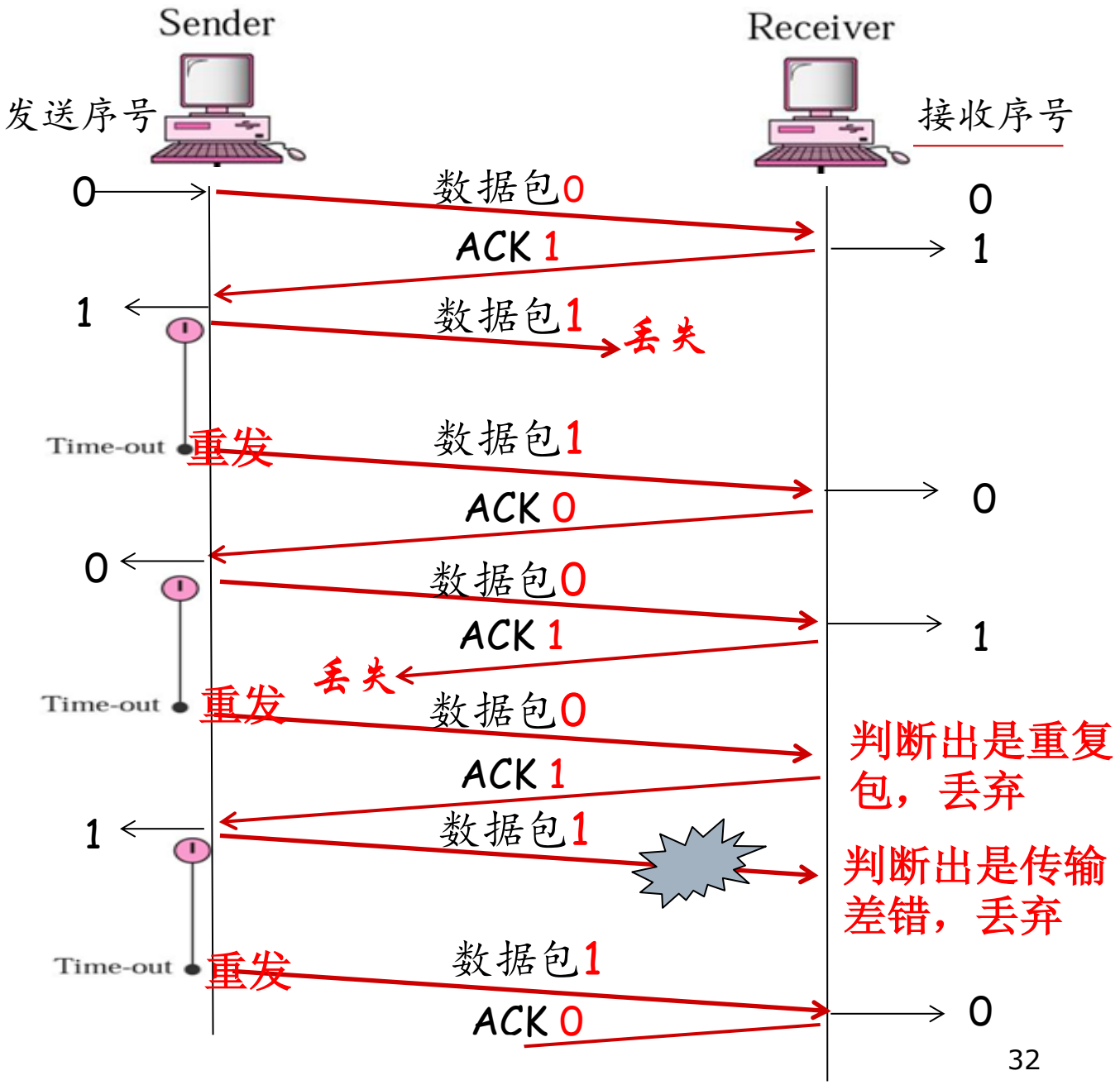
- ACK被延误的情形
  - 如果ACK没有序号，重发的ACK可能被误判为对丢失的后续数据的确认
  - 无法发现数据丢失！



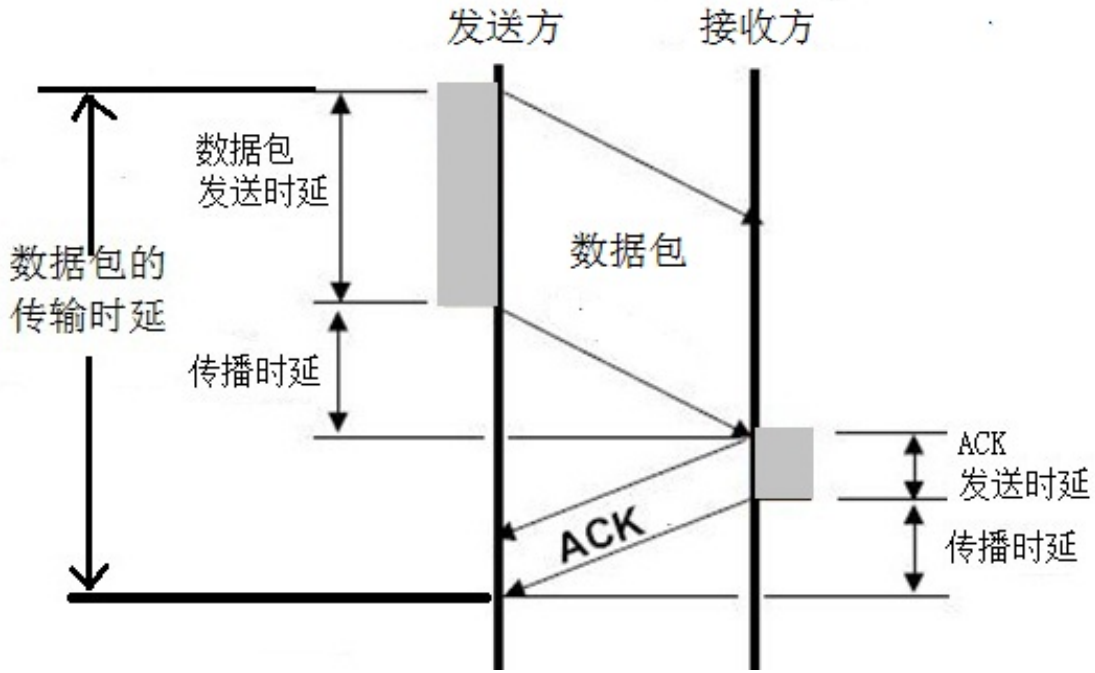
RDT3.0: 完整的停等ARQ协议

- 发送方发出一个数据包后，停下来等待，收到来自接收方的ACK后才能继续发送
- 数据包中有校验字段，发送方/接收方使用该字段来检查传输错误
- 如果接收方发现传输差错，则丢弃收到的数据包
- 如果接收方收到重复包，则重发上一个数据包的ACK
- 数据包和ACK中均包含1位序号，用于检查重复包
- 发送方维护一个定时器，如果定时器超时而仍未收到ACK，则重发数据包

完整的停等协议


Sender
Receiver
发送序号
接收序号
0
数据包0
0
ACK 1
1
1
数据包1
丢失
Time-out
重发
数据包1
ACK 0
0
0
数据包0
ACK 1
1
丢失
Time-out
重发
数据包0
判断出是重复包，丢弃
ACK 1
1
数据包1
判断出是传输差错，丢弃
Time-out
重发
数据包1
ACK 0
0

RDT3.0: 停等协议的性能（1）



传输时延 = 数据包发送时延 + ACK发送时延 + 2 × 传播时延 = $T_t + 2T_p$

- 前提：传输无差错，忽略节点的处理时延和排队时延，忽略ACK的发送时延

RDT3.0: 停等协议的性能（2）

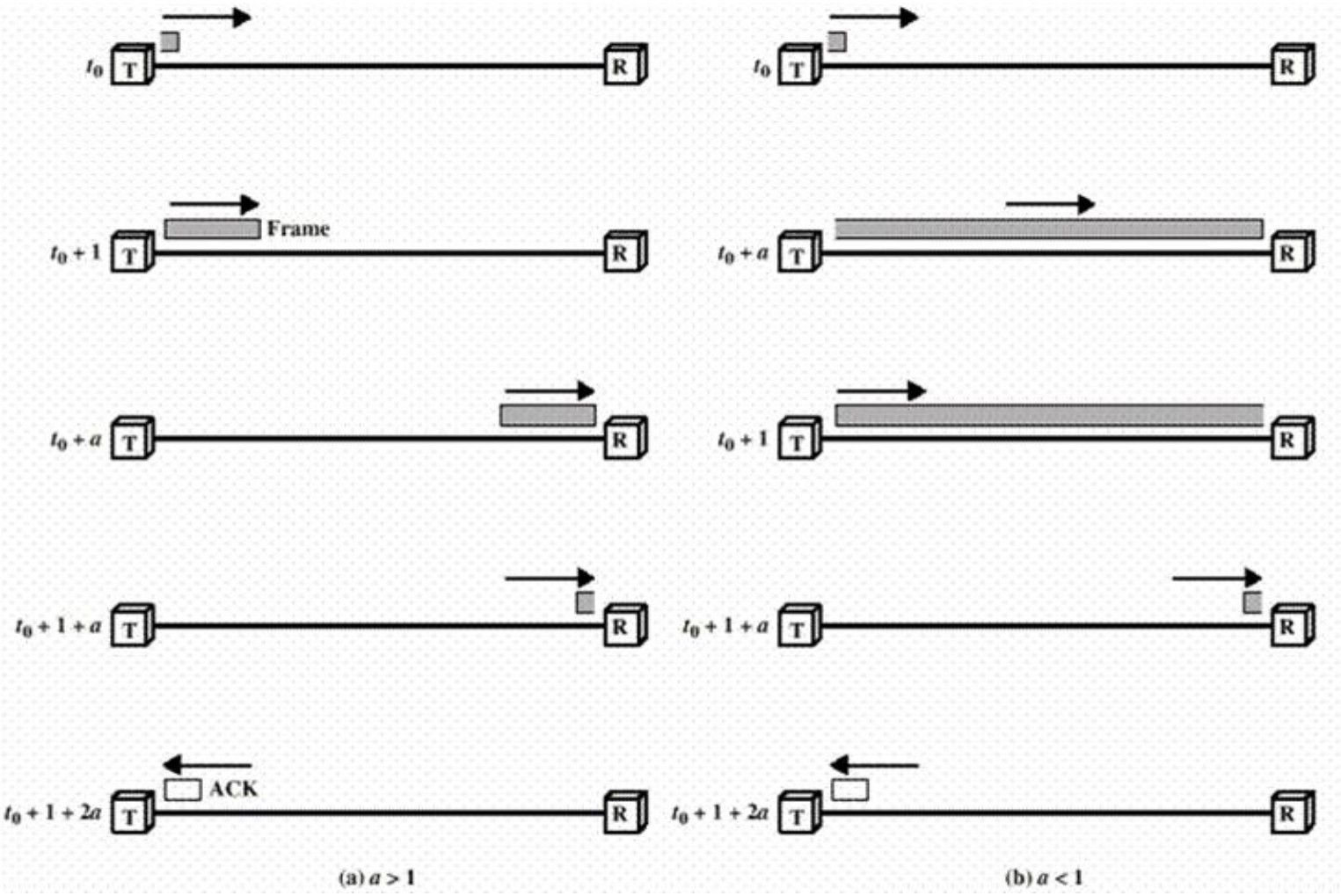
- 信道利用率：发送时间/成功传输需要的总时间

$$U = \frac{T_t}{T} = \frac{T_t}{T_t + 2T_p} = \frac{1}{1 + 2\alpha}$$

$\alpha = \frac{T_p}{T_t}$，表征传播时延对数据传输的影响

$$T_t = \frac{\text{数据包长（位）}}{\text{信道带宽（}bps\text{）}}, \quad T_P = \frac{\text{信道长度 / 距离（米）}}{\text{信号传播速度（米 / 秒）}}$$

停等协议中不同α的传输情形



- 归一化：$T_t=1$, $T_p=\alpha$

[Stallings]

停等协议：性能

- 示例: 某卫星信道数据率为1 Mbps， 传播时延为270 毫秒，数据包长为1000字节

$$\alpha = \frac{T_p}{T_t} = \frac{270 \times 10^{-3}}{1000 \times 8 \div 10^6} = 33.75$$

  - 信道利用率:

$$U = \frac{1}{1+2\alpha} = 0.015 = 1.5\%$$

  - 相当于实际的传输速率只有15Kbps，即约1.8KB/秒
  - 停等协议限制了对于物理资源的使用!

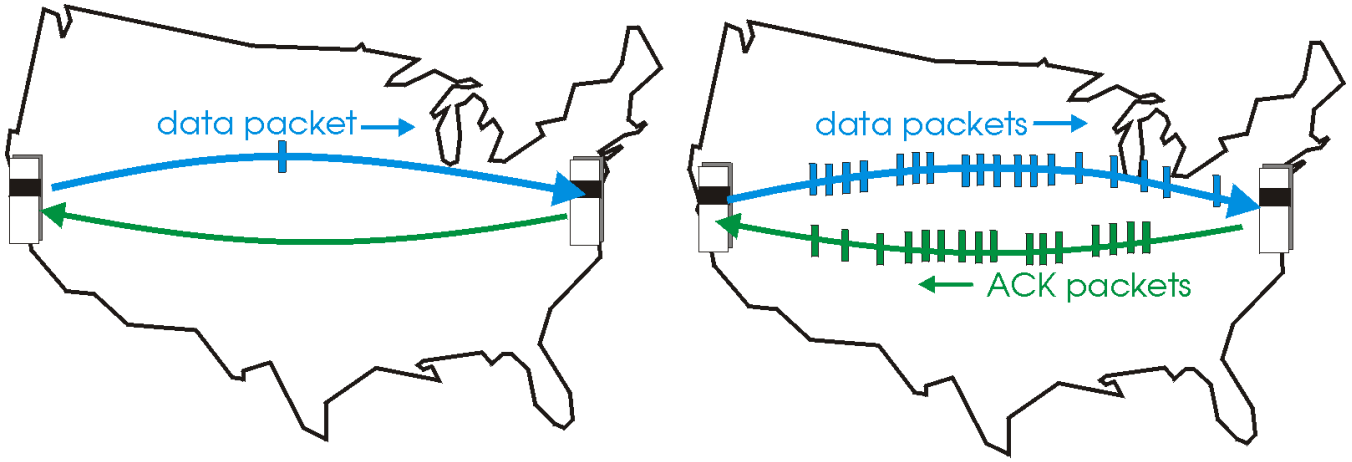
信道利用率

- 信道有效性的比率
- $= \dfrac{\text{发送时间}}{\text{成功传输需要的总时间}}$
- 或者$= \dfrac{\text{实际数据率}}{\text{带宽（最大数据率）}}$

改进: 流水线协议

流水线(Pipelining): 在收到ACK之前，发送方可以连续发送多个数据包

- 需要增加序号范围
- 发送方（和接收方）需要缓存数据包

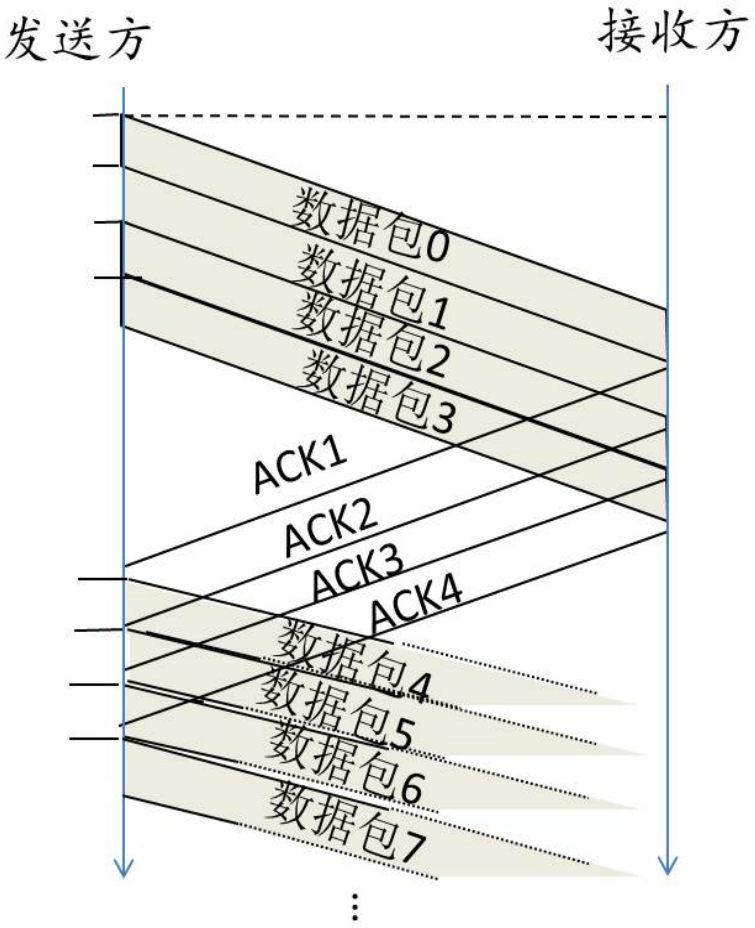


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

[Kurose]

流水线协议的性能：信道利用率提高



1 Mbps 的卫星信道， 传播时延为270 毫秒，数据包长为1000字节

- 一次可以发送4个数据包

$$U = \frac{4T_t}{T} = \frac{4}{1+2\alpha} = 6\%$$

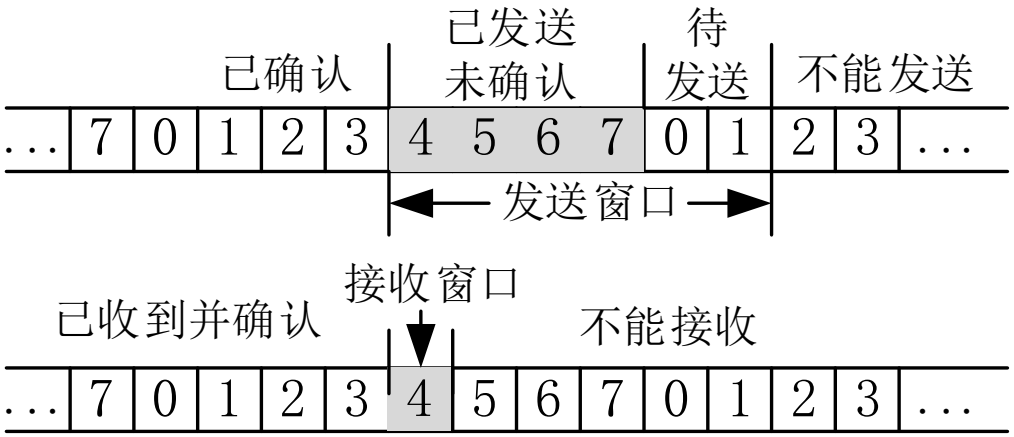
- 代表协议
  - Go-back-N ARQ
  - 选择重传ARQ

RDT4.0：连续ARQ协议

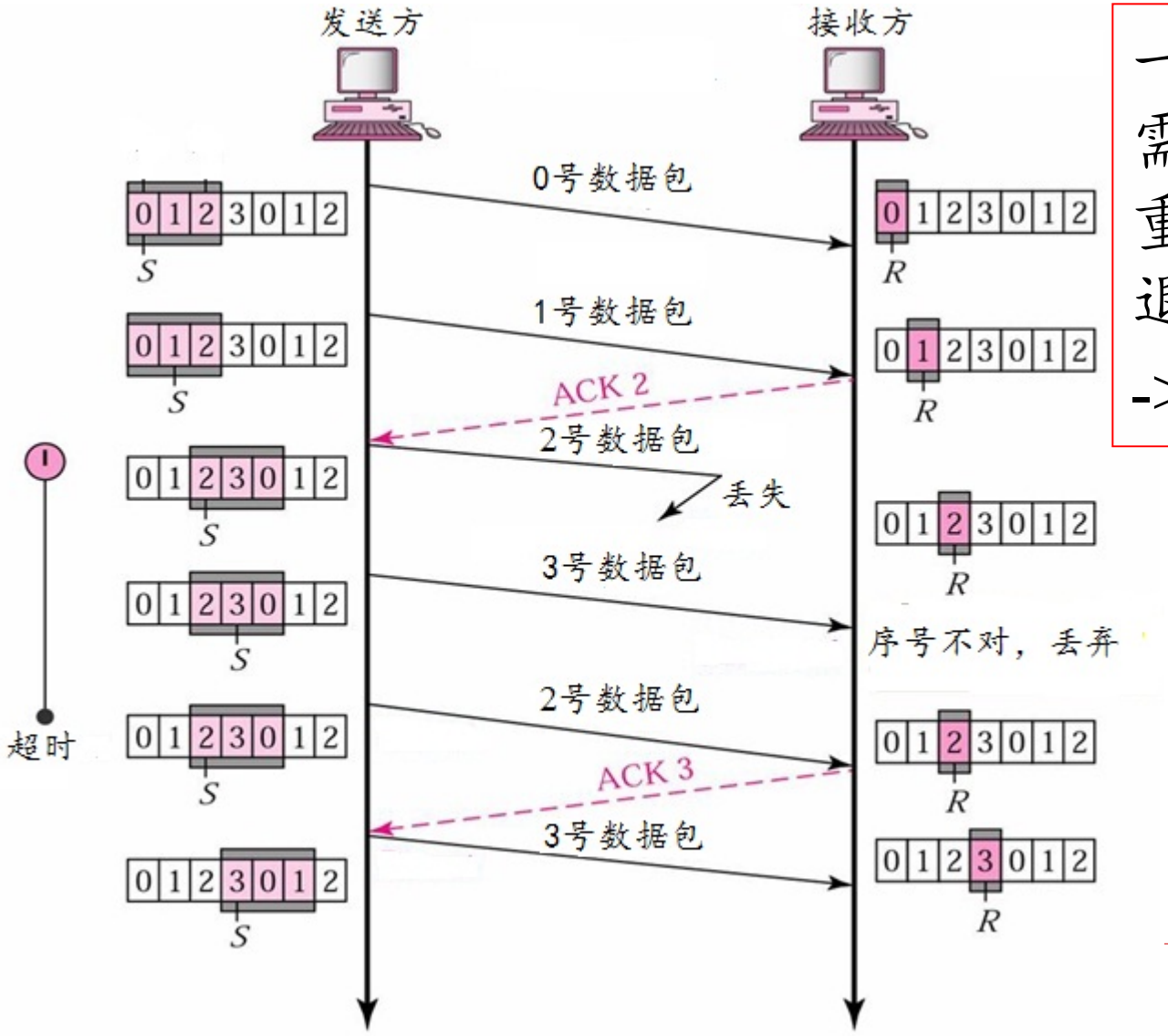
- 发送方在收到ACK之前可以连续发出 N个数据包
  - 缓存已发送但未经确认的数据包
  - 对发送的每个数据报进行计时，重发超时而未收到ACK的数据包
- 接收方一次只能接收一个数据包
  - 只能缓存一个数据包
  - 收到序号不符的数据包，将丢弃

连续ARQ：滑动窗口

- 序号：循环使用，n位序号，序号空间[0, 2^n-1]
- 发送窗口大小：N, 即收到ACK之前可连续发出的数据包个数
- 接收窗口大小：1
- 示例：n=3, N=6



连续ARQ: 数据包丢失示例



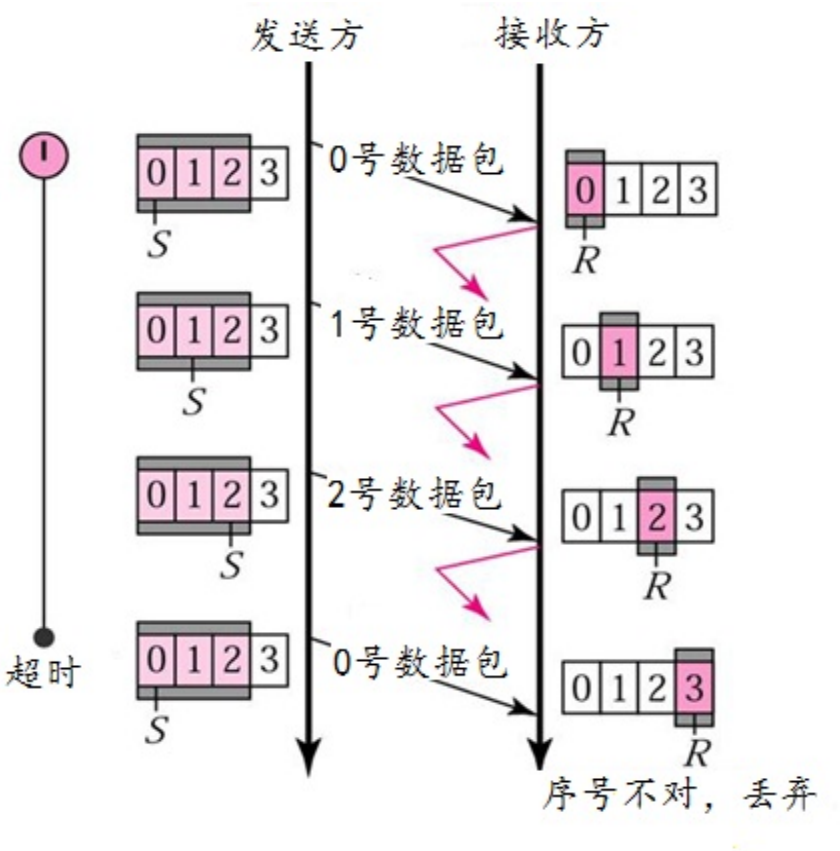
一旦出错，发送方需要从出错包开始，重传后续所有包，退回N步

-> Go-back-N

[Forouzan]

连续ARQ: 发送窗口的最大值

- n位序号，最大发送窗口：$N_{max} = 2^n - 1$

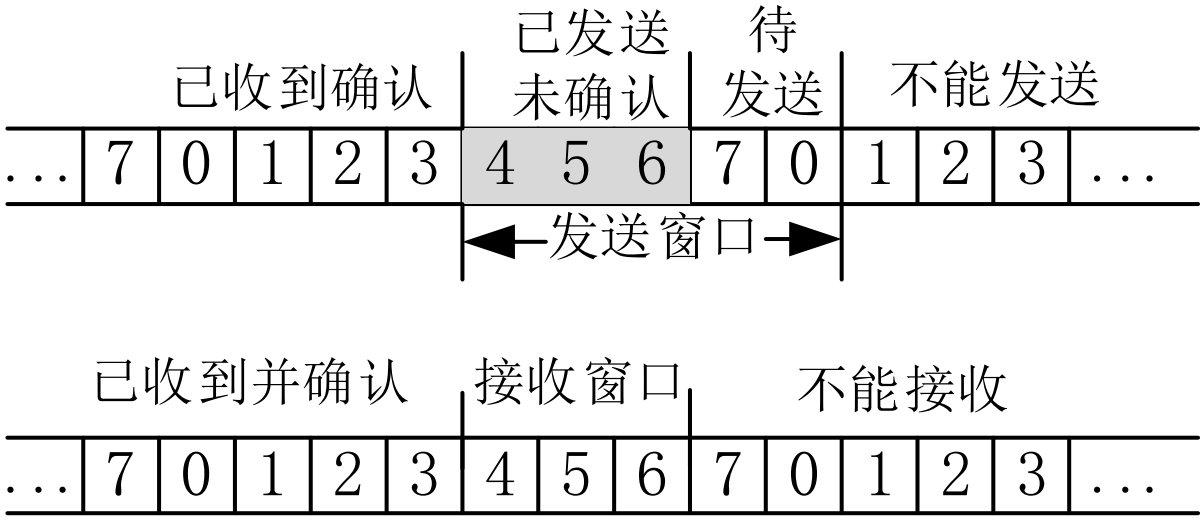


连续ARQ：如何减少重传？

RDT5.0：选择重传ARQ协议

- SR：Selective Repeat
- 接收方可以缓存不按序到达的数据包
  - 等待缺少的序号前的数据包都到达后，按序交付上层
  - 接收窗口大小：可以缓存的最大包个数
- 发送方只需重传差错/丢失的数据包

选择重传协议：发送窗口和接收窗口



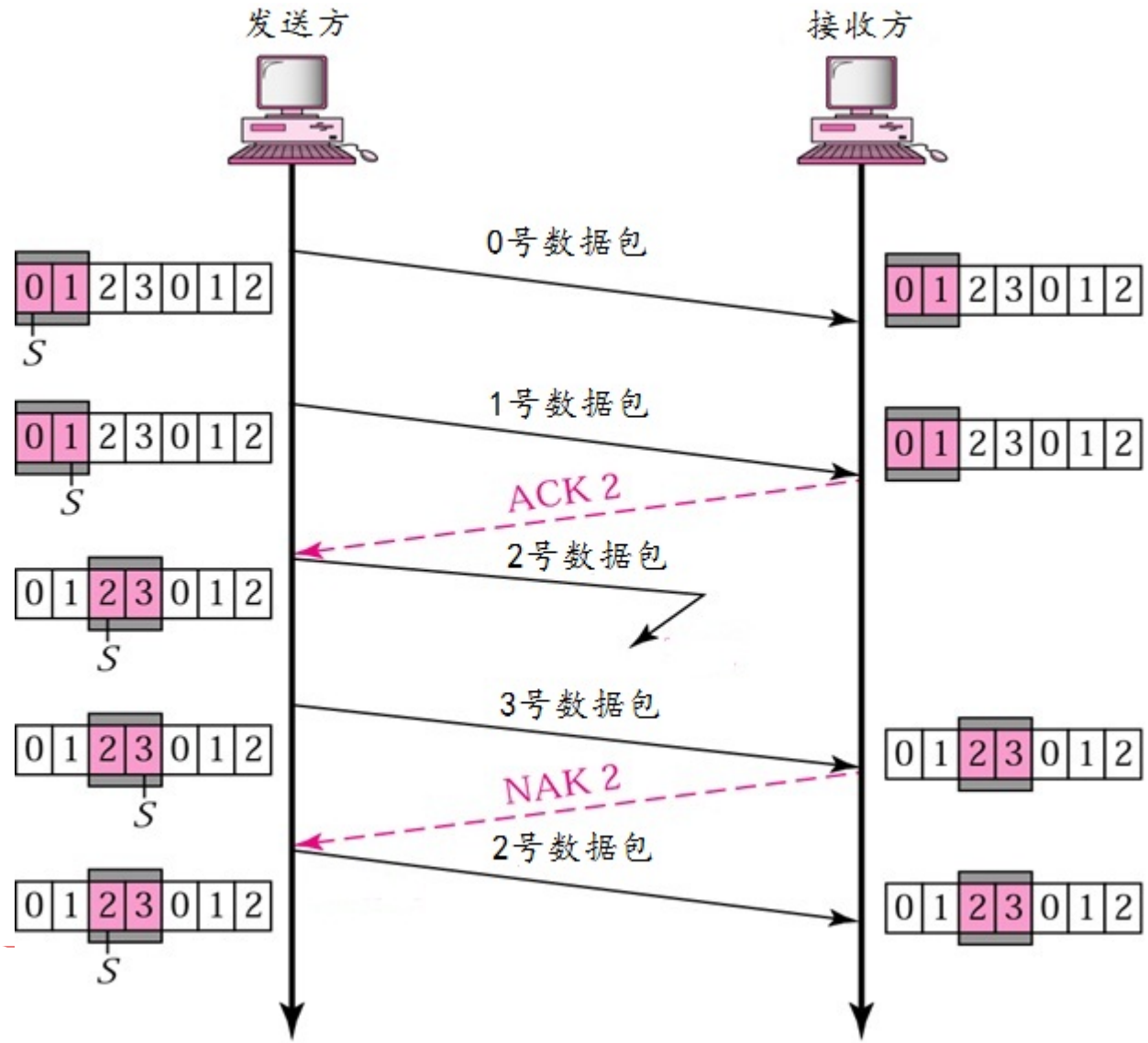
- 发送窗口W_S，接收窗口W_R，n位序号

$$W_S + W_R \leq 2^n,$$

一般取$W_S = W_R = 2^{(n-1)}$

为什么？参照Go-Back-N的最大窗口思考！

选择重传：数据包丢失示例



[Forouzan]

可靠的数据传输协议小结（1）

- 什么是可靠的数据传输？
  - 无传输差错：收到的数据=发送的数据
  - 无数据丢失
  - 无数据重复
  - 无失序：接收数据顺序=发送数据顺序

可靠的数据传输协议小结（2）

- **停止-等待ARQ协议**
  - 发送方一次只发送一个数据包
  - 数据包中包含校验信息，以检测传输差错
  - 对于无错的数据包，接收方发送ACK
  - 发送方使用定时器，解决数据包丢失问题
  - 数据包和ACK中均包含一位序号，解决重复数据包的问题
  - 信道利用率：$U = \frac{T_t}{T} = \frac{T_t}{T_t + 2T_p} = \frac{1}{1 + 2\alpha}$
  - 问题：一次只发送一个数据包，导致信道利用率低
  - 解决方案：一次发送多个数据包

可靠的数据传输协议小结（3）

- **Go-Back-N协议**
  - 发送窗口≤2^n-1，接收窗口=1
  - 出错时，要重传自出错包开始的所有已发送的数据包（回退N步）
  - 最大信道利用率：$$U=\frac{T_t}{T}=\frac{NT_t}{T_t+2T_p}=\frac{2^n-1}{1+2\alpha}$$
- **选择重传协议**
  - 发送窗口=接收窗口= 2^{n-1}
  - 只重传出错的数据包
  - 最大信道利用率：$$U=\frac{T_t}{T}=\frac{NT_t}{T_t+2T_p}=\frac{2^{n-1}}{1+2\alpha}$$

可靠的数据传输协议小结（4）

机制	用途和说明
校验和	用于检测一个传输数据包中的比特错误
定时器	用于超时 / 重传一个数据包，可能因为数据包或ACK在信道中丢失了
序号	用于为从发送方流向接收方的数据包按顺序编号
确认	接收方用于告诉发送方一个数据包或一组数据包已经被正确接收了，通常携带着数据包序号
否认	接收方用于告诉发送方某个数据包未被正确接收
发送窗口、流水线	发送方允许被限制发送落在窗口内的数据包，可以增加信道利用率

内容提要

- 3.1 传输层的功能及服务
- 3.2 可靠数据传输的原理
- *3.3 无连接传输协议：UDP*
  - 使用UDP的应用及常用端口
  - UDP的特点
  - 伪报头及校验和
- 3.4 面向连接的传输协议：TCP
- 3.5 拥塞控制的主要原理
- 3.6 传输层的安全隐患

UDP: 用户数据报协议[RFC 768]

- 简单高效的传输层协议
- 提供“尽力而为(best effort)”服务
  - UDP数据报可能丢失
  - 接收的顺序可能与发送顺序不一致
- ***无连接协议***
  - 在发送数据之前，发送端和接收端没有握手(handshaking)
  - 每个UDP数据报都是独立的，和其他的数据报无关
- 比IP增强了：
  - 多路复用/多路分解

为什么要使用UDP?

- 无需连接建立，时延低
- 简单：无状态
- 报头短，开销低
- 无拥塞控制：不考虑网络状况，UDP数据可以尽快发送

自私！

在传输层使用UDP的应用

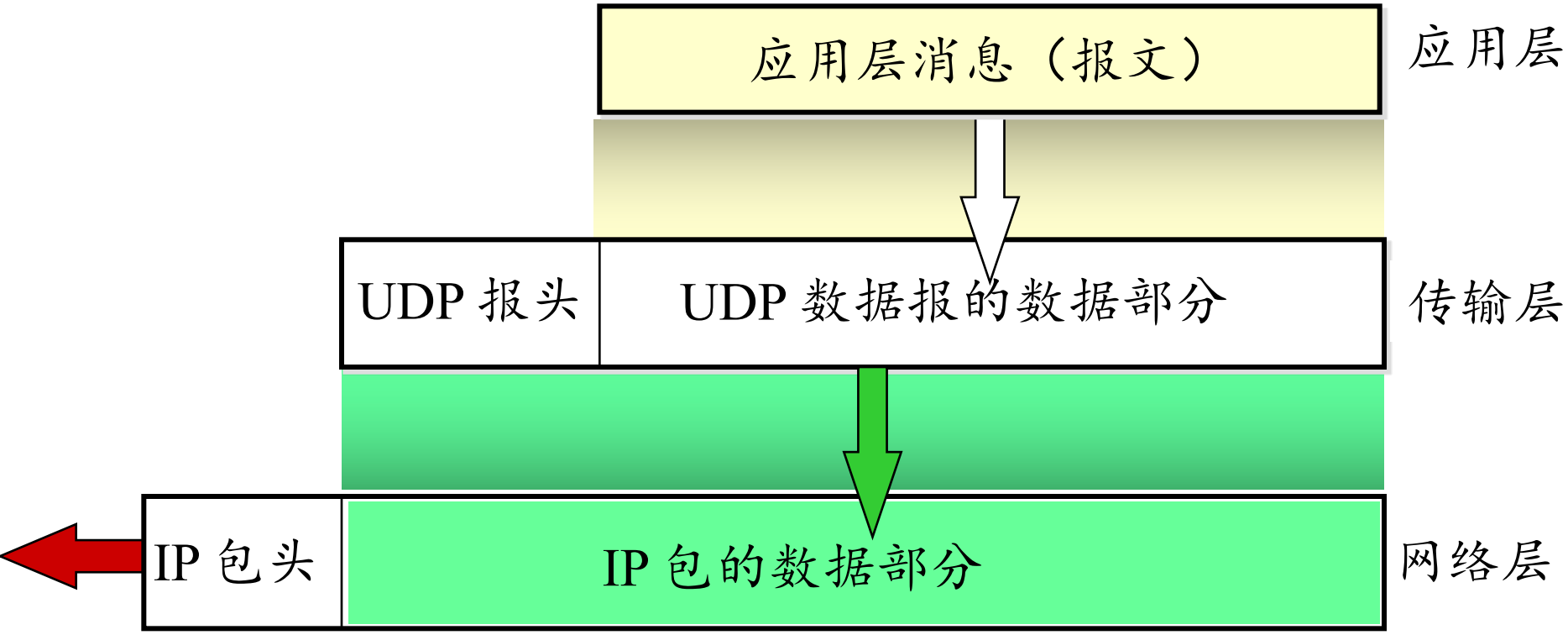
- 多媒体应用
  - 低时延
  - 高速率
- 少量、频繁的数据传送
  - DNS
  - 网络管理协议(SNMP)
  - 路由表更新
- 如果要在UDP之上增加可靠性，由应用层负责

UDP的熟知端口（常用端口）

端口号	应用协议名称	功能描述
7	Echo	将接收到的数据报原样发回发送端
13	Daytime	服务器端返回当前的日期和时间
53	DNS	域名服务
67	Bootps	DHCP的服务器端口
68	Bootpc	DHCP的客户端口
69	TFTP	简单文件传输协议
111	RPC	远程过程调用
123	NTP	网络时间协议
161	SNMP	简单网络管理协议，代理在此端口接收请求消息
162	SNMP	简单网络管理协议，管理者在此端口接收trap（差错报告）消息

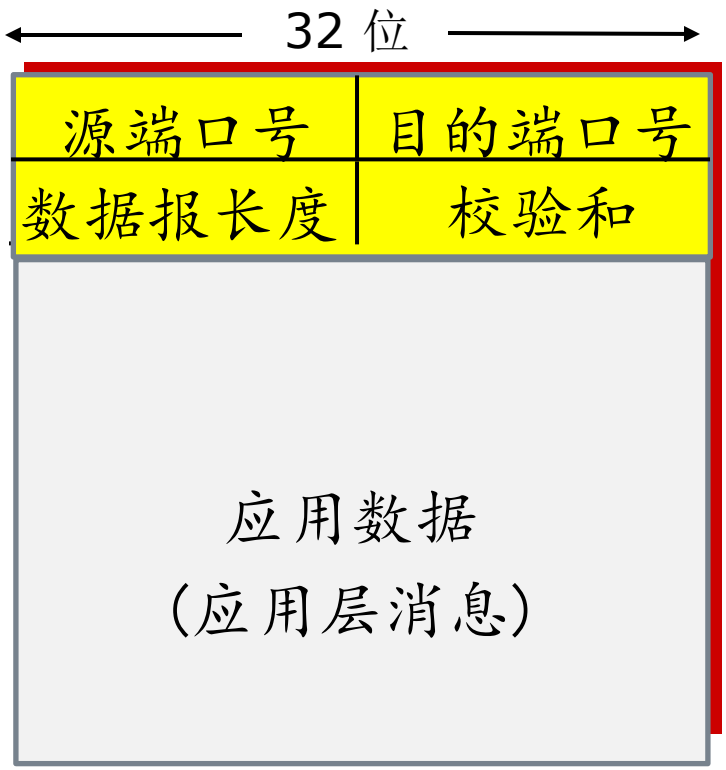
UDP面向消息（报文）

- UDP直接在应用层消息前面加上报头
- 保留应用层消息的边界



UDP的数据报格式

- 报头：固定为8字节
  - 源端口号：16位
  - 目的端口号：16位
  - 数据报长度：16位，以字节为单位，报头+数据，最大长度65535字节
  - 校验和：16位



UDP校验和(checksum，检查和)

目的：检查数据传输中是否发生错误

发送端:

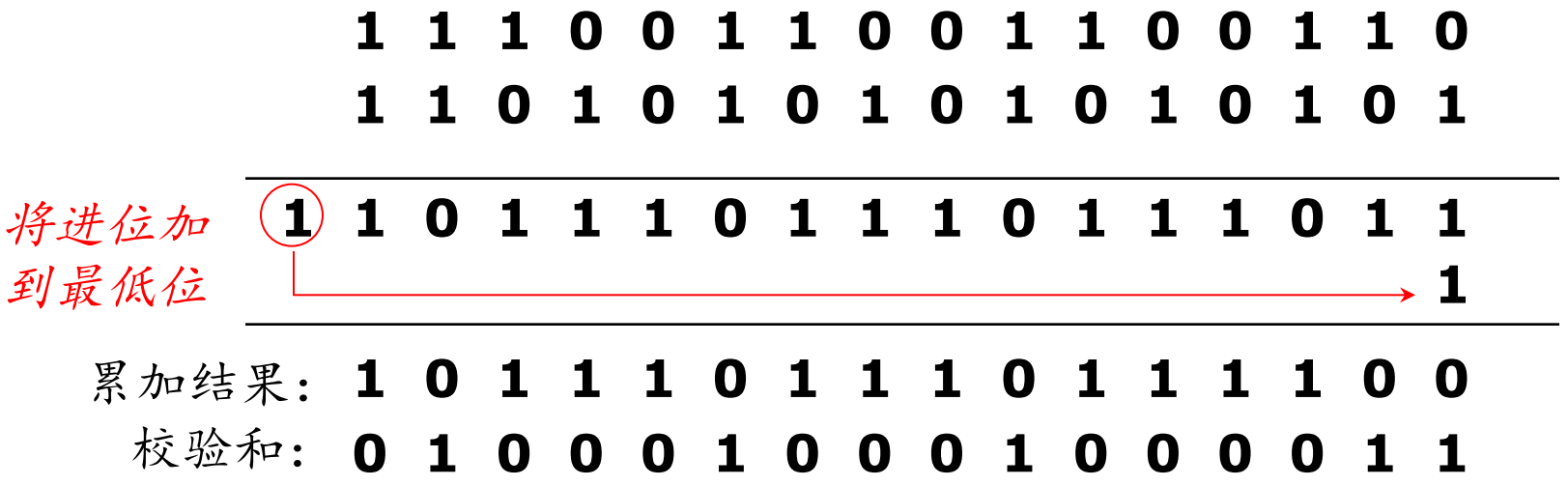
- 将UDP数据报按16位为一组 分成多个组
  - 校验和字段值设为0
  - 最后一组不足16位则补0
- 计算出校验和(16位)
- 将校验和值填入校验和字段

接收端:

- 将收到的UDP数据报按16位一组划分，计算校验和
- 检查计算结果是否=0
  - 为0：未检查出错误
  - 不为0：传输有错

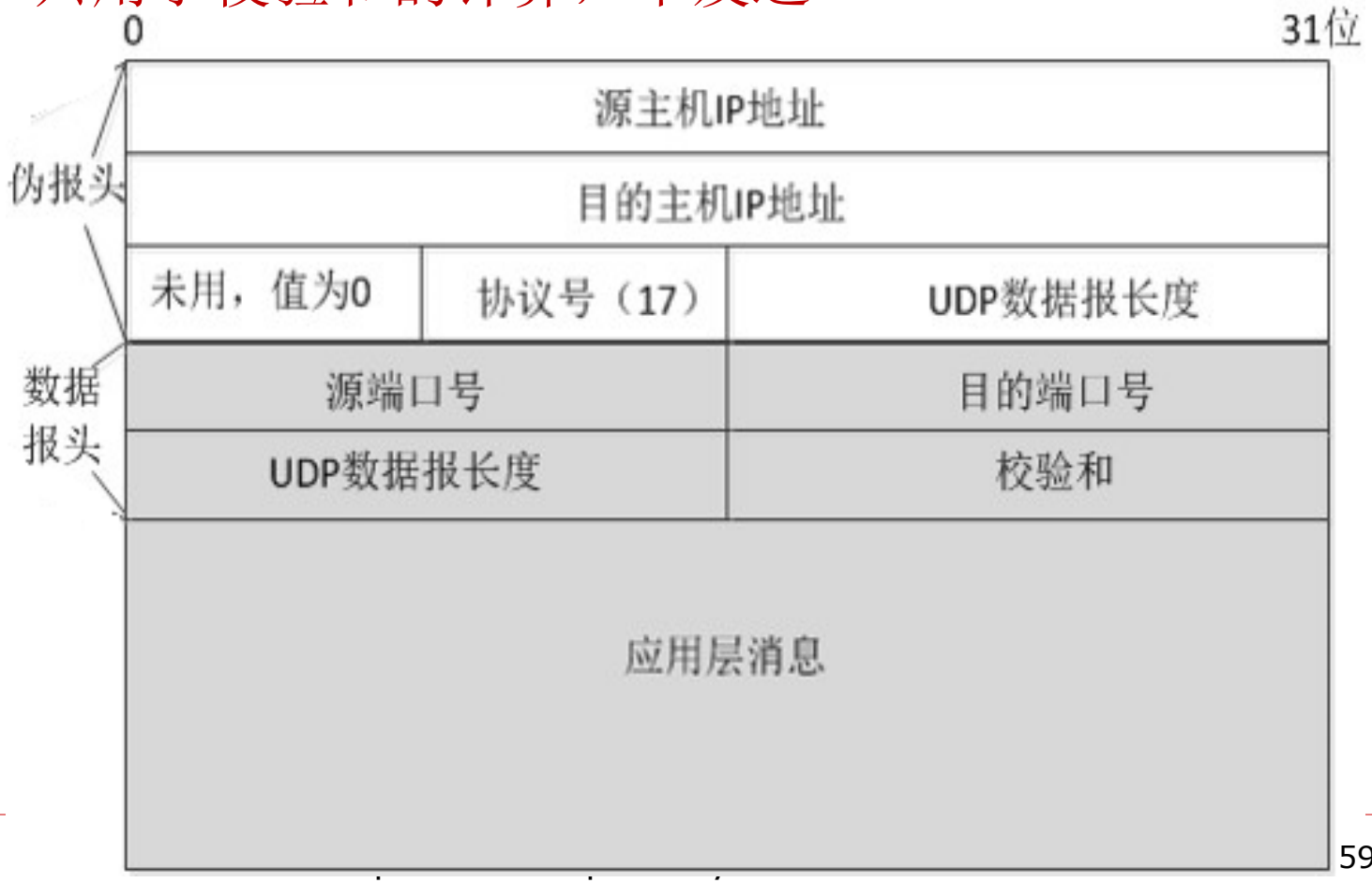
校验和计算方法

- 将数据按16位一组来分组
- 各组对应位相加
- 最高位的进位累加到结果的最低位
- 对最终结果取反码

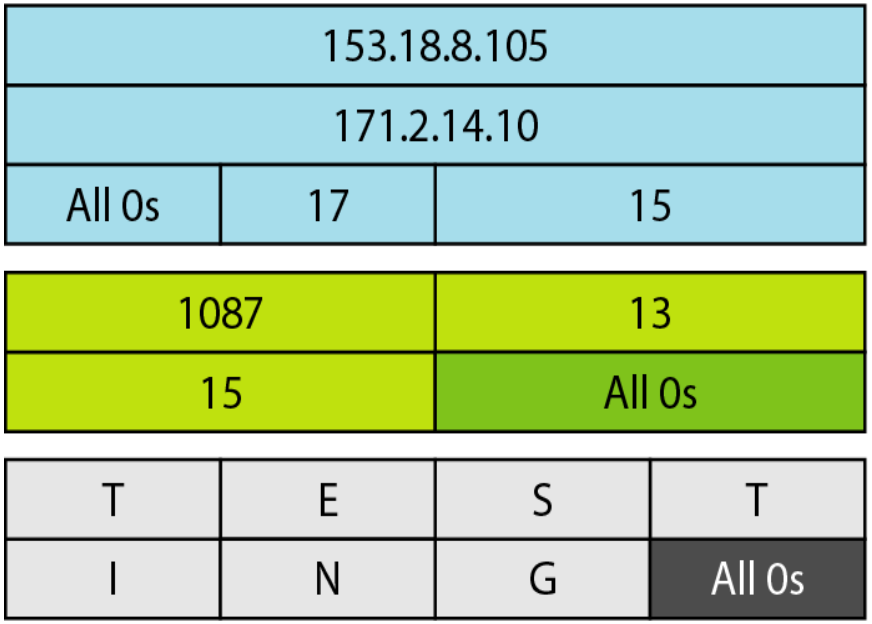


伪报头(Pseudoheader)

- 何为“伪”？
  - 只用于校验和的计算，不发送



UDP 校验和计算示例



10011001 00010010 → 153.18
00001000 01101001 → 8.105
10101011 00000010 → 171.2
00001110 00001010 → 14.10
00000000 00010001 → 0 and 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0 (checksum)
01010100 01000101 → T and E
01010011 01010100 → S and T
01001001 01001110 → I and N
01000111 00000000 → G and 0 (padding)

10010110 11101011 → Sum
01101001 00010100 → Checksum

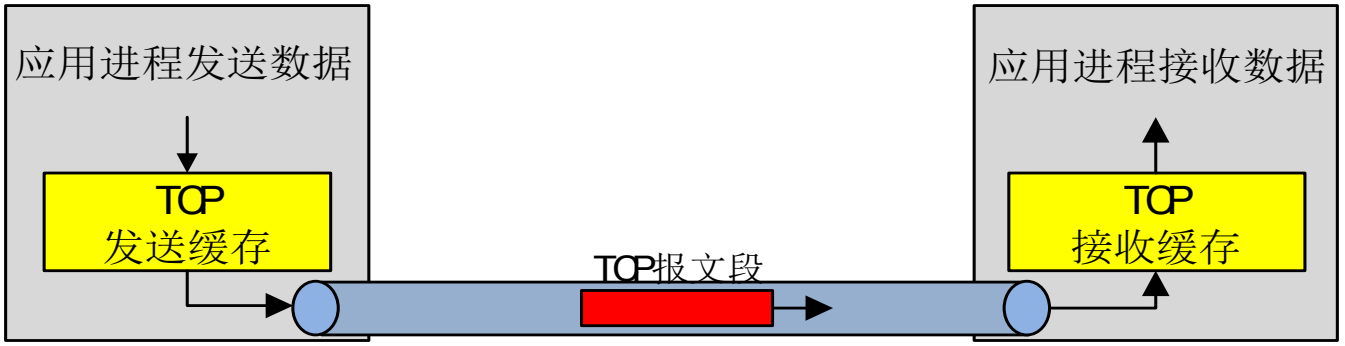
[Forouzan]

内容提要

- 3.1 传输层的功能及服务
- 3.2 可靠数据传输的原理
- 3.3 无连接传输协议：UDP
- *3.4 面向连接的传输协议：TCP*
  - TCP的报文段结构
  - 连接管理
  - 可靠数据传输
  - 流量控制
- 3.5 拥塞控制的主要原理
- 3.6 传输层的安全隐患

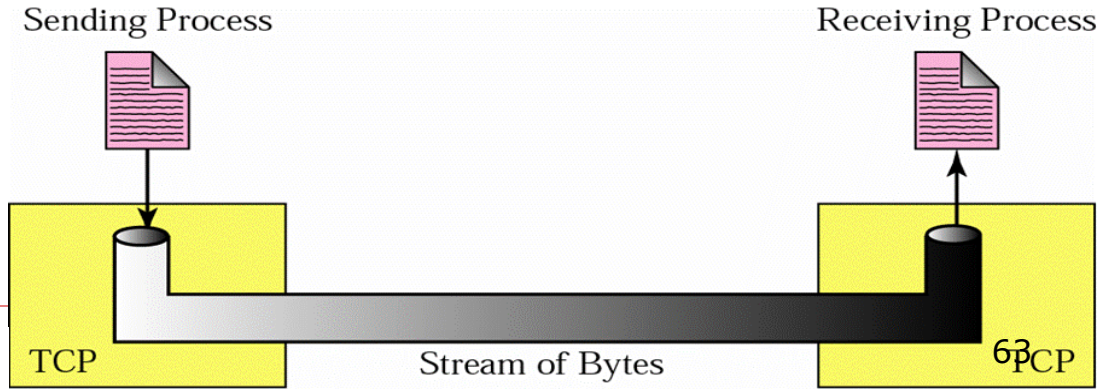
TCP：要点(1) RFC: 793, 1122, 1323, 2018, 2581

- 面向连接
  - 数据发送之前，发送端和接收端要握手(handshaking)，建立逻辑连接，协商参数，如起始序号、窗口长度等
- 全双工数据传输(full duplex):
  - 在一个连接之上，可以同时进行双向数据传输
- 端到端传输
  - 1-1：单播(unicast)



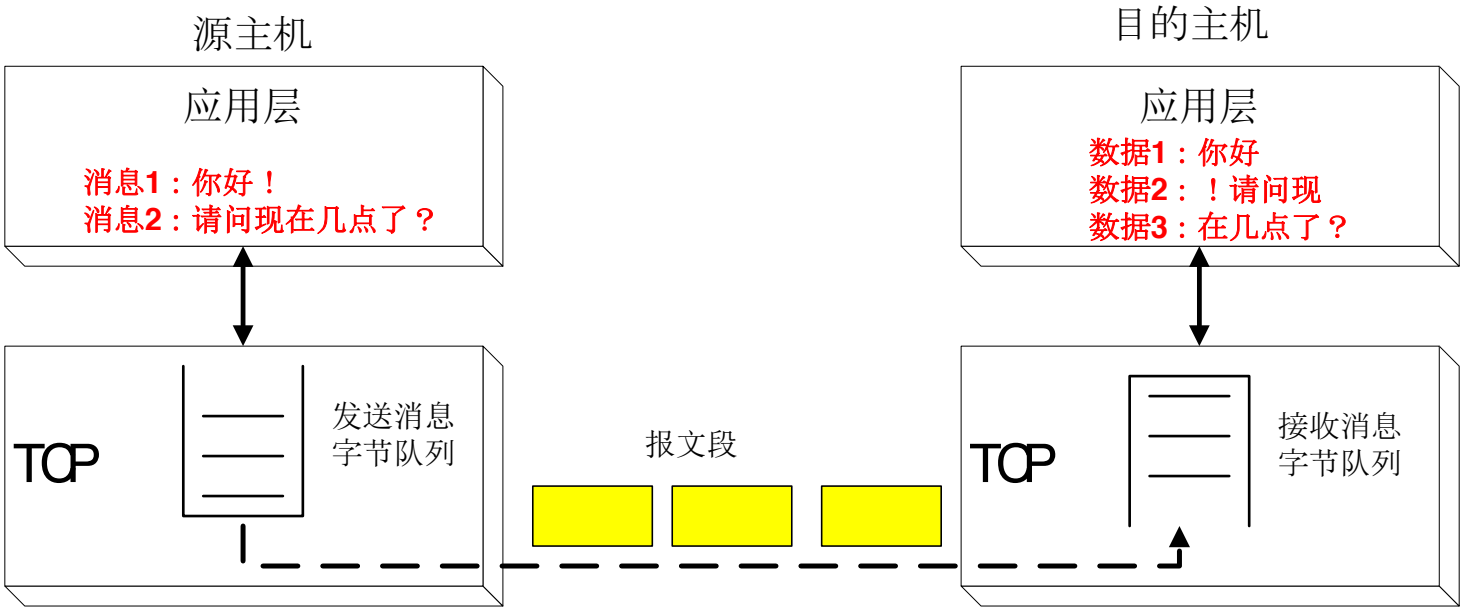
TCP要点(2)

- 采用连续ARQ协议，流水线方式工作
  - 发送窗口长度由流量控制和拥塞控制算法决定
- 流量控制(Flow Control)
  - 接收端可以限制发送端发送速度（发送窗口长度）
- 拥塞控制(Congestion Control)
  - 根据网络状况，可以限制发送端的发送速度（发送窗口长度）
- 可靠、按序的***字节流***传输
  - 窗口长度以字节为单位



TCP的字节流传输示例

- TCP不能保证上层消息的边界



TCP的熟知端口

端口号	应用协议名称	功能描述
7	Echo	将接收到的数据报原样发回发送端
13	Daytime	服务器端返回当前的日期和时间
20	FTP	文件传输协议，数据连接的端口号
21	FTP	文件传输协议，控制连接的端口号
23	TELNET	终端仿真协议
25	SMTP	简单邮件传输协议
53	DNS	域名服务
80	HTTP	超文本传输协议
110	POP3	邮局协议

TCP的报文段

报文段里数据部分第一个字节的编号

要接收的下一个字节的编号

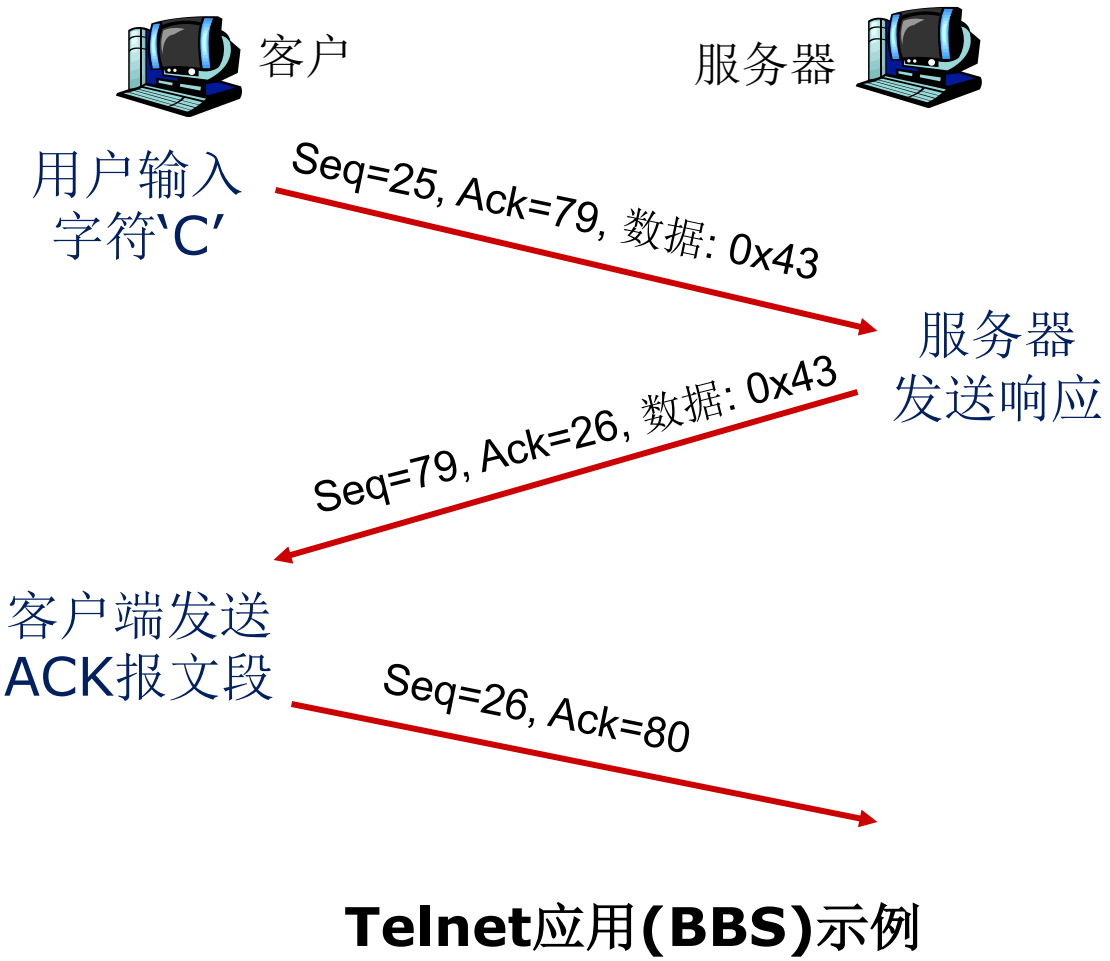
段头长度，以4字节为单位

能够接收的字节数，用于流量控制

校验和，与UDP校验 方法相同

TCP：发送序号和接收序号

- 发送序号（Seq）
  - 报文段中数据部分第一字节的编号
  - 发送端维护，本端上一次发送序号+数据长度
- 接收序号（Ack）
  - 要从对端接收的下一个字节的编号
  - 对端的发送序号+数据长度

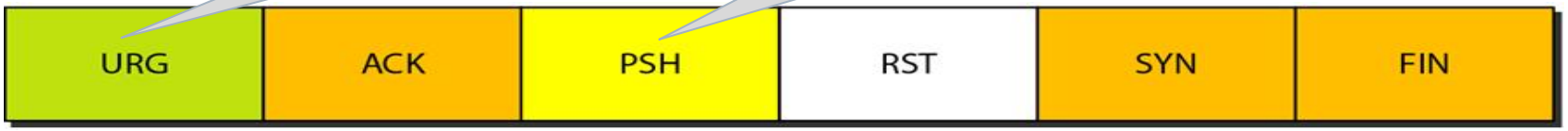


Telnet应用(BBS)示例

控制字段

紧急数据：要求接收端尽快处理，而不必在接收缓存中排队

Push：要求接收端立刻将现有缓存数据交付应用层，而不必等待后续数据

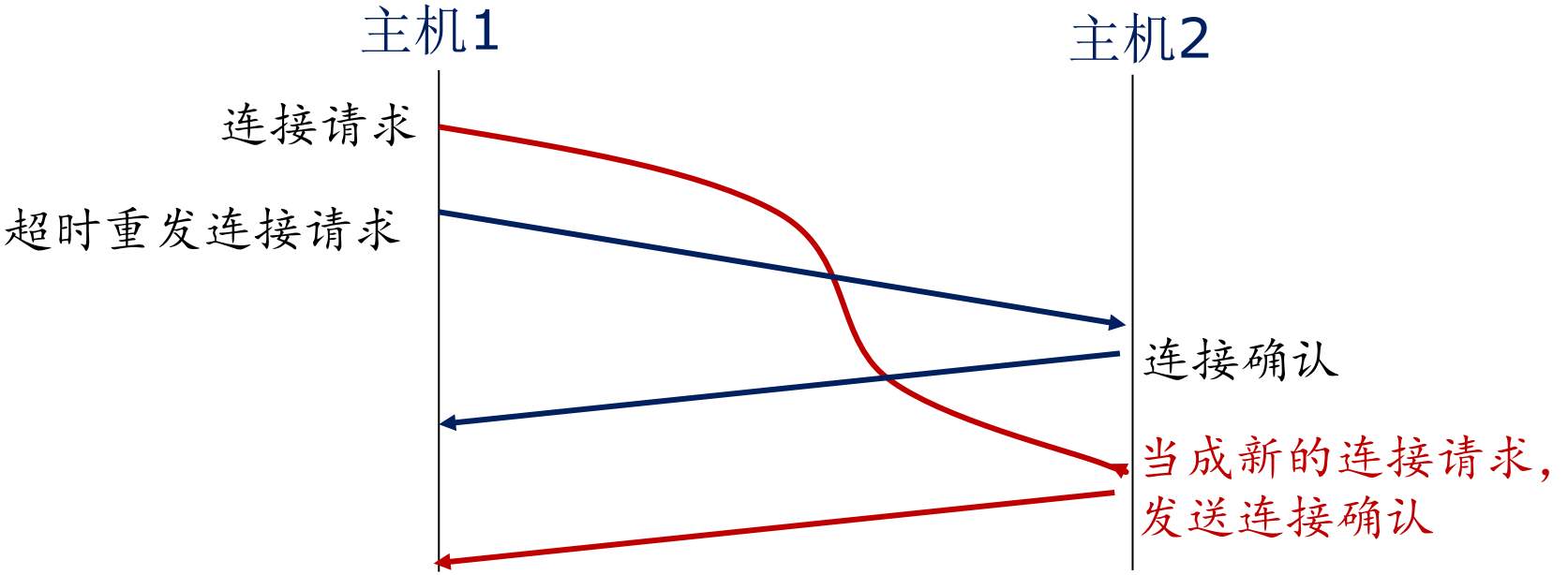


- 置为1有效

控制位	功能
URG	表示报文中携带了紧急数据，“紧急数据指针”包含紧急数据最后一字节的序号
ACK	表示“接收序号”有效
PSH	表示报文段中的数据应快速提交给应用层
RST	表示复位TCP连接
SYN	表示连接建立的请求或响应
FIN	表示连接释放的请求或响应

TCP连接管理：连接建立（1）

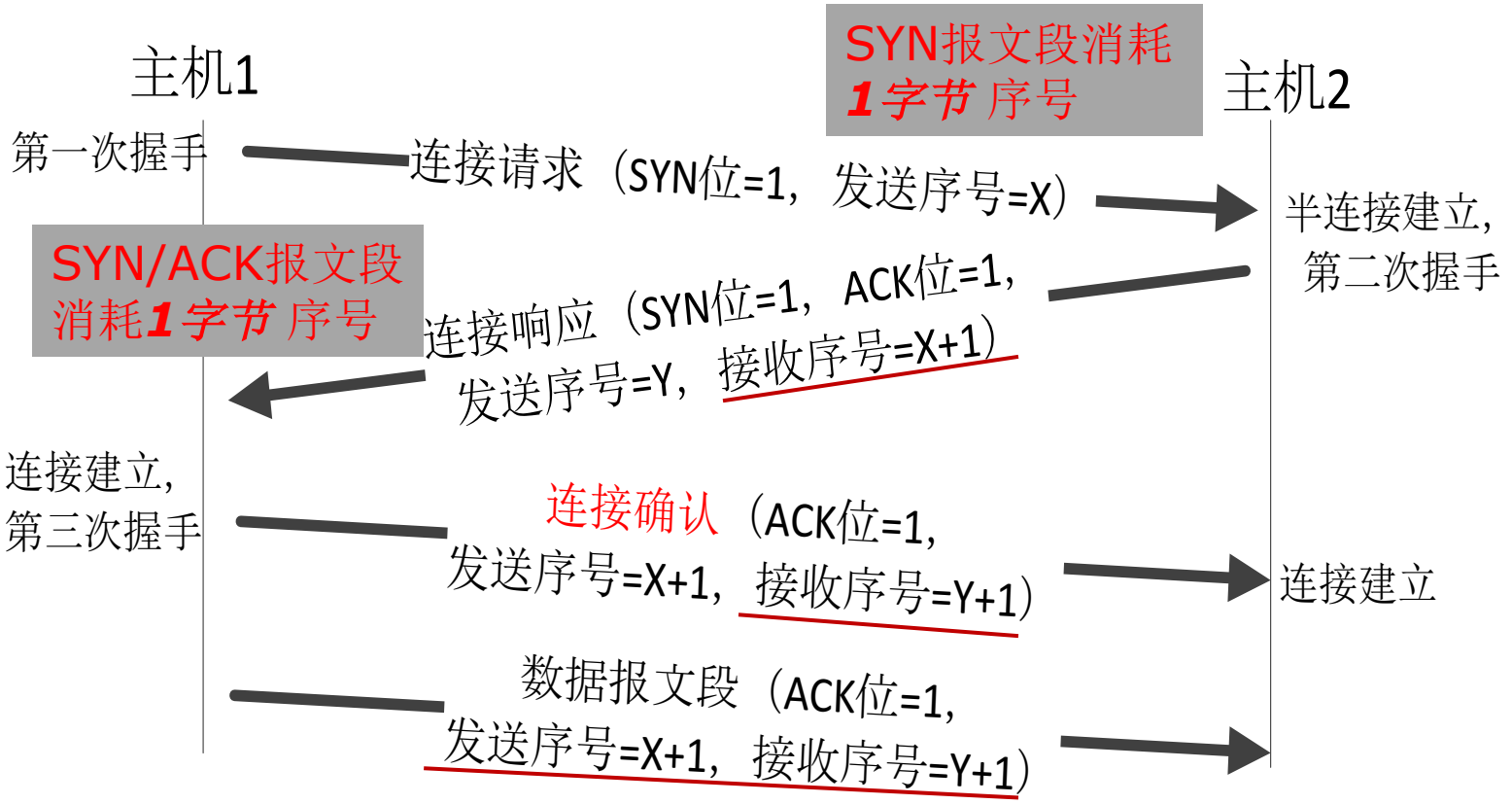
- 在传送数据之前，发送端和接收端之间要先建立连接
  - 协商起始序号、接收窗口长度等参数
- 连接建立：两次握手机制不能保证可靠性
  - 失效的重复连接请求（old duplicate）导致错误地建立连接



TCP的连接建立

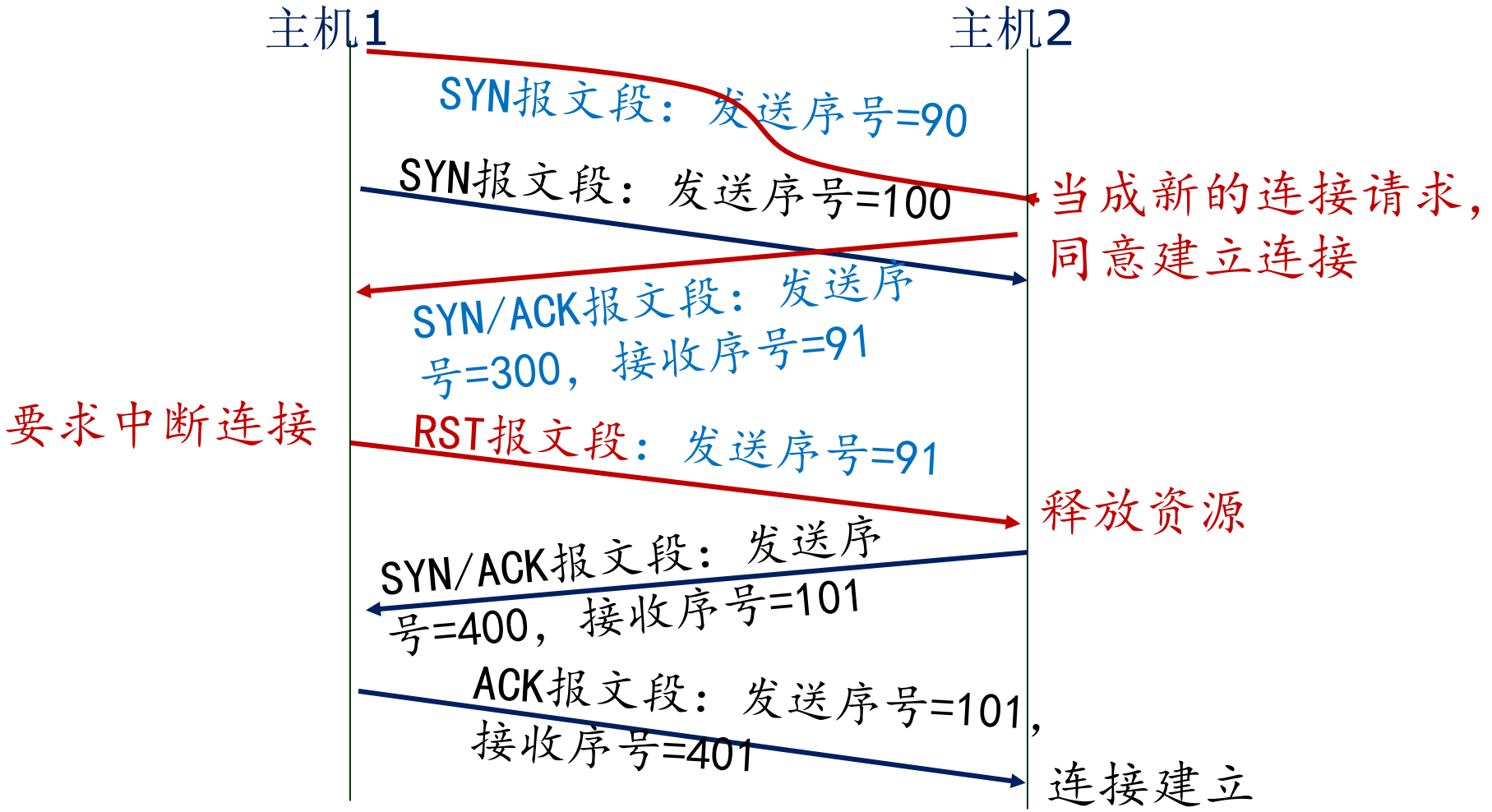
- 连接建立：三次握手
  - 第1步：TCP客户端发送SYN报文段（连接请求）
    - SYN位=1
    - 指定客户端的发送序号，无数据
  - 第2步：TCP服务器端返回SYN/ACK报文段（连接响应）
    - SYN位=1 ACK位=1
    - 指定服务器端的发送序号，无数据
  - 第3步：TCP客户端回应ACK报文段（连接确认）
    - ACK位=1
    - 报文段内可能包含数据

TCP连接建立示例


主机1
主机2
SYN报文段消耗
1字节 序号
第一次握手
连接请求（SYN位=1，发送序号=X）
半连接建立，
第二次握手
SYN/ACK报文段
消耗1字节 序号
连接响应（SYN位=1，ACK位=1，
发送序号=Y，接收序号=X+1）
连接建立，
第三次握手
连接确认（ACK位=1，
发送序号=X+1，接收序号=Y+1）
连接建立
数据报文段（ACK位=1，
发送序号=X+1，接收序号=Y+1）

三次握手解决了失效的重复连接请求（old duplicate）的问题

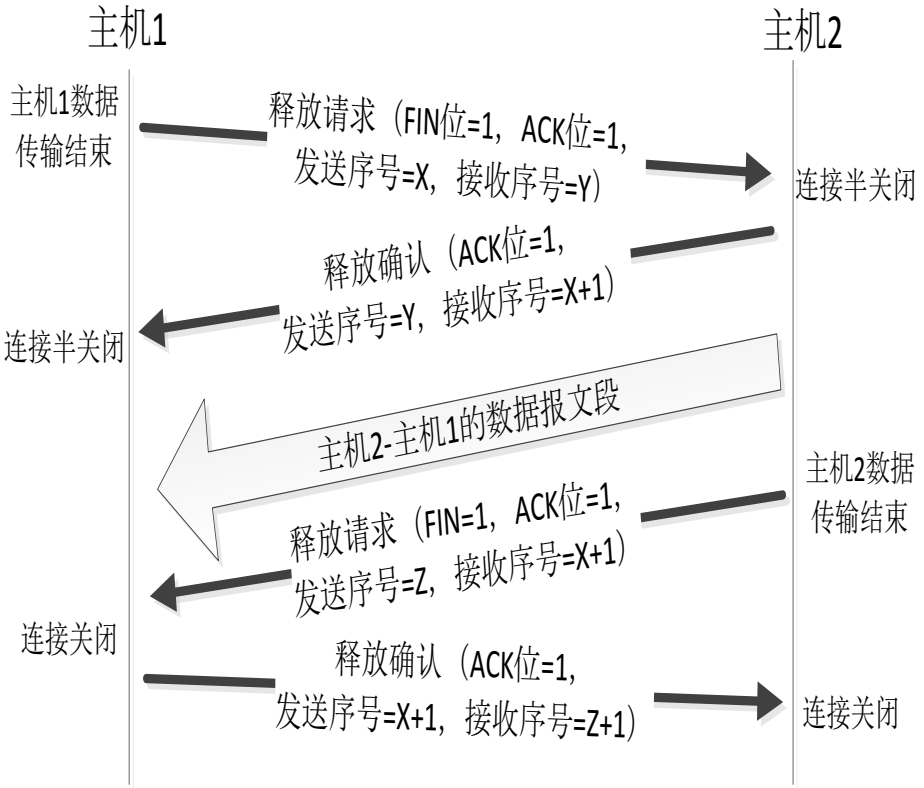
[RFC793]



TCP的连接释放：4步释放

- 两个方向分别释放：半关闭连接
- 第1步：主机1发送FIN报文段
- 第2步：主机2返回ACK报文段，主机1-2方向的连接被关闭，资源被释放
- 第3步：主机2发送FIN报文段
- 第4步：主机1返回ACK报文段，主机2-1方向的资源被释放，整个连接关闭

FIN报文段消耗***1*字节**序号



因特网中普遍采用

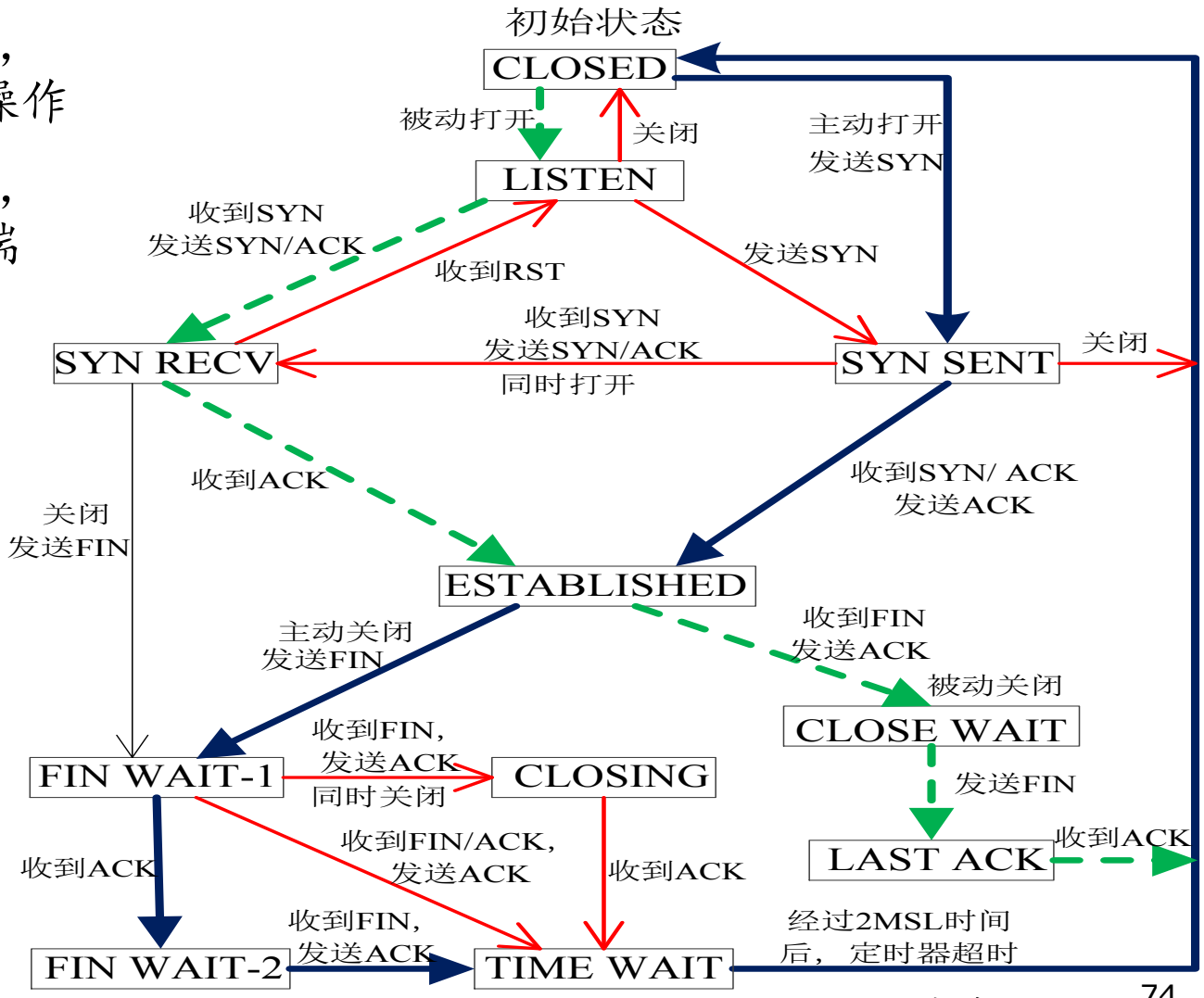
TCP连接管理：状态转移图

主动打开/关闭，通常是客户端操作

被动打开/关闭，通常是服务器端操作

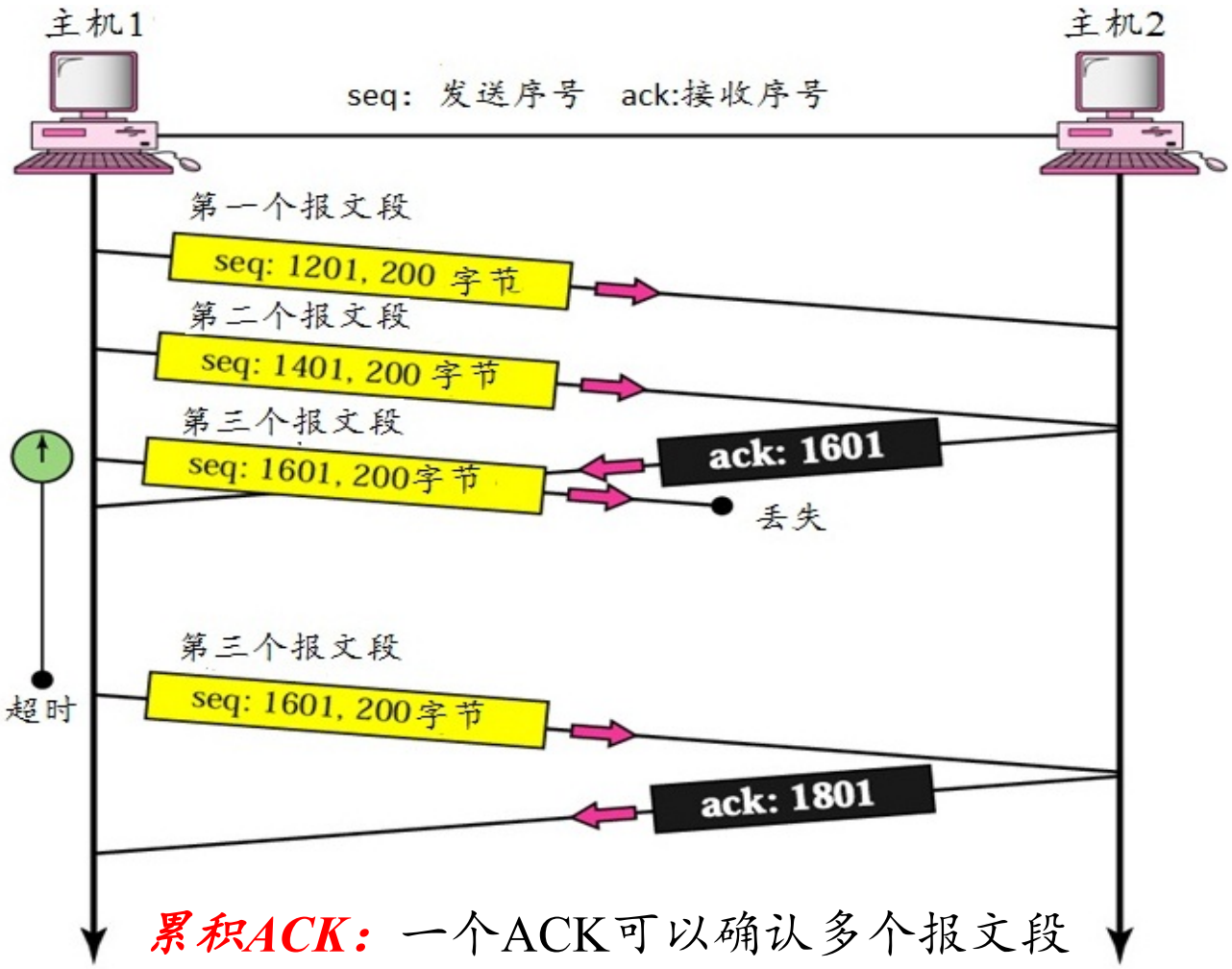
异常转移

只需了解



MSL：最大报文段生命期

TCP的可靠数据传输：序号管理

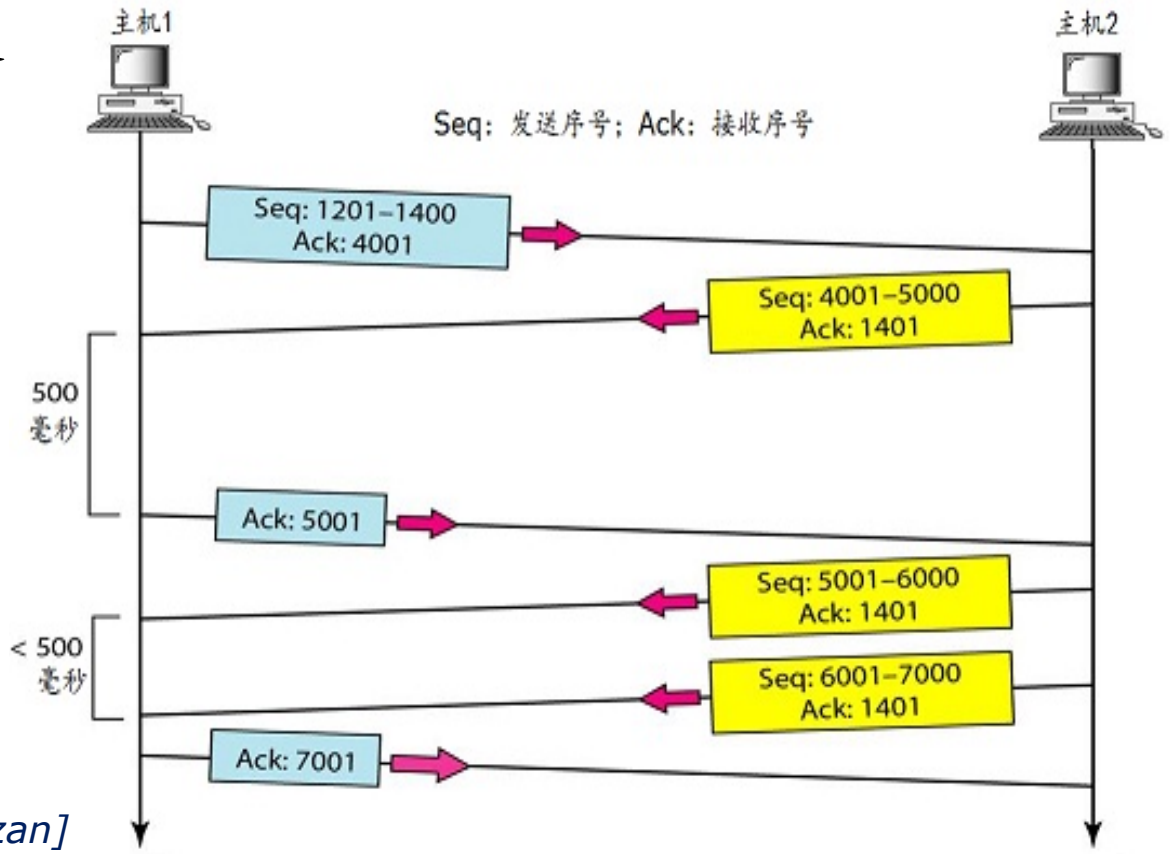


累积ACK： 一个ACK可以确认多个报文段

[Forouzan]

ACK定时器

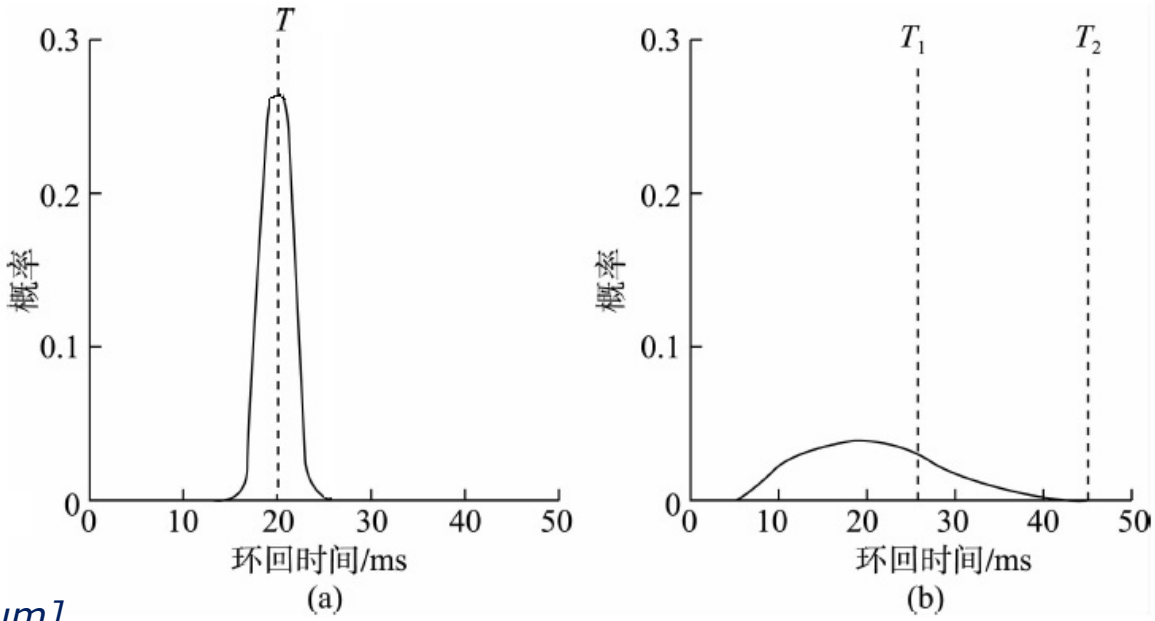
- TCP优先使用捎带确认方式：在回传的报文段中捎带接收序号
- ACK定时器：500毫秒，若超时，则发送无数据的ACK报文段



[Forouzan]

重传超时间隔的设置

- RTO：Retransmission TimeOut
- RTT：Round Trip Time，环回时延
- 通信双方直接相连的环境下（如LAN），RTT可以用均值 来度量（如下图a）；但对于可能跨越多个网络的TCP通信，不能用均值来代表（如下图b）



[Tanenbaum]

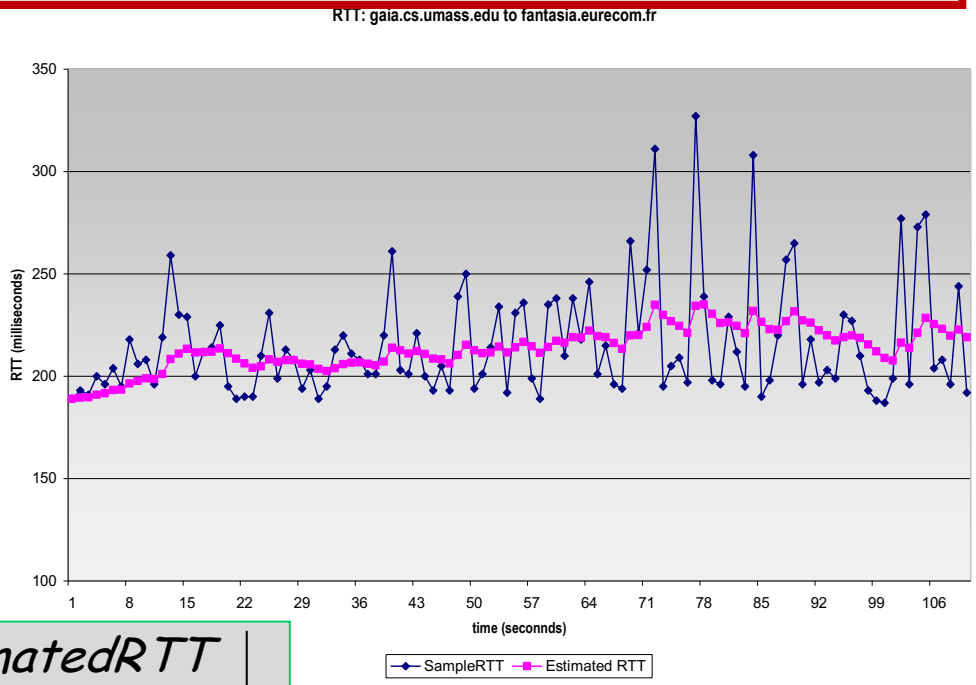
TCP环境下RTO的估计

- RTT的估值

$$RTT\text{估值} = \alpha \times RTT\text{估值的历史值} + (1-\alpha) \times RTT\text{的测量值}$$

$$RTT\text{测量值} = ACK\text{到达时刻} - \text{TCP报文段发送时刻}$$

  - α：平衡因子，$0 \leqslant \alpha \leqslant 1$

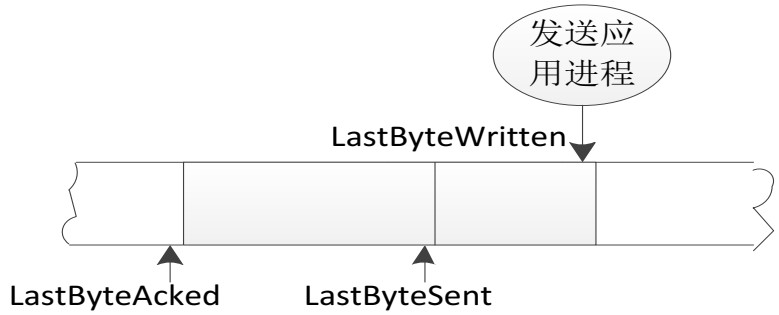


- RTO的估值
  - Jackson算法（了解）

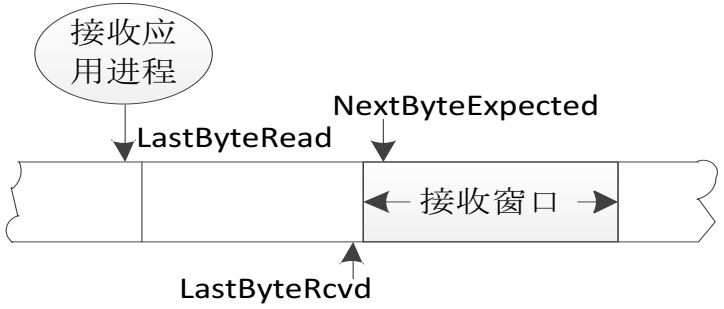
$$Difference = |SampleRTT - EstimatedRTT|$$
$$Deviation = \alpha \times Deviation + (1-\alpha) \times Difference$$
$$RTO = EstimatedRTT + 4 \times Deviation$$

TCP的流量控制

- 流量控制的目的：防止发送方发送过快而导致接收方缓存溢出，造成数据丢失
- 基本方法：接收方反馈，限制发送方的发送数据量
  - 报文段中“**接收窗口**”字段说明了接收方最多可以接收的数据字节数
  - 发送方在收到下一个ACK之前发送的数据总量不能超过接收窗口值
- 发送窗口与接收窗口

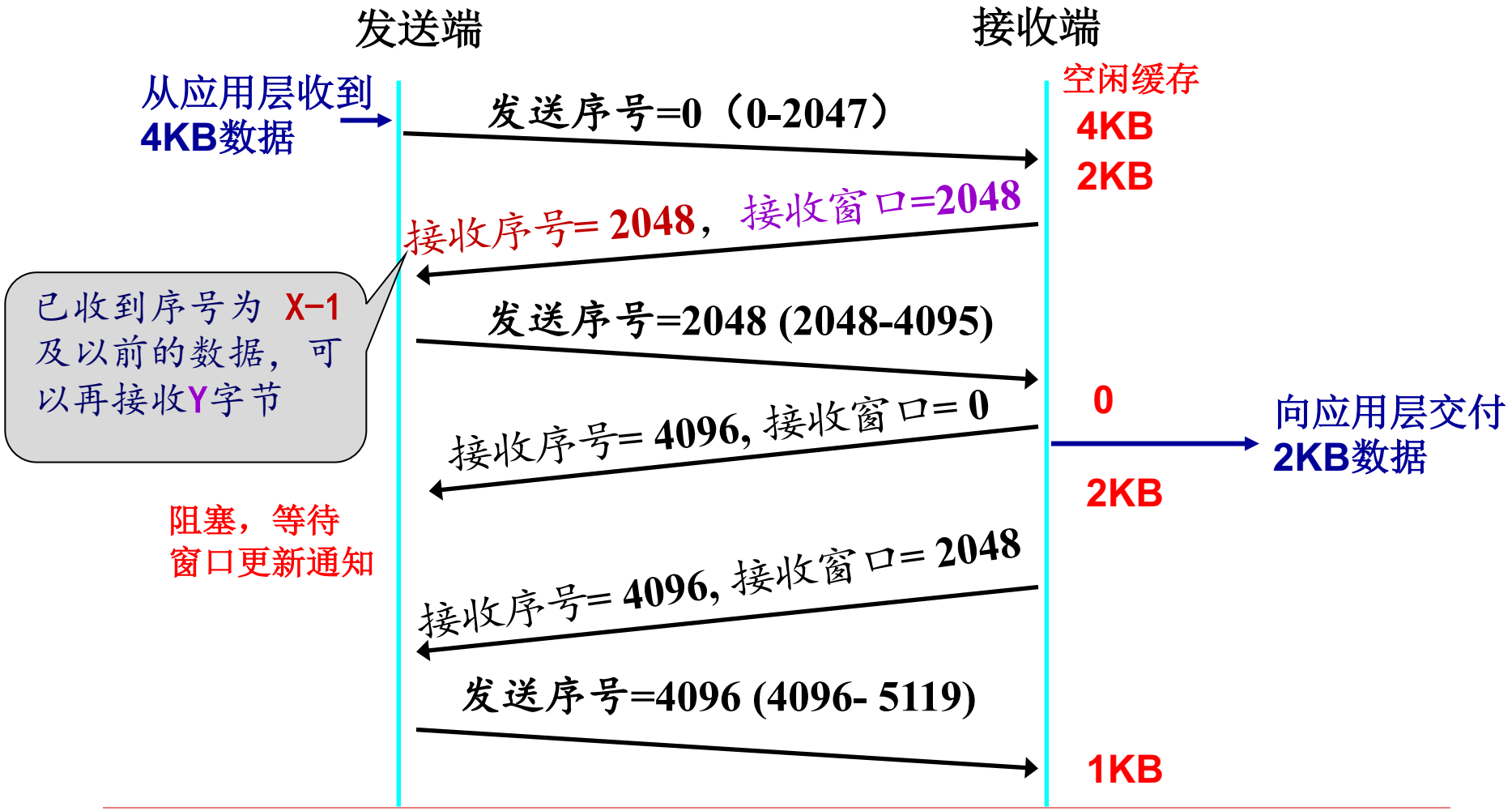


(a) 发送窗口



(b) 接收窗口

TCP流量控制：示例


发送端
接收端
从应用层收到
4KB数据
空闲缓存
4KB
2KB
发送序号=0（0-2047）
接收序号= 2048，接收窗口=2048
已收到序号为 X-1
及以前的数据，可
以再接收Y字节
发送序号=2048 (2048-4095)
0
向应用层交付
2KB数据
接收序号= 4096, 接收窗口= 0
2KB
阻塞，等待
窗口更新通知
接收序号= 4096, 接收窗口= 2048
发送序号=4096 (4096- 5119)
1KB

TCP的传输效率问题

- MSS：最大报文段长度，一个报文段中的最大数据长度，在建立连接时协商
  - 通常受底层物理网络的限制，例如在以太网上MSS=1460字节
- 报文段中的数据量接近MSS，则传输效率高
- 某些应用每次发送的数据量较小
  - 交互类应用，如Telnet，每次只发送1字节数据
  - 传输效率低！

提高传输效率：Nagle算法

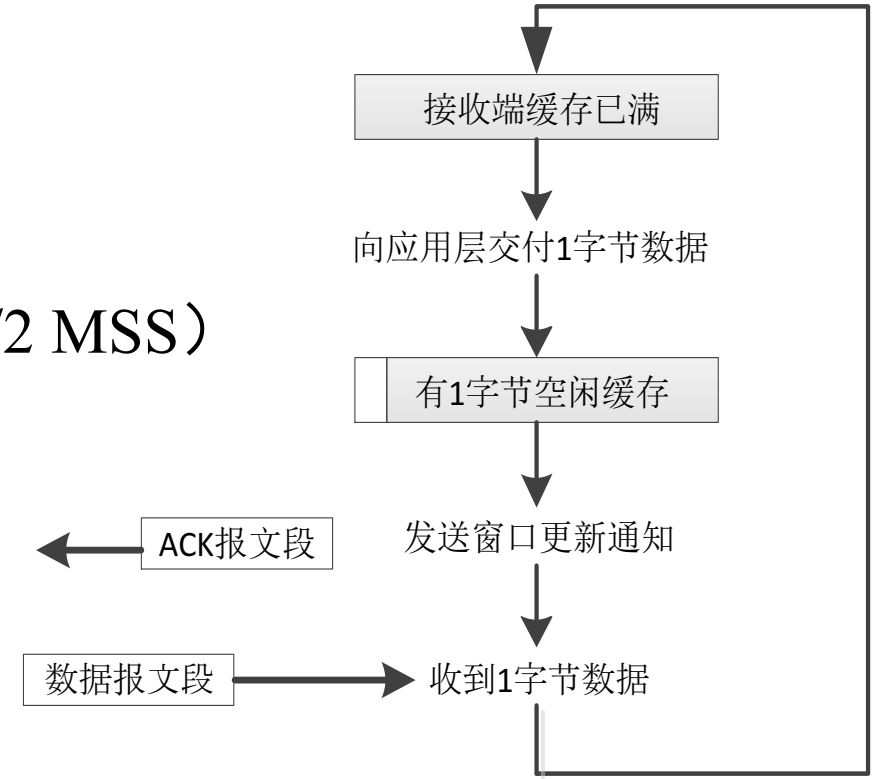
- 发送端策略
  - 一次尽量发送较大的数据量

```
If 应用层数据长度≥MSS and 接收窗口≥MSS,
        发送数据长度为MSS的报文段
Else
        If 有未确认的数据,
                将数据缓存等待接收端的确认
        Else 立即发送全部数据
```

傻瓜窗口问题及Clark算法

- 接收端频繁发送少量窗口更新通知，导致传输低效

- Clark算法：接收端策略
  - 只在有较大缓存（至少为1/2 MSS）时，才发送窗口更新通知



TCP的要点

- 连接管理：三次握手建立连接，4步释放连接
- 采用字节序号、捎带确认
- 动态滑动窗口（发送窗口不超过对端通知的接收窗口值）
- RTT估算：

$$RTT\text{估值} = \alpha \times RTT\text{估值的历史值} + (1-\alpha) \times RTT\text{的测量值}$$

- 提高传输效率
  - Nagle算法：一次尽量发送较大的数据量
  - Clark算法：只在有较大缓存时，才发送窗口更新通知

内容提要

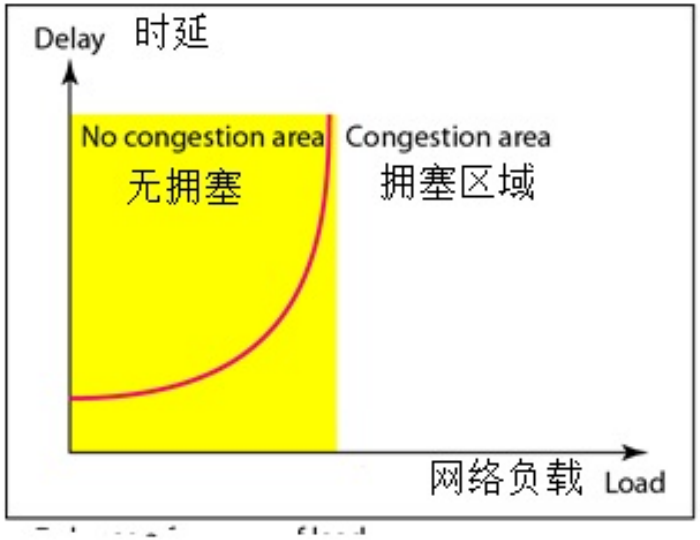
- 3.1 传输层的功能及服务
- 3.2 可靠数据传输的原理
- 3.3 无连接传输协议：UDP
- 3.4 面向连接的传输协议：TCP
- *3.5 拥塞控制的主要原理*
  - 拥塞的概念
  - 拥塞控制的主要方法
  - TCP的拥塞控制：慢启动、AIMD、快速重传
- 3.6 传输层的安全隐患

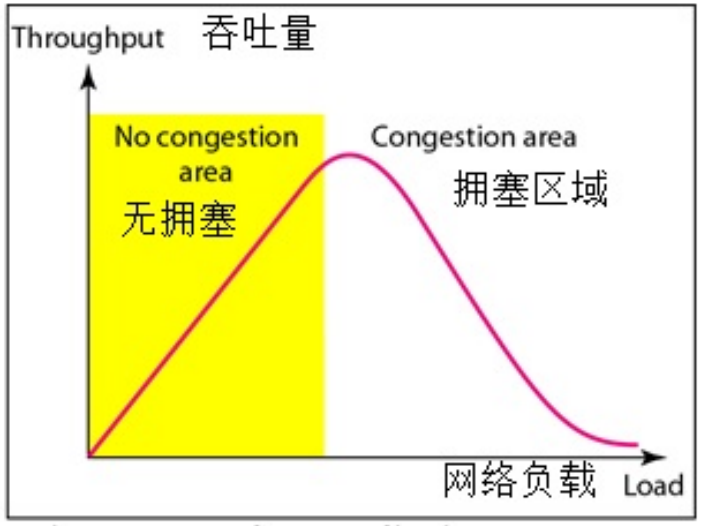
什么是拥塞？

- Congestion, informally: “too many sources sending too much data too fast for *network* to handle”
- 网络边缘主机发送到网络中的负载超出了网络的承载能力，即导致拥塞
- 网络拥塞的特征
  - 时延增大
    - 由于在路由器缓存中排队而导致
  - 数据包丢失
    - 由于路由器的缓存溢出而导致
- 重传“治标不治本”，甚至可能使拥塞更加严重！

网络拥塞时的性能

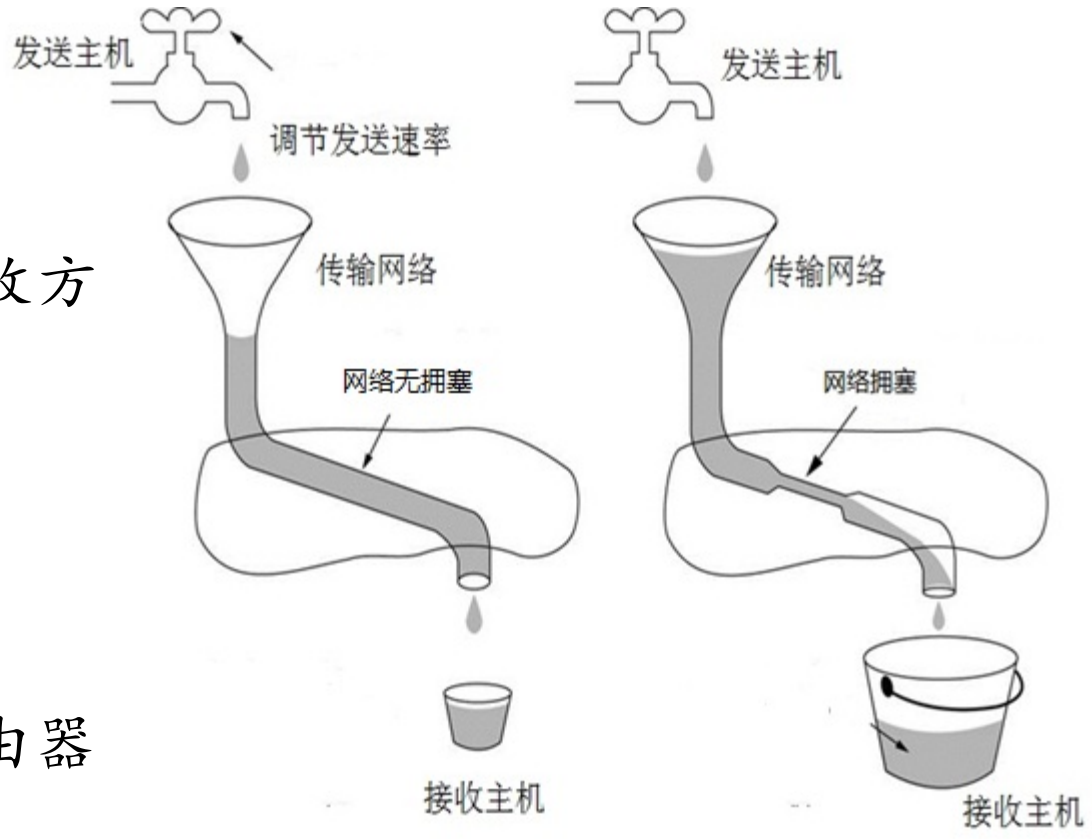
- 随着网络负载的增加
  - 传输时延增大
  - 吞吐量下降





拥塞控制 vs 流量控制

- 流量控制
  - 局部问题
  - 仅限于发送方和接收方之间
- 拥塞控制
  - 全局问题
  - 涉及全网所有的路由器和主机



[Tanenbaum]

拥塞控制：开环方法

- 通过对于传输协议的精心设计来避免拥塞
- 和拥塞控制相关的策略
  - 重传策略：合理设计超时定时器
  - 滑动窗口协议：选择重传ARQ 比连续ARQ协议更有助于避免拥塞；
  - ACK策略：捎带确认或者累积ACK可以减少网络负载；
  - 丢弃策略：对于某些应用（例如话音业务），适量丢弃数据包；
  - 接纳控制（Admission control）：在建立连接时可以使用接纳控制来限制源主机的发送速率，以避免拥塞。

拥塞控制方法：闭环方法

- 检测拥塞，在可能出现拥塞时，调整主机的发送速率以避免或缓解拥塞

端到端的拥塞控制:

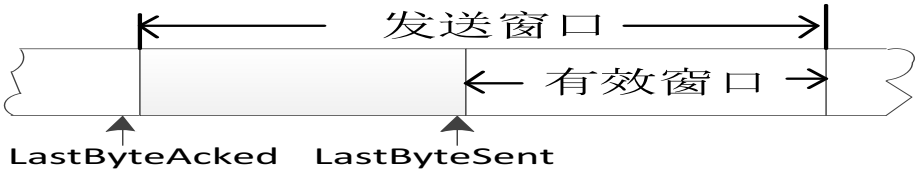
- 网络不提供拥塞通知（反馈）
- 端系统(主机)根据包丢失或时延来推断发生了拥塞
- TCP采用此方法

网络辅助进行拥塞控制：

- 在发生拥塞时，路由器向端系统提供反馈
  - 在数据包中包含1-2位拥塞通知信息（如TCP/IP的ECN/ECE/CWR)

TCP拥塞控制方法

- 端到端的拥塞控制
  - 发送端检测拥塞，通过减少拥塞窗口长度来降低发送数据率
- 拥塞窗口(cwnd)
  - 由发送端维护的一个变量
  - 表示最多可以向网络中发送的字节数
- 发送端的发送窗口长度
  - Min (接收窗口, 拥塞窗口)
- 有效发送窗口长度



TCP 拥塞控制：细节

- 发送端限制发送速率：

  LastByteSent-LastByteAcked

  ≤ cwnd

- 近似

$$\text{发送速率} = \frac{\text{cwnd}}{\text{RTT}} \quad \text{字节/秒}$$

- cwnd是动态变化的，取决于网络拥塞状况

发送端如何感知拥塞?

- 当定时器超时或收到3个重复ACK时，即认为有数据丢失
- 发送端随即降低发送速率(降低**cwnd**)

3个机制：

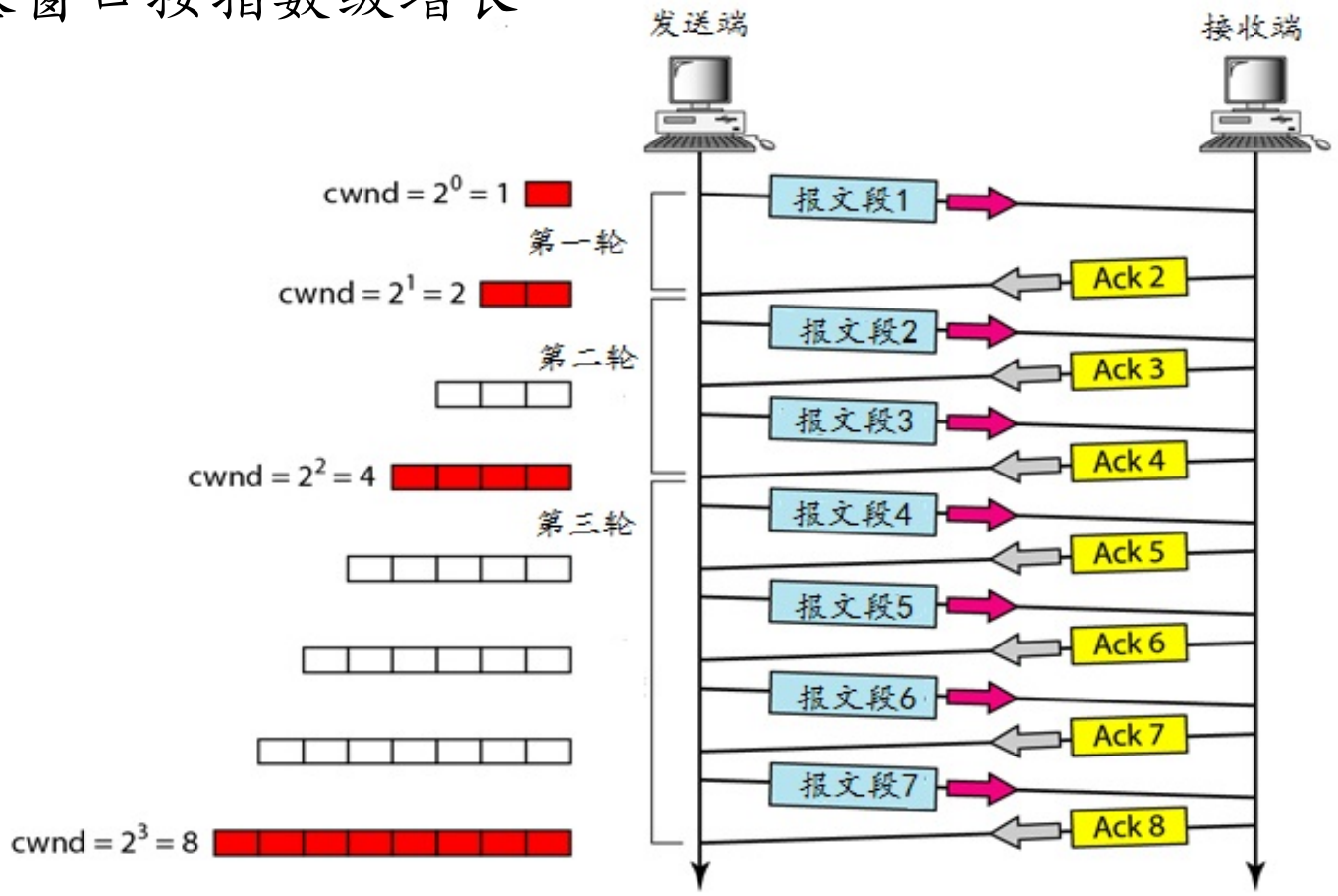
- 慢启动(slow start)
- AIMD
- 根据拥塞情况调整策略

慢启动(Slow Start)阶段

- 启动速率很低
- TCP连接建立时， 拥塞窗口 = 1 MSS
  - 示例: MSS = 1000 字节， RTT = 200 ms
  - 启动速率 = 5k 字节/秒
- 按指数级增加发送速率，直至发现数据丢失
  - 每收到1个ACK，cwnd+1
  - 每一轮发送（窗口内的数据全部发完），cwnd增加为原来的**2**倍
- 增长到何时？
  - 达到一个预先设定的阈值
  - 或者，出现数据丢失

慢启动示例

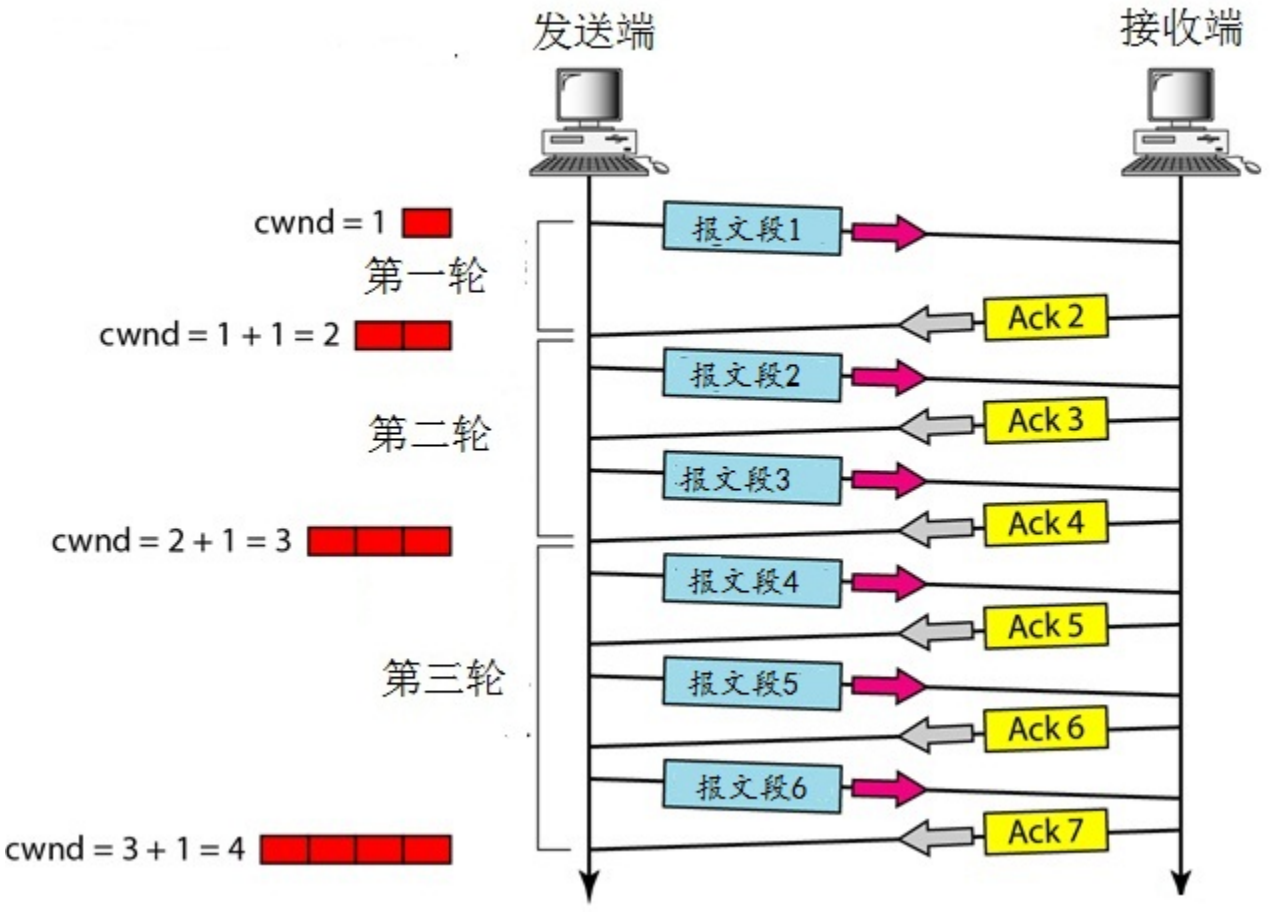
- 拥塞窗口按指数级增长



拥塞避免（CA）阶段

- 拥塞窗口达到阈值
  - 降低发送速度的增长，按线性增长
  - AI: Additive Increase，按加法增长
  - 每收到一个ACK，$cwnd = cwnd + \frac{1}{\text{本轮开始的}cwnd}$
  - 一轮结束，cwnd+1
- 增长到何时？
  - 出现数据丢失

AI示例


发送端
接收端
cwnd = 1
第一轮
cwnd = 1 + 1 = 2
第二轮
cwnd = 2 + 1 = 3
第三轮
cwnd = 3 + 1 = 4
报文段1
Ack 2
报文段2
Ack 3
报文段3
Ack 4
报文段4
Ack 5
报文段5
Ack 6
报文段6
Ack 7

出现数据丢失：MD

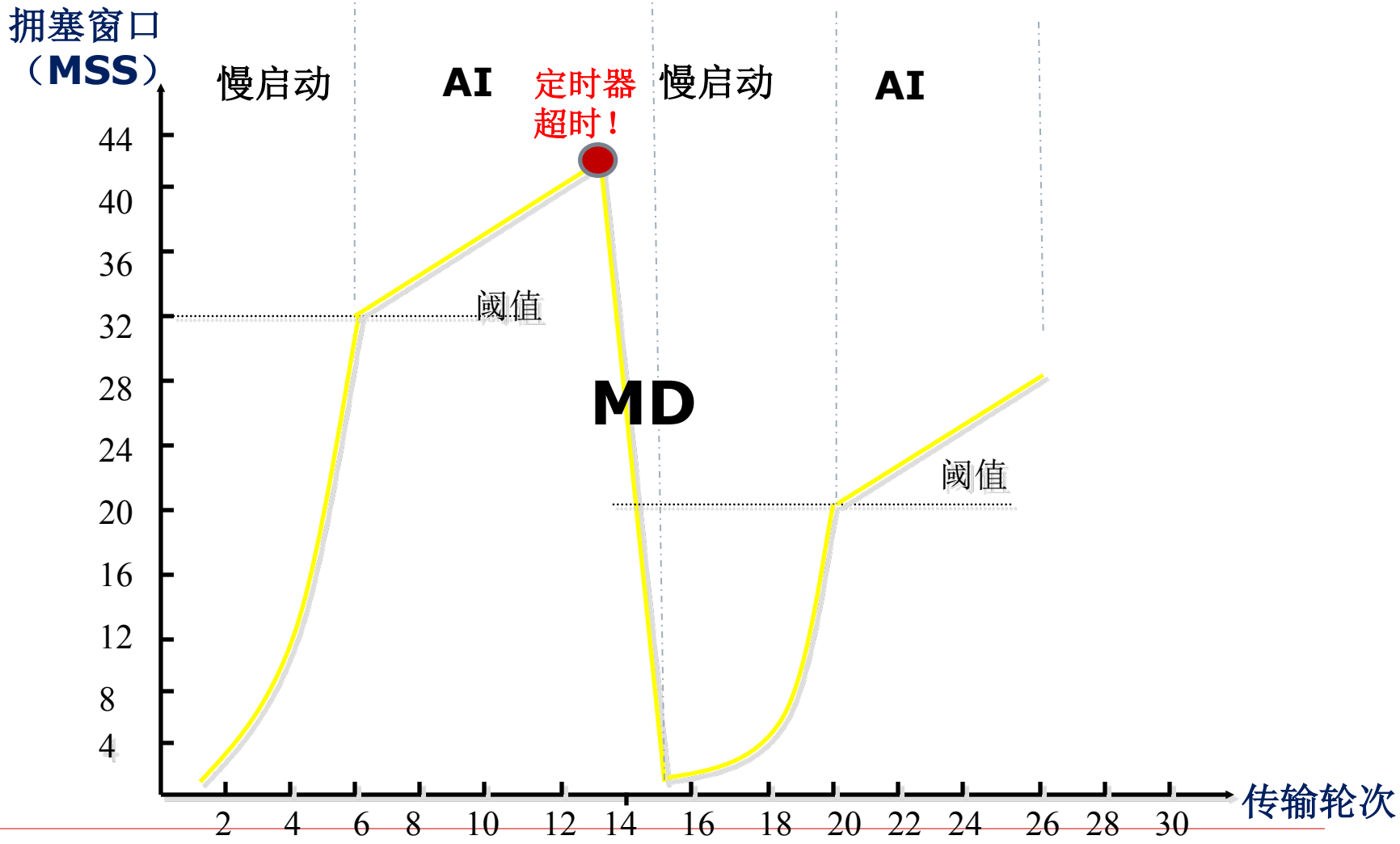
- MD：Multiplicative Decrease，按乘法减小
- 出现重发定时器超时
  - 阈值=当前拥塞窗口的一半
  - 拥塞窗口减为1
  - 开始新的“慢启动”
- 收到3次重复的ACK
  - 阈值=当前拥塞窗口的一半
  - 拥塞窗口=阈值
  - 开始新的AI，快速恢复

依据：

- 3个重复ACK表示网络还能够传输一些报文段
- 定时器超时则表示拥塞告警

根据拥塞的不同情况调整策略

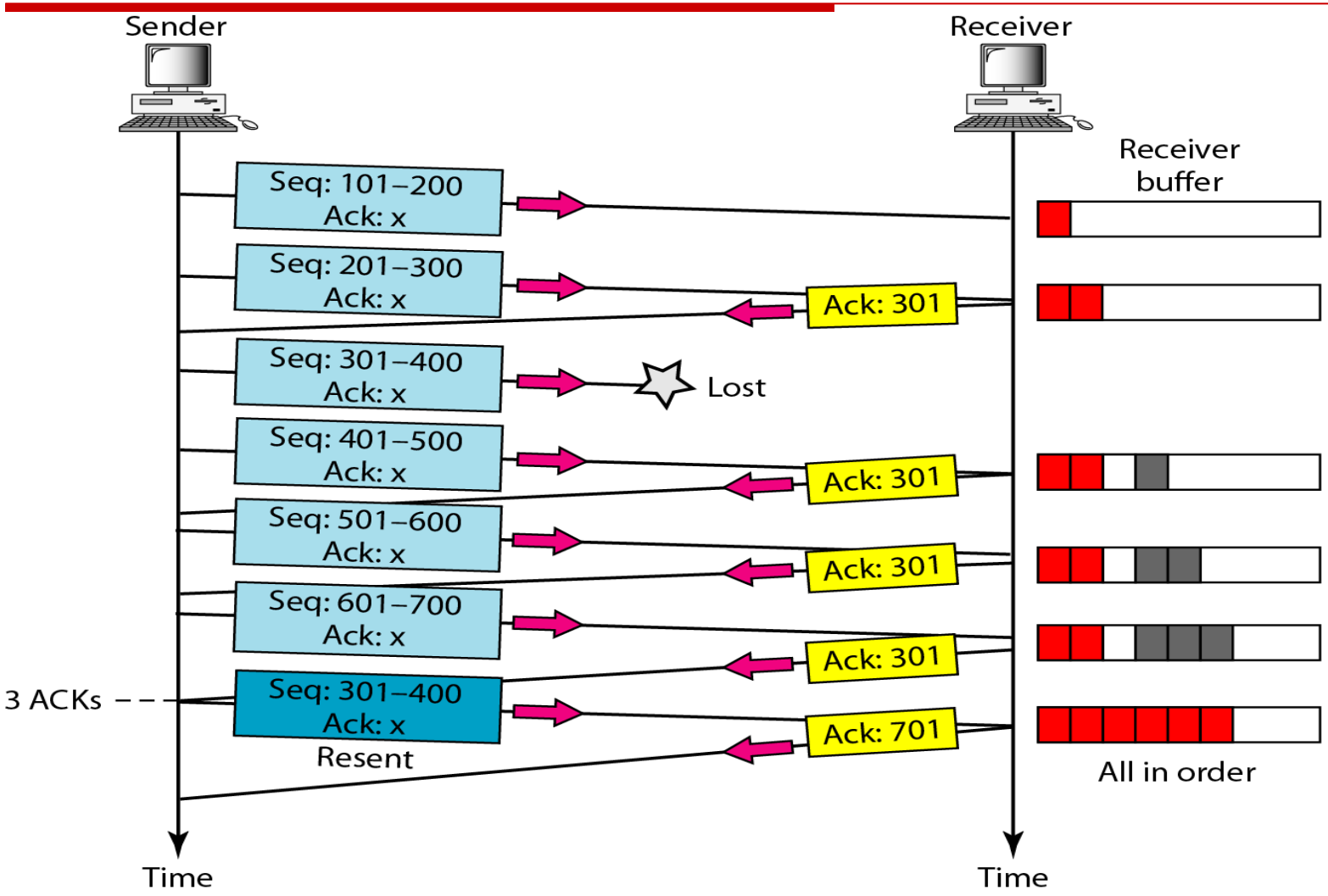
慢启动及AIMD示例：定时器超时



快速重传机制

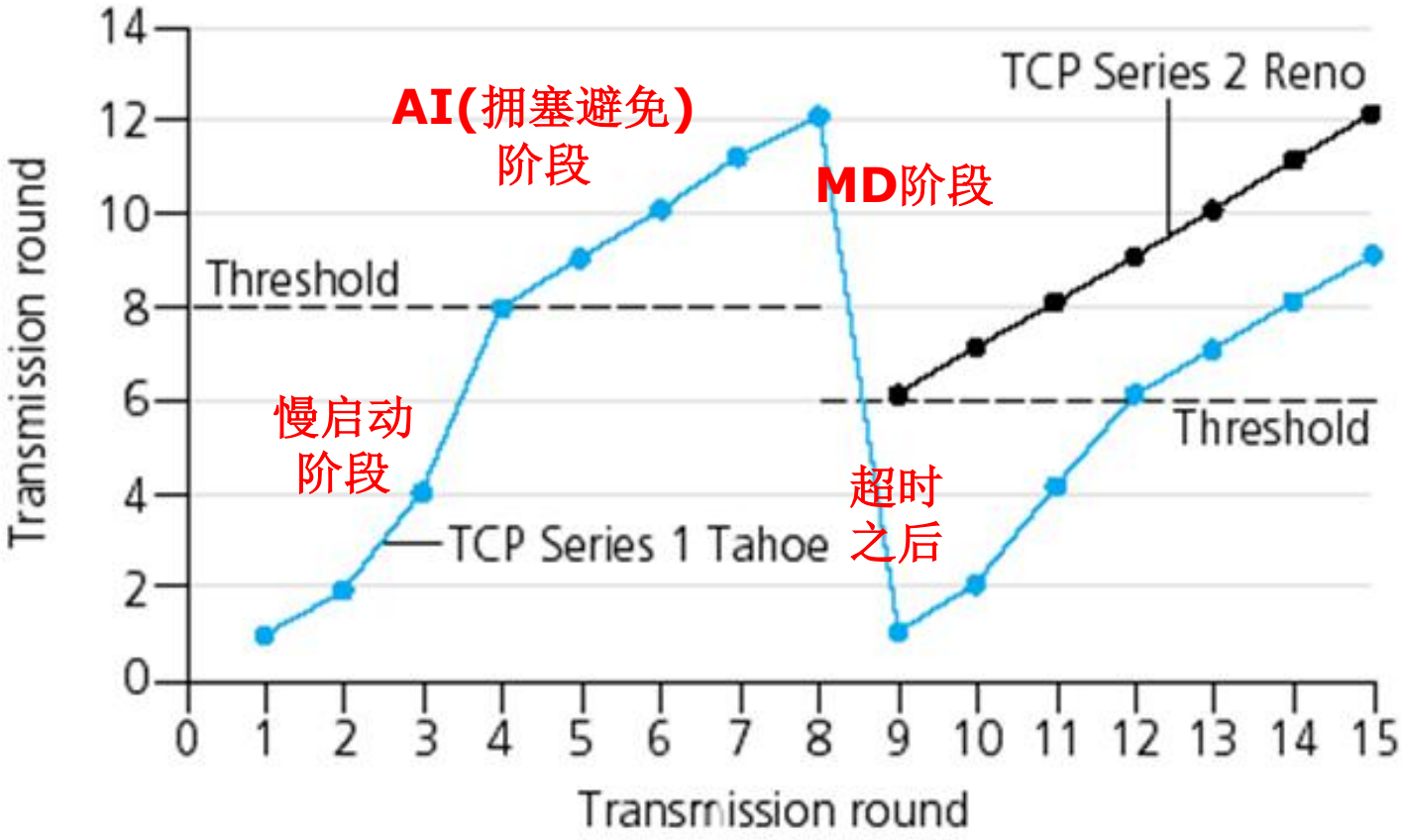
- 超时重传等待时间过长
  - 传输效率低
- 快速重传：通过收到重复的ACK来发现报文段丢失
  - 在报文段丢失时，发送端往往会收到多个重复的ACK
  - 如果发送端收到3个重复的ACK，则立刻重传对应的报文段，而不必等待定时器超时

快速重传示例


Sender
Receiver
Receiver buffer
Seq: 101–200 Ack: x
Seq: 201–300 Ack: x
Ack: 301
Seq: 301–400 Ack: x
Lost
Seq: 401–500 Ack: x
Ack: 301
Seq: 501–600 Ack: x
Ack: 301
Seq: 601–700 Ack: x
Ack: 301
3 ACKs
Seq: 301–400 Ack: x
Resent
Ack: 701
All in order
Time
Time

TCP拥塞控制：快速回复示例

3个重复ACK之后

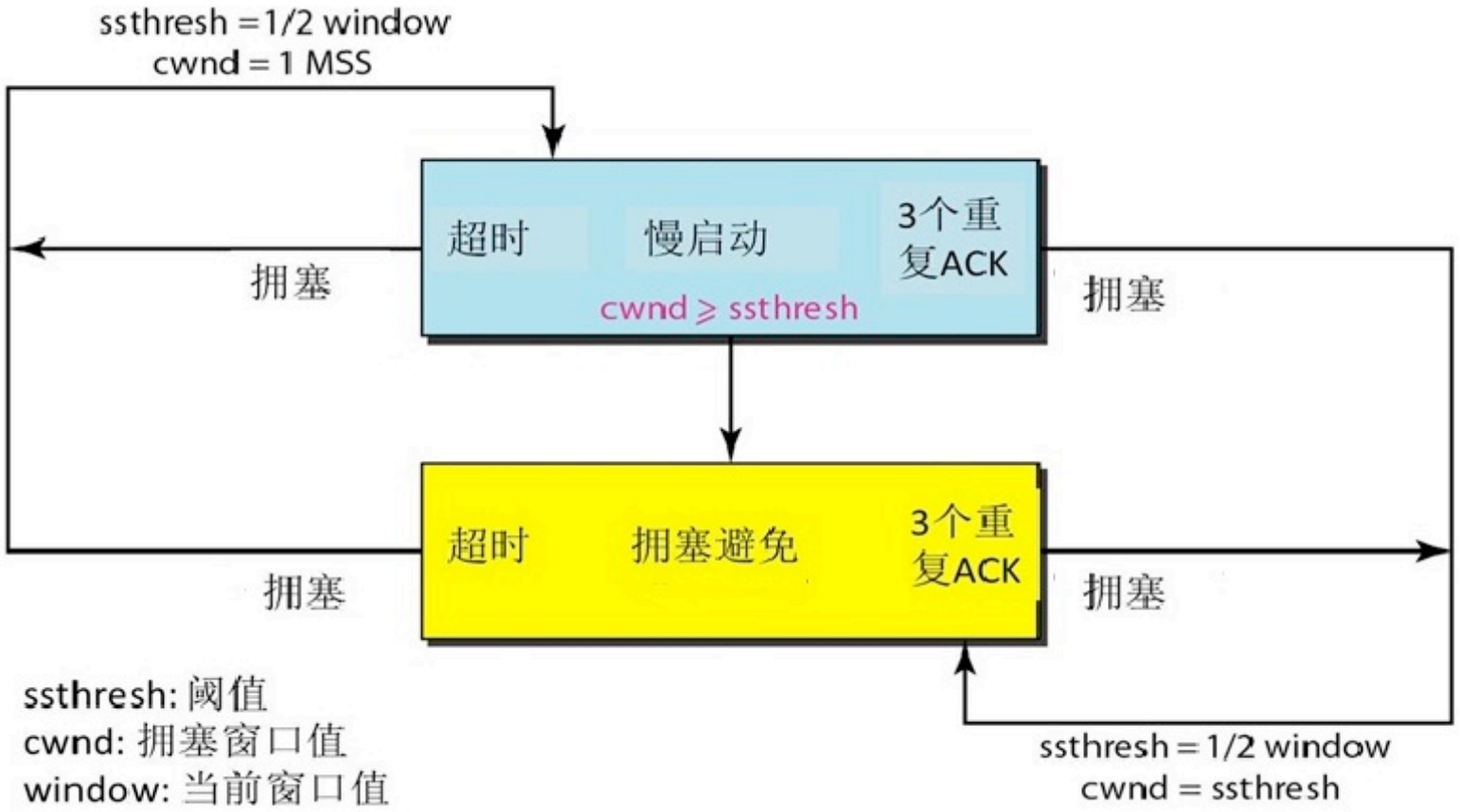


- 阈值：检测到数据丢失时，阈值=cwnd/2

TCP拥塞控制策略小结

- 在拥塞窗口大小cwnd未达到阈值之前，发送端处于**慢启动**阶段，窗口按指数级增长
- 当cwnd达到阈值之后，发送端处于**拥塞避免**阶段，窗口按线性增长
- 当收到**3个重复的ACK**，新阈值=cwnd/2 ，cwnd=新阈值
- 当**定时器超时**，新阈值=cwnd/2 ，cwnd =1

TCP拥塞控制策略


ssthresh =1/2 window
cwnd = 1 MSS
超时
慢启动
3个重复ACK
cwnd ⩾ ssthresh
拥塞
拥塞
超时
拥塞避免
3个重复ACK
拥塞
拥塞
ssthresh: 阈值
cwnd: 拥塞窗口值
window: 当前窗口值
ssthresh = 1/2 window
cwnd = ssthresh

内容提要

- 3.1 传输层的功能及服务
- 3.2 可靠数据传输的原理
- 3.3 无连接传输协议：UDP
- 3.4 面向连接的传输协议：TCP
- 3.5 拥塞控制的主要原理
- *3.6 传输层的安全隐患*

对传输层的攻击

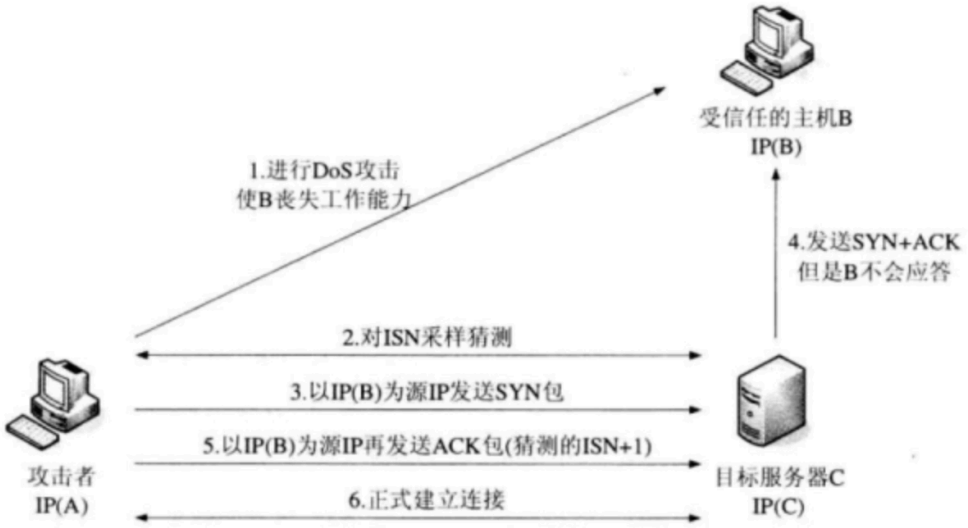
- 端口扫描
- 基于TCP序列号预测的攻击
- SYN洪泛攻击
- TCP会话劫持
- LAND攻击
- UDP洪泛攻击

端口扫描

- 依次扫描常用端口或指定端口，尝试进行连接，查看其端口是否开放（活动端口）
- 扫描分类
  - 全TCP连接（容易被发现）
  - 半打开式扫描（SYN扫描）
  - FIN扫描
  - 第三方扫描（代理扫描）

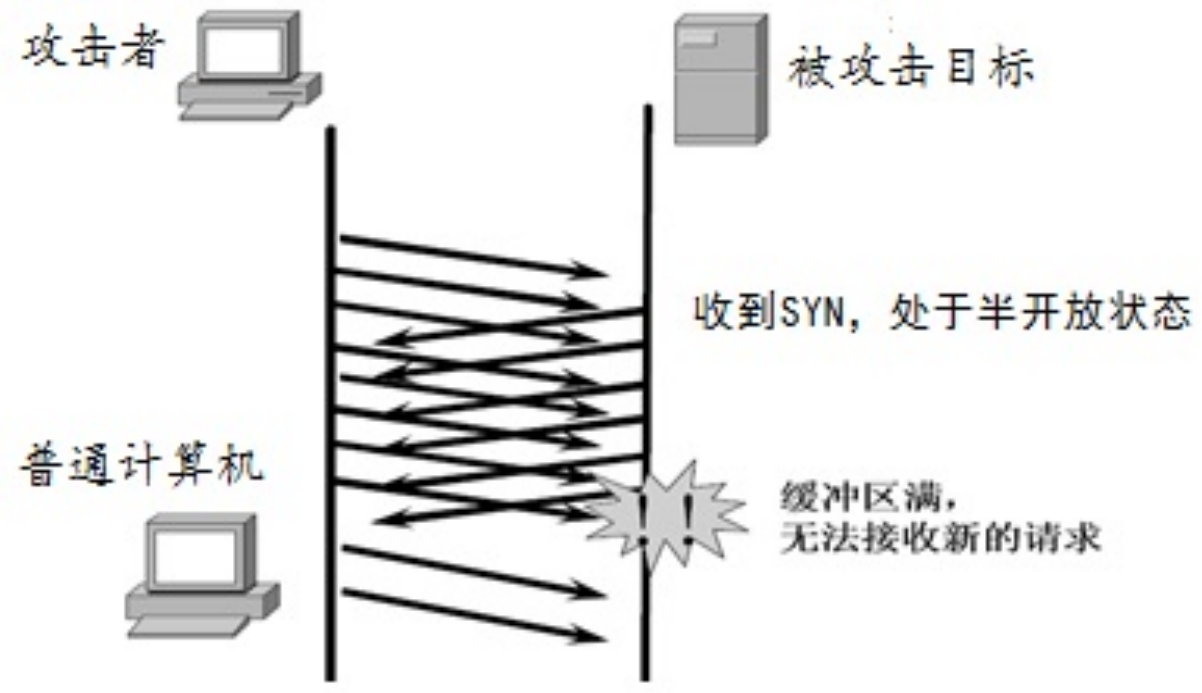
基于TCP序列号预测的攻击

- 信任主机之间的地址欺骗
  - 攻击者预测TCP连接的初始序列号(ISN)
  - 通过测量来回传输路径，得到进攻主机到目标主机之间数据包传送的RTT
  - 预测下一次连接的ISN



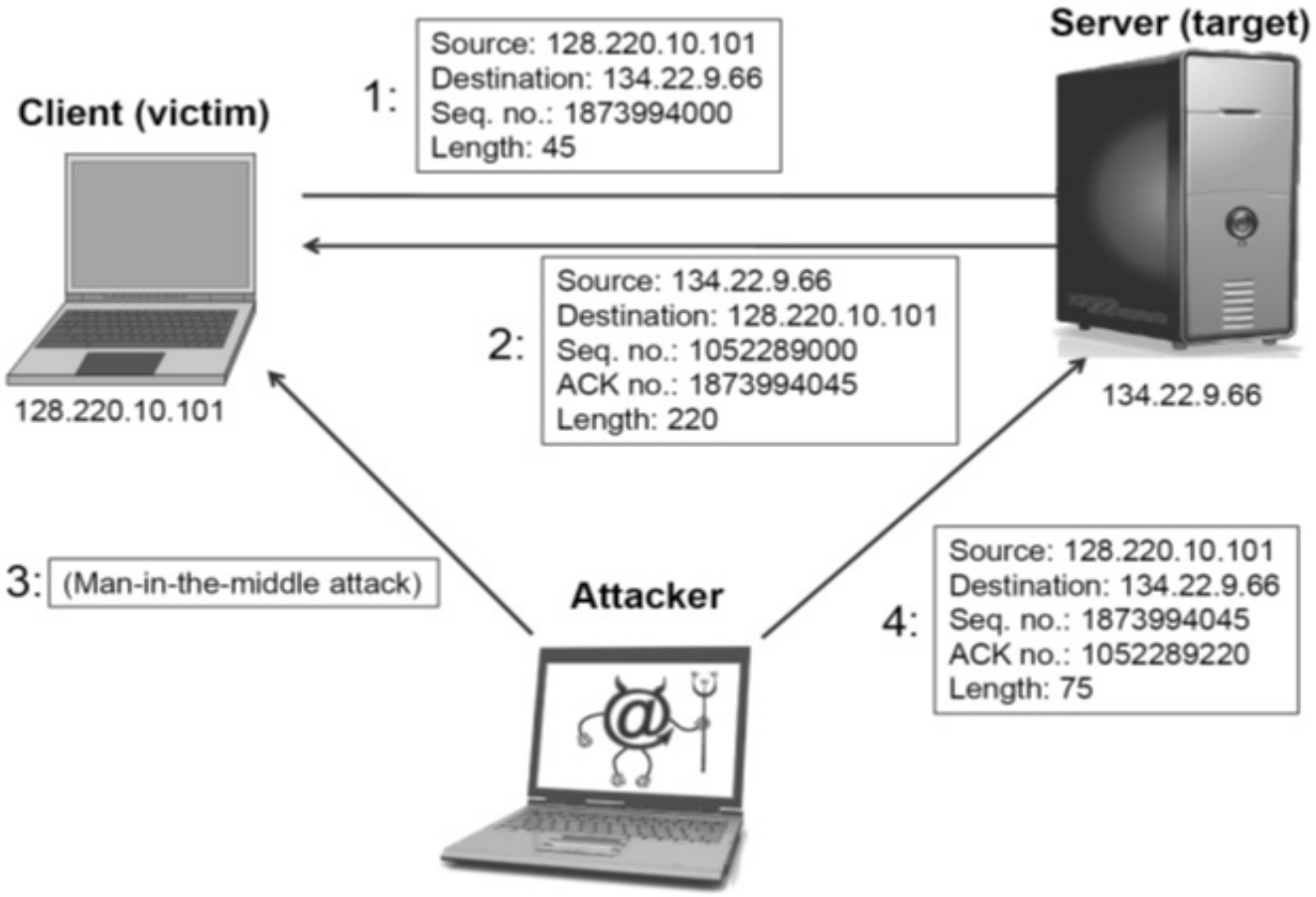
SYN洪泛DOS攻击

- SYN洪泛：发送大量SYN报文段，耗尽TCP服务器的资源（CPU、内存、网络带宽...）
- DOS：拒绝服务



TCP会话劫持

- 结合嗅探、欺骗技术，采用中间人攻击方式
- 窃听、复制或修改数据



LAND攻击

- LAND：LAN拒绝（LAN Denial）
- 伪造源IP地址/端口号为目标主机的IP地址/端口号，发送SYN报文段给目标主机
  - 源IP地址=目的IP地址
- 拒绝服务：导致空连接，消耗目标主机资源

UDP洪泛攻击

- 随机向目标主机（如域名服务器）的各个UDP端口发送大量数据报
- 耗尽目标主机的CPU或内存，最终导致目标主机瘫痪

本章小结

- 传输层服务的主要原理
  - 多路复用/多路分解
  - 可靠的数据传输
  - 流量控制
- 因特网的传输层协议要点
  - UDP：服务、报头结构
  - TCP：服务、报头结构、连接管理、可靠传输原理、流量控制原理
  - 拥塞控制概念及TCP的拥塞控制原理

报头结构不用背，掌握主要字段的功能

版权说明

- 本讲义中有部分图片来源于下列教材所附讲义：
  - Jim Kurose, Keith Ross，Computer Networking: A Top Down Approach, 7th Edition，Pearson/Addison Wesley，April 2016，引用时标记为*[Kurose]*;
  - Andrew S. Tanenbaum, Computer Networks, Fourth Edition, 清华大学出版社（影印版），2004，引用时标记为*[Tanenbaum]*;
  - 谢希仁，计算机网络，第五版，电子工业出版社，2008年1月,引用时标记为*[谢]*;
  - Behrouz A. Forouzan，Data Communications and Networking，Fourth Edition, McGraw-Hill Higher Education, 2007年1月，引用时标记为*[Forouzan]*
  - “Data and Computer Communications”, William Stallings, 引用时标记为*[Stallings]*

本章勘误表(1)

页码	位置	原文	更正
70	3.1标题下第9行	发送方	发送进程
72	倒数第3行	校验和方法	16位校验和
74	倒数第2条消息	ACK(3)	ACK(2)
74	倒数第8行	其要性	其必要性
79	标题4下面第2行	即序号出错包之后	即序号在出错包之后
79	标题4下面第5行	序号在的正确到达的数据包	序号超前的正确到达的数据包
81	表中第5行	简单邮件传输协议	简单文件传输协议
81	2 标题下第3行	而且在默认情况下，UDP只对数据报头进行校验，不对数据报包含的应用层消息进行校验	如果UDP的校验和字段值为0，则表示没有进行校验

本章勘误表(2)

页码	位置	原文	更正
82	图3-18上数第4行	在默认情况下，UDP对伪报头和数据报头两部分进行校验	UDP对伪报头和整个UDP数据包进行校验
82	图3-18	默认校验区间	去掉，参见本讲义p57图
83	图3-19	数据2：请问现	数据2：！请问现
84	图3-20	SIN	FIN
85	倒数第3行	如图3-23所示	如图3-22所示
86	图3-22	连接确认（SYN位=1,ACK位=1,发送序号=X+1,接收序号=Y+1）	参见本讲义p69图

本章勘误表(3)

页码	位置	原文	更正
86	图3-23	连接确认	连接响应，参见本讲义p70图
86	图3-23	确认（SYN位=1,ACK位=1,发送序号=X+1,接收序号=Y+1）	参见本讲义p70图
88	图3-27	原图中虚线和实线分不清	参见本讲义p72图
91	第4行	表明RTT值对历史值的依赖程序	表明RTT值对历史值的依赖程度
91	倒数第5行	因此支持捎带应答	因此支持捎带确认
92	图3-31	Ack:14001	Ack:4001
97	图3-38	$2^3=4$	$2^2=4$