计算机网络原理

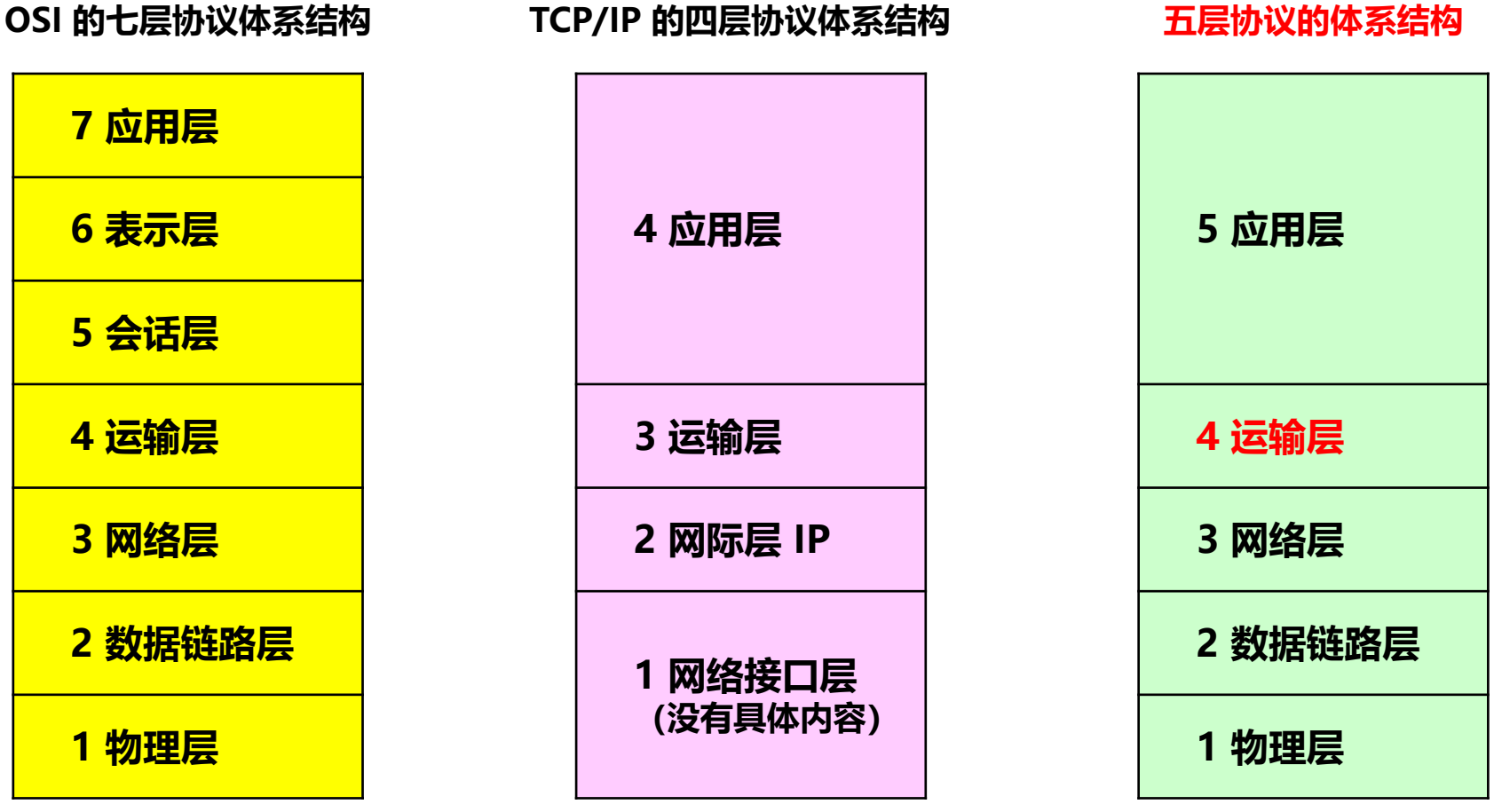
第5章：运输层

阮晓龙

13938213680 / rxl@hactcm.edu.cn
http://network.xg.hactcm.edu.cn

河南中医药大学信息管理与信息系统教研室
信息技术学院网络与信息系统科研工作室
河南中医药大学医疗健康信息工程技术研究所

2022.3


OSI 的七层协议体系结构
7 应用层
6 表示层
5 会话层
4 运输层
3 网络层
2 数据链路层
1 物理层
TCP/IP 的四层协议体系结构
4 应用层
3 运输层
2 网际层 IP
1 网络接口层
（没有具体内容）
五层协议的体系结构
5 应用层
4 运输层
3 网络层
2 数据链路层
1 物理层

本章教学计划

- 运输层协议概述
- 用户数据报协议 UDP

UDP

- 传输控制协议 TCP
- 可靠传输的工作原理
- TCP 报文段的首部格式
- TCP 可靠传输的实现
- TCP 的流量控制
- TCP 的拥塞控制
- TCP 的运输连接管理

TCP

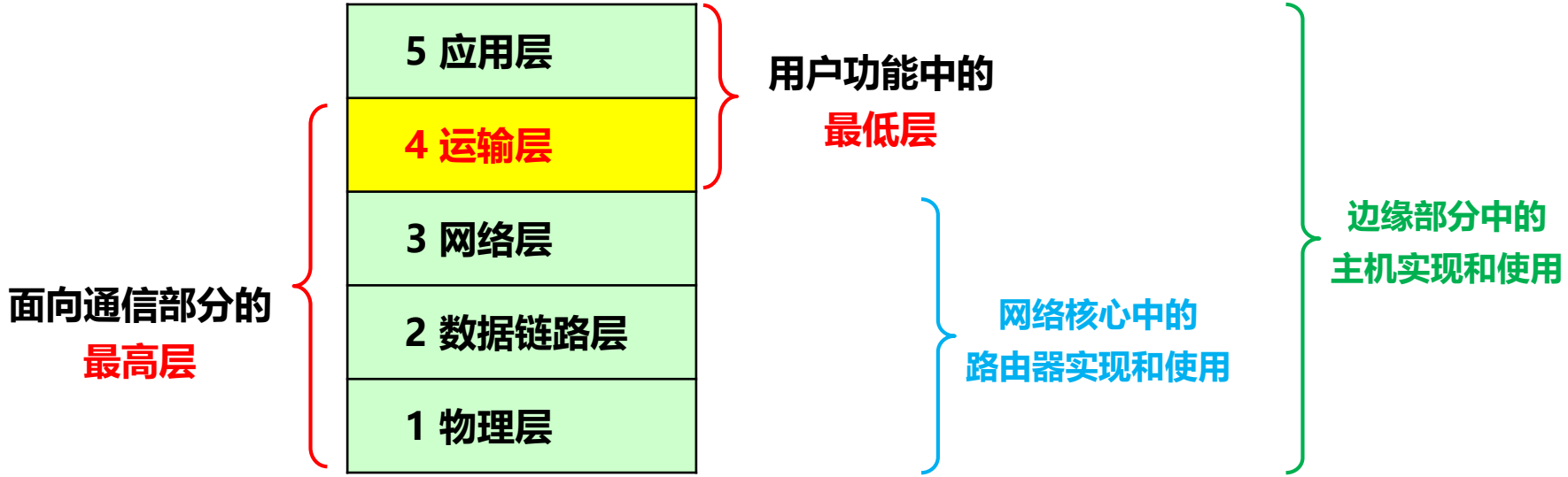
本章教学计划

- 概括介绍运输层协议的特点，以及进程之间的通信和端口等重要概念。
- 讲述UDP和TCP协议。
- 重点讨论较为复杂的TCP协议：
  - TCP协议和可靠传输原理
  - TCP的三个重要问题：滑动窗口、流量控制、拥塞控制
  - TCP连接管理

本章教学计划

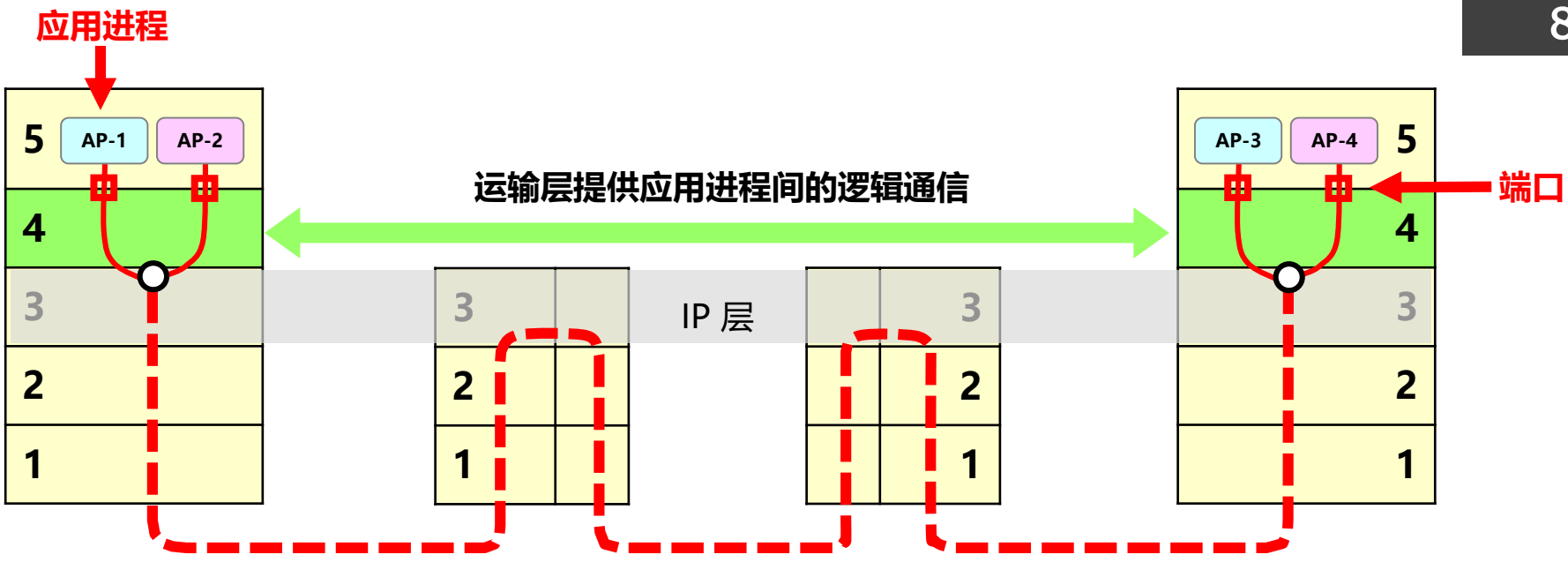
- 运输层是整个网络体系结构中的关键层次之一，本部分的重要概念有：
  - 运输层为相互通信的应用进程提供逻辑通信。
  - 端口和套接字
  - UDP的特点
  - TCP的特点
  - 在不可靠的网络上实现可靠传输的工作原理，以及停止等待协议和ARQ协议
  - TCP的滑动窗口、流量控制、拥塞控制
  - TCP的连接管理

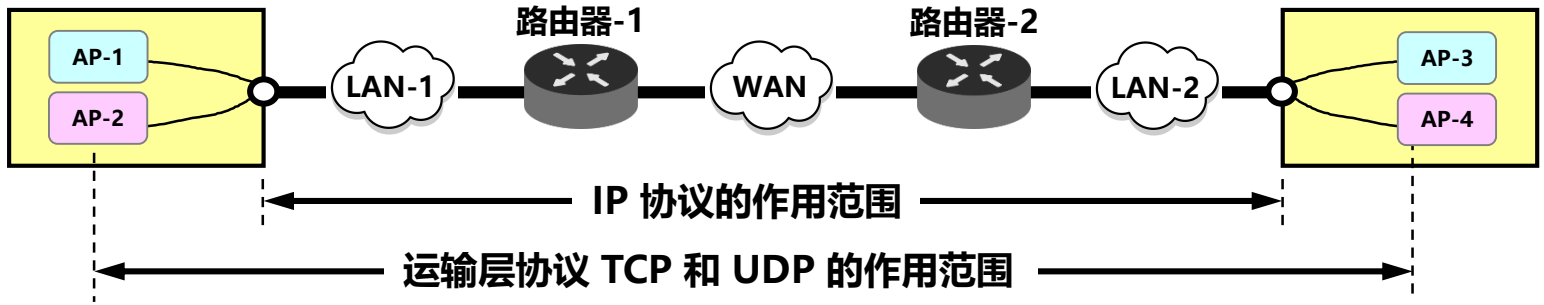
1. 运输层协议概述


5 应用层
4 运输层
3 网络层
2 数据链路层
1 物理层
面向通信部分的
最高层
用户功能中的
最低层
网络核心中的
路由器实现和使用
边缘部分中的
主机实现和使用

1. 运输层协议概述

- 从通信和信息处理的角度看，运输层向它上面的应用层提供通信服务。
  - 属于面向通信部分的最高层，同时也是用户功能中的最低层。
- 当网络边缘部分中的两个主机使用网络核心部分的功能进行端到端通信。
  - 只有位于网络边缘部分的主机的协议栈才有运输层。
  - 网络核心部分中的路由器在转发分组时都只用到下三层的功能。


应用进程
5
AP-1
AP-2
4
3
2
1
运输层提供应用进程间的逻辑通信
IP 层
AP-3
AP-4
端口


AP-1
AP-2
LAN-1
路由器-1
WAN
路由器-2
LAN-2
AP-3
AP-4
IP 协议的作用范围
运输层协议 TCP 和 UDP 的作用范围

1. 运输层协议概述

屏蔽作用

运输层向高层用户屏蔽了下面网络核心的细节，它使应用进程看见的就是好像在两个运输层实体之间有一条端到端的逻辑通信信道。

1. 运输层协议概述

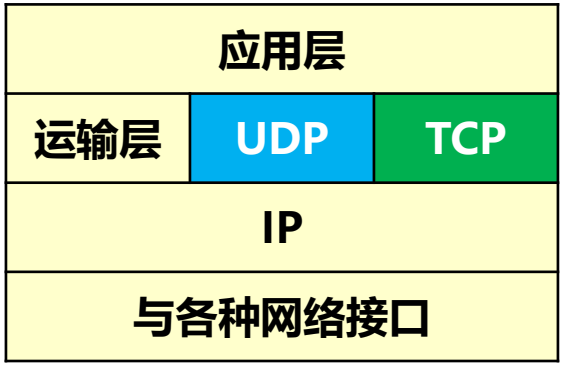
- 运输层的主要作用：
  - 运输层为应用进程之间提供端到端的逻辑通信，网络层为主机之间提供逻辑通信。
  - 运输层要对收到的报文进行差错检测，网络层对IP数据报首部提供首部数据的校验而不检查数据部分。
  - 根据应用程序的不同需求，运输层需要有两种不同的运输协议：
    - 面向连接的TCP
    - 无连接的UDP

1. 运输层协议概述

运输层的两个主要协议

TCP/IP 的运输层有两个主要协议：

- 用户数据报协议 UDP (User Datagram Protocol)
- 传输控制协议 TCP (Transmission Control Protocol)



TCP/IP 体系中的运输层协议

1. 运输层协议概述

- 两种不同的运输协议：
  - 运输层向高层用户屏蔽了其下层网络核心的细节（如网络拓扑、所采用的路由选择协议等），它使应用进程看见的就是好像在两个运输层实体之间有一条端到端的逻辑通信信道。
  - 当运输层采用面向连接的TCP协议时，尽管网络核心是不可靠且只提供尽最大努力交付服务，逻辑通信信道相当于全双工模式的可靠信道。
  - 当运输层采用无连接的UDP协议时，逻辑通信信道是不可靠信道。

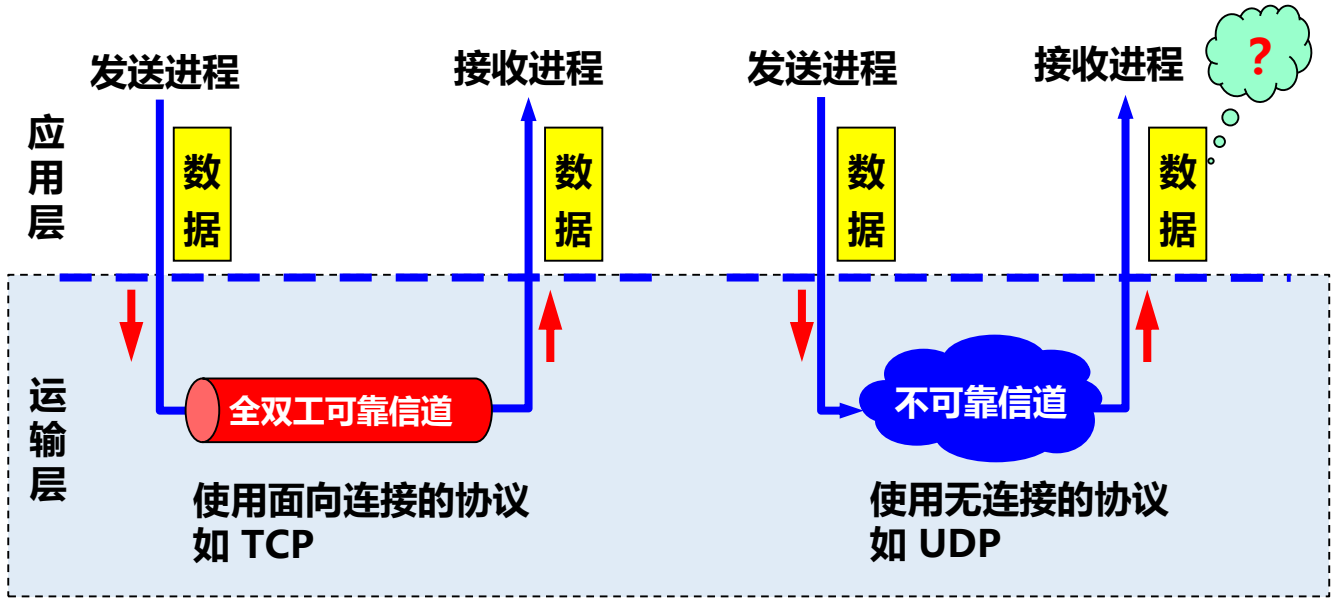
1. 运输层协议概述

- TCP/IP的运输层有两个不同的协议，均为因特网标准。
  - 用户数据报协议 UDP(User Datagram Protocol)，RFC 768
  - 传输控制协议 TCP(Transmission Control Protocol)，RFC 793

- 两个对等运输实体在通信时传送的数据单位叫作运输协议数据单元 TPDU(Transport Protocol Data Unit)。
  - TCP传送的数据单位协议是TCP 报文段 (segment)
  - UDP传送的数据单位协议是UDP 报文或用户数据报 (Datagram)

1. 运输层协议概述

可靠信道与不可靠信道



1. 运输层协议概述

UDP

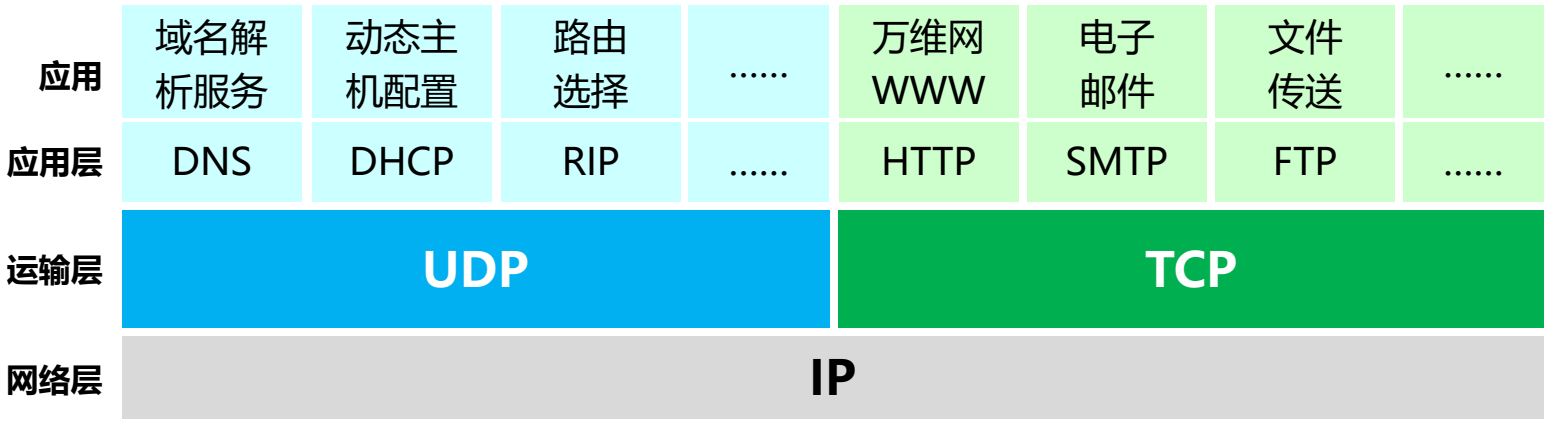
- 传送数据之前不需要先建立连接。
- 收到 UDP 报后，不需要给出任何确认。
- 不提供可靠交付，但是一种最有效的工作方式。

TCP

- 提供可靠的、面向连接的运输服务。
- 不提供广播或多播服务。
- 开销较多。

1. 运输层协议概述

使用 UDP 和 TCP 的典型应用和应用层协议



1. 运输层协议概述

- 两个主机进行通信，实际上就是两个主机中的应用进程互相通信。
  - 应用进程之间的通信又称为端到端的通信。
- 运输层的一个很重要功能就是复用和分用。
  - 应用层不同进程的报文通过不同的端口向下交到运输层，再往下就共用网络层提供的服务。
- 运输层提供应用进程间的逻辑通信。
  - “逻辑通信”的意思是：运输层之间的通信好像是沿水平方向传送数据。但事实上这两个运输层之间并没有一条水平方向的物理连接。

1. 运输层协议概述

- 运输层的重要功能就是复用和分用。
  - 复用：应用层所有的应用进程都可通过运输层再传送到IP层。
  - 分用：运输层从IP层收到数据后必须交付指明的应用进程。

- 这就说明：给应用层的每个应用进程赋予一个非常明确的标志是非常重要的。
- 应用层的应用进程必须要有明确的标识系统。

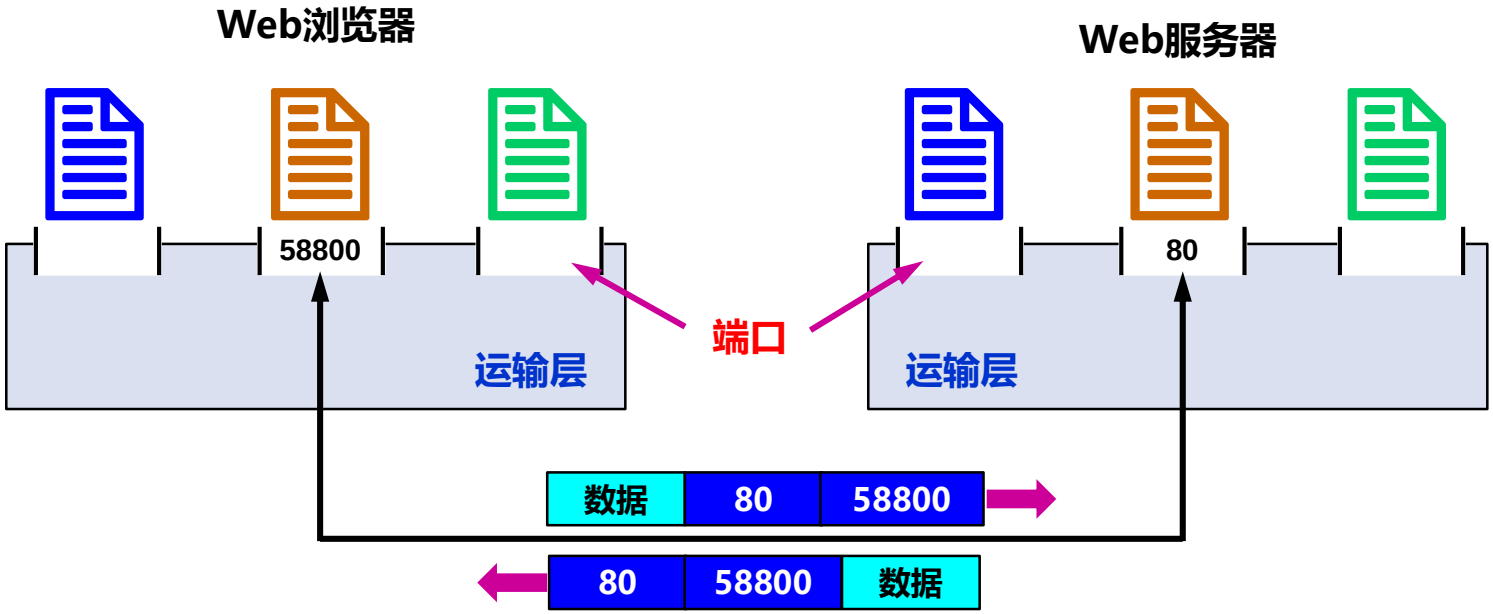
1. 运输层协议概述

- 讨论：应用进程标识的几种可能性。

1. 运输层协议概述

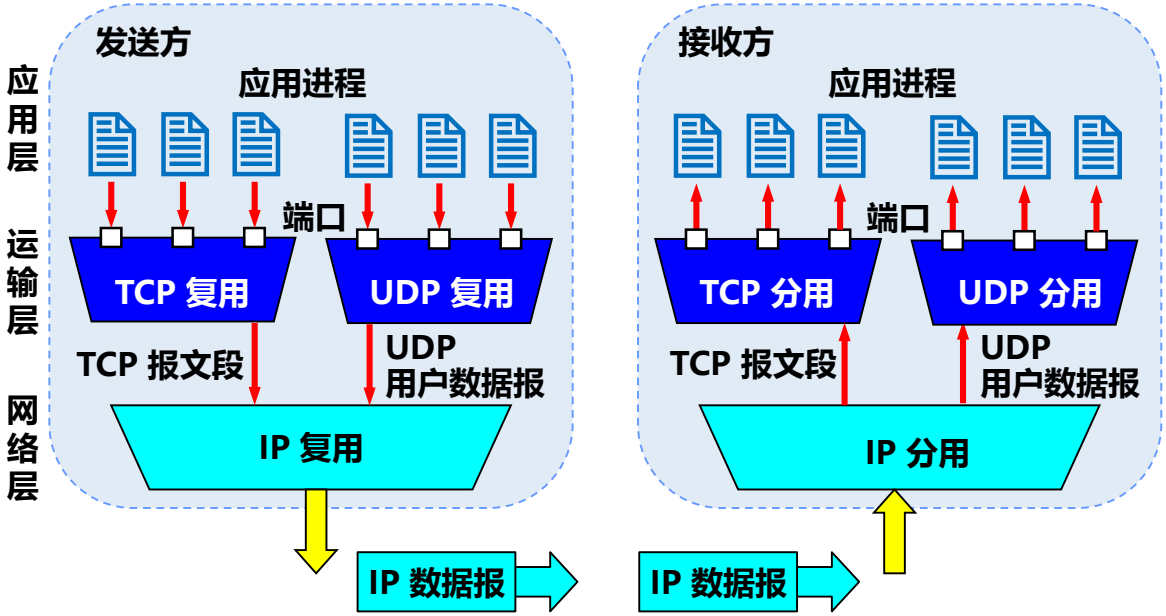
- 解决应用进程标识问题的方法就是在运输层使用协议端口号(protocol port number)，或通常简称为端口(port)。
- 虽然通信的终点是应用进程，但可以把端口想象是通信的终点，因为只要把要传送的报文交到目的主机的某一个合适的目的端口，剩下的工作（即最后交付目的进程）就由 TCP来完成。

1. 运输层协议概述



1. 运输层协议概述

基于端口的复用和分用功能



1. 运输层协议概述

- 注意：软件端口与硬件端口
  - 在协议栈层间的抽象的协议端口是软件端口。
  - 路由器或交换机上的端口是硬件端口。
  - 硬件端口是不同硬件设备进行交互的接口，而软件端口是应用层的各种协议进程与运输实体进行层间交互的一种地址。
  - 协议端口和硬件端口是完全无关的两个概念。

1. 运输层协议概述

- TCP/IP的运输层使用一个16位端口号进行端口标志。
- 端口号只具有本地意义，即端口号只是为了标志本计算机应用层中的各进程。
- 在因特网中不同计算机的相同端口号是没有联系的。
- 16位端口号就表明端口号的取值为0~65535，共计65536个不同端口号。

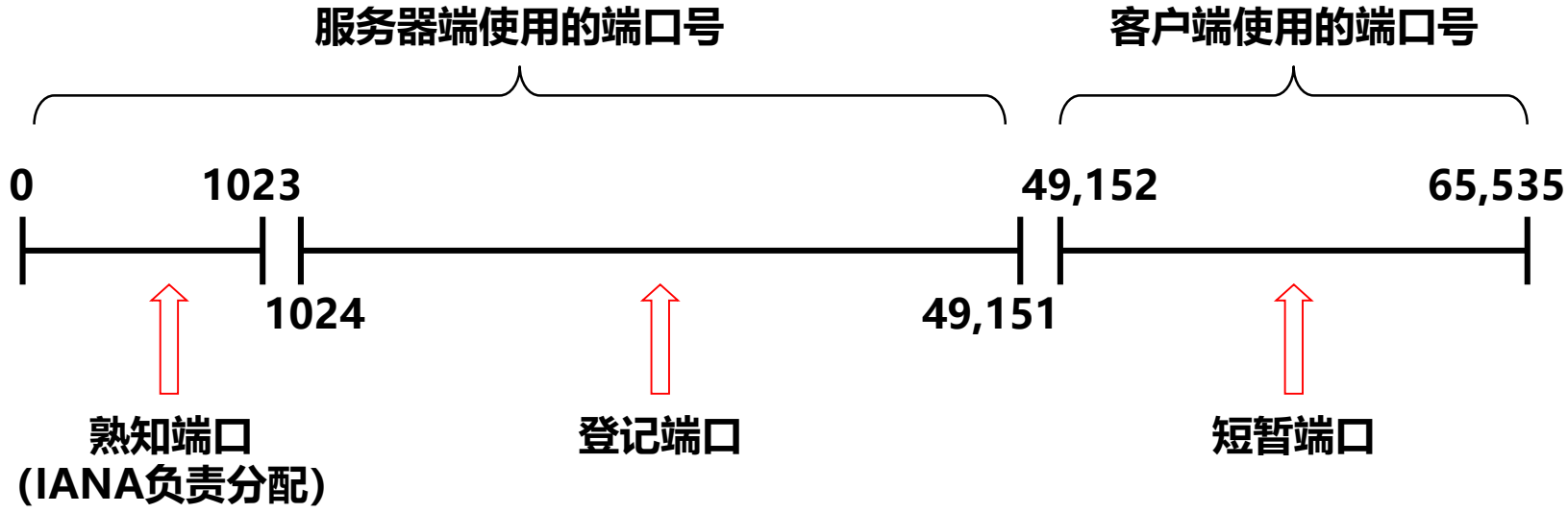
1. 运输层协议概述

- 端口分为两类：
  - 服务器端使用的端口号：
    - 熟知端口号（wellknown port number）或系统端口号，数值为0~1023。IANA把这些端口号指派给TCP/IP最重要的一些应用程序，让所有用户都知道。
    - 登记端口号，数值为1024~49151。这类端口是为没有熟知端口号的应用程序使用的，使用这类端口号必须在IANA按照规定的手续登记，以防止重复。
  - 讨论：我是否可以自由使用服务器端使用的端口号？

1. 运输层协议概述

- 端口分为两类：
  - 客户端使用的端口号：
    - 数值为49152~65535，留给客户进程选择暂时使用。当服务器进程收到客户进程的报文时，就知道了客户进程所使用的动态端口号。
    - 通信结束后，这个端口号可供其他客户进程以后使用。

1. 运输层协议概述


服务器端使用的端口号
客户端使用的端口号
0
1023
1024
49,151
49,152
65,535
熟知端口
(IANA负责分配)
登记端口
短暂端口

1. 运输层协议概述

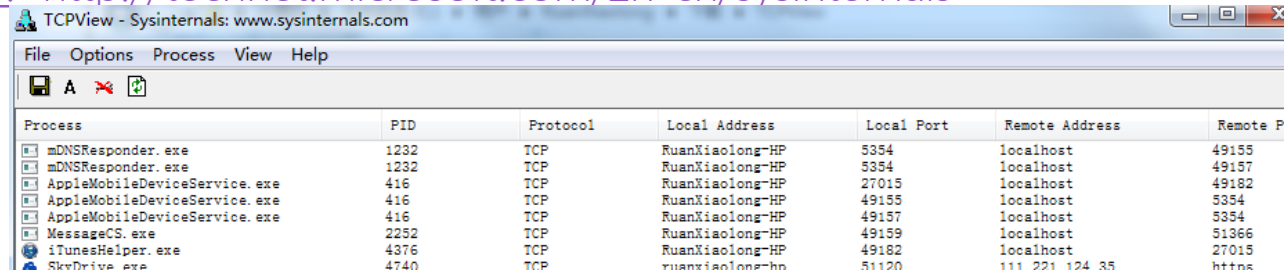
- IANA定义的常见服务端口：
  - http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

常用熟知端口号

应用程序	FTP	SSH	TELNET	SMTP	DNS	TFTP	HTTP	POP	SNMP
端口号	21	22	23	25	53	69	80	110	161

1. 运输层协议概述

- 演示：查看Windows系统中端口占用情况：
  - Windows系统端口：http://support.microsoft.com/kb/832017
  - 通过控制台命令netstat查看端口占用情况
    - Netstat -an
    - Netstat –ano|findstr 80
  - 使用TCPView查看端口占用情况
    - 官网：http://technet.microsoft.com/zh-cn/sysinternals



1. 运输层协议概述

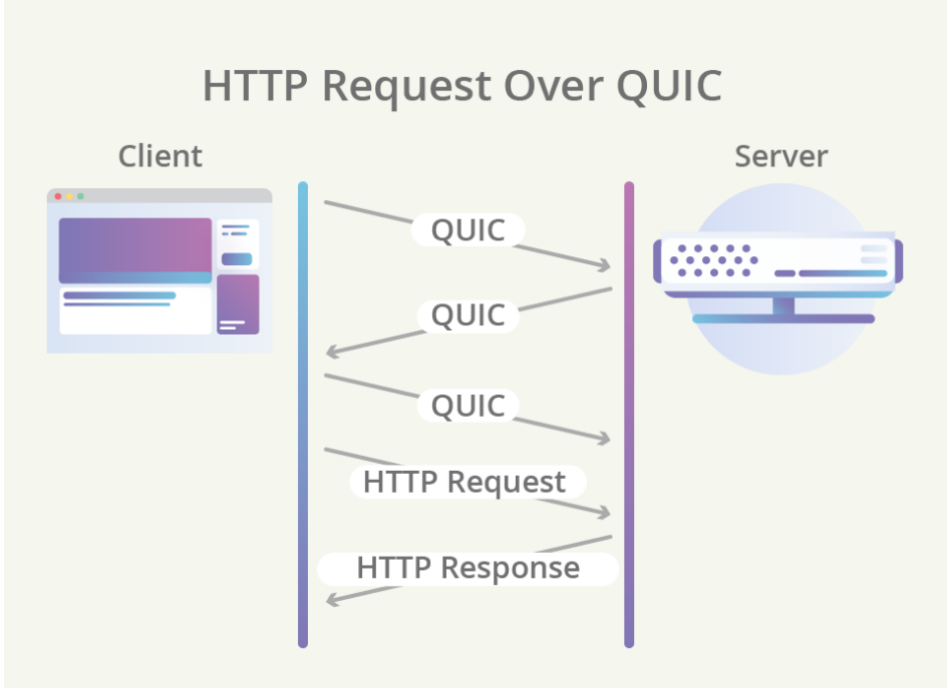
- 演示：查看Linux系统中端口占用情况：
  - Linux系统端口：/etc/services

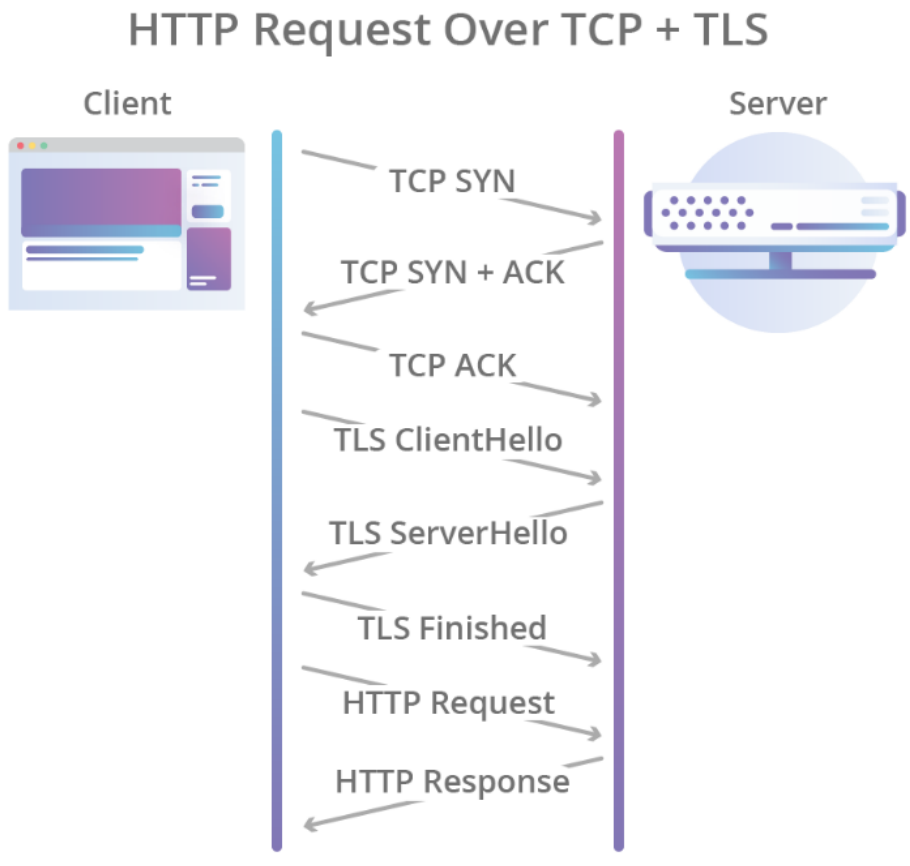
```
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
tacacs          49/tcp                          # Login Host Protocol (TACACS)
tacacs          49/udp
re-mail-ck      50/tcp                          # Remote Mail Checking Protocol
re-mail-ck      50/udp
domain          53/tcp                          # Domain Name Server
domain          53/udp
```

  - 使用Netstat命令查看活动端口。

2. 用户数据报协议 UDP

- UDP只在IP数据报服务之上增加了很少一点的功能：
  - 使用端口实现复用和分用
  - 实现差错检测
- UDP用户数据报提供不可靠的交付，但UDP在某些方面有其特殊的优点。


HTTP Request Over QUIC
Client
Server
QUIC
QUIC
QUIC
HTTP Request
HTTP Response

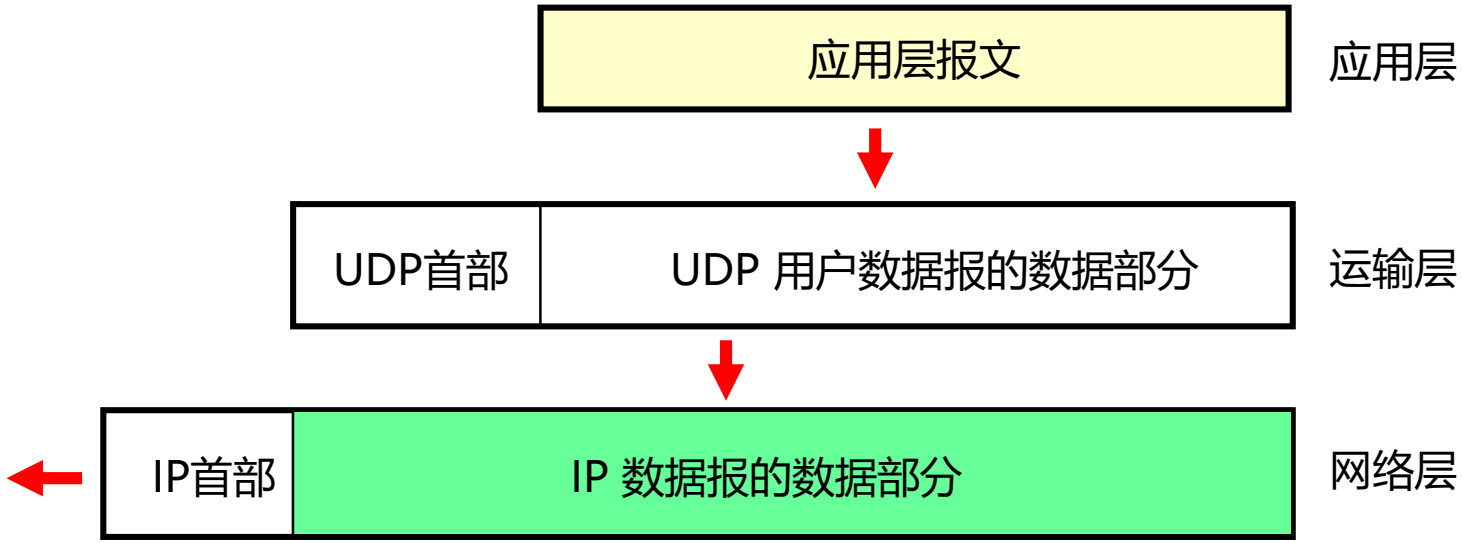

HTTP Request Over TCP + TLS
Client
Server
TCP SYN
TCP SYN + ACK
TCP ACK
TLS ClientHello
TLS ServerHello
TLS Finished
HTTP Request
HTTP Response

2. 用户数据报协议 UDP

- UDP的主要特点：
  - 无连接。即发送数据之前不需要建立连接。
  - 使用尽最大努力交付。即不保证可靠交付，同时也不使用拥塞控制。
  - 面向报文。
    - 应用层交下来的报文，UDP在添加首部后就直接交付给网络层，不做拆分合并，而是保留报文的边界。
    - 对于收到的UDP报文，去除首部后直接提交给应用层。
  - 没有拥塞控制。
    - 网络出现拥塞也不会使源主机的发送速率降低，很适合多媒体通信的要求。
  - 支持一对一、一对多、多对一和多对多的交互通信。
  - 首部开销小。只有8个字节。

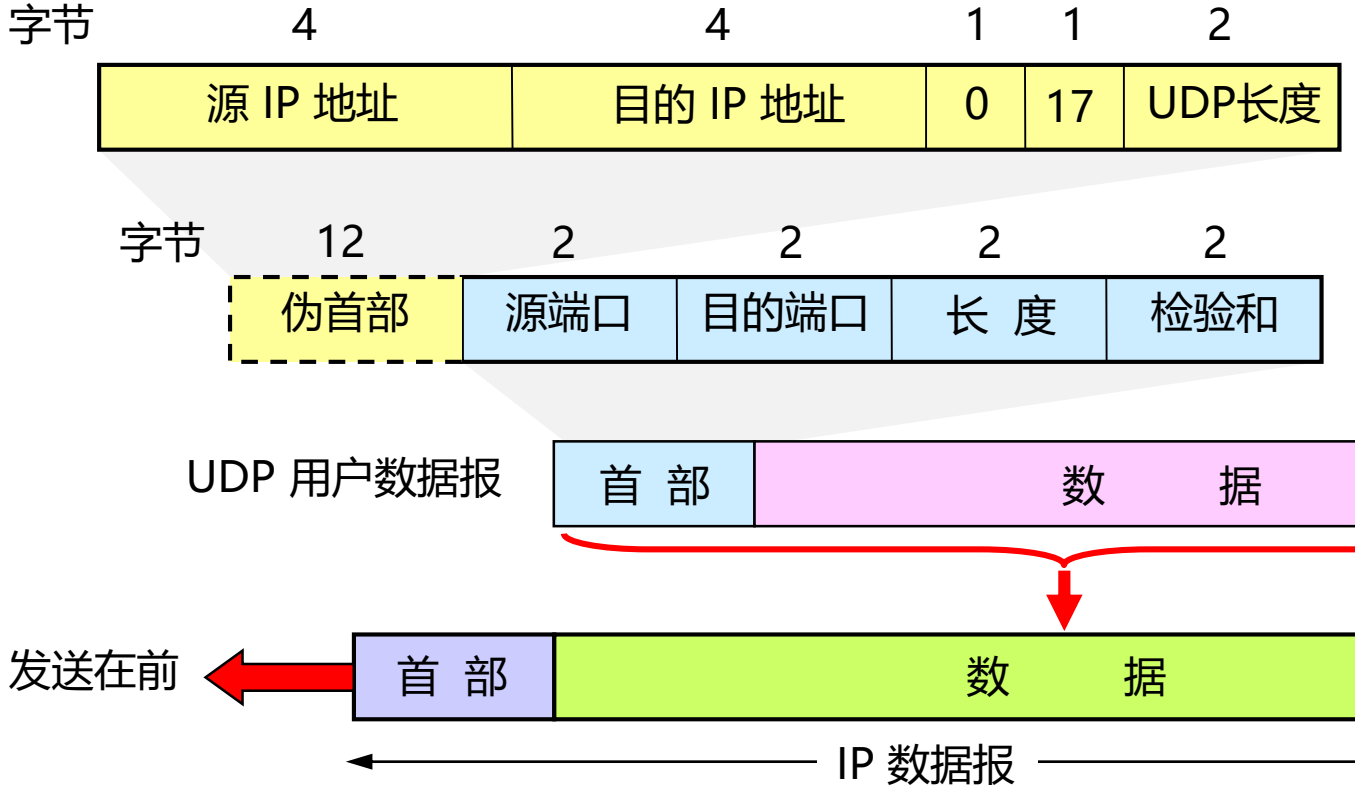
2. 用户数据报协议 UDP

UDP 是面向报文的



- 发送方 UDP 对应用层交下来的报文，既不合并，也不拆分，按照样发送。
- 接收方 UDP 对 IP 层交上来的 UDP 用户数据报，去除首部后就原封不动地交付上层的应用进程，一次交付一个完整的报文。

2. 用户数据报协议 UDP



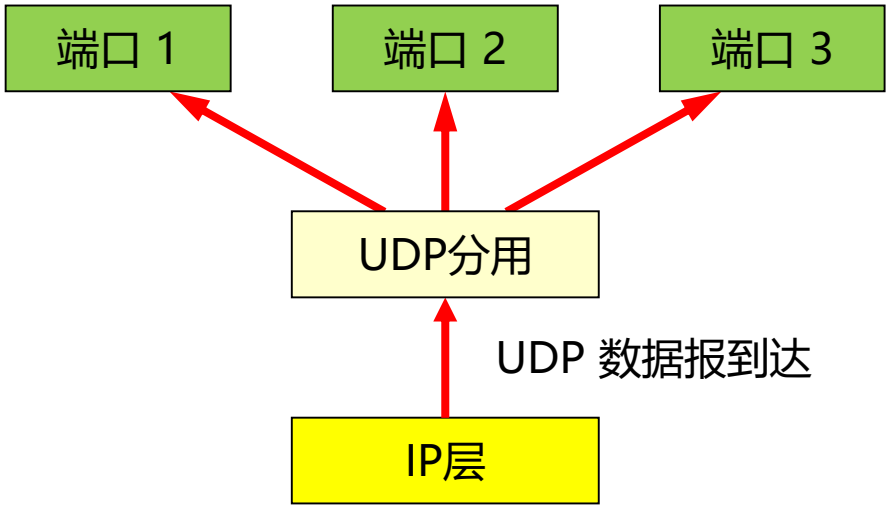
- ✓ 用户数据报 UDP 有两个字段：数据字段和首部字段。
- ✓ 首部字段有 8 个字节，由 4 个字段组成，每个字段都是 2 个字节。

2. 用户数据报协议 UDP

- 用户数据报UDP有两个字段：数据字段和首部字段。
- 首部字段很简单，只有8个字节，由四个字段组成，每个字段的长度都是两个字节。
- 各字段意义为：
  - 源端口：源端口号。在需要对方回信时选用。不需要时可用全0。
  - 目的端口：目的端口号。在终点交付报文时必须要使用到。
  - 长度：UDP用户数据报的长度，最小值为8。
  - 检验和：检测UDP用户数据报在传输时是否有错。

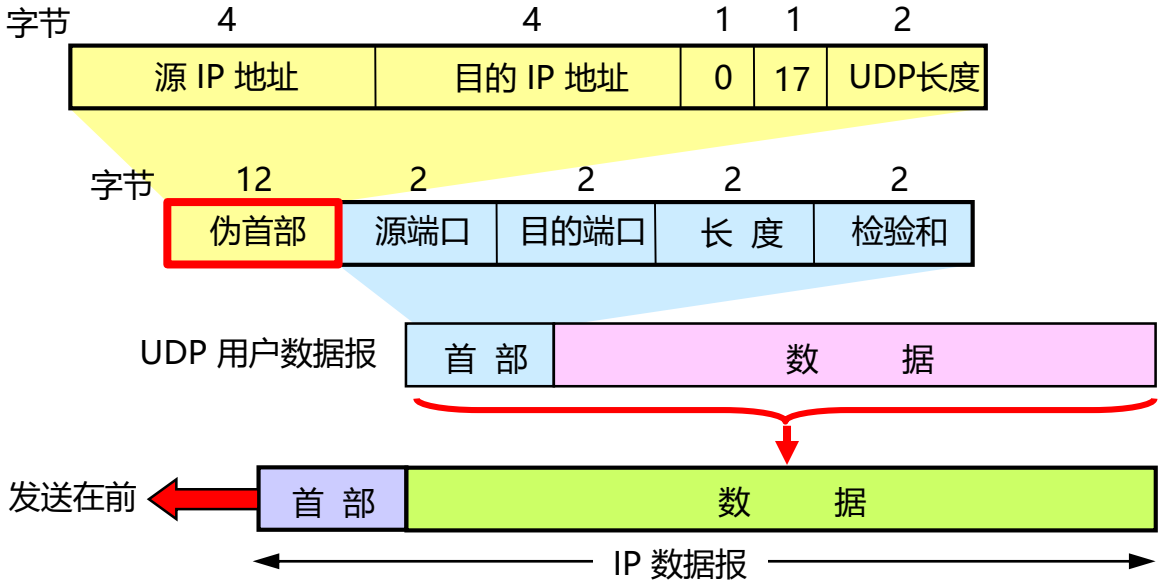
2. 用户数据报协议 UDP

- UDP 基于端口的分用
  - 当运输层从IP层收到UDP数据报时，就根据首部中的目的端口，把UDP数据报通过相应的端口，上交到相应的应用进程。



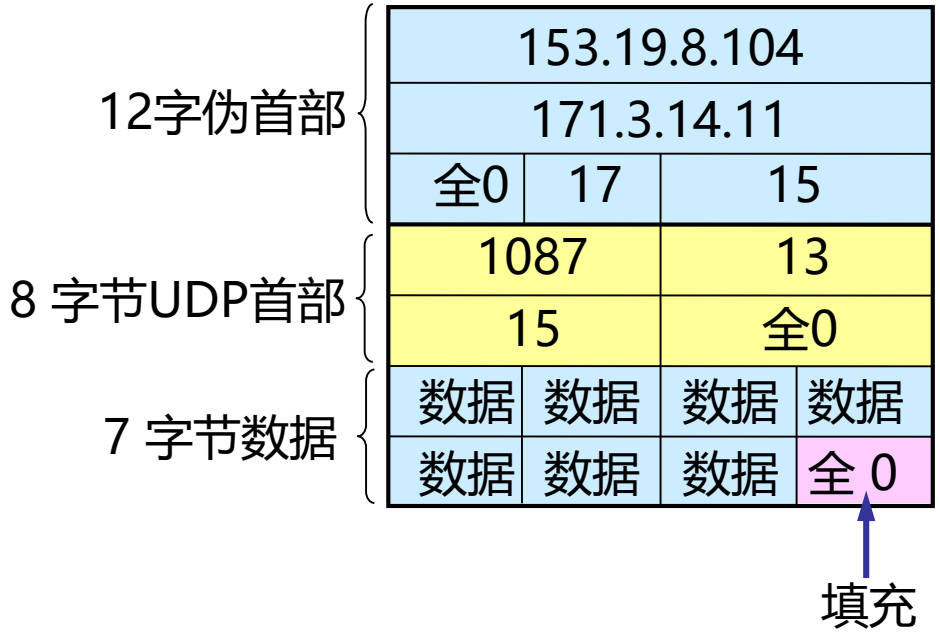
2. 用户数据报协议 UDP

- 在计算校验和时，在UDP用户数据报之前增加12个字节的伪首部。
  - 只在计算时使用，既不向上也不向下传送，仅为了计算校验和。

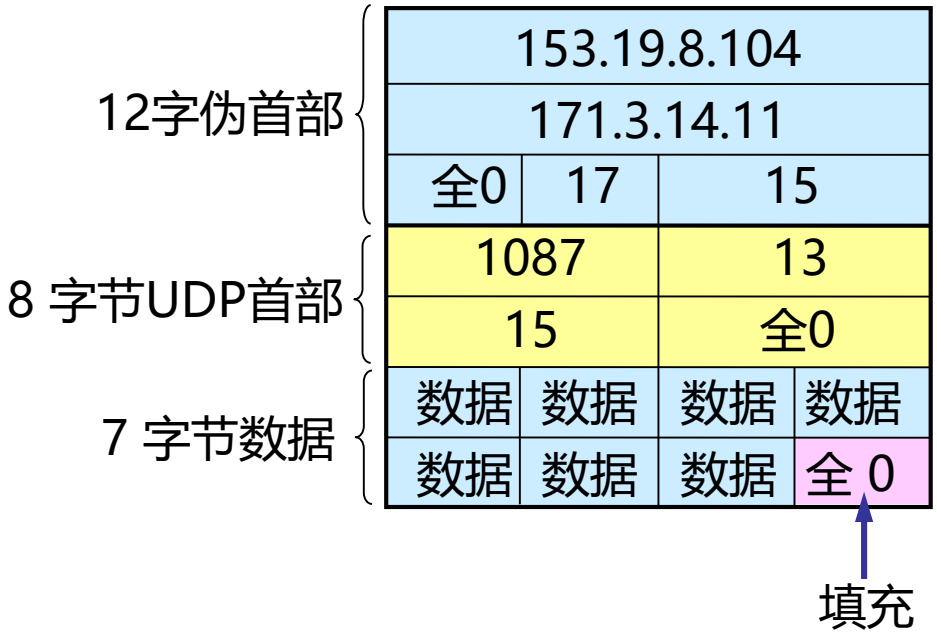


2. 用户数据报协议 UDP

计算UDP校验和

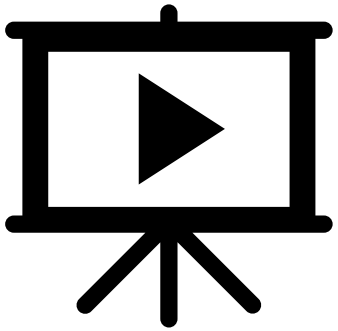


2. 用户数据报协议 UDP



10011001 00010011 → 153.19
00001000 01101000 → 8.104
10101011 00000011 → 171.3
00001110 00001011 → 14.11
00000000 00010001 → 0 和 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0 (检验和)
01010100 01000101 → 数据
01010011 01010100 → 数据
01001001 01001110 → 数据
01000111 00000000 → 数据和0

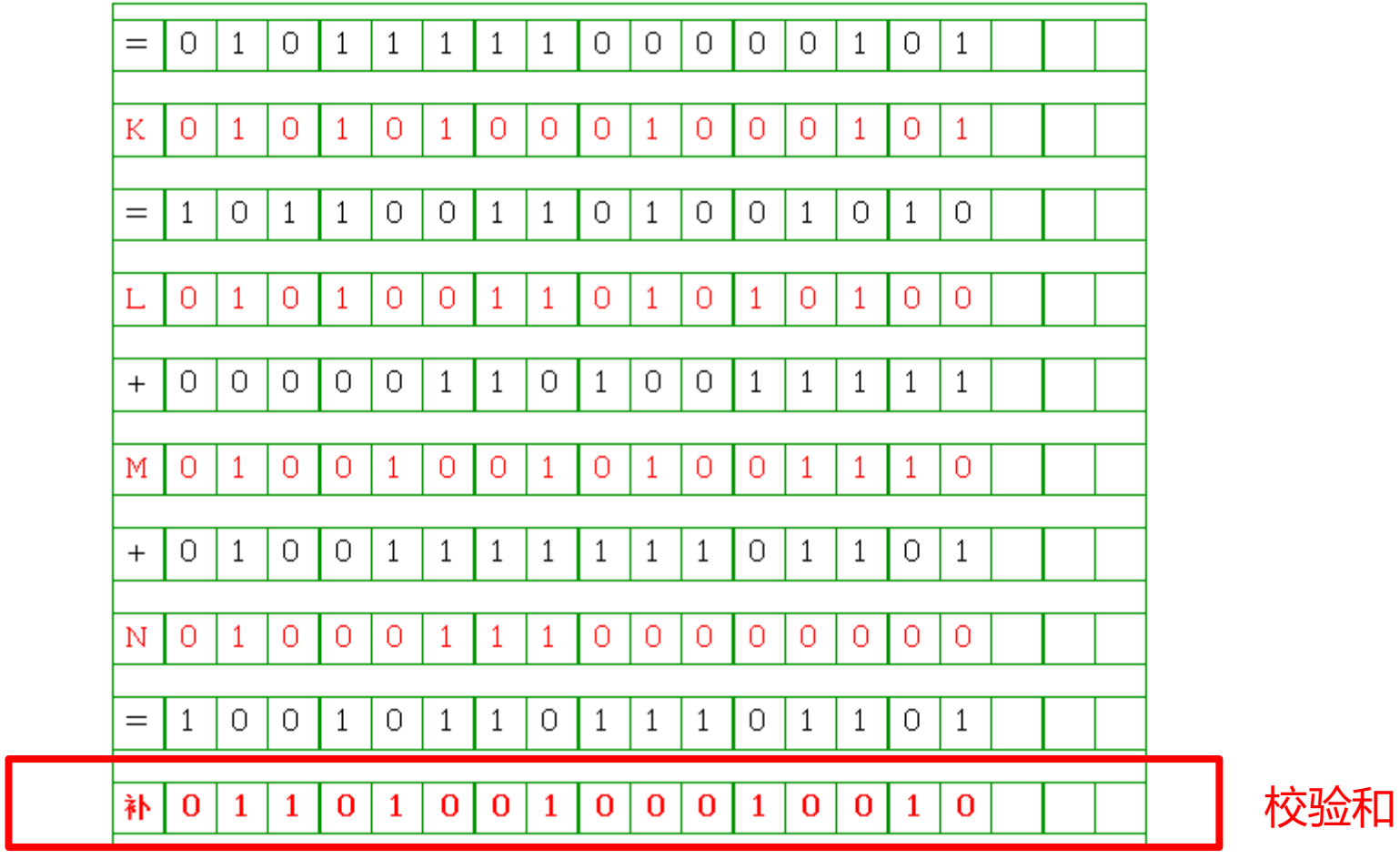
2. 用户数据报协议 UDP



UDP检验和的计算

A	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1			
B	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0	0			
=	1	0	1	0	0	0	0	1	0	1	1	1	1	0	1	1			
C	1	0	1	0	1	0	1	1	0	0	0	0	0	0	1	1			
+	0	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1			
D	0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	1			
=	0	1	0	1	1	0	1	0	1	0	0	0	1	0	1	0			
E	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1			
=	0	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1			
F	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1			
=	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0			

=	0	1	0	1	1	0	1	0	1	0	0	1	1	0	1	1			
F	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1			
=	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0			
G	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1			
=	0	1	0	1	1	1	1	0	1	1	1	0	1	0	0	1			
H	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1			
=	0	1	0	1	1	1	1	0	1	1	1	1	0	1	1	0			
I	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1			
=	0	1	0	1	1	1	1	1	0	0	0	0	0	1	0	1			
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
=	0	1	0	1	1	1	1	1	0	0	0	0	0	1	0	1			


=
0 1 0 1 1 1 1 1 0 0 0 0 0 1 0 1
K
0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1
=
1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 0
L
0 1 0 1 0 0 1 1 0 1 0 1 0 1 0 0
+
0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1
M
0 1 0 0 1 0 0 1 0 1 0 0 1 1 1 0
+
0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1
N
0 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0
=
1 0 0 1 0 1 1 0 1 1 1 0 1 1 0 1
补
0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0
校验和

2. 用户数据报协议 UDP

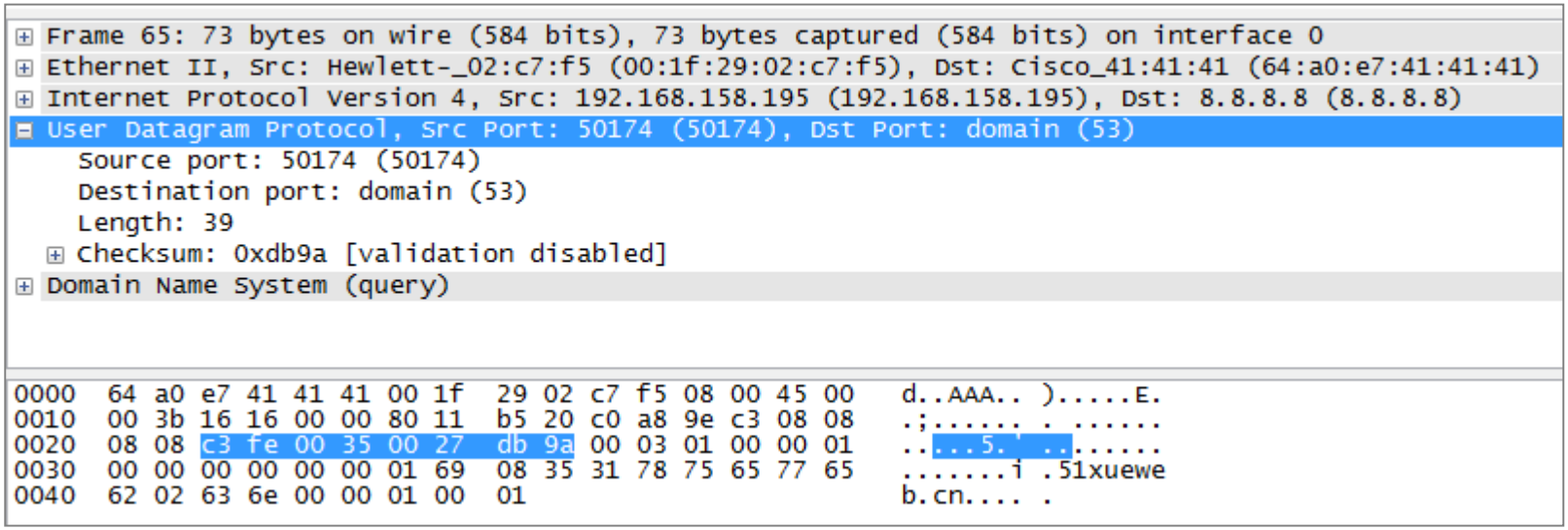
- 二进制反码运算求和的计算方法：

  0+0=1，0+1=1，1+1=0且进位1

- 使用二进制反码运算求和的地方：《计算机网络(第8版)》
  - IP数据报的首部校验和，P139
  - ICMP报文的校验和，P146
  - UDP校验和，P218
  - TCP校验和，P226
    - 在TCP/IP中，除了FCS使用CRC算法外，基本上所有校验都是通过二进制反码运算求和进行的。

2. 用户数据报协议 UDP

- 实例：UDP校验和的计算



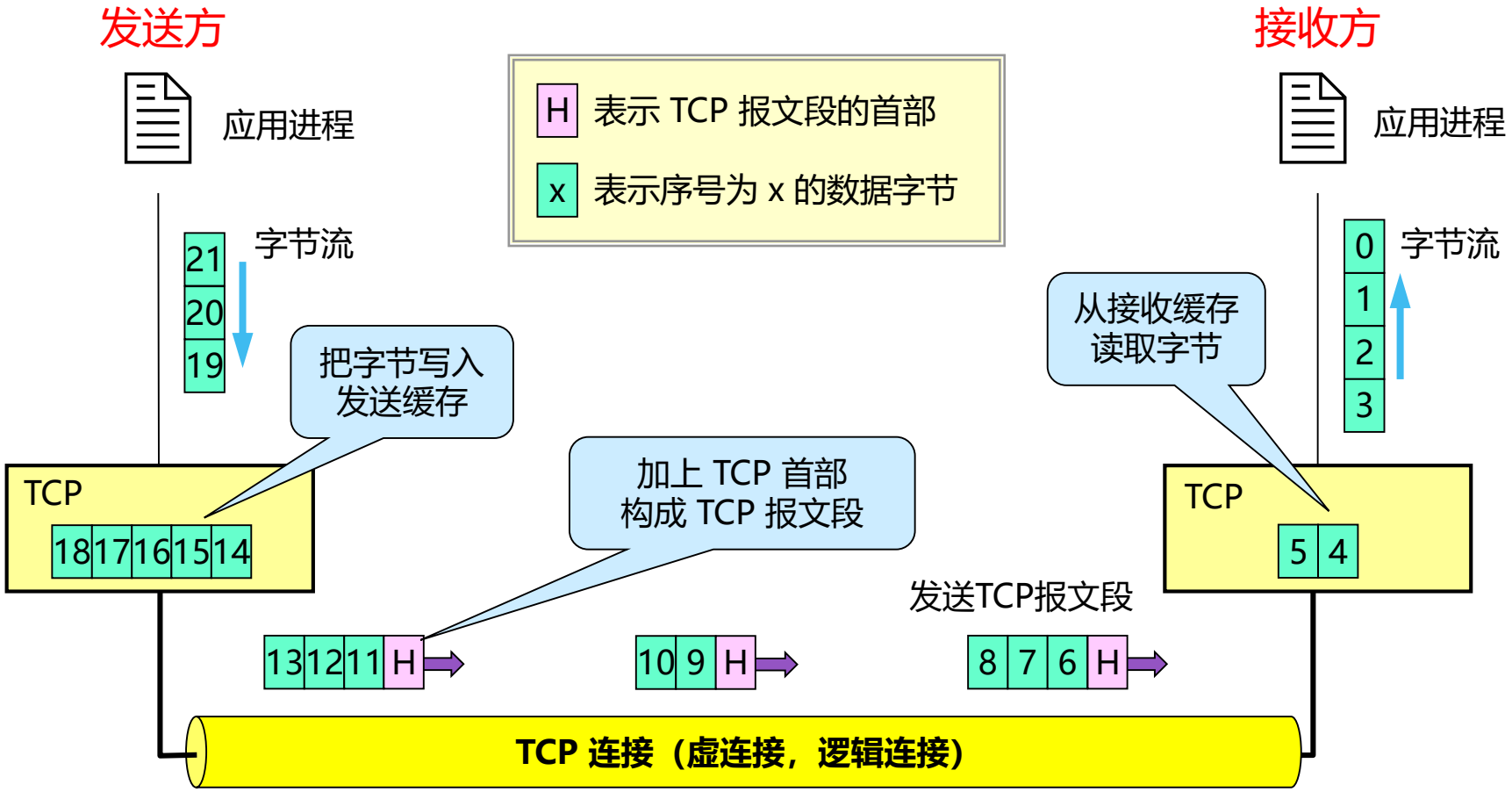
3. 传输控制协议 TCP

- TCP是TCP/IP体系中最复杂的一个协议，其主要特点有：
  - TCP是面向连接的运输层协议。
  - TCP连接只能有两个端点(endpoint)，每一条TCP连接只能是点对点的（一对一）。
  - TCP提供可靠交付的服务。
  - TCP提供全双工通信。
  - TCP面向字节流。

3. 传输控制协议 TCP

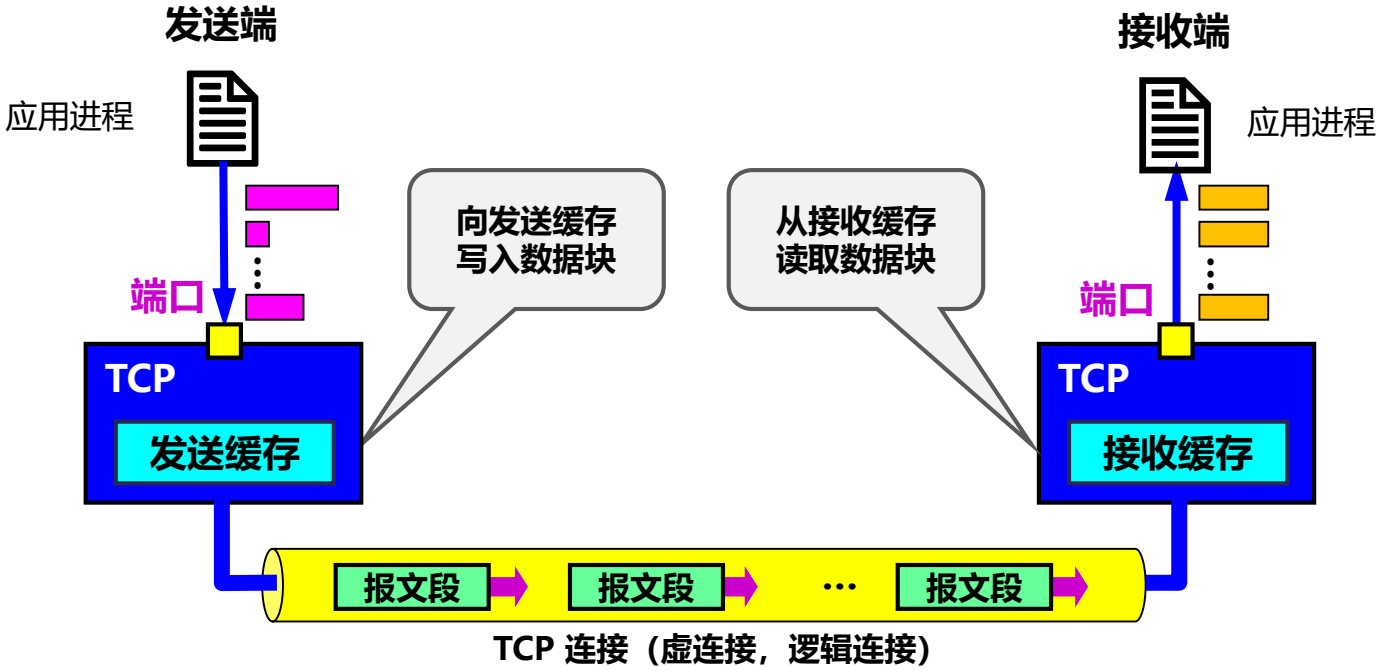
- TCP是面向字节流，TCP中的“流（stream）”指的是流入到进程或从进程流出的字节序列。
- 面向字节流的含义是：
  - 虽然应用程序和TCP的交互是一次一个数据块，但TCP把应用程序交下来的数据看成仅仅是一连串的无结构的字节流。
  - TCP并不知道传送的字节流的含义。
- TCP 面向流的概念
  - TCP 不保证接收方应用程序所收到的数据块和发送方应用程序所发出的数据块具有对应大小的关系。
  - 但接收方应用程序收到的字节流必须和发送方应用程序发出的字节流完全一样。

3. 传输控制协议 TCP


发送方
应用进程
字节流
21
20
19
把字节写入
发送缓存
TCP
18 17 16 15 14
H 表示 TCP 报文段的首部
x 表示序号为 x 的数据字节
加上 TCP 首部
构成 TCP 报文段
13 12 11 H
10 9 H
8 7 6 H
发送TCP报文段
TCP 连接（虚连接，逻辑连接）
接收方
应用进程
字节流
0
1
2
3
从接收缓存
读取字节
TCP
5 4

3. 传输控制协议 TCP

- TCP 不关心应用进程一次把多长的报文发送到 TCP 缓存。
- TCP 根据对方给出的窗口值和当前网络拥塞程度来决定一个报文段应包含多少个字节，形成 TCP 报文段。



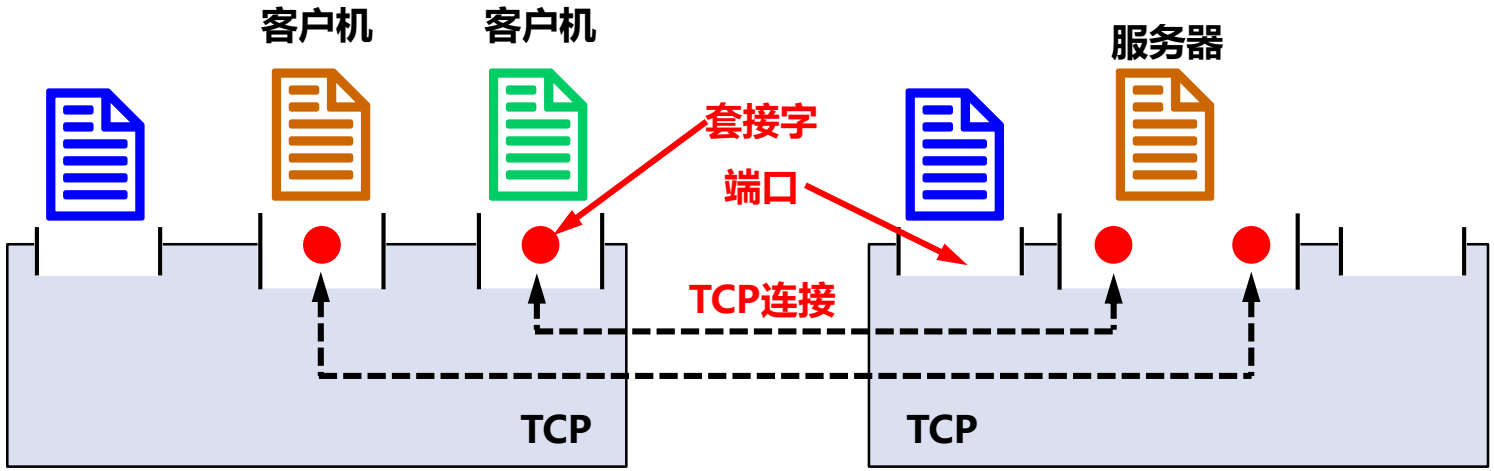
3. 传输控制协议 TCP

- 对TCP面向字节流的补充说明：
  - 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。

3. 传输控制协议 TCP

- TCP把连接作为最基本的抽象。
  - TCP的许多特征都与TCP是面向连接的这个基本特征有关。
- 每一条TCP连接有两个端点。
  - 但是TCP连接的端点不是主机，不是主机的IP地址，不是应用进程，也不是运输层的协议端口。
- TCP连接的端点叫做套接字(socket)或插口。
  - 根据RFC793的定义：端口号拼接到(contatenated with) IP地址即构成了套接字。

3. 传输控制协议 TCP



3. 传输控制协议 TCP

套接字 socket = (IP地址: 端口号)

每一条 **TCP** 连接唯一地被通信两端的两个端点（即两个套接字）所确定。即：

TCP连接::={socket1, socket2}
TCP连接::={(IP1:port1),(IP2:port2)}

3. 传输控制协议 TCP

TCP连接::={socket1, socket2}
TCP连接::={(IP1:port1),(IP2:port2)}

IP1、IP2 分别是两个端点主机的IP地址，
Port1、Port2 分别是两个端点主机中的端口号。
TCP连接的两个套接字就是 socket1、socket2。

3. 传输控制协议 TCP

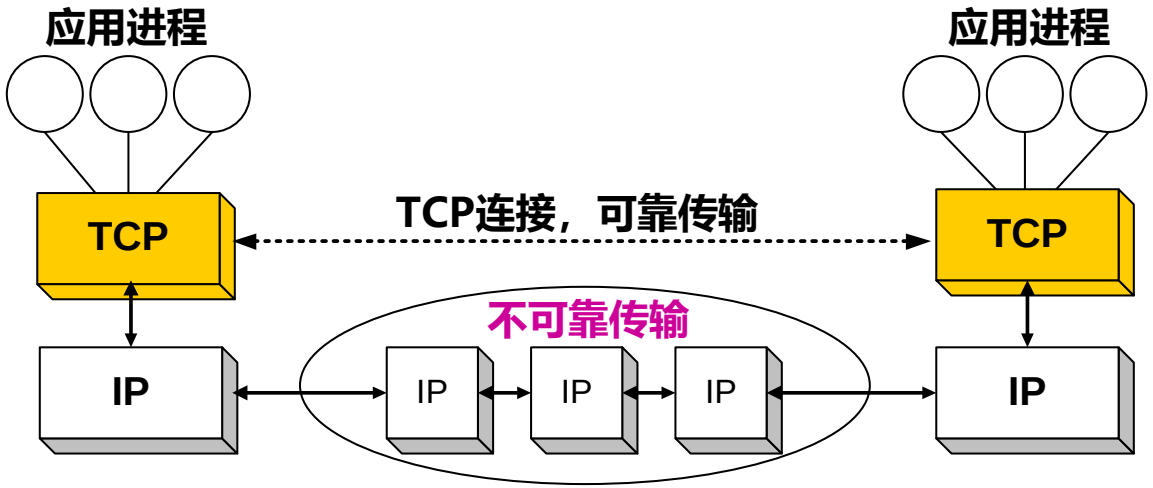
- Socket在不同的场景上有多种表示：
  - 允许应用程序访问连网协议的应用编程接口API称为socket API，简称为socket。
  - socket API中使用的一个函数名也叫作socket。
  - 调用socket函数的端点称为socket。
  - 调用socket函数时其返回值称为socket描述符，可简称为socket。
  - 在操作系统内核中连网协议的Berkeley实现，称为socket实现。？

4. 可靠传输的工作原理

- TCP 发送的报文段是交给 IP 层传送的，但是 IP 层只能提供尽最大努力服务，也就是说，TCP 下面的网络所提供的是不可靠的传输。
- TCP 必须采用适当的措施才能使两个运输层之间的通信变得可靠。

4. 可靠传输的工作原理

IP 网络所提供的是不可靠的传输



4. 可靠传输的工作原理

- 理想的传输条件有两个特点：
  - 传输信道不产生错误。
  - 不管发送方以多快的速度发送数据，接收方总是来得及处理收到的数据。
  - 在理想传输条件下，不需要采取任何措施就能够实现可靠传输，但是理想的传输条件是不存在的。
- 为了实现可靠传输就需要使用一些可靠传输协议。
  - 当发生差错时让发送方重传出现差错的数据
  - 在接收方来不及处理收到的数据时，及时告诉发送方适当降低发送数据的速度。

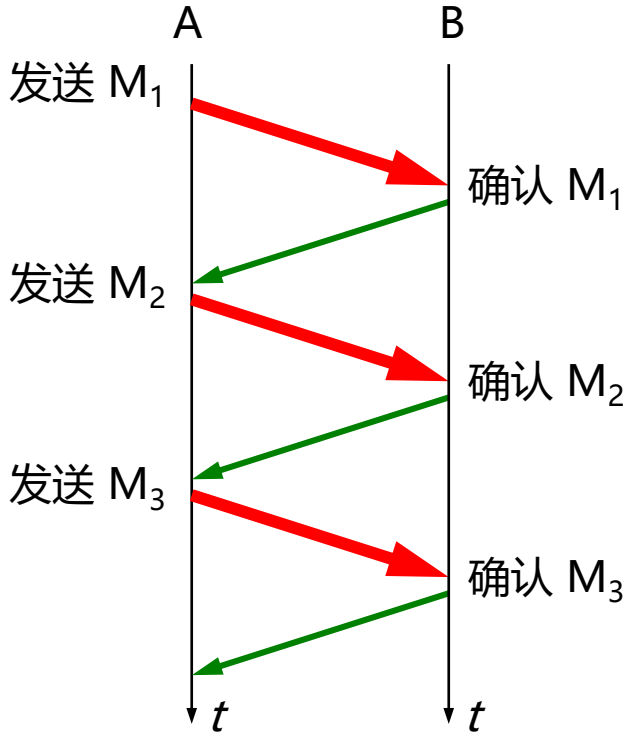

确认
数据验证实现传输错误避免
协商
重传与速率协商沟通实现错误传输保障

4. 可靠传输的工作原理

- 全双工通信的双方既是发送方又是接收方。
  - 为了讨论问题的方便，仅考虑从 A 向 B 传送数据，A 发送数据叫做发送方，B 接收数据并发送确认，叫做接收方。
  - 本部分讨论可靠传输原理，因此把传送的数据单元叫做分组，不考虑在哪一个层次上传送。
- 停止等待就是每发送完一个分组就停止发送，等待对方确认。收到确认后，在发送下一个分组。

4. 可靠传输的工作原理

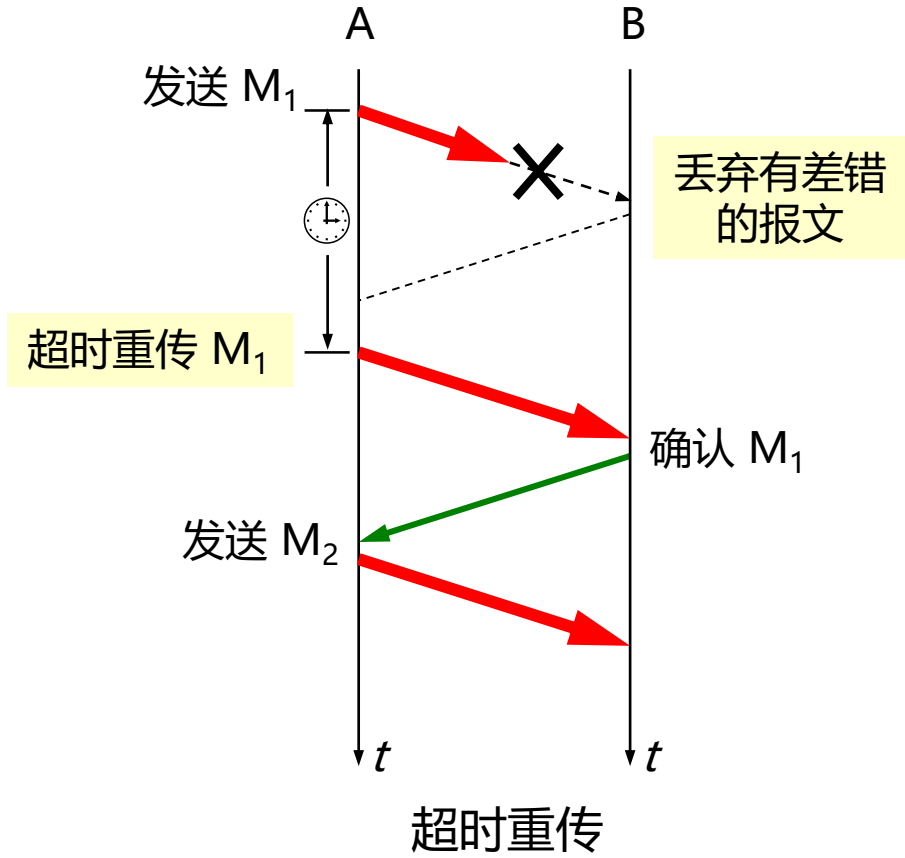
- A 发送分组 M1，发完就暂停发送，等待 B 的确认。
- B 收到 M1，就向A发送确认。
- A 收到了对 M1 的确认后，就再发送下一个分组 M2。
- 在收到 B 对 M2 的确认后，就再发送下一个分组 M3。



无差错情况

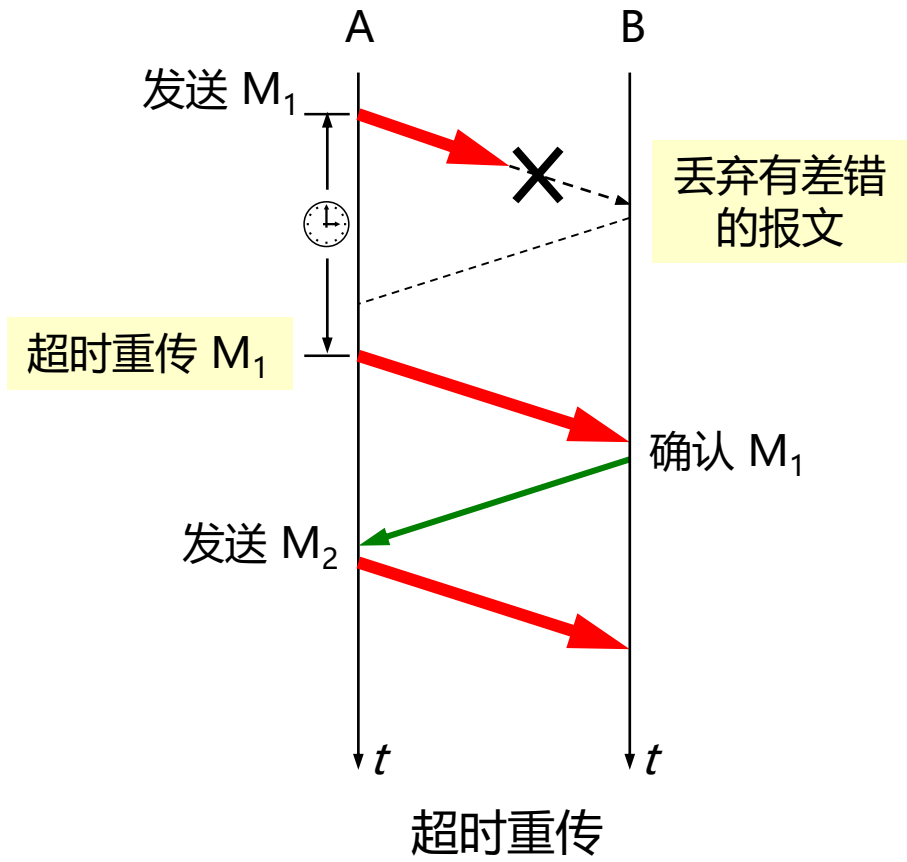
4. 可靠传输的工作原理

- 分组在传输过程中出现差错。B接收M1时检测出了错误，就丢弃M1，或者是M1在传输过程中丢失了。
- B没有向A发送确认信息。
- A超过一定时间没有收到确认，就认为发送的分组丢失，就重传M1。
- A在发送分组后设置超时计时器。



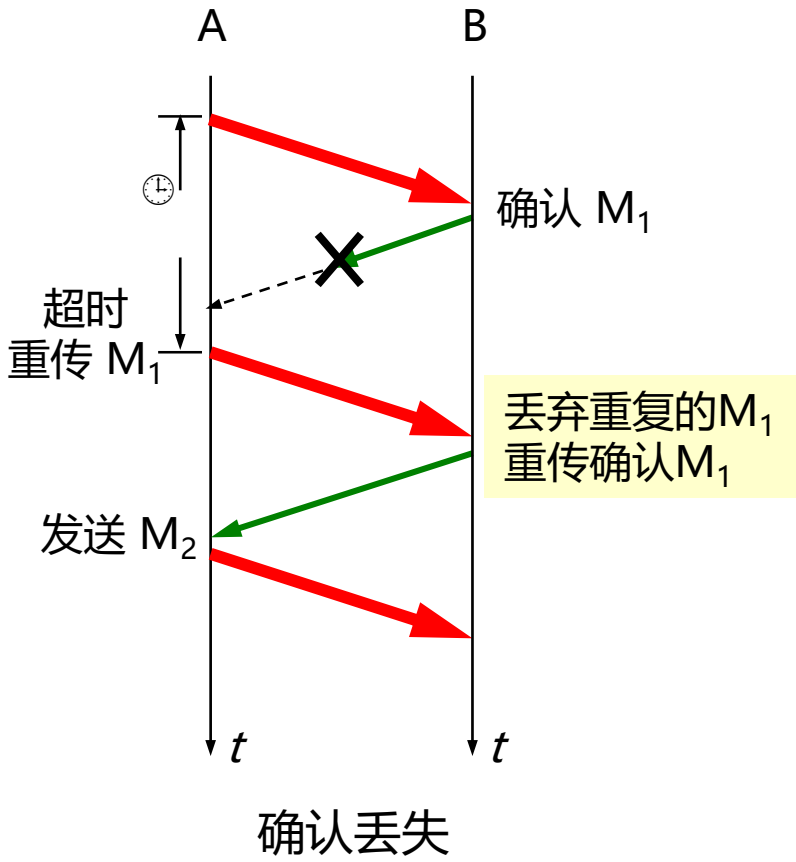
4. 可靠传输的工作原理

- A在发送完一个分组后，必须暂时保留已发送的分组的副本。
- 分组和确认分组都必须进行编号，才能够明确是哪一个发送出去的分组收到了确认。
- 超时计时器的重传时间应当比数据在分组传输的平均往返时间更长一些。



4. 可靠传输的工作原理

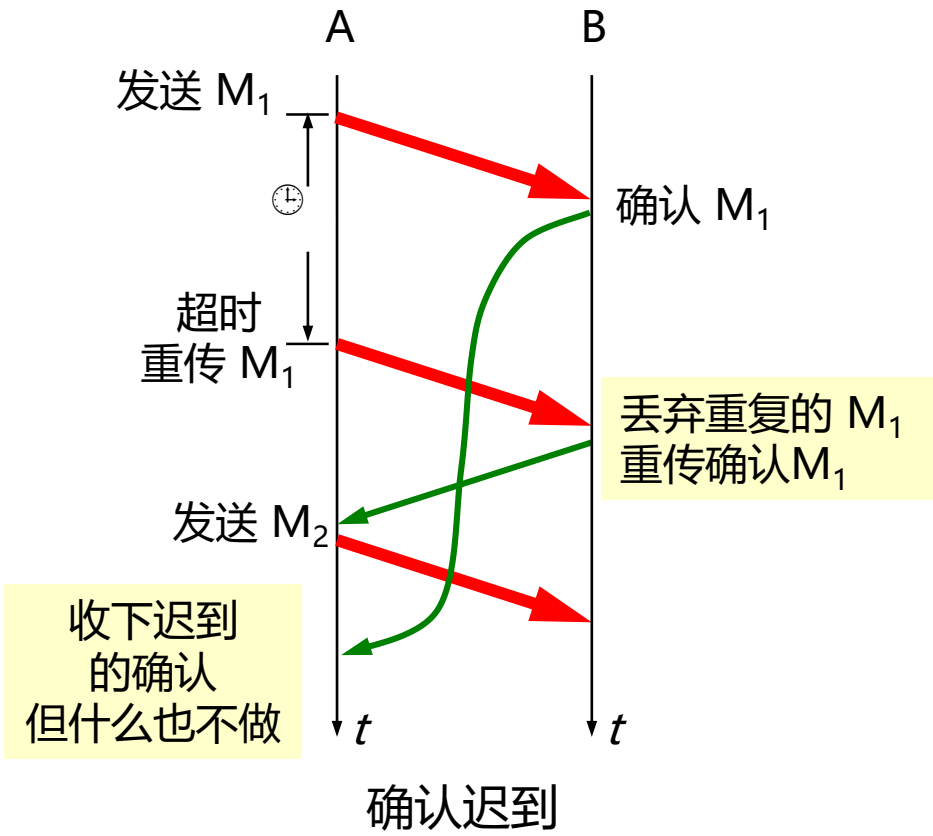
- A发送M1，B收到M1。
- B发送的M1确认丢失，A在设定的超时重传时间内没有收到确认。
- A在超时后重传M1，B收到重传M1后，丢弃重传M1数据，并且再次确认M1。



确认丢失

4. 可靠传输的工作原理

- 传输过程没有出现差错，但是B对分组M1的确认迟到。
- A会收到重复的确认，A收到重复确认后直接丢弃。
- B会收到重发的M1，B收到重复数据后直接丢弃，并重新确认分组。



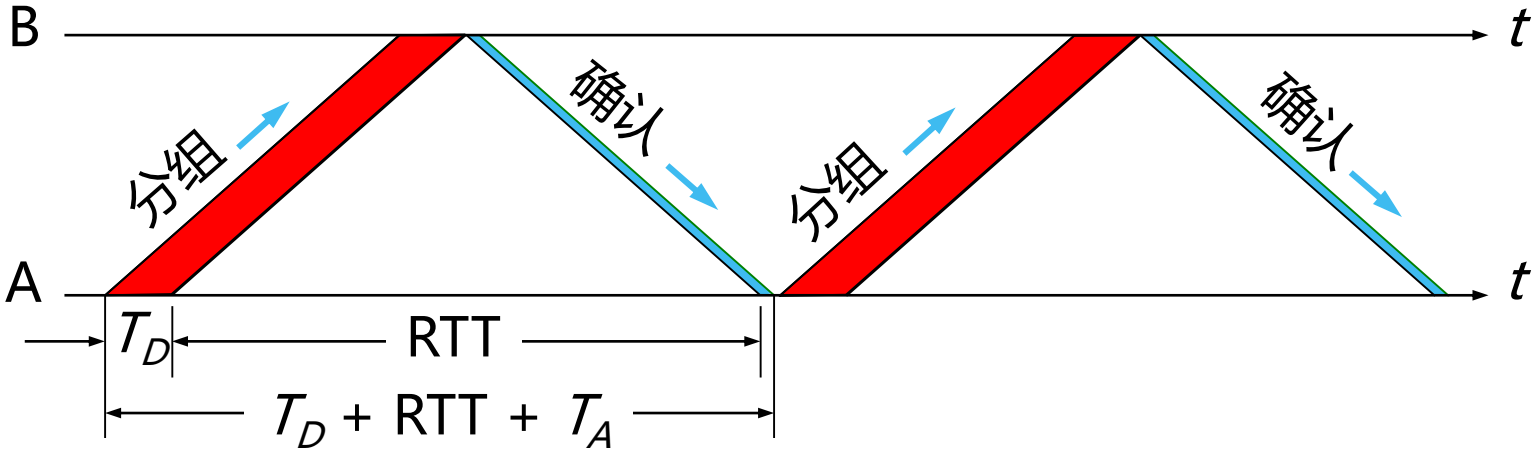
确认迟到

4. 可靠传输的工作原理

- 上述描述的确认和重传机制，就可以在不可靠的传输网络上实现可靠的通信。
- 上述的可靠传输协议常称为自动重传请求ARQ（Automatic Repeat reQuest）。
  - 就是说，重传的请求是自动进行的，接收方不需要请求发送方重传某个出错的分组。
- 发送方和接收方达成默契，只要在超时重传时间内没有得到确认，就自动重发分组。

4. 可靠传输的工作原理

- 停止等待协议的优点是简单，但缺点是信道利用率太低。

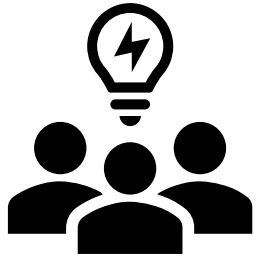


信道利用率

$$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A}$$

4. 可靠传输的工作原理

- 计算：
  - 假定1200km的信道的往返时间 RTT=20ms，分组长度为 1200bit，发送速率为 1Mbps，若忽略处理时间和 TA，那么信道利用率为多少？
  - 如果把发送速率提升为 10Mbps，则信道利用率为多少？
  - 如果把发送速率提升为 100Mbps，则信道利用率为多少？



4. 可靠传输的工作原理

- 在上述计算中，如果RTT远大于分组发送时间，那么信道利用率将非常低。
- 如果考虑到出现差错后的分组重传，则信道利用率还要进一步降低。

4. 可靠传输的工作原理

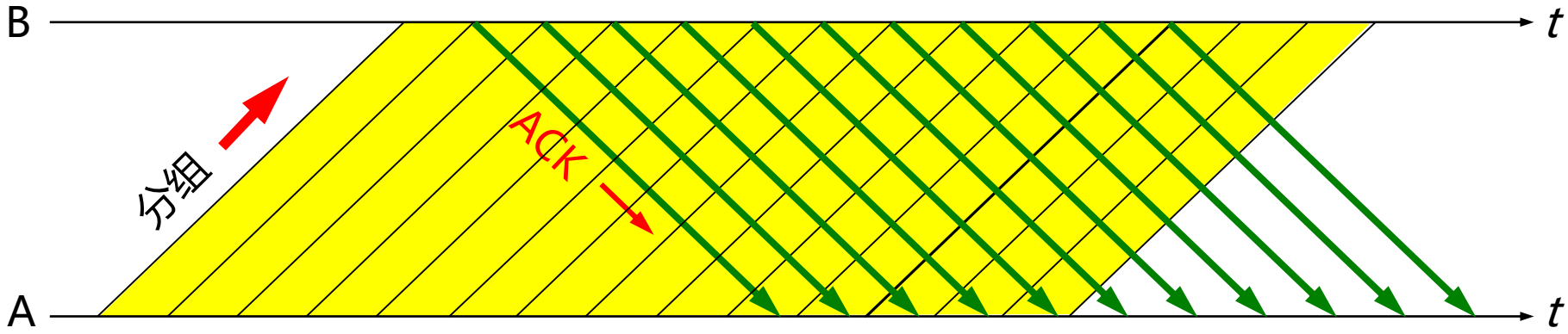
- 停止等待协议要点：
  - 停止等待：发送方每次只发送一个分组。在收到确认后再发送下一个分组。
  - 暂存：在发送完一个分组后，发送方必须暂存已发送的分组的副本，以备重发。
  - 编号：对发送的每个分组和确认都进行编号。
  - 超时重传：发送方为发送的每个分组设置一个超时计时器。
    - 若超时计时器超时位收到确认，发送方会自动超时重传分组。
    - 超时计时器的重传时间应当比数据在分组传输的平均往返时间更长一些，防止不必要的重传。
  - 简单，但信道利用率太低。

4. 可靠传输的工作原理

- 提高传输效率：流水线传输
  - 为了提高传输效率，发送方可以不使用低效率的停止等待协议，而是采用流水线传输。
  - 流水线传输就是发送方可连续发送多个分组，不必每发完一个分组就停顿下来等待对方的确认。
  - 由于信道上一直有数据不间断地传送，这种传输方式可获得很高的信道利用率。

4. 可靠传输的工作原理

- 提高传输效率：流水线传输
  - 流水线传输时，最为常见的是连续ARQ协议和滑动窗口协议。



4. 可靠传输的工作原理

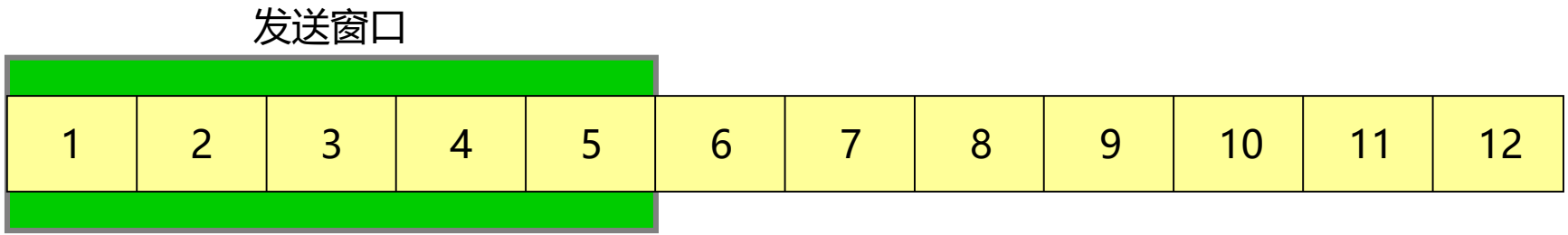
- 滑动窗口协义比较复杂，是TCP协议的精髓。
- 本部分不涉及到细节问题，先讨论连续ARQ协议的基本概念。
- 在讨论之前，需要先明确和重复一个概念：
  - TCP是全双工通信。
  - TCP连接的通信两端可以同时发送和接收数据。

4. 可靠传输的工作原理

- 连续ARQ协议的基本思想：
  - 发送窗口：
    - 发送方维持一个发送窗口，位于发送窗口内的分组都可被连续发送出去，而不需要等待对方的确认。
  - 发送窗口滑动：
    - 发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。
  - 累积确认：
    - 接收方对按序到达的最后一个分组发送确认，表示：到这个分组为止的所有分组都已正确收到了。

4. 可靠传输的工作原理

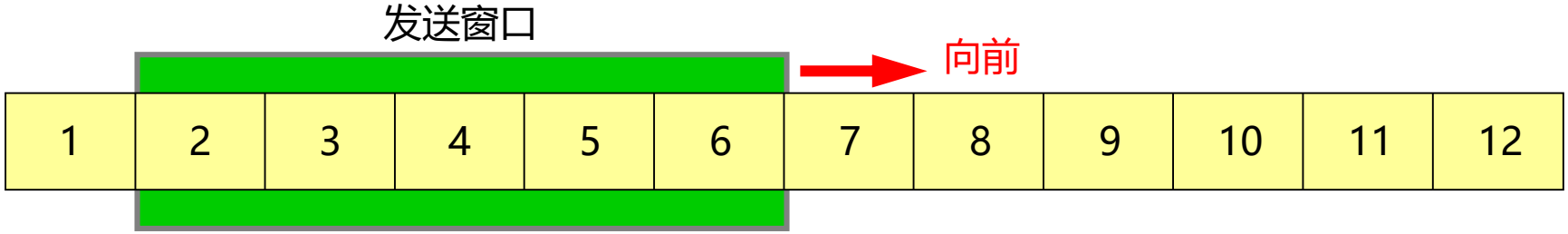
- 发送窗口。
  - 位于发送窗口内的5个分组都可以连续发出去，而不需要等待对方的确认。



发送方维持发送窗口（发送窗口是 5）

4. 可靠传输的工作原理

- 发送窗口
  - 连续ARQ协议规定，发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。
  - 当接收方发送一个确认而发送方收到后，发送窗口向前移动一个分组，则第6个分组就可以发送。



收到一个确认后发送窗口向前滑动

4. 可靠传输的工作原理

- 接收方通常会采用累积确认的方式。
  - 接收方不必对收到的分组逐个发送确认，而是在收到几个分组后，对按序到达的最后一个分组发送确认。
  - 这就表示：到这个分组为止的所有分组都已经正确收到了。
- 累积确认的优缺点：
  - 优点是：容易实现，即使确认丢失也不必重传。？
  - 缺点是：不能向发送方反映出接收方已经正确收到的所有分组的信息。

4. 可靠传输的工作原理

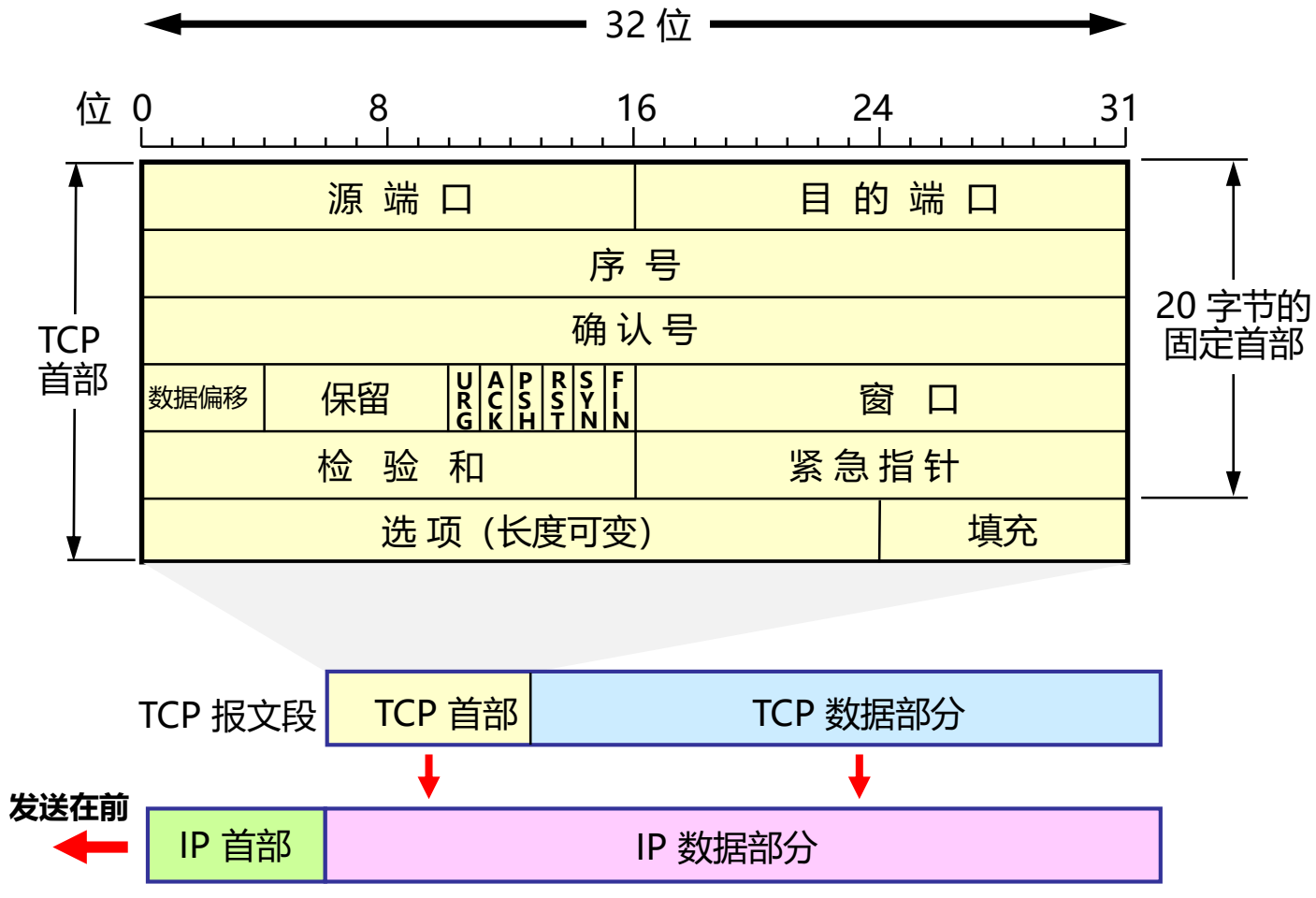
- 累积确认的优缺点：
  - 优点是：容易实现，即使确认丢失也不必重传。？
  - 缺点是：不能向发送方反映出接收方已经正确收到的所有分组的信息。
- Go-back-N：
  - 如果发送方发送了前5个分组，而中间的第3个分组丢失了。这时接收方只能对前两个分组发出确认。发送方无法知道后面三个分组的下落，而只好把后面的三个分组都再重传一次。
  - 这就叫做Go-back-N（回退 N），表示需要再退回来重传已发送过的N个分组。
  - 当通信线路质量不好时，连续ARQ协议会带来负面的影响。

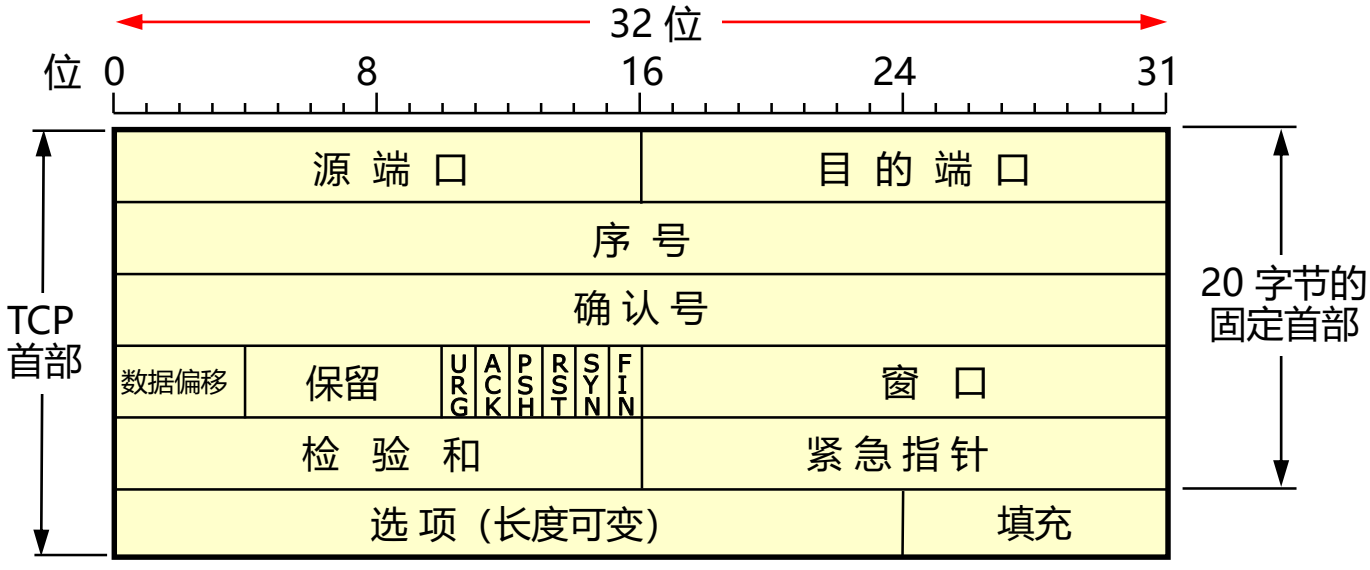
5. TCP 报文段的首部格式

- TCP在通信时是面向字节流的，但是TCP传送的数据单元却是报文段。
- TCP报文段分为首部和数据两部分，而TCP的全部功能都体现在它首部中各字段的作用。
  - 要想理解TCP的工作原理，就必须要熟练掌握TCP首部各字段的作用。

5. TCP 报文段的首部格式

- TCP报文段首部的前20个字节是固定，后面有4n字节是根据需要而增加的选项（n是整数）。
- TCP首部的最小长度为20字节，最大为60字节。

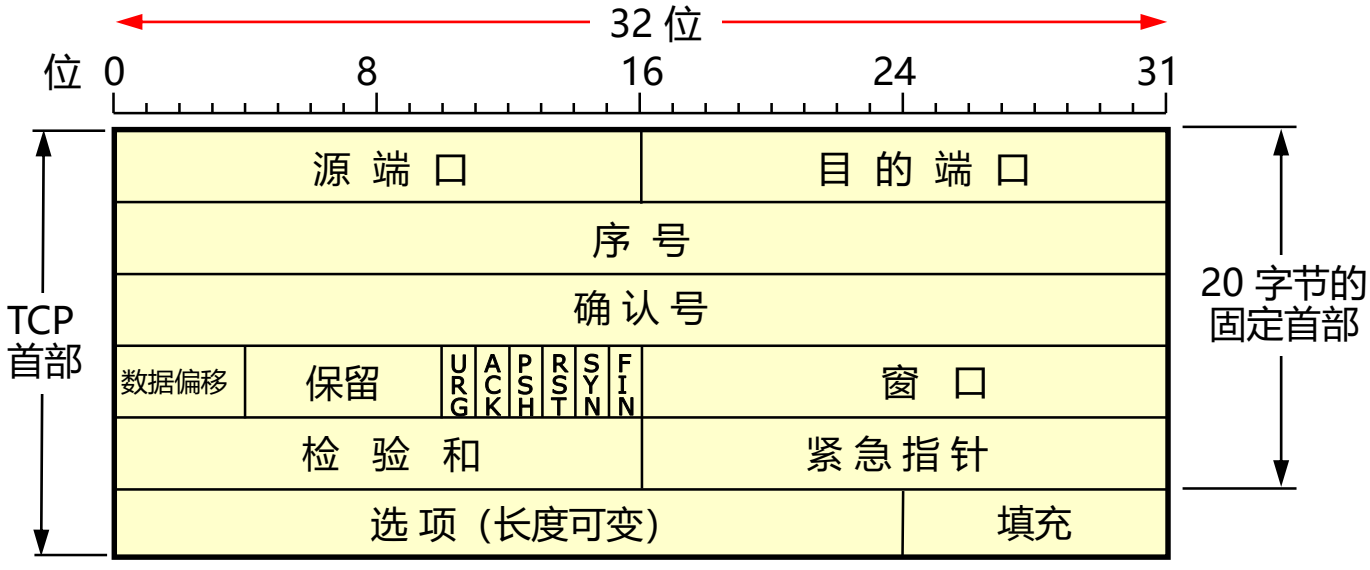

32 位
位 0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项（长度可变）
填充
TCP 首部
20 字节的固定首部
TCP 报文段
TCP 首部
TCP 数据部分
发送在前
IP 首部
IP 数据部分



源端口和目的端口字段：
各2字节。端口是运输层与应用层的服务接口。运输层的复用和分用功能都要通过端口才能实现。

序号字段：
4字节。TCP连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值指的是本报文段所发送数据的第一个字节的序号。

确认号字段：
4字节。是期望收到对方的下一个报文段的数据的第一个字节的序号。如果确认号为N，则表明序号N-1为止的所有数据都已经正确收到。

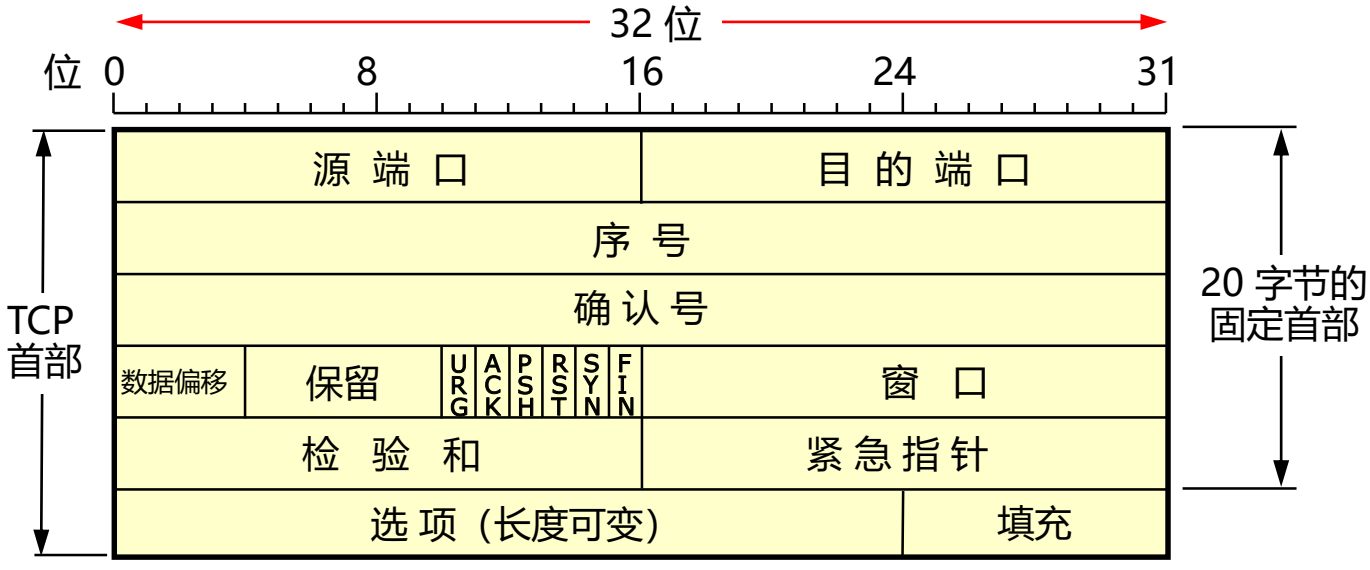


数据偏移（即首部长度）：

4位。它指出TCP报文段的数据起始处距离 TCP报文段的起始处有多远。 “数据偏移”的单位是32位字（以4字节为计算单位）。因此TCP报文段首部最大长度为：$(2^4-1)*4$字节=60字节。

保留字段：

6位。保留为今后使用，目前应置为0。



紧急URG(URGent)：

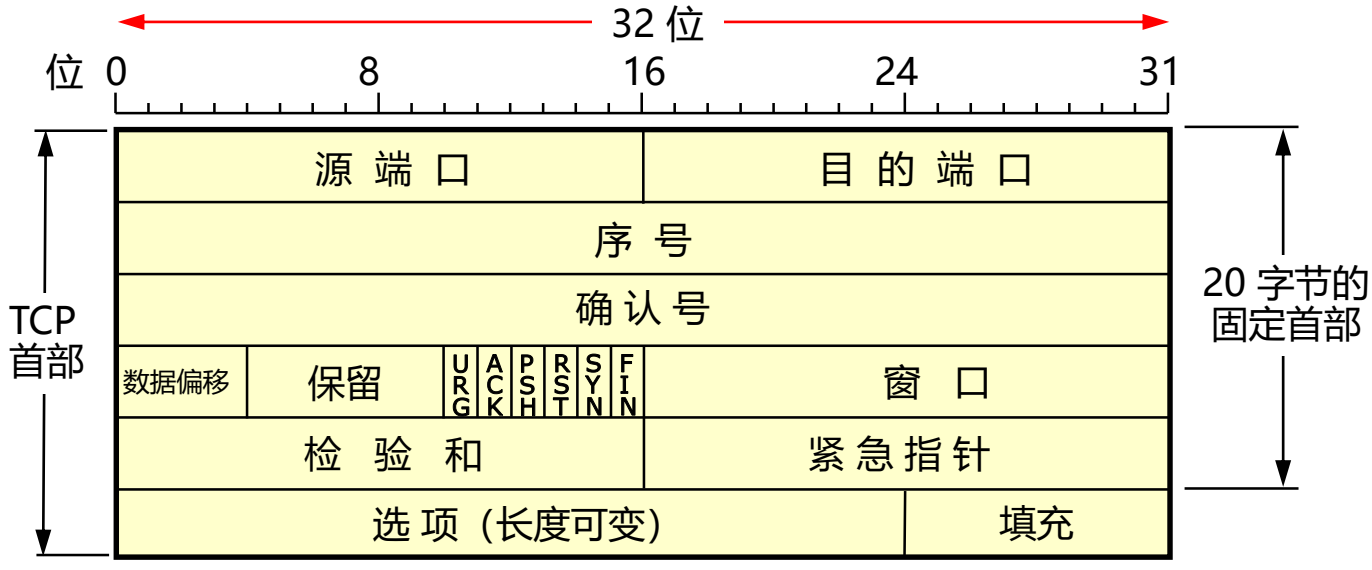
1位。当URG=1时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。

确认ACK(ACKnowlegment)：

1位。当ACK=1时，表明确认号字段有效。当ACK=0时，表明确认号字段无效。

推送PSH(PuSH) ：

1位。接收端收到TCP报文段的PSH=1时，此报文会尽快地交付接收应用进程，而不再等到整个缓存都填满后再向上交付。



复位RST(ReSeT)：

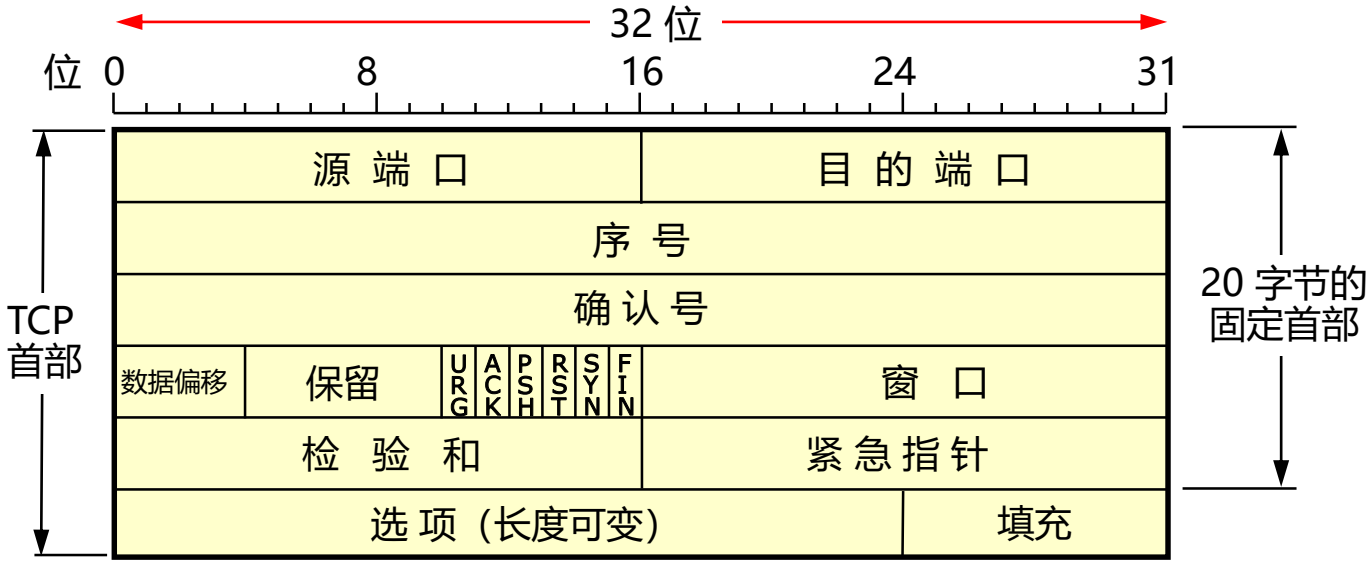
1位。当RST=1时，表明TCP连接中出现严重差错（如主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。

同步SYN(SYNchronization)：

1位。当SYN=1且ACK=0时，表示这是一个连接请求报文段，如果对方同意建立连接，则SYN=1且ACK=1。因此当SYN=1，表示这是一个连接请求或连接接受报文。

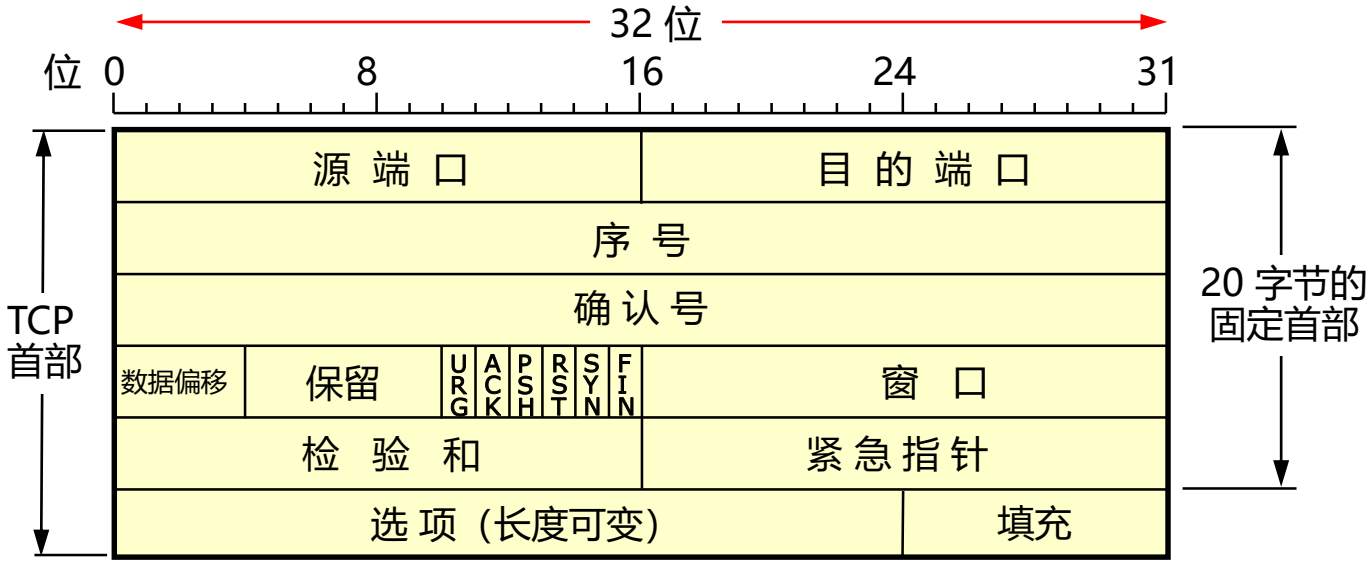
终止FIN(FINis)：

1位。用来释放一个连接。当FIN=1表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。



窗口：

2字节。窗口指的是发送本报文段的一方的接受窗口。窗口值告诉对方：从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量。由于接收方的数据缓存空间是有限的，窗口值作为接收方让发送方设置其发送窗口的依据。其数据单位为字节。在ACK=1时才有效。

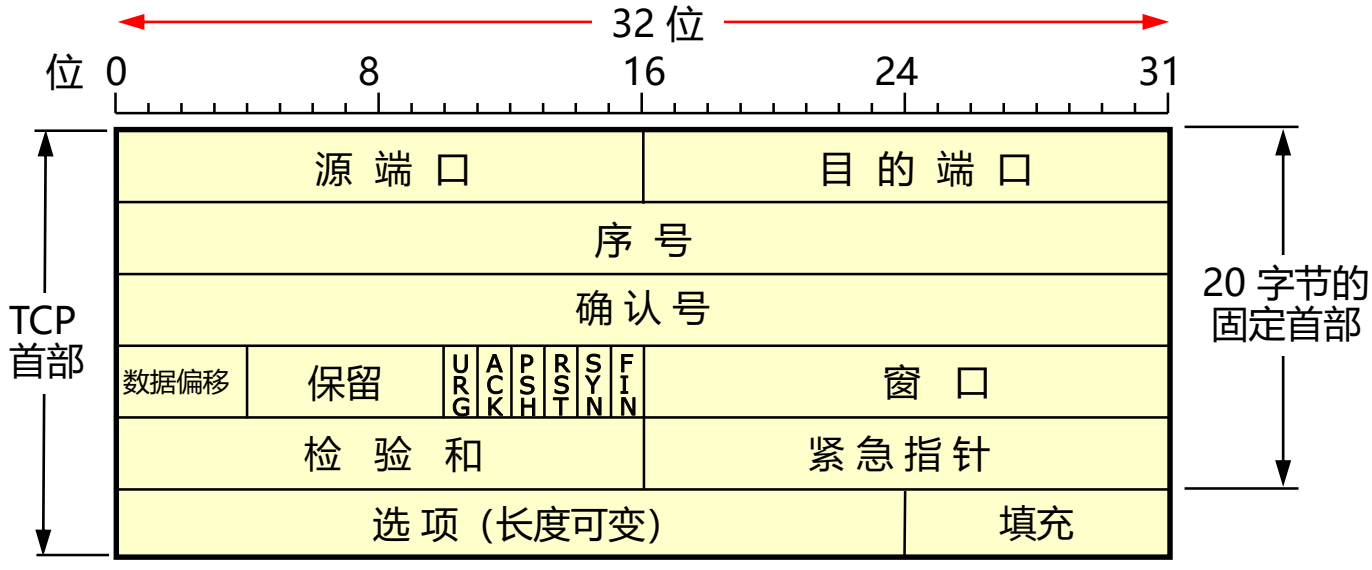


检验和：

2字节。检验和字段检验的范围包括首部和数据这两部分。和UDP用户数据报一样，在计算校验和时，要在TCP报文段的前面加上12字节的伪首部。伪首部的格式和UDP用户数据报的结构一样，但要把协议号修改为6。校验和的计算方法和UDP用户数据报的计算方法相同。

紧急指针：

16位。该字段在UGR=1时起效。指出在本报文段中的紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。因此紧急指针指出了紧急数据的末尾在报文段中的位置。当所有紧急数据都处理完毕后，TCP就告诉应用程序恢复到正常操作。



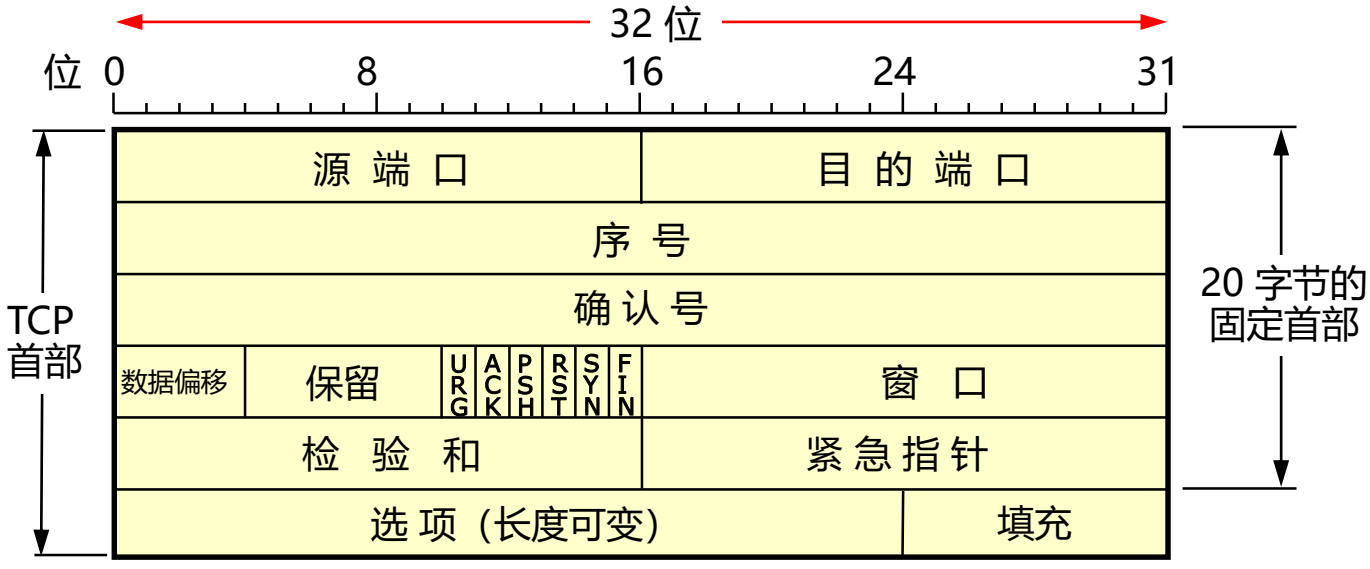
选项：

0-40字节。

TCP 最初只规定了一种选项，即最大报文段长度 MSS。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报文的最大长度=MSS+TCP首部长度

随着因特网的发展，逐步增加了新的选项。如窗口扩大选项、时间戳选项、选择确认选项等。



填充：为什么要有填充选项？填充的内容是什么？

5. TCP 报文段的首部格式

- 现场讨论：
  - 根据提供的报文进行分析，并回答六个问题：
  - 问题一：来源和目的MAC地址是什么？
  - 问题二：来源和目的IP地址是什么？
  - 问题三：来源和目的端口是什么？
  - 问题四：运输层协议是什么？
  - 问题五：应用层协议是什么？

5. TCP 报文段的首部格式

```
64 a0 e7 41 41 41 00 1f  29 02 c7 f5 08 00 45 00
00 3f 75 45 00 00 80 11  72 af c0 a8 9e c3 d3 45
20 08 d8 ef 00 35 00 2b  75 1e b7 ab 01 00 00 01
00 00 00 00 00 00 03 77  77 77 09 77 69 72 65 73
68 61 72 6b 03 6f 72 67  00 00 01 00 01
```

```
⊞ Frame 4: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
⊟ Ethernet II, Src: Hewlett-_02:c7:f5 (00:1f:29:02:c7:f5), Dst: Cisco_41:41:41 (64:a0:e7:41:41:41)
  ⊟ Destination: Cisco_41:41:41 (64:a0:e7:41:41:41)
      Address: Cisco_41:41:41 (64:a0:e7:41:41:41)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ⊟ Source: Hewlett-_02:c7:f5 (00:1f:29:02:c7:f5)
      Address: Hewlett-_02:c7:f5 (00:1f:29:02:c7:f5)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
⊟ Internet Protocol Version 4, Src: 192.168.158.195 (192.168.158.195), Dst: 211.69.32.8 (211.69.32.8)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 63
    Identification: 0x7545 (30021)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: UDP (17)
  ⊞ Header checksum: 0x72af [correct]
    Source: 192.168.158.195 (192.168.158.195)
    Destination: 211.69.32.8 (211.69.32.8)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ User Datagram Protocol, Src Port: 55535 (55535), Dst Port: domain (53)
    Source port: 55535 (55535)
    Destination port: domain (53)
    Length: 43
  ⊞ Checksum: 0x751e [validation disabled]
⊟ Domain Name System (query)
    [Response In: 5]
    Transaction ID: 0xb7ab
  ⊞ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊞ Queries

0000  64 a0 e7 41 41 41 00 1f  29 02 c7 f5 08 00 45 00   d..AAA.. ).....E.
0010  00 3f 75 45 00 00 80 11  72 af c0 a8 9e c3 d3 45   .?uE.... r......E
0020  20 08 d8 ef 00 35 00 2b  75 1e b7 ab 01 00 00 01    ....5.+ u.......
0030  00 00 00 00 00 00 03 77  77 77 09 77 69 72 65 73   .......w ww.wires
0040  68 61 72 6b 03 6f 72 67  00 00 01 00 01           hark.org .....
```

```
64 a0 e7 41 41 41 00 1f  29 02 c7 f5 08 00 45 00
01 b0 11 cb 40 00 80 06  cb e5 c0 a8 9e c3 da 1c
e2 0e f6 67 24 67 5e 04  27 63 42 a1 b7 44 50 18
40 29 fa 02 00 00 47 45  54 20 2f 69 6d 61 67 65
73 2f 6c 6f 67 6f 2f 71  69 73 68 69 5f 6c 6f 67
6f 5f 31 34 30 30 38 31  38 37 36 32 2e 70 6e 67
20 48 54 54 50 2f 31 2e  31 0d 0a 41 63 63 65 70
74 3a 20 74 65 78 74 2f  68 74 6d 6c 2c 20 61 70
70 6c 69 63 61 74 69 6f  6e 2f 78 68 74 6d 6c 2b
78 6d 6c 2c 20 2a 2f 2a  0d 0a 41 63 63 65 70 74
2d 4c 61 6e 67 75 61 67  65 3a 20 7a 68 2d 43 4e
0d 0a 55 73 65 72 2d 41  67 65 6e 74 3a 20 4d 6f
7a 69 6c 6c 61 2f 35 2e  30 20 28 57 69 6e 64 6f
77 73 20 4e 54 20 36 2e  31 3b 20 54 72 69 64 65
6e 74 2f 37 2e 30 3b 20  72 76 3a 31 31 2e 30 29
20 6c 69 6b 65 20 47 65  63 6b 6f 0d 0a 41 63 63
65 70 74 2d 45 6e 63 6f  64 69 6e 67 3a 20 67 7a
69 70 2c 20 64 65 66 6c  61 74 65 0d 0a 48 6f 73
74 3a 20 63 75 61 67 77  2e 68 61 63 74 63 6d 2e
65 64 75 2e 63 6e 3a 39  33 31 39 0d 0a 49 66 2d
4d 6f 64 69 66 69 65 64  2d 53 69 6e 63 65 3a 20
46 72 69 2c 20 32 33 20  4d 61 79 20 32 30 31 34
20 30 34 3a 31 39 3a 32  32 20 47 4d 54 0d 0a 49
66 2d 4e 6f 6e 65 2d 4d  61 74 63 68 3a 20 22 66
65 30 32 39 65 2d 31 30  36 63 38 2d 34 66 61 30
39 38 38 62 65 30 37 66  36 22 0d 0a 44 4e 54 3a
20 31 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20
4b 65 65 70 2d 41 6c 69  76 65 0d 0a 0d 0a
```

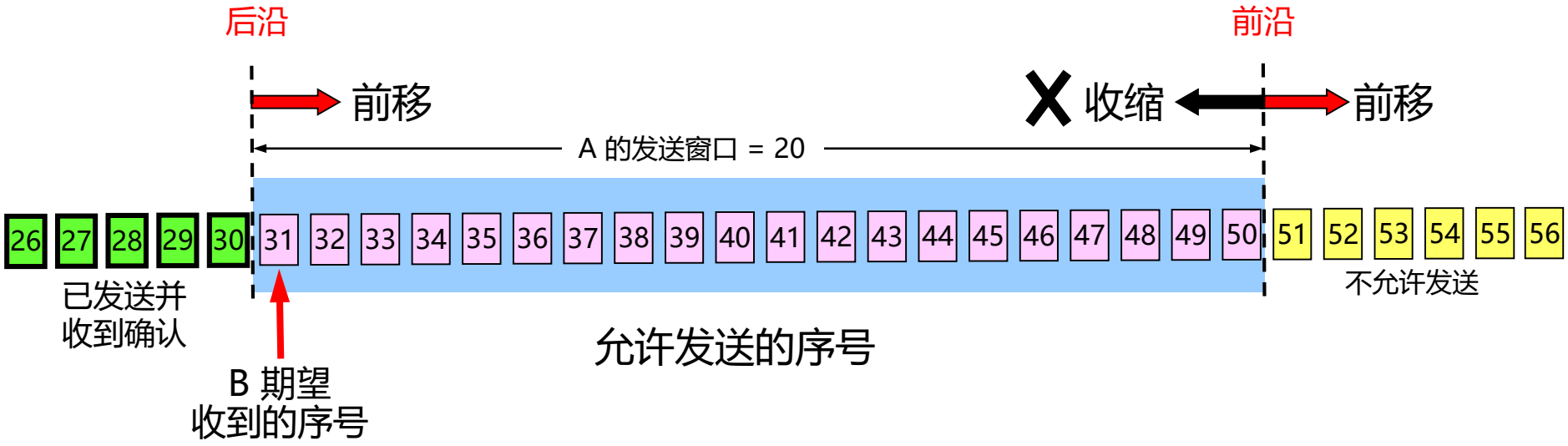
```
⊞ Frame 8539: 446 bytes on wire (3568 bits), 446 bytes captured (3568 bits) on interface 0
⊟ Ethernet II, Src: Hewlett-_02:c7:f5 (00:1f:29:02:c7:f5), Dst: Cisco_41:41:41 (64:a0:e7:41:41:41)
  ⊟ Destination: Cisco_41:41:41 (64:a0:e7:41:41:41)
      Address: Cisco_41:41:41 (64:a0:e7:41:41:41)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ⊟ Source: Hewlett-_02:c7:f5 (00:1f:29:02:c7:f5)
      Address: Hewlett-_02:c7:f5 (00:1f:29:02:c7:f5)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
⊟ Internet Protocol Version 4, Src: 192.168.158.195 (192.168.158.195), Dst: 218.28.226.14 (218.28.226.14)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 432
    Identification: 0x11cb (4555)
  ⊞ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (6)
  ⊞ Header checksum: 0xcbe5 [correct]
    Source: 192.168.158.195 (192.168.158.195)
    Destination: 218.28.226.14 (218.28.226.14)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ Transmission Control Protocol, Src Port: 63079 (63079), Dst Port: 9319 (9319), Seq: 1, Ack: 1, Len: 392
    Source port: 63079 (63079)
    Destination port: 9319 (9319)
    [Stream index: 20]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 393    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x018 (PSH, ACK)
    Window size value: 16425
    [Calculated window size: 65700]
    [Window size scaling factor: 4]
  ⊞ Checksum: 0xfa02 [validation disabled]
  ⊞ [SEQ/ACK analysis]
⊟ Hypertext Transfer Protocol
  ⊟ GET /images/logo/qishi_logo_1400818762.png HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): GET /images/logo/qishi_logo_1400818762.png HTTP/1.1\r\n]
```

```
0000  64 a0 e7 41 41 41 00 1f  29 02 c7 f5 08 00 45 00   d..AAA.. ).....E.
0010  01 b0 11 cb 40 00 80 06  cb e5 c0 a8 9e c3 da 1c   ....@... ........
0020  e2 0e f6 67 24 67 5e 04  27 63 42 a1 b7 44 50 18   ...g$g^. 'cB..DP.
0030  40 29 fa 02 00 00 47 45  54 20 2f 69 6d 61 67 65   @)....GE T /image
0040  73 2f 6c 6f 67 6f 2f 71  69 73 68 69 5f 6c 6f 67   s/logo/q ishi_log
0050  6f 5f 31 34 30 30 38 31  38 37 36 32 2e 70 6e 67   o_140081 8762.png
0060  20 48 54 54 50 2f 31 2e  31 0d 0a 41 63 63 65 70    HTTP/1. 1..Accep
0070  74 3a 20 74 65 78 74 2f  68 74 6d 6c 2c 20 61 70   t: text/ html, ap
```

6. TCP 可靠传输的实现

- TCP的滑动窗口是以字节为单位的。
  - 假定A收到了B发来的确认报文段，其中窗口是20（字节），而确认号是31（表示希望收到的下一个序号为31，序号30为止的数据已经收到）。
  - 根据窗口和确认号，A就可以构造出自己的发送窗口。
  - 说明：为了讨论方便，仅假定数据传输只在一个方向进行，即A发送数据，B给出确认。

6. TCP 可靠传输的实现



6. TCP 可靠传输的实现

- TCP的滑动窗口是以字节为单位的。
  - 发送窗口里面的序号表示允许发送的序号。
    - 窗口越大，发送发就可以在收到对方确认之前连续发送更多的数据，从而获得更高的传输效率。
  - 发送窗口后沿的后面部分表示已发送且收到了确认。
    - 发送窗口前沿的前面部分表示不允许发送的。
  - 发送窗口的位置由窗口前沿和后沿的位置共同确定。

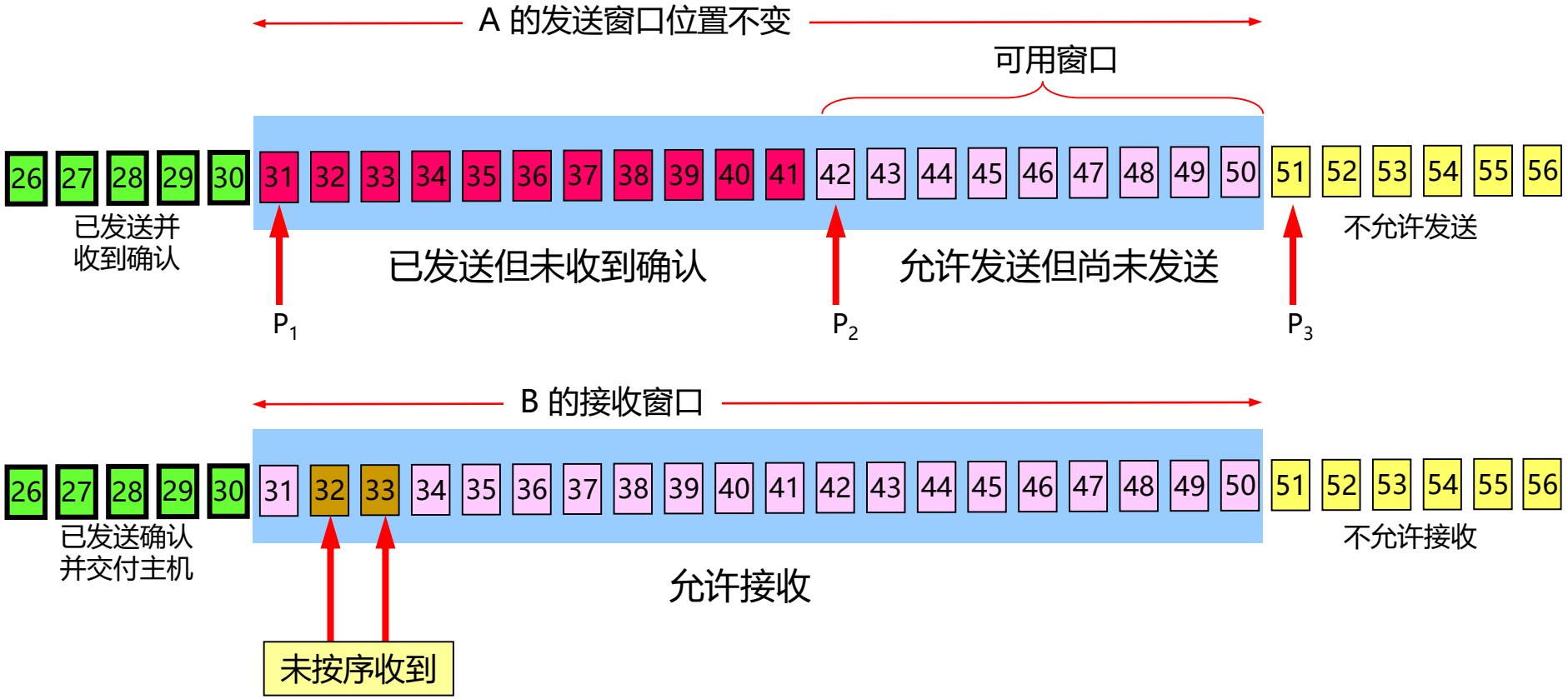
6. TCP 可靠传输的实现

- 发送窗口后沿的变化情况有两种可能：不动和前移。
  - 没有收到新的确认，就不动；
  - 收到了新的确认，就前移。
  - 发送窗口后沿不可能向后移动，因为不能够撤销已经收到的确认。
- 发送窗口前沿的变化情况有三种可能：不动、前移和收缩。
  - 没有收到新的确认或者收到新的确认但窗口变小都可能造成前沿不动。
  - 在对方通知的窗口缩小时，发送窗口前沿也可能向后收缩，但是TCP标准强烈不造成这种做法。

6. TCP 可靠传输的实现

- 假定A发送了序号31-41的数据。这时，发送窗口位置并未改变，但是31-41的11个字节为已发送但未收到确认。42-50的9个字节为允许发送但尚未发送。
- 要描述一个发送窗口的状态需要三个指针：P1、P2、P3。
  - P3-P1 = A的发送窗口
  - P2-P1 = 已发送但尚未收到确认的字节数
  - P3-P2 = 允许发送但尚未发送的字节数

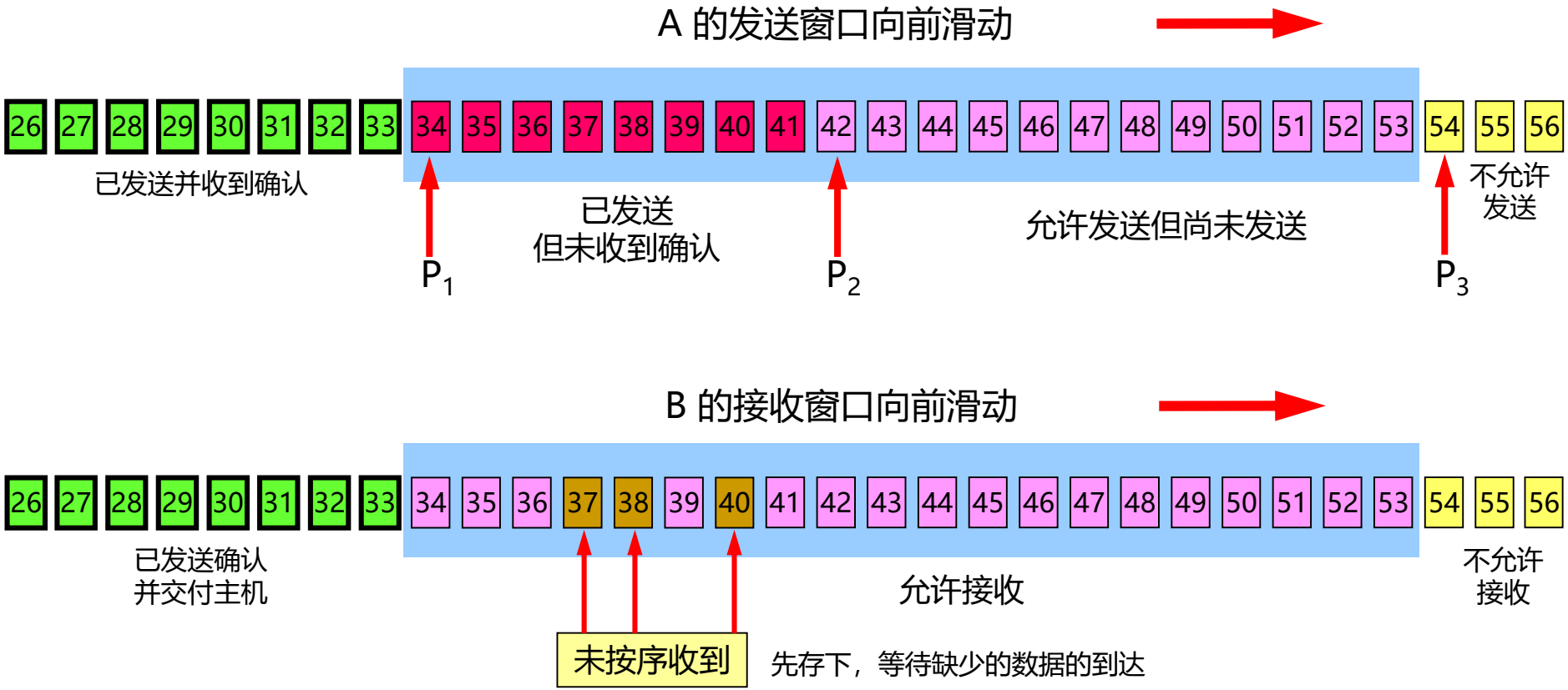
6. TCP 可靠传输的实现



6. TCP 可靠传输的实现

- 假定B收到了序号为31的数据，并把序号为31-33的数据交付主机，然后B删除这些数据。接着把接收窗口向前移动3个序号，同时给A发送确认，窗口值仍为20，确认号为34。同时B收到了37、38、40的数据，但是没有按序到达。
- A收到B的确认后，就可以把发送窗口向前滑动3个序号，但指针P2不动。A的可用窗口增大，可发送的序号范围是42-53。

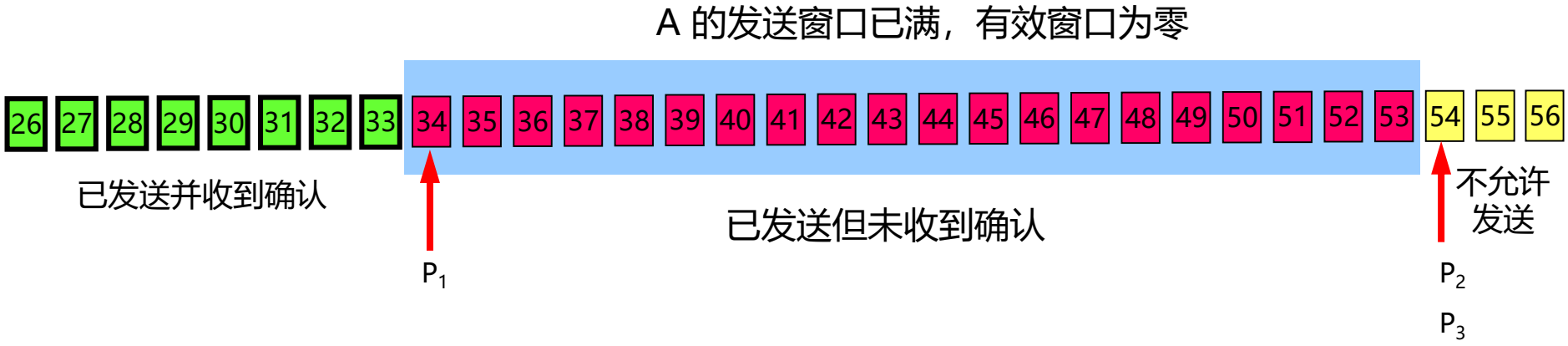
6. TCP 可靠传输的实现



6. TCP 可靠传输的实现

- A在继续发送完序号42-53的数据后，指针P2向前移动和P3重合。发送窗口内的序号都已用完，但还没有再收到确认。
- 由于A的发送窗口已满，可用窗口已减小到零，因此停止发送数据。

6. TCP 可靠传输的实现



6. TCP 可靠传输的实现

- 发送方的应用进程把字节流写入TCP的发送缓存，接收方的应用进程从TCP的接收缓存中读取字节流。
- 窗口和缓存的关系对于研究TCP可靠传输非常重要。
- 在讨论之前说明两点：
  - 缓存空间和序号空间都是有限的，且都是循环使用的。
  - 实际上缓存或窗口中的字节数都是非常大的。

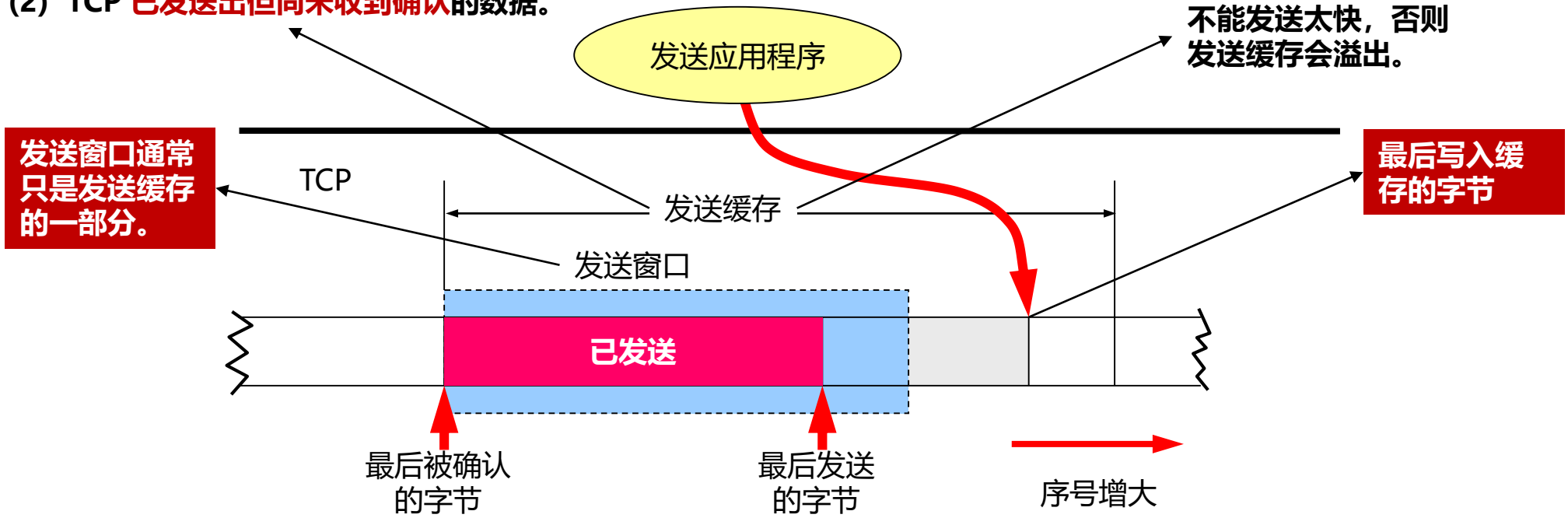
6. TCP 可靠传输的实现

暂时存放：

(1) 发送应用程序传送给发送方 TCP 准备发送的数据；

(2) TCP 已发送出但尚未收到确认的数据。

发送缓存与发送窗口

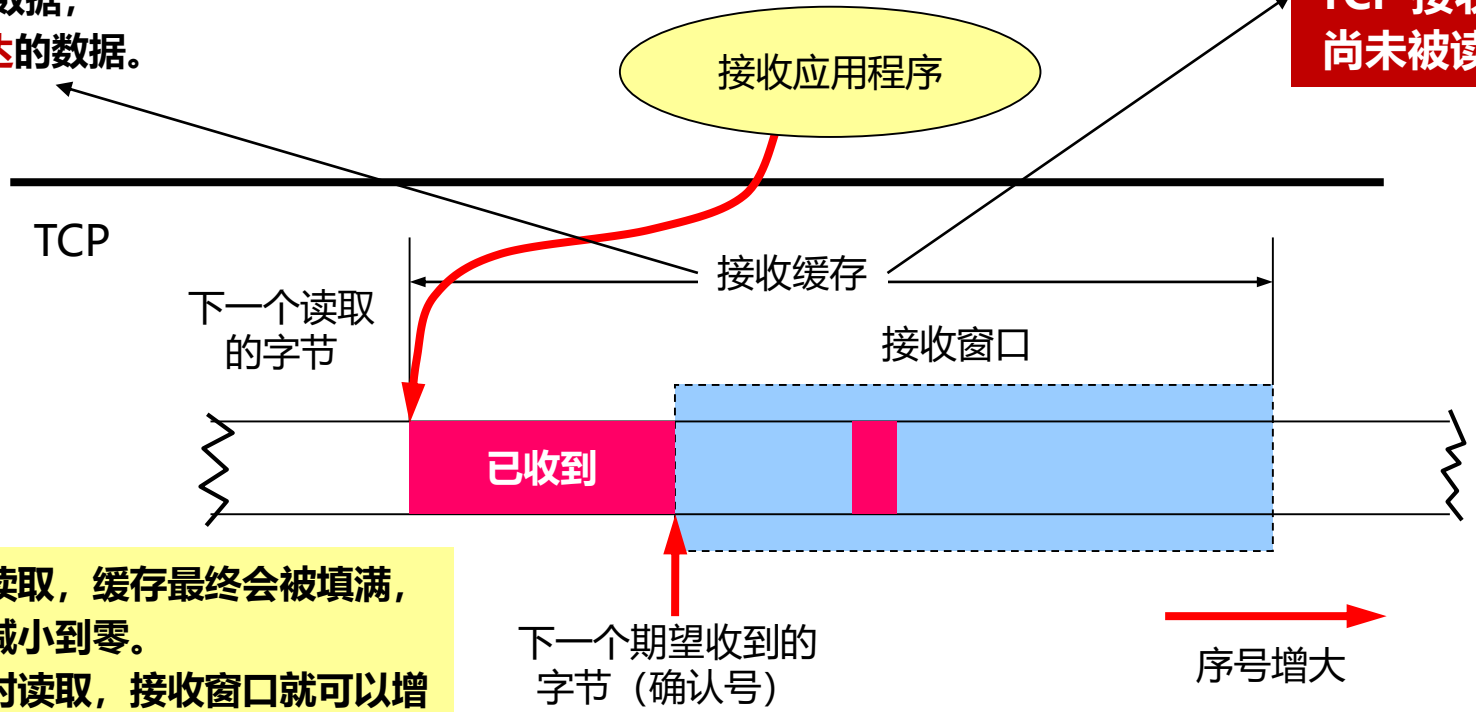


缓存中的字节数 = 发送应用程序最后写入缓存的字节 - 最后被确认的字节

6. TCP 可靠传输的实现

暂时存放：
(1) 按序到达的、但尚未被接收应用程序读取的数据；
(2) 未按序到达的数据。

接收缓存与接收窗口



接收方的应用进程从 TCP 接收缓存中读取尚未被读取的字节。

若不能及时读取，缓存最终会被填满，使接收窗口减小到零。
如果能够及时读取，接收窗口就可以增大，但最大不能超过接收缓存的大小。

6. TCP 可靠传输的实现

- 关于TCP窗口强调三点：

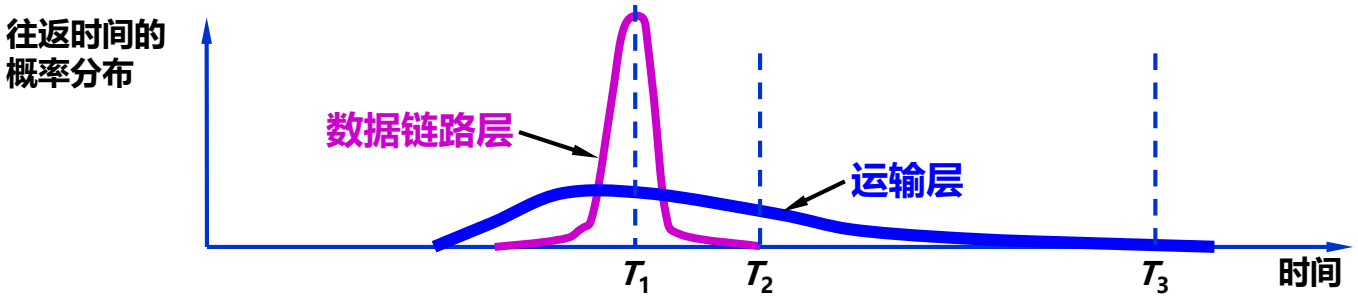
① 发送窗口是根据接收窗口设置的，但在同一时刻，发送窗口并不总是和接收窗口一样大（因为有一定的时间滞后）。

② 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。

③ TCP 要求接收方必须有累积确认的功能，以减小传输开销。接收方可以在合适的时候发送确认，也可以在自己有数据要发送时把确认信息顺便捎带上。但接收方不应过分推迟发送确认，否则会导致发送方不必要的重传，捎带确认实际上并不经常发生。

6. TCP 可靠传输的实现

- TCP的发送方在规定的时间内没有收到确认就要**重传**已发送的报文段，这种重传的概念是简单的。
- 重传时间的选择却是 TCP 最复杂的问题之一。
- 互联网环境复杂，IP 数据报所选择的路由变化很大，导致运输层的往返时间 (RTT) 的变化也很大。



6. TCP 可靠传输的实现

- TCP 超时重传时间设置
  - 不能太短，否则会引起很多报文段的不必要的重传，使网络负荷增大。
  - 不能过长，会使网络的空闲时间增大，降低了传输效率。

- 运输层的超时计时器的超时重传时间究竟应设置为多大呢？
  - TCP 采用了一种自适应算法，它记录一个报文段发出的时间，以及收到相应的确认的时间。
  - 这两个时间之差就是报文段的往返时间 RTT。

6. TCP 可靠传输的实现

加权平均往返时间 RTT_S

- 加权平均往返时间 RTT_S 又称为平滑的往返时间。

新的 RTT_S = (1 − α) × (旧的 RTT_S) + α × (新的 RTT 样本)

其中，$0 \leq \alpha < 1$ 。

若 $\alpha \to 0$，表示 RTT 值更新较慢。

若 $\alpha \to 1$，表示 RTT 值更新较快。

RFC 6298 推荐的 α 值为 1/8，即 0.125。

6. TCP 可靠传输的实现

超时重传时间 RTO

- RTO (Retransmission Time-Out) 应略大于加权平均往返时间 RTT_S。
- RFC 6298 建议 RTO：

$$RTO = RTT_S + 4 \times RTT_D$$

其中：RTT_D 是 RTT 偏差的加权平均值。

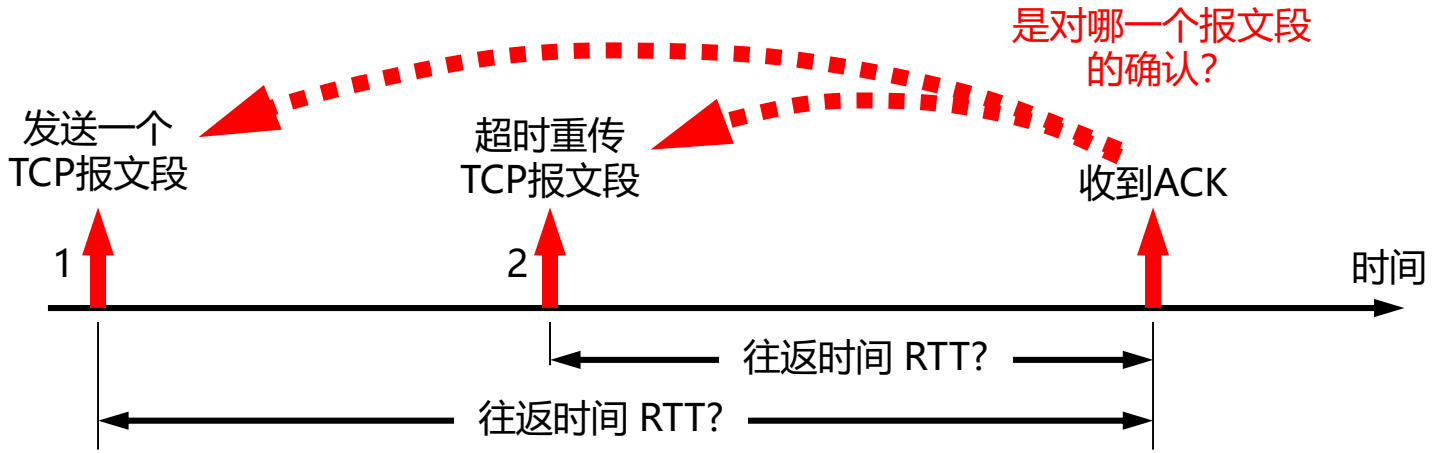
- RFC 6298 建议 RTT_D：

$$\text{新的 } RTT_D = (1 - \beta) \times (\text{旧的 } RTT_D) + \beta \times |RTT_S - \text{新的 RTT 样本}|$$

其中：β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

6. TCP 可靠传输的实现

- 往返时间 (RTT) 的测量相当复杂
  - 超时重传报文段后，如何判定此确认报文段是对原来的报文段的确认，还是对重传报文段的确认？
  - TCP 报文段1 没有收到确认。重传（即报文段2）后，收到了确认报文段ACK。
  - 如何判定此确认报文段是对原来的报文段1 的确认，还是对重传的报文段2 的确认？



6. TCP 可靠传输的实现

- Karn算法：
  - 在计算平均往返时间 RTT 时，只要报文段重传了，就不采用其往返时间样本。这样得出的加权平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。
  - 报文段每重传一次，就把 RTO 增大一些：

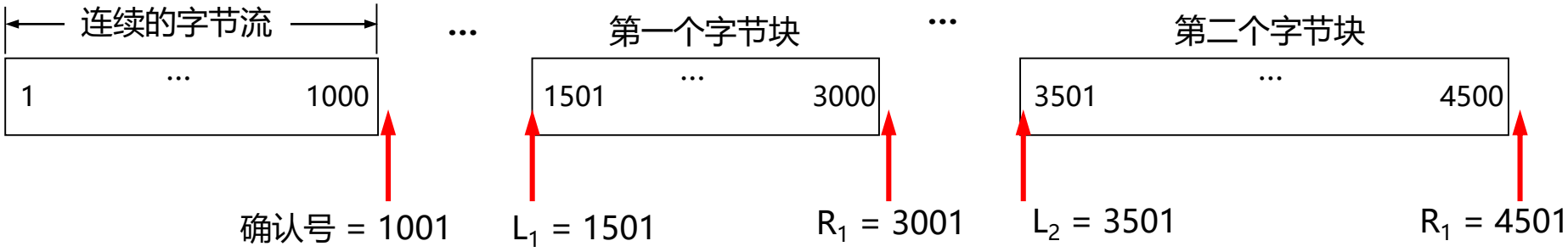
$$\text{新的 RTO} = \gamma \times \text{(旧的 RTO)}$$

  - 系数γ 的典型值是2。
  - 当不再发生报文段的重传时，才根据报文段的往返时延更新平均往返时延 RTT 和超时重传时间 RTO 的数值。
    - 实践证明，这种策略较为合理。

6. TCP 可靠传输的实现

- 如果收到的报文段无差错，只是未按序号，中间缺少一些序号的数据，那么是否设法只传送缺少的数据而不重传已经正确到达接收方的数据？
  - TCP采用了选择确认的方式来处理。
  - 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，但要把这些信息准确地告诉发送方，使发送方不要再重复发送这些已收到的数据。

6. TCP 可靠传输的实现

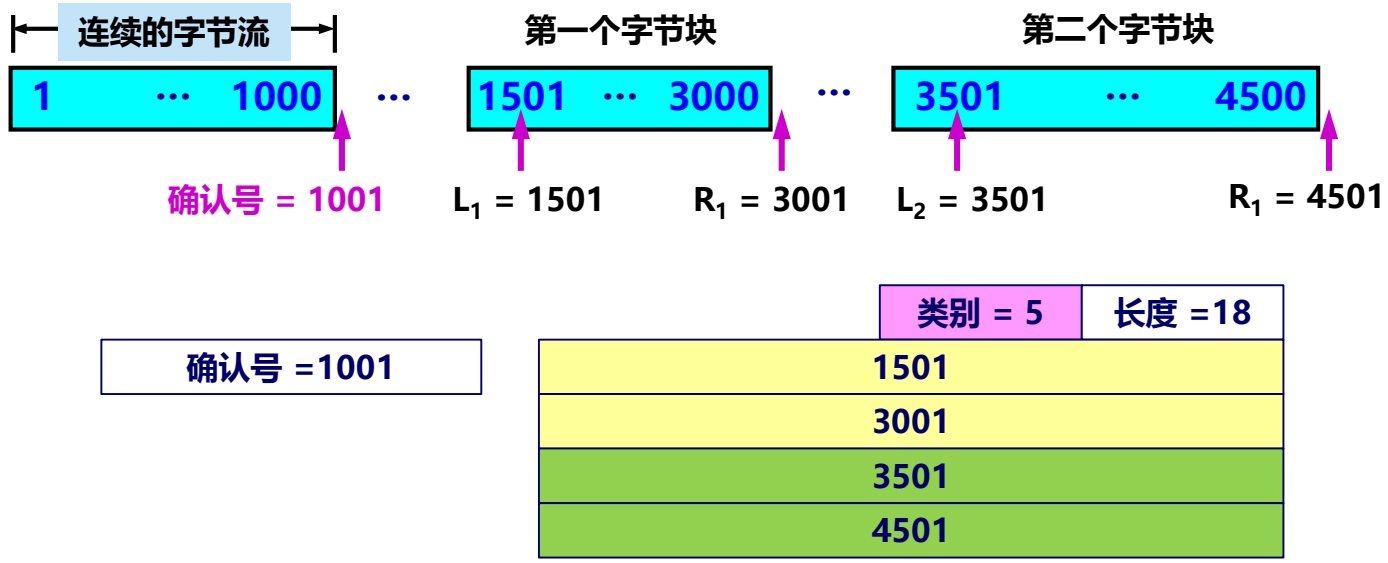


- 和前后字节不连续的每一个字节块都有两个边界：左边界和右边界。
  - 第一个字节块的左边界 L1=1501，但右边界 R1=3001。
    - 左边界指出字节块的第一个字节的序号，但右边界减 1 才是字节块中的最后一个序号。
  - 第二个字节块的左边界 L2=3501，而右边界 R2=4501。

6. TCP 可靠传输的实现

- RFC 2018 的规定：
  - 如果要使用选择确认，那么在建立 TCP 连接时，就要在 TCP 首部的选项中加上 “允许SACK” 的选项，而双方必须都事先商定好。
  - 如果使用选择确认，那么原来首部中的“确认号字段”的用法仍然不变。只是以后在 TC P报文段的首部中都增加了 SACK 选项，以便报告收到的不连续的字节块的边界。
  - 由于首部选项的长度最多只有 40 字节，而指明一个边界就要用掉 4 字节，因此在选项中最多只能指明 4 个字节块的边界信息。

6. TCP 可靠传输的实现



左边界 = 第一个字节的序号，右边界 = 最后一个字节序号 + 1。

7. TCP 流量控制

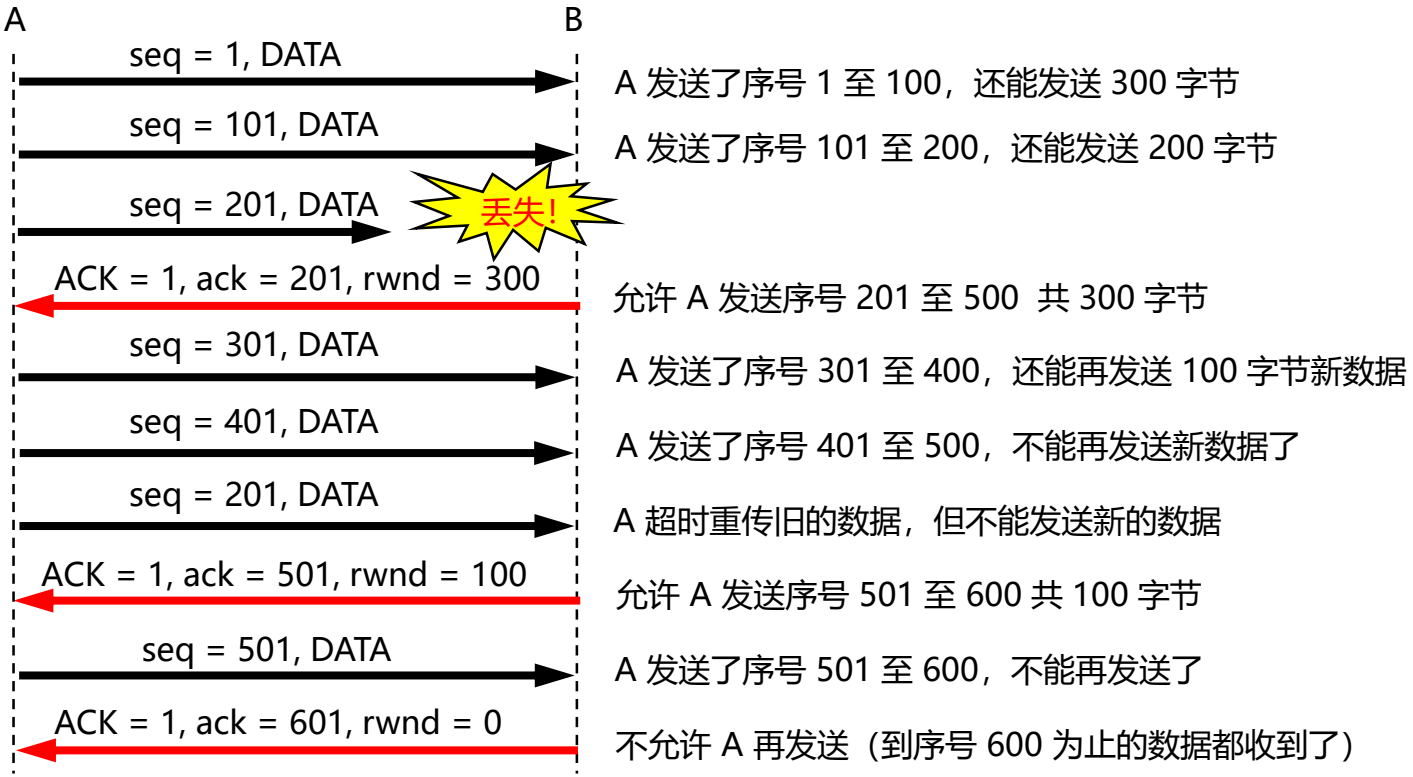
- 一般说来，总是希望数据传输得更快一些。
  - 但如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失。
- 流量控制 (flow control)：让发送方的发送速率不要太快，既要让接收方来得及接收，也不要使网络发生拥塞。
- 利用滑动窗口机制可以很方便地在 TCP 连接上实现流量控制。

7. TCP 流量控制

- TCP 在发送数据时，发送方的发送窗口不能超过接收方给出的接收窗口的数值。
  - 注意：TCP窗口的单位是字节，不是报文段。
- TCP 就利用这一原理实现流量控制。
  - 当接收方接收数据的速度变慢时，就在确认报文中告诉发送方接收窗口的大小，发送方根据接收窗口的大小确定发送窗口大小。
  - 从而达到控制流量的目的。

7. TCP 流量控制

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）”。



7. TCP 流量控制

- 持续计时器 (persistence timer)
  - TCP 为每一个连接设有一个持续计时器。
  - 只要 TCP 连接的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。

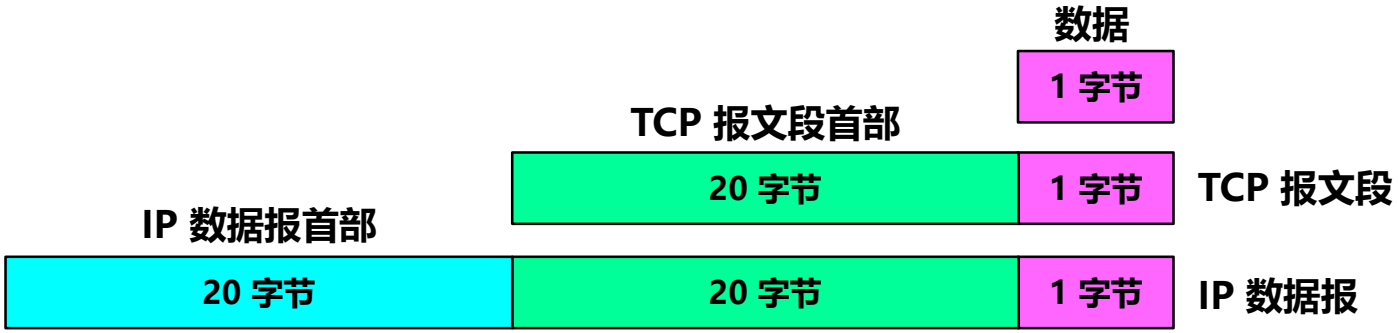
7. TCP 流量控制

- 应用进程把数据传送到 TCP 的发送缓存后，剩下的发送任务就由 TCP 具体控制。
- 如何控制 TCP 发送报文段的时机是非常复杂的，TCP 有不同的机制来控制 TCP 报文段的发送时机。
  - 第一种机制：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：由发送方的应用进程指明要求发送报文段，即 TCP 支持推送 (push) 操作。
  - 第三种机制：发送方的一个计时器期限到了，这时就把当前已有的缓存数据装入报文段（但长度不能超过 MSS）发送出去。

7. TCP 流量控制

糊涂窗口综合症

糊涂窗口综合症：每次仅发送一个字节或很少几个字节的数据时，有效数据传输效率变得很低的现象。

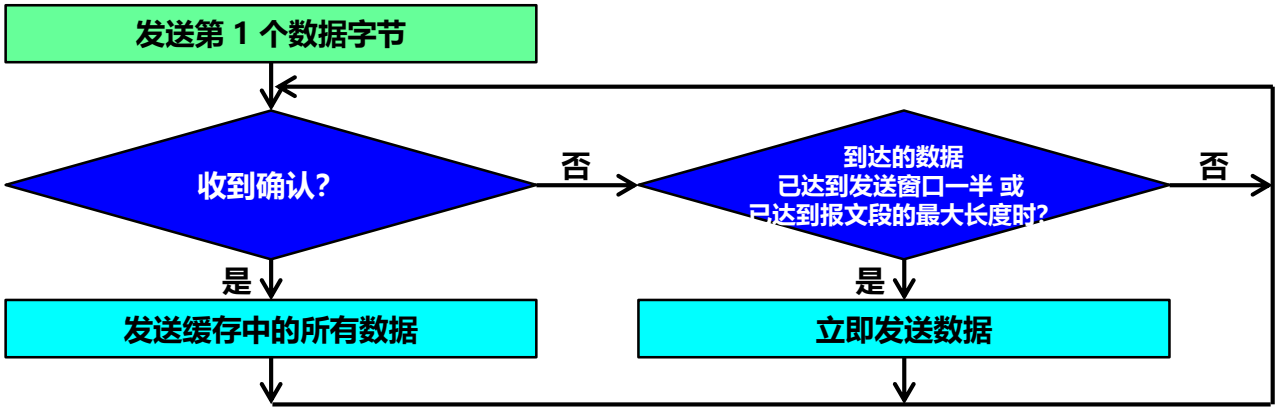


此时，有效数据传输效率 = 1/41 = 2.44%

7. TCP 流量控制

发送方糊涂窗口综合症

- 发送方 TCP **每次接收到一字节**的数据后就发送。
- 发送一个字节需要形成 41 字节长的 IP 数据报，效率很低。
- **解决方法**：使用 Nagle 算法。



7. TCP 流量控制

接收方糊涂窗口综合症

- 原因：接收方应用进程消耗数据太慢，例如：每次只读取一个字节。

不允许 A 再发送（到序号 600 为止的数据都收到了）

应用进程读取 1 字节。接收缓存大小 = 1 字节

允许 A 发送序号 601 至 601 共 1 字节

A 发送了序号 601 至 601 字节，不能再发送了

应用进程读取 1 字节。接收缓存大小 = 1 字节

允许 A 发送序号 602 至 602 共 1 字节

A 发送了序号 602 至 602 字节，不能再发送了

7. TCP 流量控制

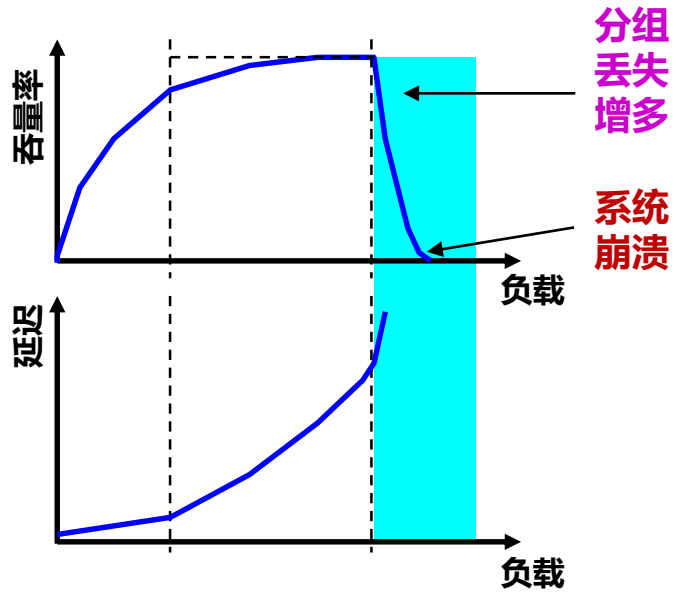
接收方糊涂窗口综合症

- 原因：接收方应用进程消耗数据太慢，例如：每次只读取一个字节。
- **解决方法：**让接收方等待一段时间，使得或者接收缓存已有足够空间容纳一个最长的报文段，或者等到接收缓存已有一半空闲的空间。只要出现这两种情况之一，接收方就发出确认报文，并向发送方通知当前的窗口大小。

上述两种方法可配合使用，使得发送方不发送很小的报文段的同时，接收方也不要在缓存刚刚有了一点小的空间就急忙把这个很小的窗口大小信息通知给发送方。

8. TCP 拥塞控制

- 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。这种情况就叫做**拥塞** (congestion)。
- 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。
- 最坏结果：系统崩溃。



8. TCP 拥塞控制

- 拥塞产生的原因：
  - 由许多因素引起。
  - 例如：
    - 节点缓存容量太小；
    - 链路容量不足；
    - 处理机处理速率太慢；
    - 拥塞本身会进一步加剧拥塞；
- 出现网络拥塞的条件是：

对资源需求的总和 > 可用资源

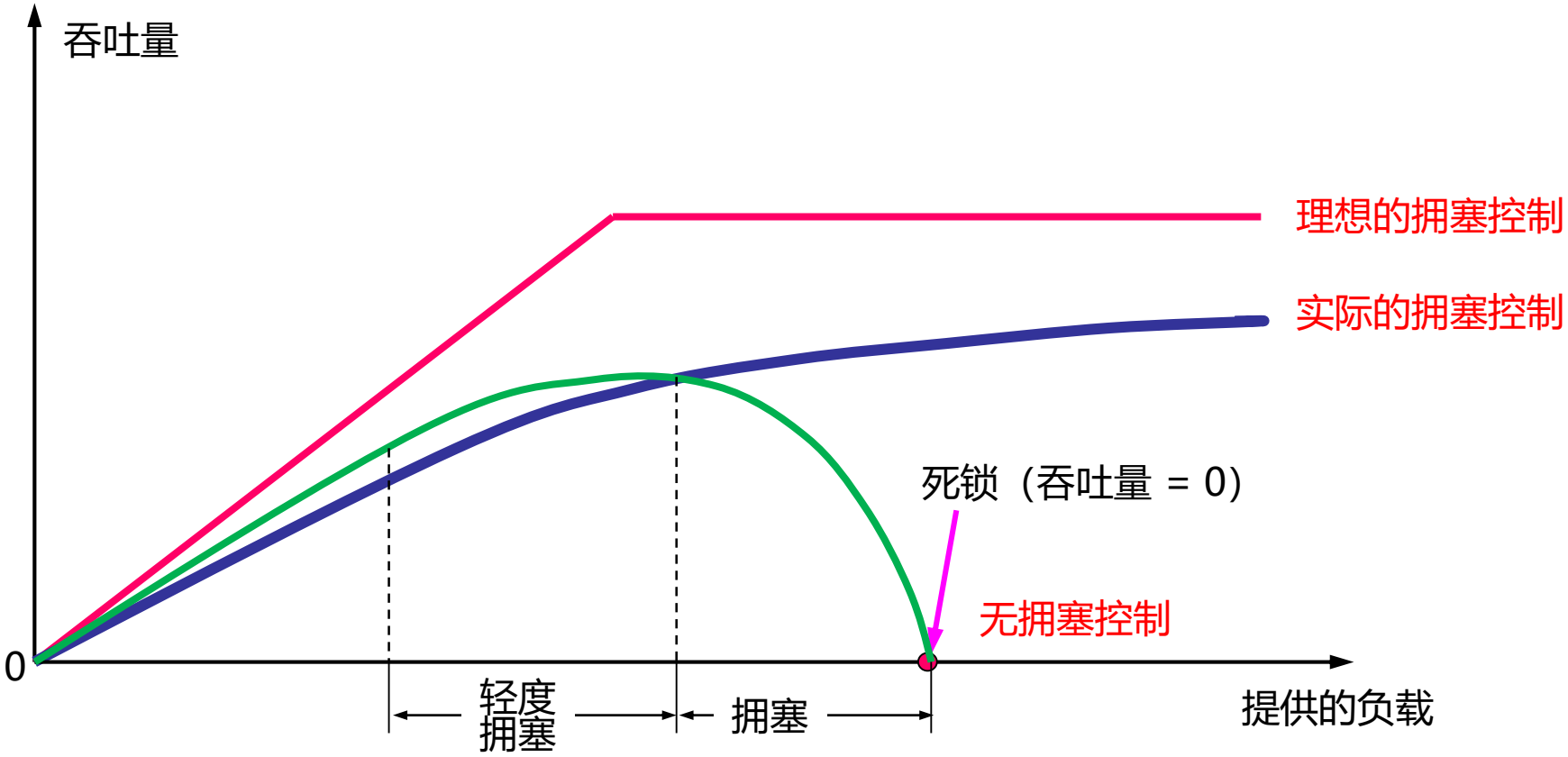
8. TCP 拥塞控制

- 网络拥塞往往是由许多因素引起的，是一个非常复杂的问题，需要综合考虑和分析。
  - 有人说："只要任意增加一些资源，例如把节点缓存的存储空间扩大，或把链路更换为更高速率的链路，或把结点处理机的运算速度提高，就可以解决网络拥塞的问题"。
  - 其实，简单的采用上述的做法，在许多情况下，不但不能解决拥塞问题，而且还可能使网络的性能变得更差。

8. TCP 拥塞控制

- 拥塞控制与流量控制的关系非常密切，但是也存在一定的差别。
  - 拥塞控制所要做的有一个前提，就是网络能够承受现有网络负荷。
  - 拥塞控制是一个全局性的过程，涉及到所有的主机、所有的路由器，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。

8. TCP 拥塞控制



8. TCP 拥塞控制

- 拥塞控制是很难设计的，因为它是一个动态的（而不是静态的）问题。
- 当前网络正朝着高速化的方向发展，这很容易出现缓存不够大而造成分组的丢失。
  - 注意：分组的丢失是网络发生拥塞的征兆，但不是原因。
- 在许多情况下，甚至正是拥塞控制本身成为引起网络性能恶化甚至发生死锁的原因。
  - 这点应特别引起重视。

8. TCP 拥塞控制

- 从大的方面来看，拥塞控制可分为开环控制和闭环控制两种方法。
  - 开环控制方法就是在设计网络时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。
  - 闭环控制是基于反馈环路的概念。属于闭环控制的措施：
    - 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。

8. TCP 拥塞控制

- 从大的方面来看，拥塞控制可分为开环控制和闭环控制两种方法。

开环控制	闭环控制
• 在设计网络时，事先考虑周全，力求工作时不发生拥塞。 • 思路：力争避免发生拥塞。 • 但一旦整个系统运行起来，就不再中途进行改正了。	• 基于反馈环路的概念。 • 根据网络当前运行状态采取相应控制措施。 • 思路：在发生拥塞后，采取措施进行控制，消除拥塞。

8. TCP 拥塞控制

闭环控制措施

1，监测

监测网络系统，检测拥塞在何时、何处发生。

2，传送

将拥塞发生的信息传送到可采取行动的地方。

3，调整

调整网络系统的运行以解决出现的问题。

8. TCP 拥塞控制

闭环控制措施

1，监测

监测网络系统，检测拥塞在何时、何处发生。

2，传送

将拥塞发生的信息传送到可采取行动的地方。

3，调整

调整网络系统的运行以解决出现的问题。

主要指标有：

① 由于缺少缓存空间而被丢弃的分组的百分数；

② 平均队列长度；

③ 超时重传的分组数；

④ 平均分组时延；

⑤ 分组时延的标准差，等等。

这些指标的上升都标志着拥塞的增长。

8. TCP 拥塞控制

闭环控制措施

1，监测

监测网络系统，检测拥塞在何时、何处发生。

2，传送

将拥塞发生的信息传送到可采取行动的地方。

3，调整

调整网络系统的运行以解决出现的问题。

1. 将拥塞发生的信息传送到产生分组的源站。
2. 在路由器转发的分组中保留一个比特或字段，用该比特或字段的值表示网络没有拥塞或产生了拥塞。
3. 周期性地发出探测分组等。

8. TCP 拥塞控制

闭环控制措施

1，监测

监测网络系统，检测拥塞在何时、何处发生。

2，传送

将拥塞发生的信息传送到可采取行动的地方。

3，调整

调整网络系统的运行以解决出现的问题。

1. 过于频繁，会使系统产生不稳定的振荡。
2. 过于迟缓，不具有任何实用价值。
3. 选择正确的时间常数是相当困难的。

8. TCP 拥塞控制

- TCP 采用基于滑动窗口的方法进行拥塞控制，属于闭环控制方法。
- TCP 发送方维持一个拥塞窗口 cwnd (Congestion Window)
- 拥塞窗口的大小取决于网络的拥塞程度，并且是动态变化的。
- 发送端利用拥塞窗口根据网络的拥塞情况调整发送的数据量。
- 发送窗口大小不仅取决于接收方窗口，还取决于网络的拥塞状况。
- 真正的发送窗口值：

真正的发送窗口值 = Min (接收方通知的窗口值，拥塞窗口值)

8. TCP 拥塞控制

- 控制拥塞窗口变化的原则
  - 只要网络没有出现拥塞，拥塞窗口就可以再增大一些，以便把更多的分组发送出去，提高网络的利用率。
  - 但只要网络出现拥塞或有可能出现拥塞，就必须把拥塞窗口减小一些，以减少注入到网络中的分组数，缓解网络出现的拥塞。
- 发送方判断拥塞的方法：隐式反馈

超时重传计时器超时	**• 网络已经出现了拥塞。**
收到 3 个重复的确认	**• 预示网络可能会出现拥塞。**

因传输出差错而丢弃分组的概率很小（远小于1 %）。

因此，发送方在超时重传计时器启动时，就判断网络出现了拥塞。

8. TCP 拥塞控制

- 1999年公布的因特网建议标准RFC 2581定义了进行拥塞控制的四种算法：
  - 慢开始（slow-start）
  - 拥塞避免（congestion avoidance）
  - 快重传（fast retransmit）
  - 快恢复（fast recovery）。
- 之后RFC 2582和RFC 3390又对这些算法进行了一些改进。
  - 在讨论拥塞控制时，为了讨论方便做如下假定：
    - 数据是单方向发送，而另一个方向只传送确认；
    - 接收方总是有足够大的缓存空间，因而发送窗口的大小由网络的拥塞程度来决定。

8. TCP 拥塞控制

- 慢开始 (Slow start)
  - 目的：探测网络的负载能力或拥塞程度。
  - 算法：由小到大逐渐增大注入到网络中的数据字节，即：由小到大逐渐增大拥塞窗口数值。
  - 2 个控制变量：

拥塞窗口 cwnd

- 初始值：2 种设置方法。
  - 1 至 2 个最大报文段 MSS （旧标准）
  - 2 至 4 个最大报文段 MSS（RFC 5681）

慢开始门限 ssthresh

- 防止拥塞窗口增长过大引起网络拥塞。

8. TCP 拥塞控制

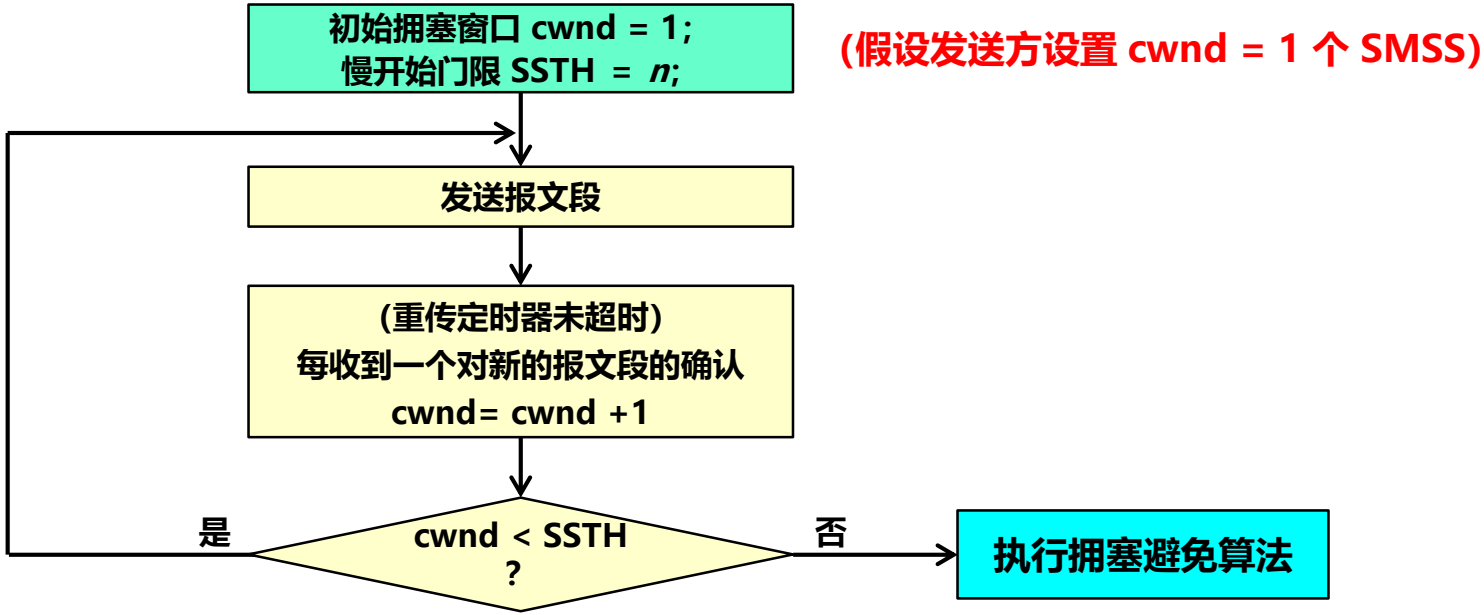
- 慢开始 (Slow start)
  - 拥塞窗口 cwnd 增大：在每收到一个对新的报文段的确认，就把拥塞窗口增加最多一个发送方的最大报文段 SMSS（Sender Maximum Segment Size）的数值。

拥塞窗口 cwnd 每次的增加量 = min (*N*, SMSS)

其中 *N* 是原先未被确认的、但现在被刚收到的确认报文段所确认的字节数。

8. TCP 拥塞控制

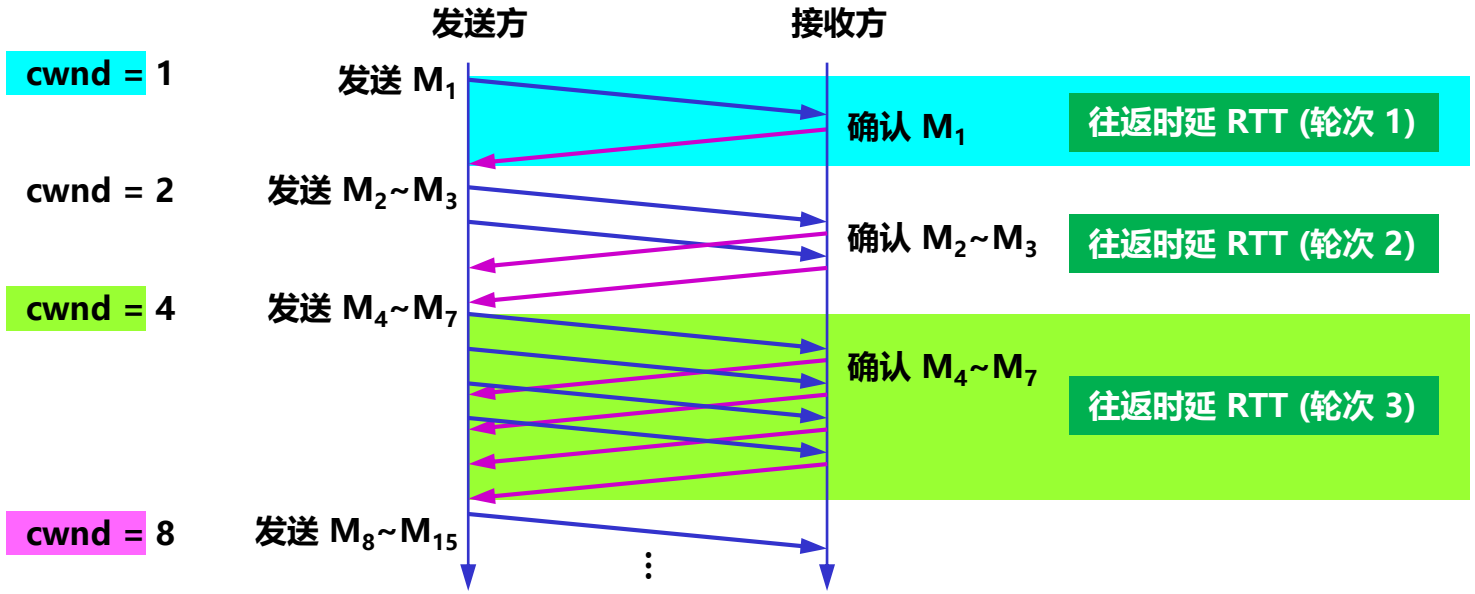
- 慢开始 (Slow start)



8. TCP 拥塞控制

- 慢开始 (Slow start)

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。



8. TCP 拥塞控制

- 慢开始 (Slow start)
  - 传输轮次（transmission round）
    - 一个传输轮次所经历的时间其实就是往返时间 RTT。
    - 传输轮次强调：把拥塞窗口 cwnd 所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。
    - 例如：拥塞窗口 cwnd = 4，这时的往返时间 RTT 就是发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间。

8. TCP 拥塞控制

- 慢开始 (Slow start)
  - 慢开始门限 ssthresh
    - 防止拥塞窗口 cwnd 增长过大引起网络拥塞。
    - 用法：
      - 当 cwnd < ssthresh 时，使用慢开始算法。
      - 当 cwnd > ssthresh 时，停止使用慢开始算法，改用拥塞避免算法。
      - 当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

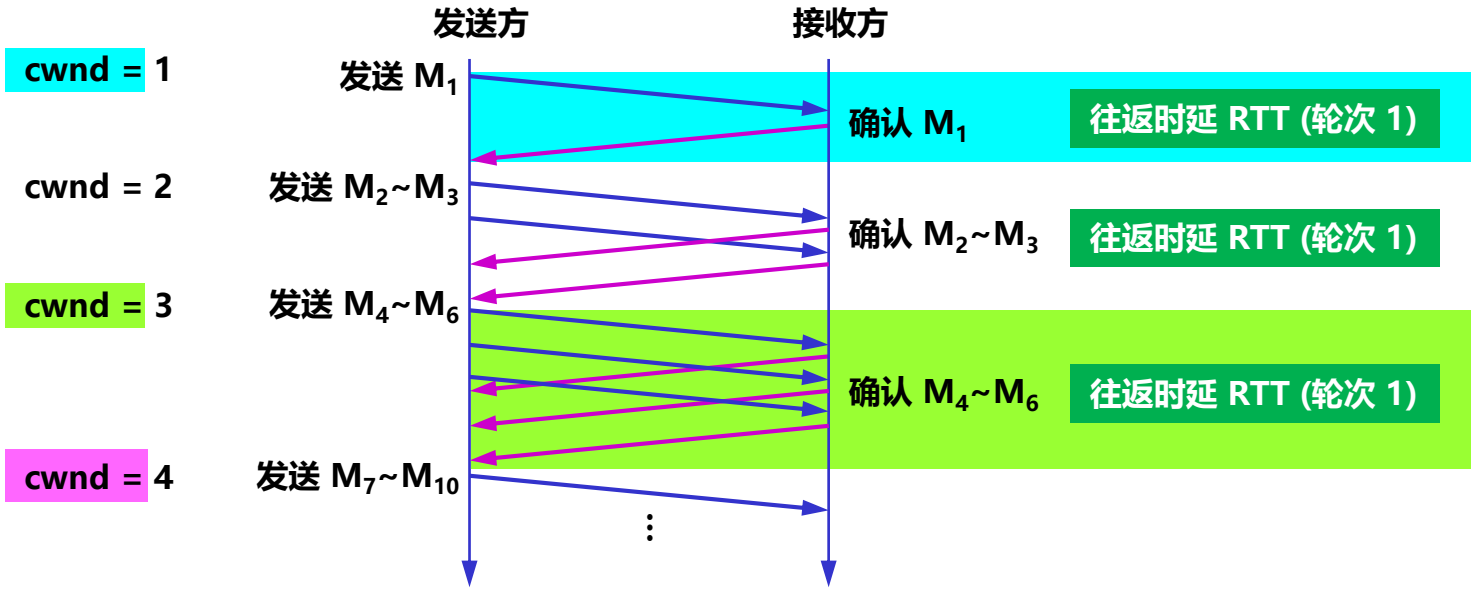
8. TCP 拥塞控制

- 拥塞避免
  - 目的：让拥塞窗口 cwnd 缓慢地增大，避免出现拥塞。
  - 拥塞窗口 cwnd 增大：每经过一个往返时间 RTT（不管在此期间收到了多少确认），发送方的拥塞窗口 cwnd = cwnd + 1。
  - 具有加法增大 AI (Additive Increase) 特点：使拥塞窗口 cwnd 按线性规律缓慢增长。
    - 拥塞避免并非完全避免拥塞，而是让拥塞窗口增长得缓慢些，使网络不容易出现拥塞。

8. TCP 拥塞控制

- 拥塞避免

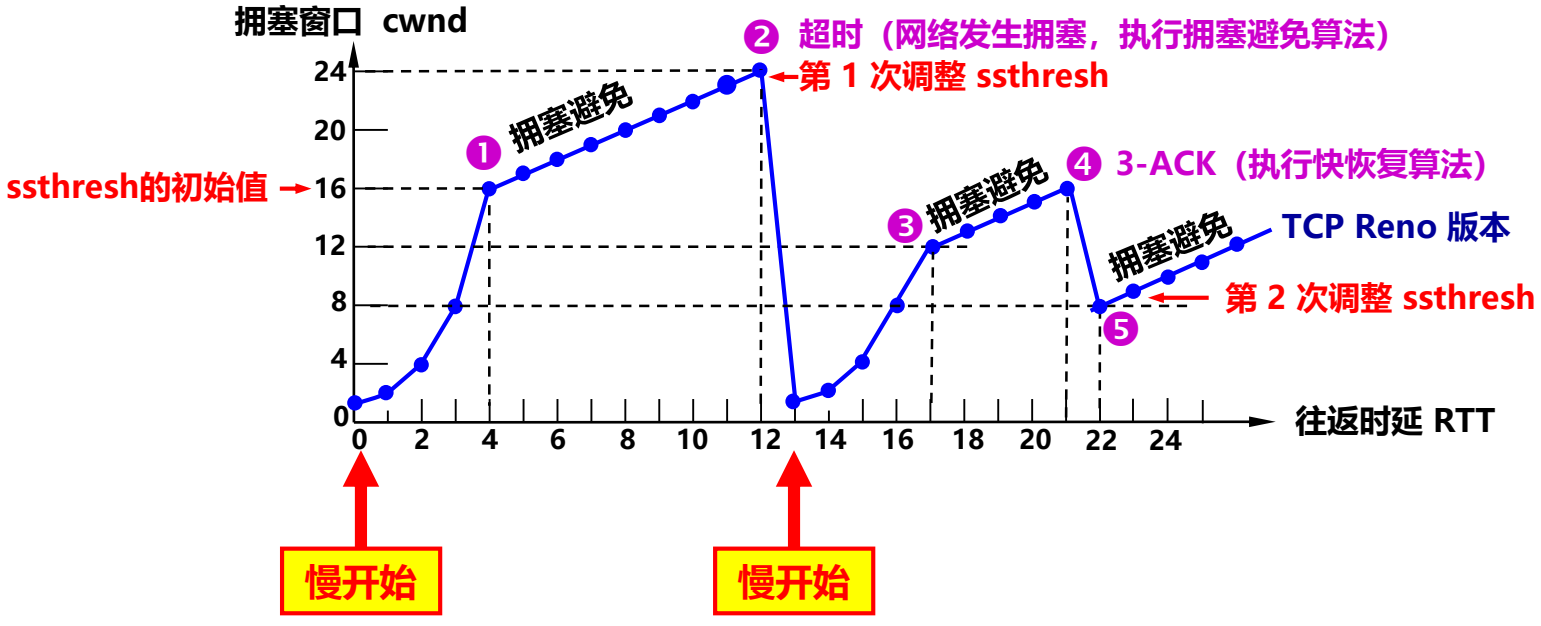
每经过一个往返时间 RTT，发送方就把拥塞窗口 cwnd 加 1。



8. TCP 拥塞控制

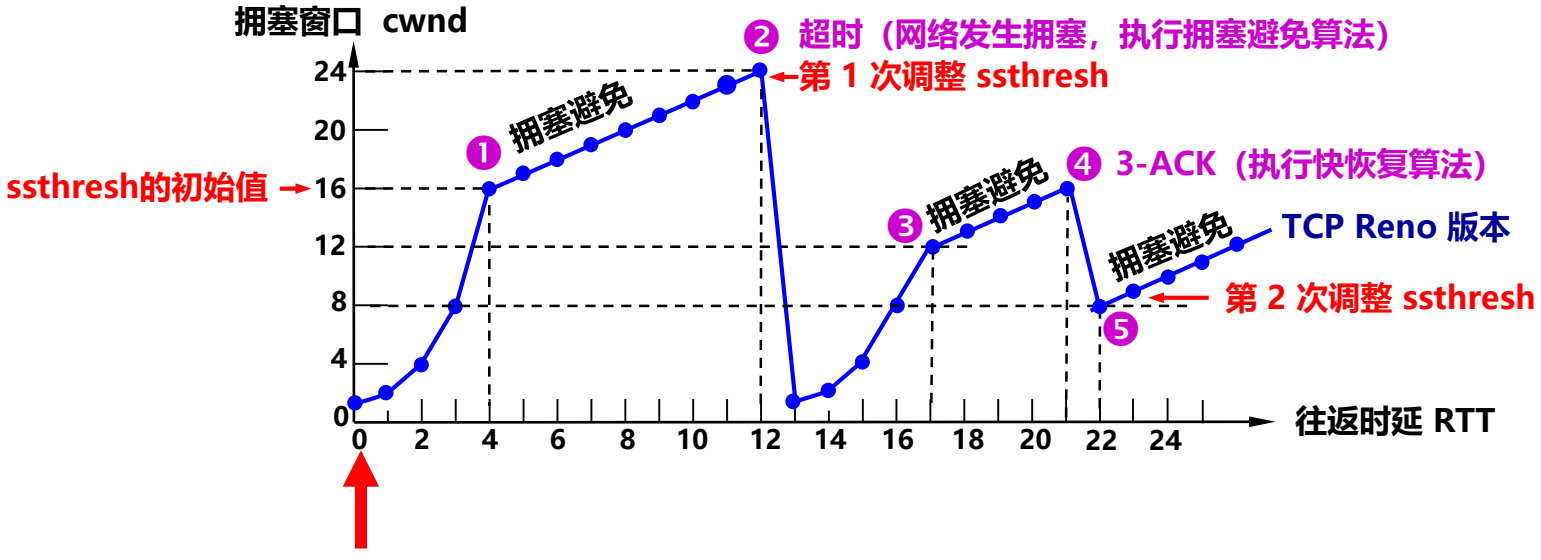
- 慢开始和拥塞避免：
  - 无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（重传定时器超时）：
    1. ssthresh = max (cwnd/2，2)
    2. cwnd = 1
    3. 执行慢开始算法
  - 目的：迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

慢开始和拥塞避免算法的实现举例



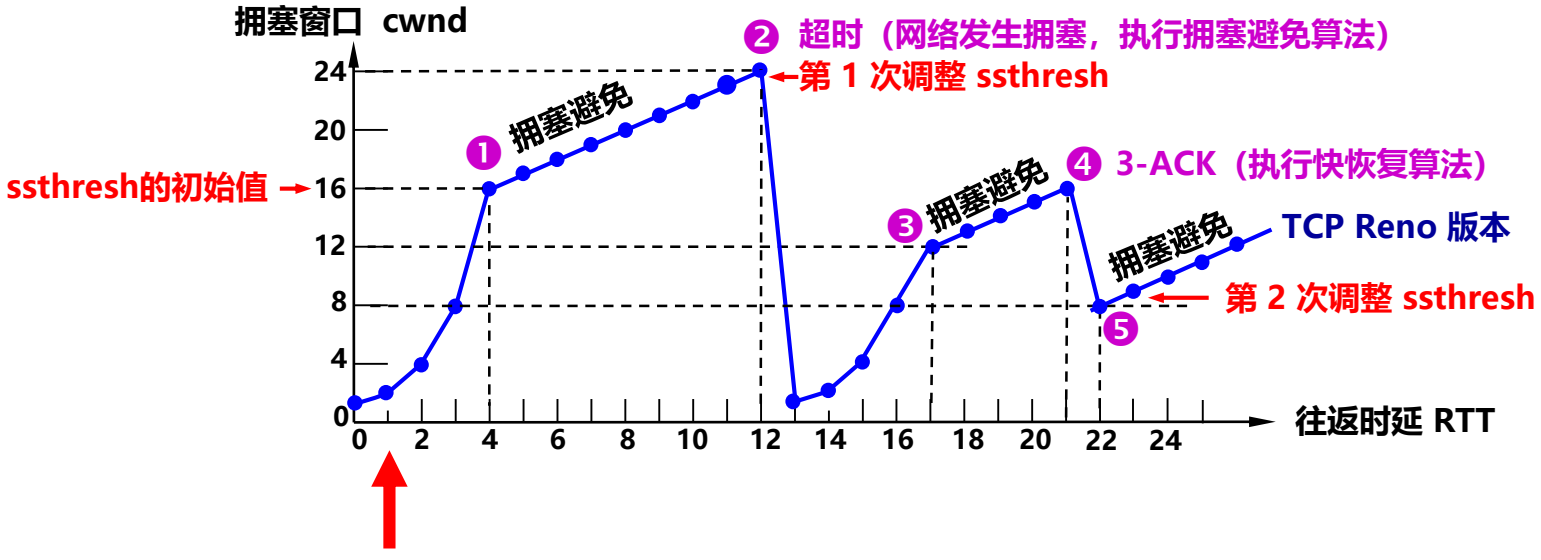
- 当 TCP 连接进行初始化时，将拥塞窗口置为 1（窗口单位不使用字节而使用报文段）。
- 将慢开始门限的初始值设置为 16 个报文段，即 ssthresh = 16。

慢开始和拥塞避免算法的实现举例



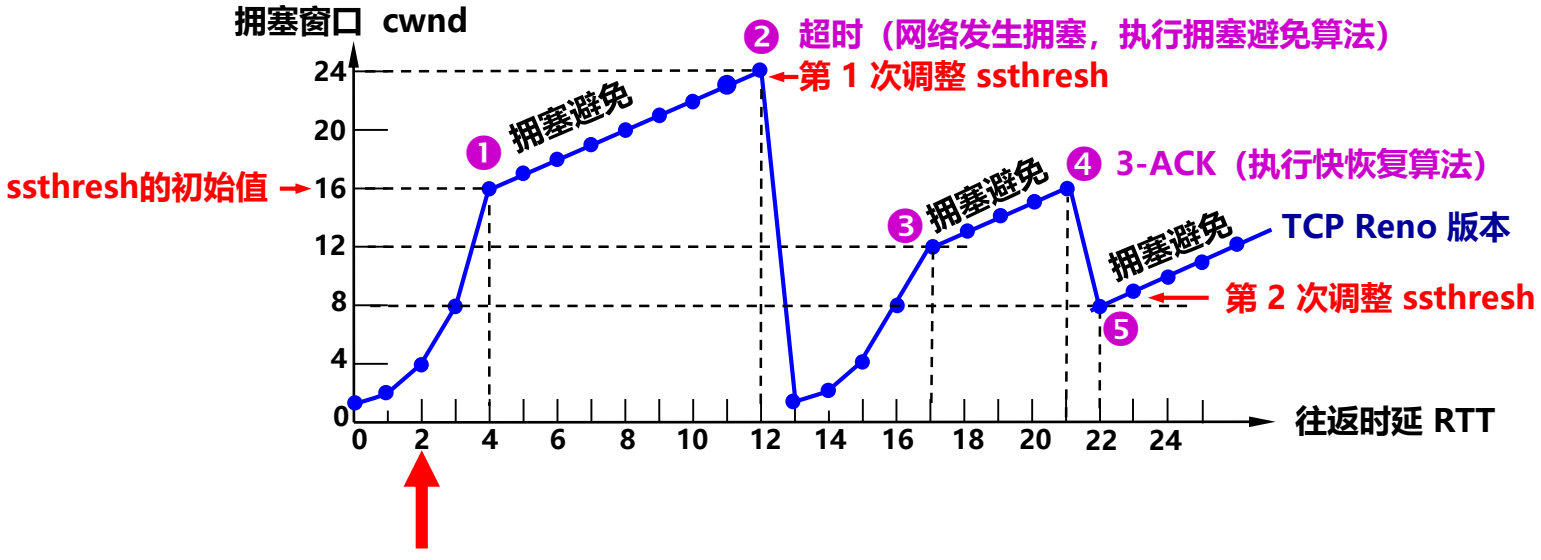
- 开始执行慢开始算法时，拥塞窗口 cwnd=1，发送第一个报文段。

慢开始和拥塞避免算法的实现举例



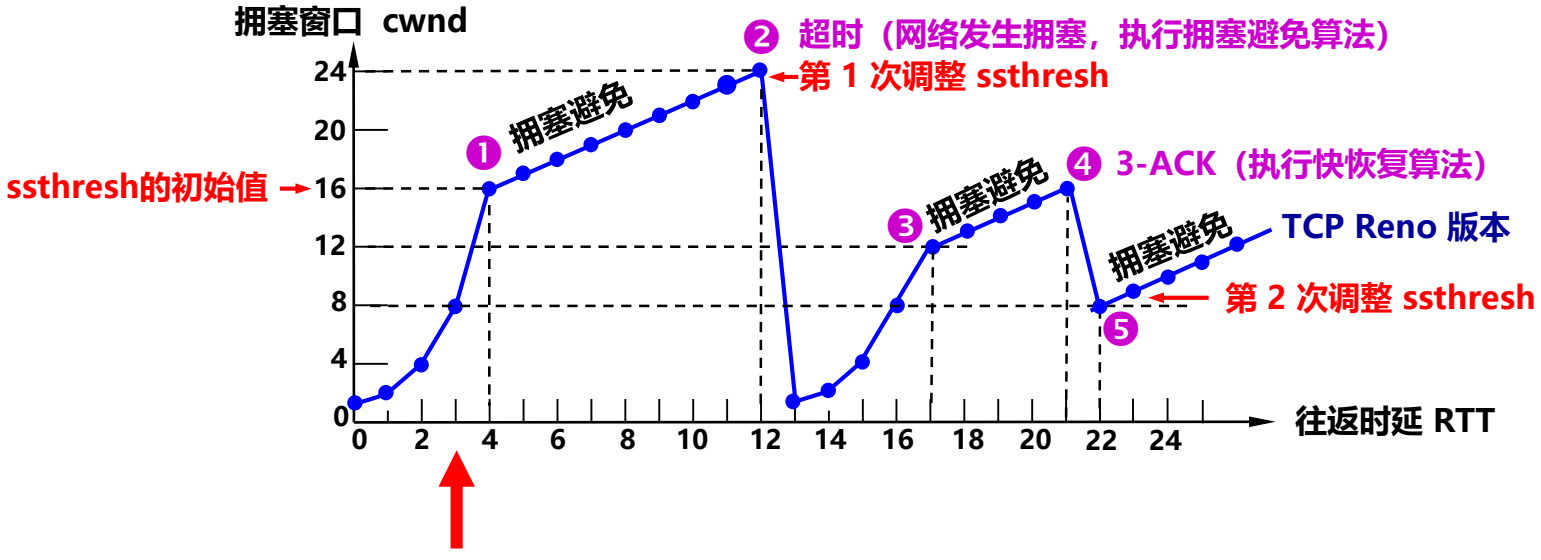
- 发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1，因此拥塞窗口 cwnd 随着往返时延 RTT 按指数规律增长。

慢开始和拥塞避免算法的实现举例



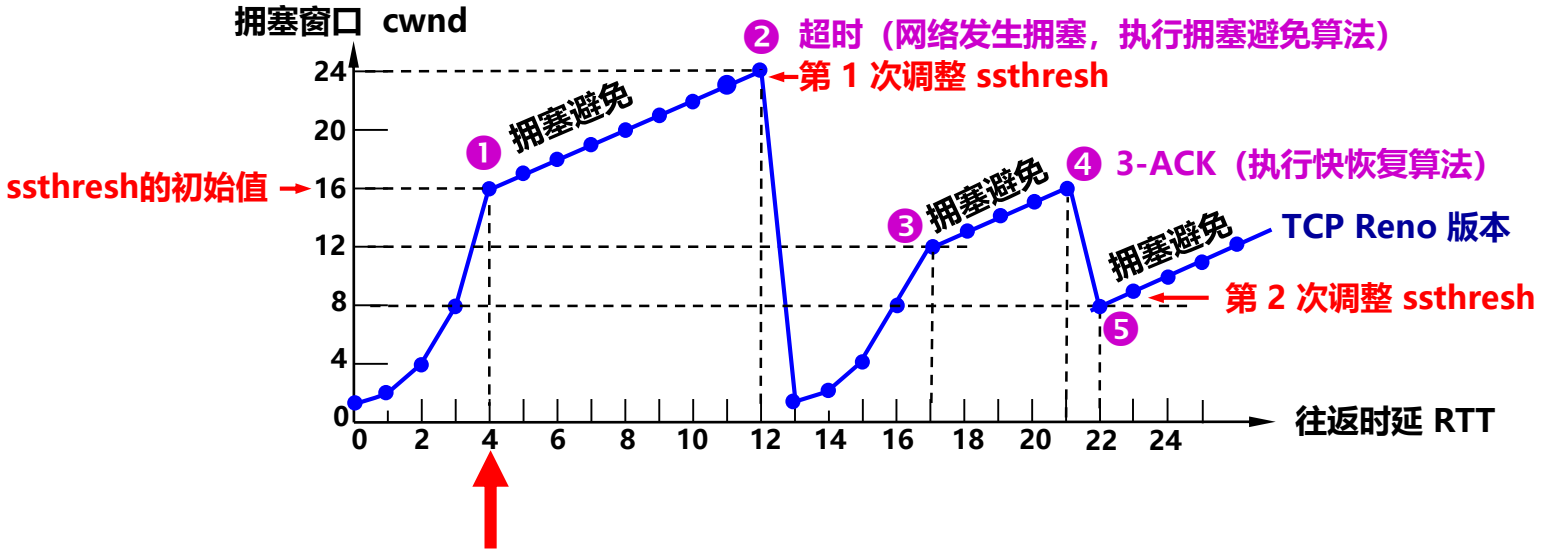
- 发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1，因此拥塞窗口 cwnd 随着往返时延 RTT 按指数规律增长。

慢开始和拥塞避免算法的实现举例



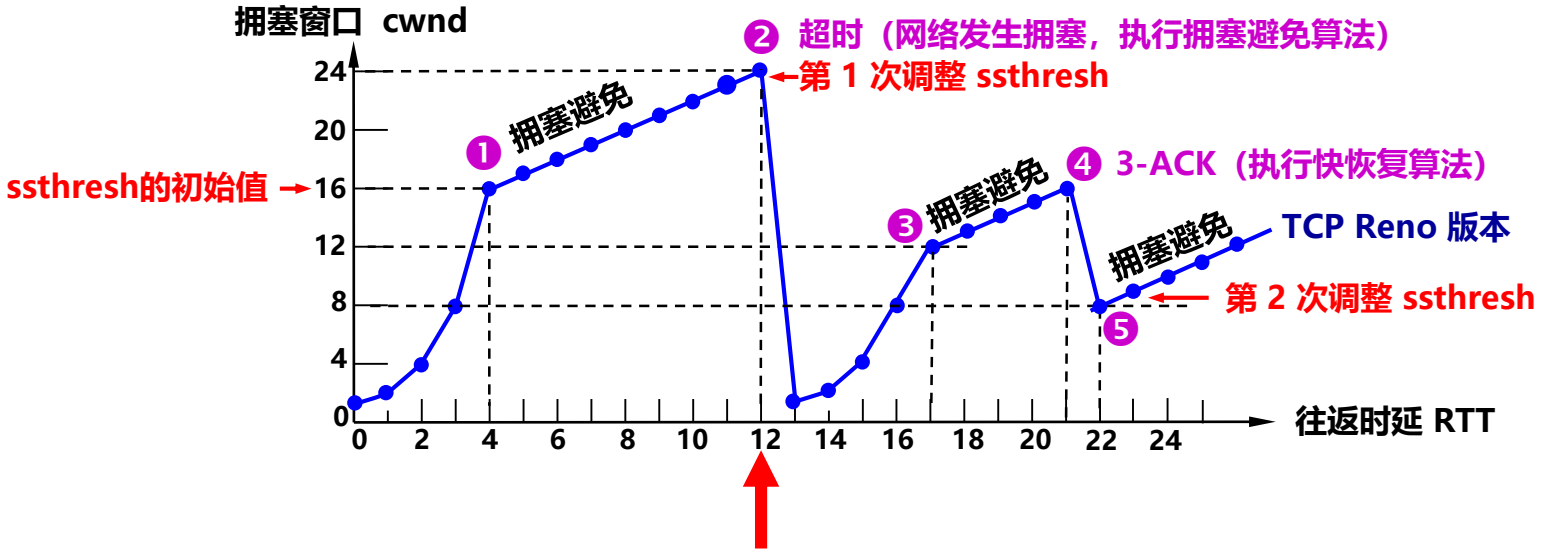
- 发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1，因此拥塞窗口 cwnd 随着往返时延 RTT 按指数规律增长。

慢开始和拥塞避免算法的实现举例



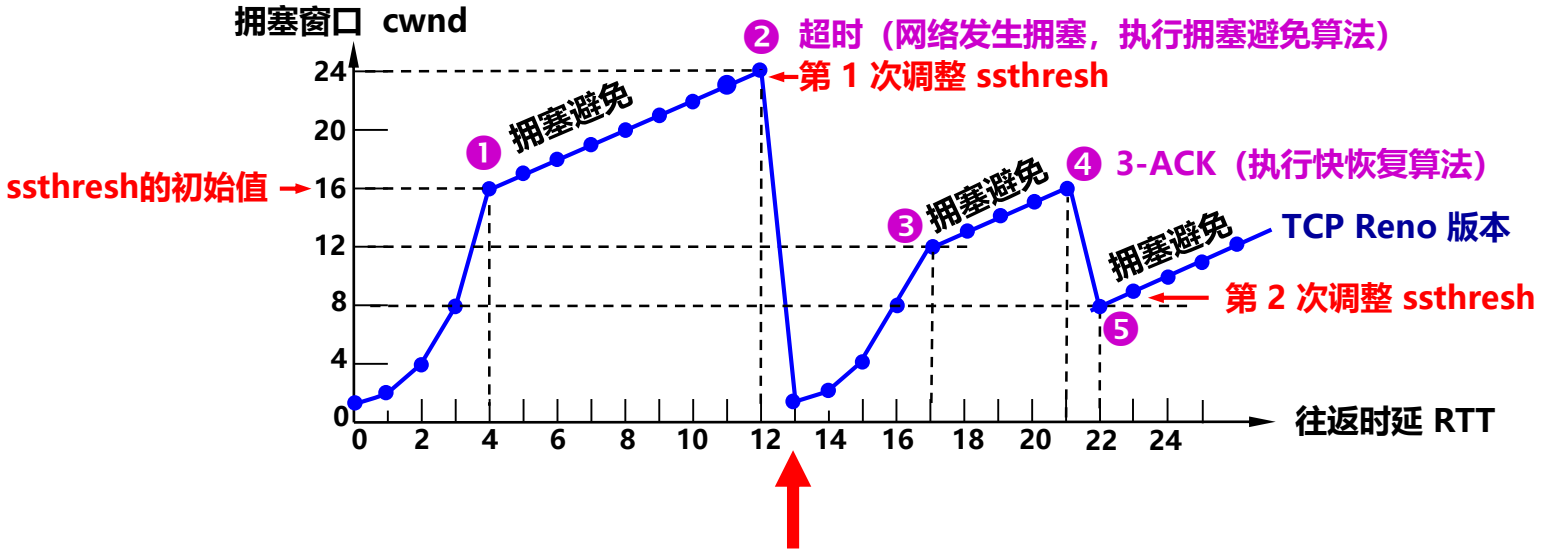
- 当拥塞窗口 cwnd 增长到慢开始门限值 ssthresh 时，改为执行拥塞避免算法，拥塞窗口按线性规律增长。

慢开始和拥塞避免算法的实现举例



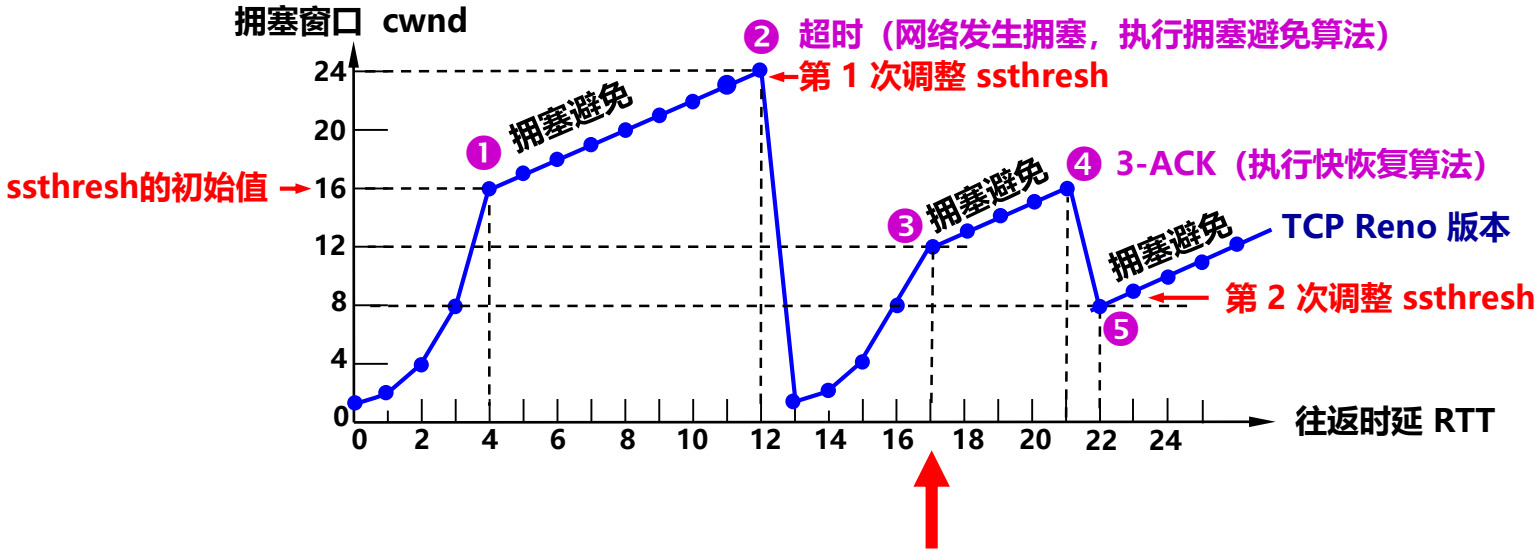
- 当拥塞窗口 cwnd = 24 时，网络出现了超时，发送方判断为网络拥塞。
- 调整门限值 ssthresh = cwnd / 2 = 12，同时设置拥塞窗口 cwnd = 1，进入慢开始阶段。

慢开始和拥塞避免算法的实现举例



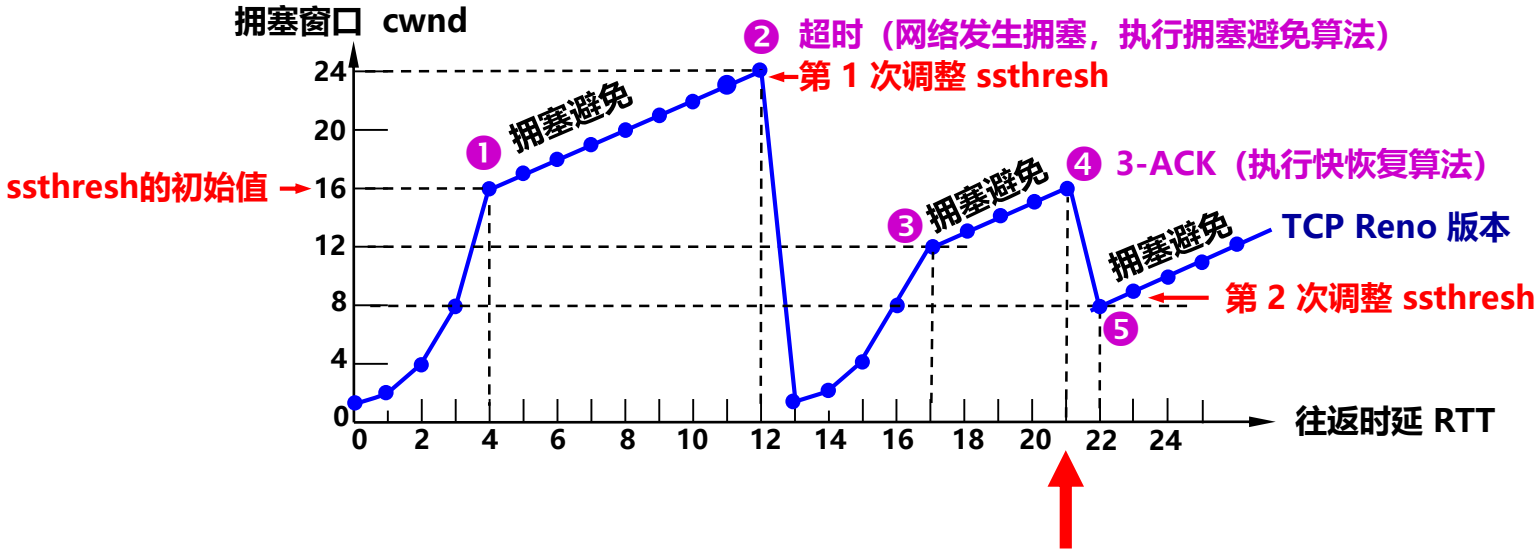
- 当拥塞窗口 cwnd = 24 时，网络出现了超时，发送方判断为网络拥塞。
- 调整门限值 ssthresh = cwnd / 2 = 12，同时设置拥塞窗口 cwnd = 1，进入慢开始阶段。

慢开始和拥塞避免算法的实现举例



- 按照慢开始算法，发送方每收到一个对新报文段的确认 ACK，就把拥塞窗口值加 1。
- 当拥塞窗口 cwnd = ssthresh = 12 时，改为执行拥塞避免算法，拥塞窗口按线性规律增大。

慢开始和拥塞避免算法的实现举例



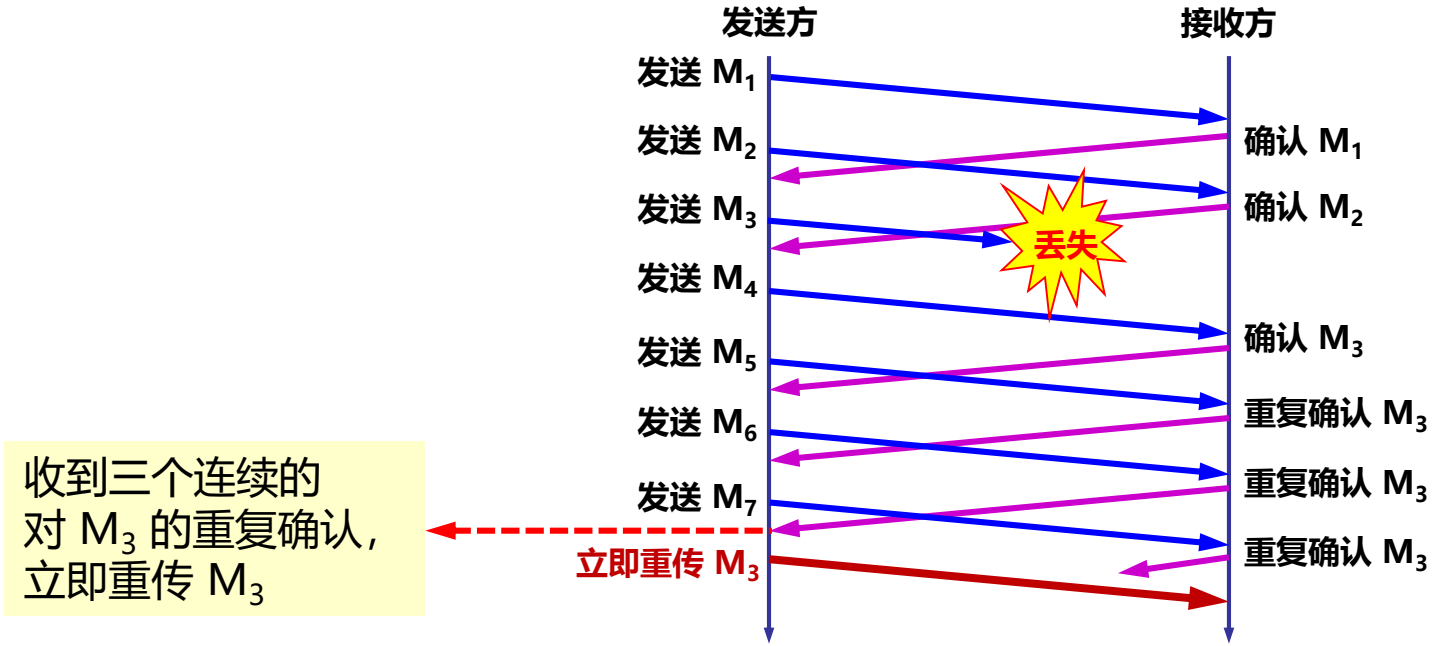
- 当拥塞窗口 cwnd = 16 时，发送方连续收到 3 个对同一个报文段的重复确认（记为 3-ACK）。
- 发送方改为执行快重传和快恢复算法。

8. TCP 拥塞控制

- 快重传 FR (Fast Retransmission) 算法
  - 目的：让发送方尽早知道发生了个别报文段的丢失。
  - 发送方只要连续收到三个重复的确认，就立即进行重传（即“快重传”），这样就不会出现超时。
  - 使用快重传可以使整个网络的吞吐量提高约 20%。
  - 快重传算法要求接收方立即发送确认，即使收到了失序的报文段，也要立即发出对已收到的报文段的重复确认。
  - 快重传并非取消重传计时器，而是在某些情况下可以更早地（更快地）重传丢失的报文段。

8. TCP 拥塞控制

- 快重传 FR (Fast Retransmission) 算法



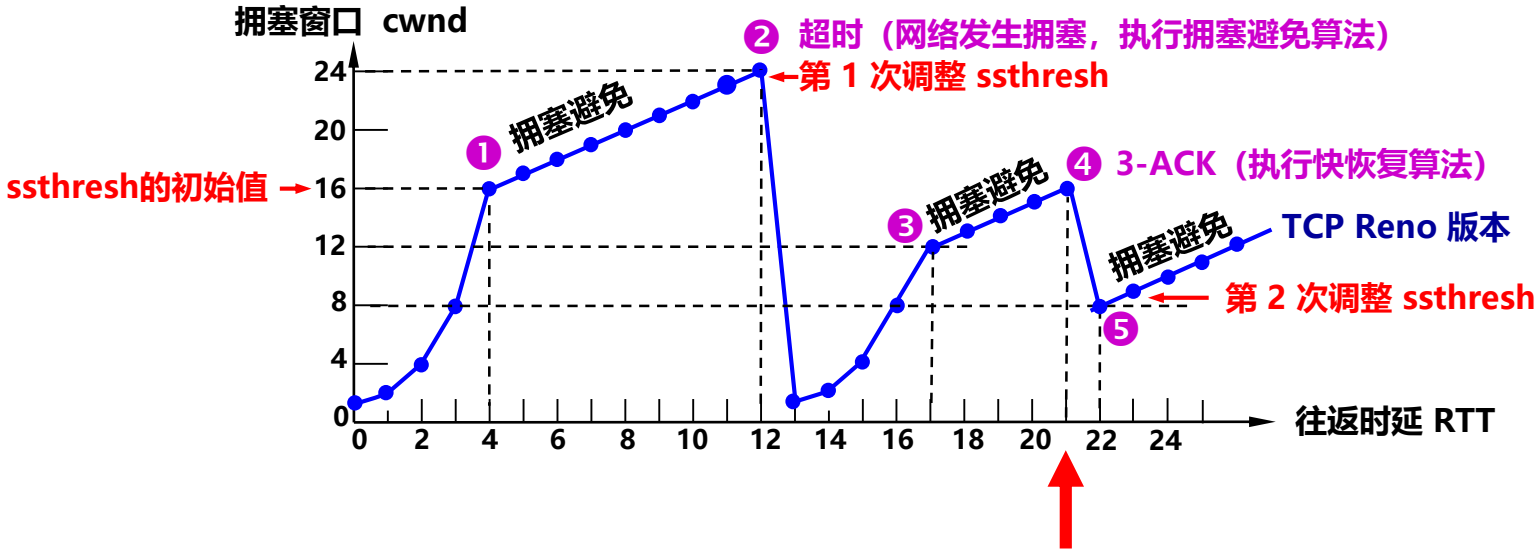
8. TCP 拥塞控制

- 快恢复 FR (Fast Recovery)算法
  - 当发送端收到连续三个重复的确认时，不执行慢开始算法，而是执行快恢复算法 FR (Fast Recovery) 算法：
    - 慢开始门限 ssthresh = 当前拥塞窗口 cwnd / 2 ；
    - 乘法减小 MD (Multiplicative Decrease) 拥塞窗口。

      新拥塞窗口 cwnd = 慢开始门限 ssthresh
    - 执行拥塞避免算法，使拥塞窗口缓慢地线性增大（加法增大 AI）。

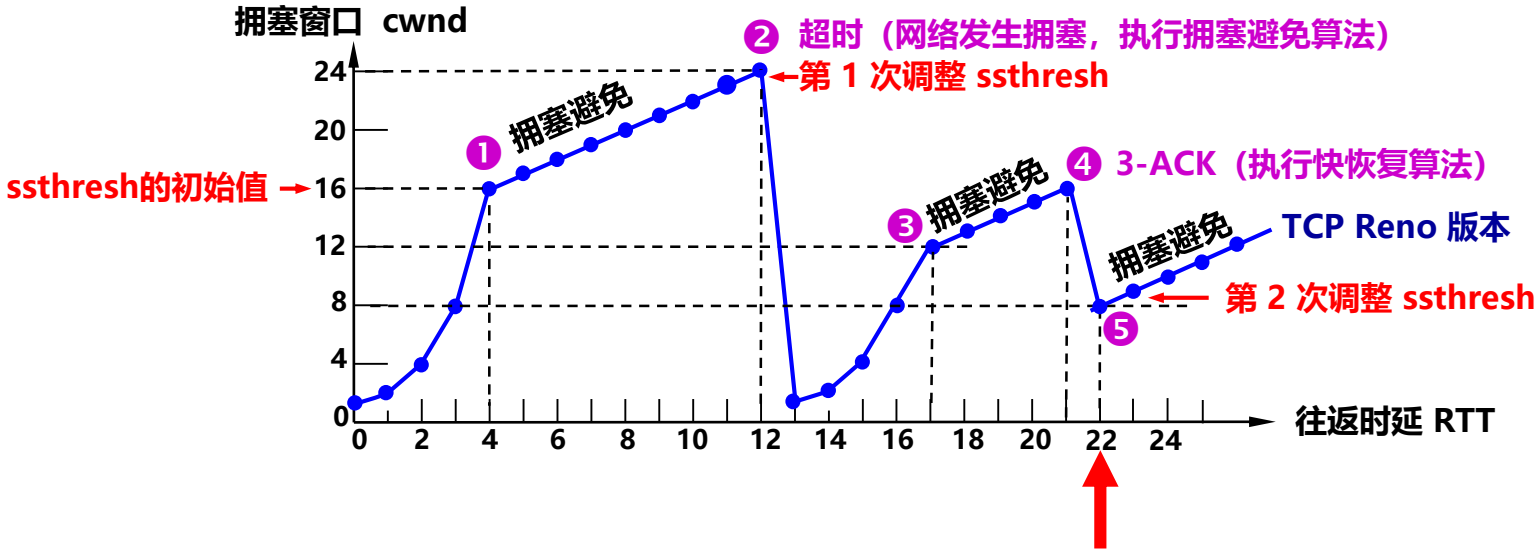
快重传算法和快恢复算法
合在一起就是所谓的 AIMD 算法，使 TCP 性能有明显改进。

慢开始和拥塞避免算法的实现举例



- 当拥塞窗口 cwnd = 16 时，发送方连续收到 3 个对同一个报文段的重复确认（记为 3-ACK）。
- 发送方改为执行快重传和快恢复算法。

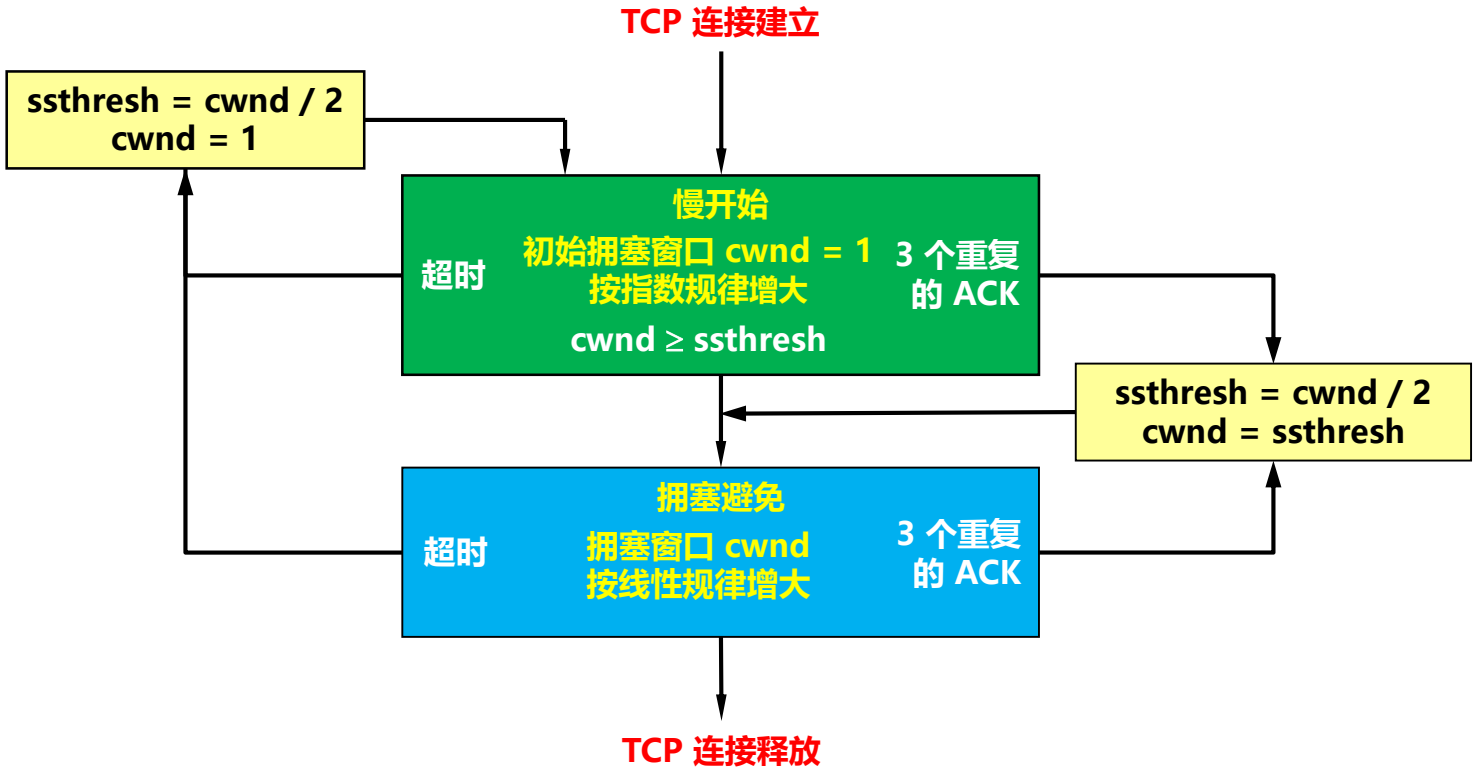
慢开始和拥塞避免算法的实现举例



- 执行快重传和快恢复算法：
- 发送方调整门限值 ssthresh = cwnd / 2 = 8，设置拥塞窗口 cwnd = ssthresh = 8，开始执行拥塞避免算法。

8. TCP 拥塞控制

- TCP 拥塞控制流程图



8. TCP 拥塞控制

- 发送窗口的上限值

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，是网络拥塞限制发送窗口的最大值。**

8. TCP 拥塞控制

- TCP 拥塞控制和网络层采取的策略有密切联系。
- 例如：
  - 若路由器对某些分组的处理时间特别长，就可能引起发送方 TCP 超时，对这些报文段进行重传。
  - 重传会使 TCP 连接的发送端认为在网络中发生了拥塞，但实际上网络并没有发生拥塞。
- 对 TCP 拥塞控制影响最大的就是路由器的分组丢弃策略。

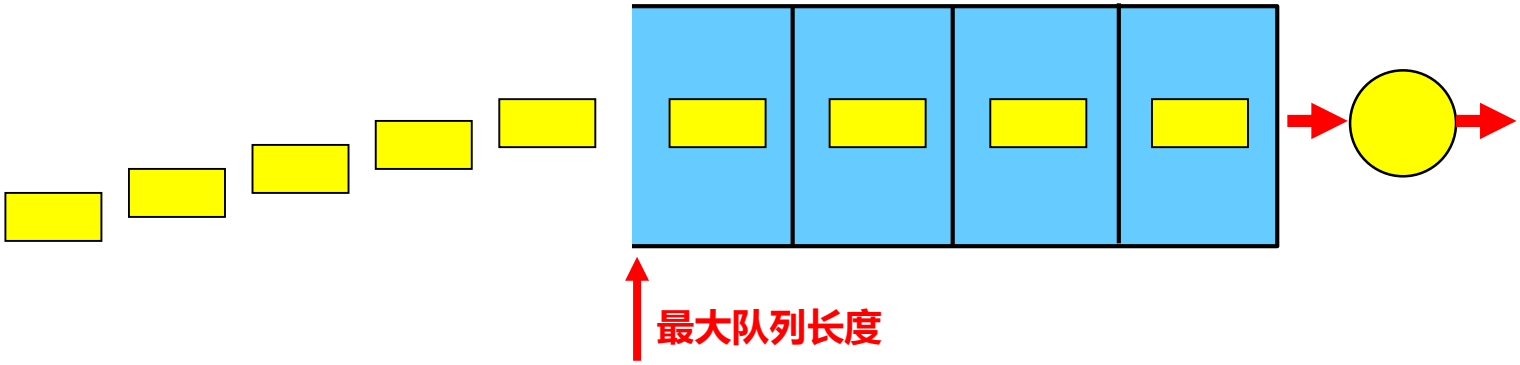
8. TCP 拥塞控制

- “先进先出”FIFO 处理规则
  - 尾部丢弃策略 (tail-drop policy)：
    - 当队列已满时，以后到达的所有分组（如果能够继续排队，这些分组都将排在队列的尾部）将都被丢弃。
  - 路由器的尾部丢弃往往会导致一连串分组的丢失，这就使发送方出现超时重传，使 TCP 进入拥塞控制的慢开始状态，结果使 TCP 连接的发送方突然把数据的发送速率降低到很小的数值。

8. TCP 拥塞控制

- 先进先出（FIFO）处理规则与尾部丢弃策略

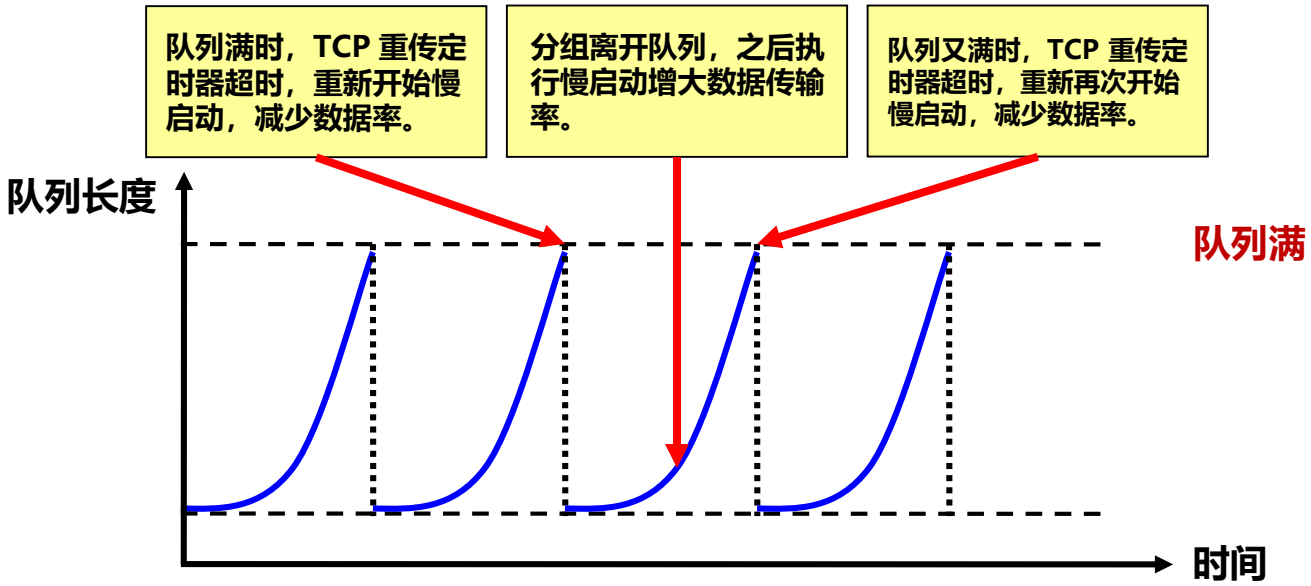
在最简单的情况下，路由器队列通常采用
先进先出 (FIFO) 处理规则与尾部丢弃策略 (tail-drop policy)。
当队列已满时，以后到达的所有分组将都被丢弃。



8. TCP 拥塞控制

- 先进先出（FIFO）处理规则与尾部丢弃策略

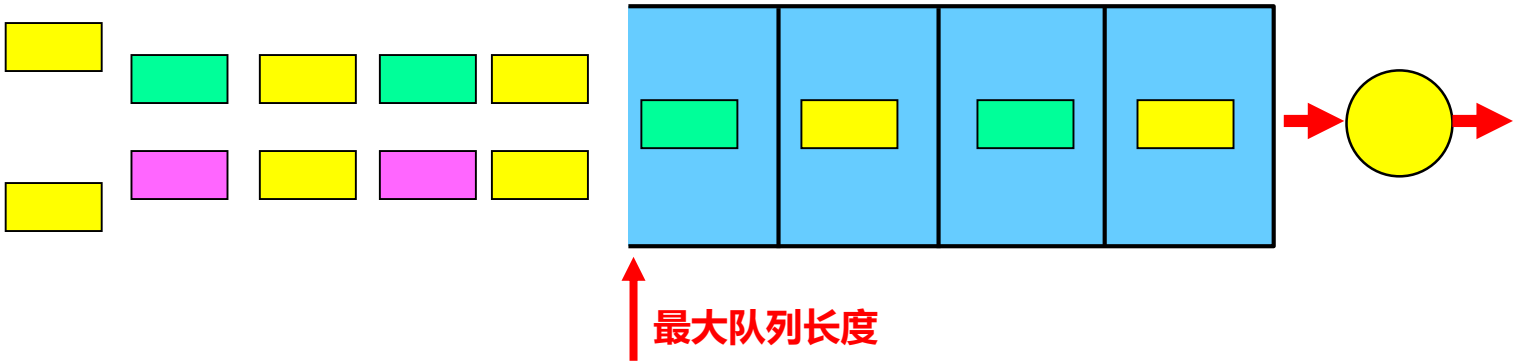
分组丢弃使发送方出现超时重传，使 TCP 连接进入慢开始状态。



8. TCP 拥塞控制

- 严重问题：全局同步

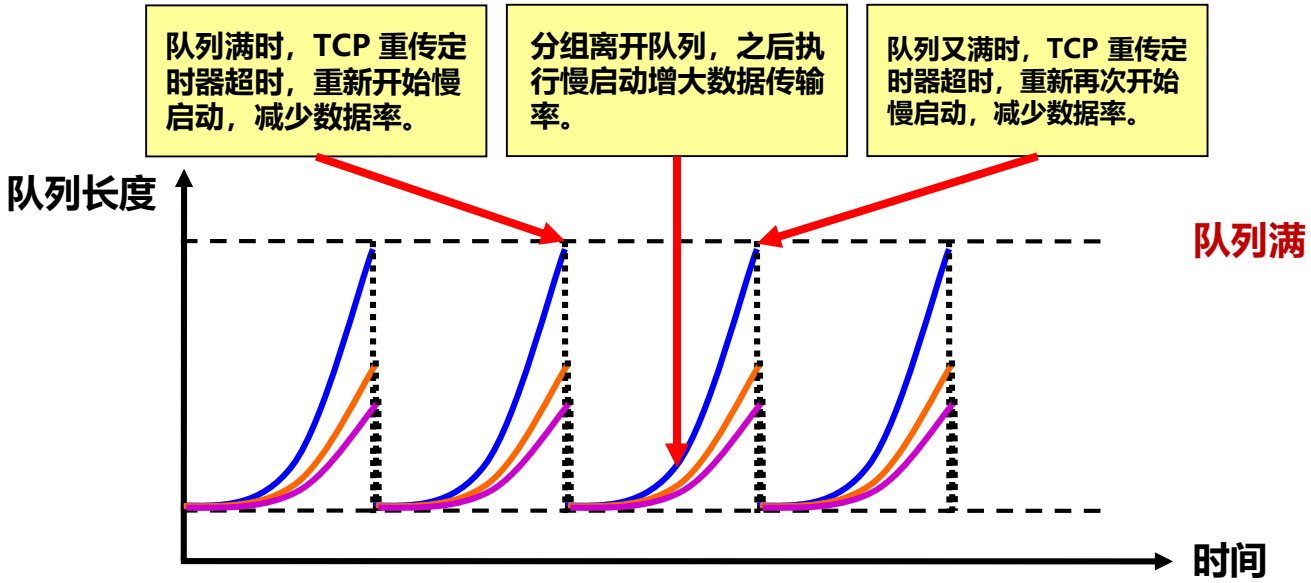
若路由器进行了尾部丢弃，
所有到达的分组都被丢弃，不论它们属于哪个 TCP 连接。



8. TCP 拥塞控制

- 严重问题：全局同步

分组丢弃使发送方出现超时重传，使多个 TCP 连接同时进入慢开始状态，发生全局同步 (global syncronization)。



8. TCP 拥塞控制

- 主动队列管理 AQM
  - 1998 年提出了主动队列管理 AQM (Active Queue Management)。
  - 主动：不要等到路由器的队列长度已经达到最大值时才不得不丢弃后面到达的分组，而是在队列长度达到某个值得警惕的数值时（即当网络拥塞有了某些拥塞征兆时），就主动丢弃到达的分组。
  - AQM 可以有不同实现方法，其中曾流行多年的就是随机早期检测 RED (Random Early Detection)。

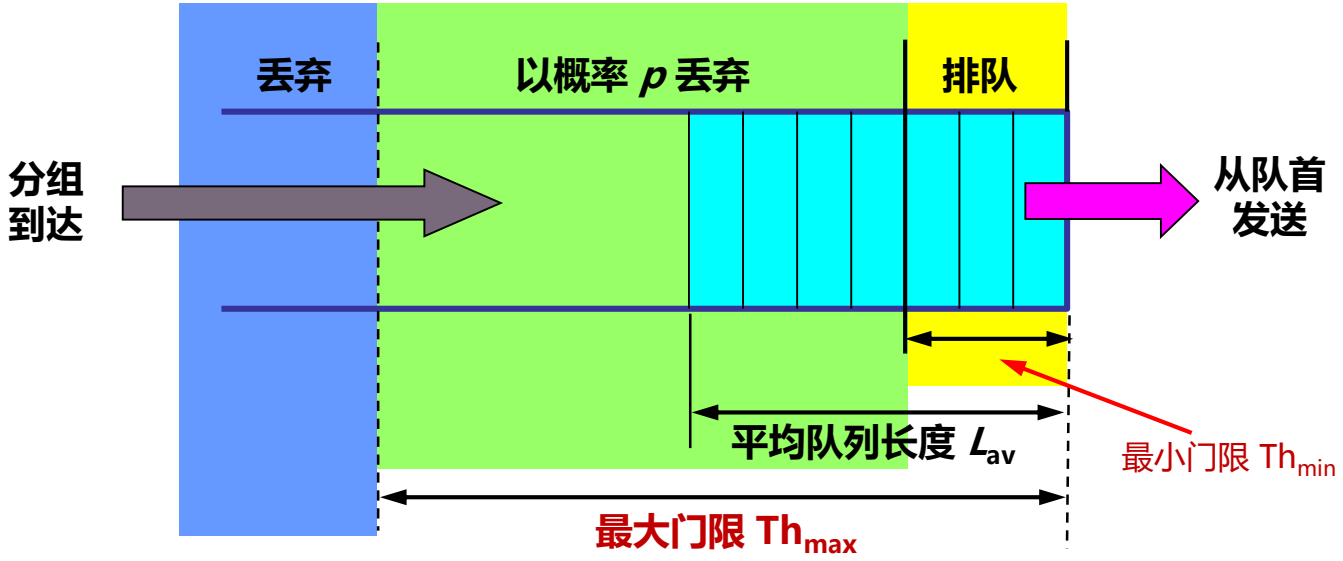
8. TCP 拥塞控制

- 随机早期检测 RED
  - 路由器队列维持两个参数：
    - 队列长度最小门限 TH_{min}
    - 队列长度最大门限 TH_{max}
  - RED 对每一个到达的分组都先计算平均队列长度 L_{AV} 。
    - 若平均队列长度小于最小门限 TH_{min}，则将新到达的分组放入队列进行排队。
    - 若平均队列长度超过最大门限 TH_{max} ，则将新到达的分组丢弃。
    - 若平均队列长度介于在最小门限 TH_{min} 和最大门限 TH_{max} 之间，则按照某一概率 p 将新到达的分组丢弃。

8. TCP 拥塞控制

- 随机早期检测 RED

RED 路由器到达队列维持两个参数：Th_{min}， Th_{max}，分成为三个区域：

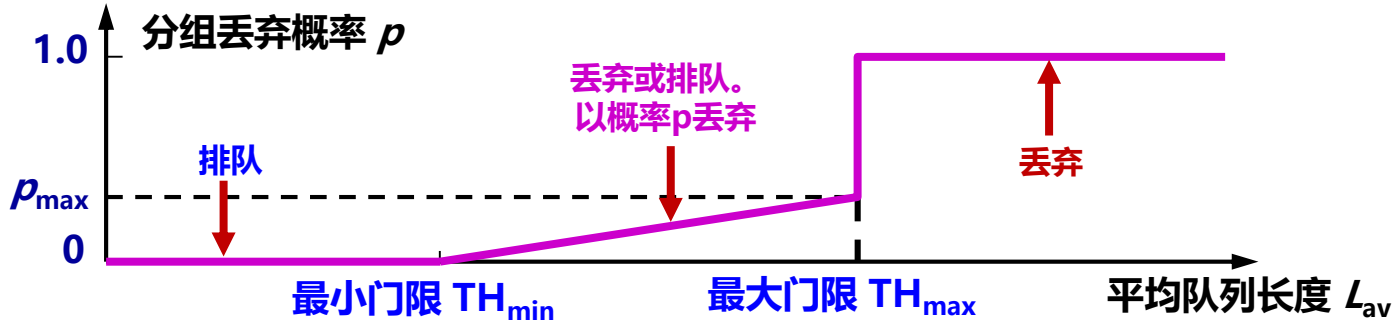


RED 对每一个到达的分组都先计算平均队列长度 L_{AV} 。

8. TCP 拥塞控制

- 随机早期检测 RED
  - 当 $L_{AV} < Th_{min}$ 时，丢弃概率 p = 0。
  - 当 $L_{AV} < Th_{max}$ 时，丢弃概率 p = 1。
  - 当 $Th_{min} \le L_{AV} \le Th_{max}$时，丢弃概率 p： 0 < p < 1 。

困难：丢弃概率 *p* 的选择，因为 *p* 并不是个常数。
例如，按线性规律变化，从 0 变到 p_{max}。



8. TCP 拥塞控制

- 随机早期检测 RED
  - 多年的实践证明，RED 的使用效果并不太理想。
  - 2015 年公布的 RFC 7567 已经把 RFC 2309 列为“陈旧的”，并且不再推荐使用 RED。
  - 但对路由器进行主动队列管理 AQM 仍是必要的。
  - AQM 实际上就是对路由器中的分组排队进行智能管理，而不是简单地把队列的尾部丢弃。
  - 现在已经有几种不同的算法来代替旧的 RED，但都还在实验阶段。

9. TCP 的运输连接管理

- TCP 是面向连接的协议。
- TCP 连接有三个阶段：
  1. 连接建立
  2. 数据传送
  3. 连接释放
- TCP 的连接管理就是使 TCP 连接的建立和释放都能正常地进行。

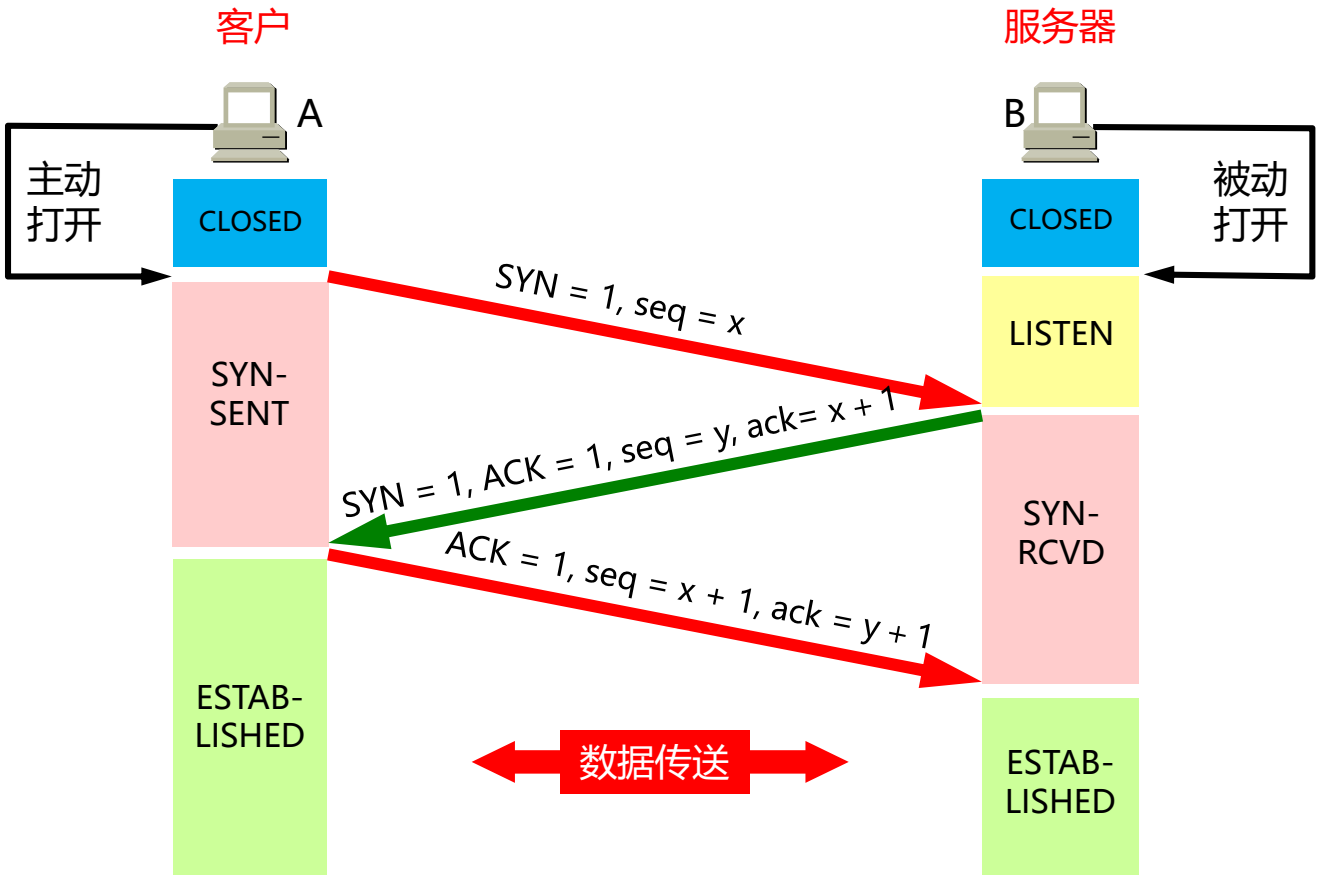
9. TCP 的运输连接管理

- TCP是面向连接的协议，在连接建立过程中要解决以下三个问题：
  1. 要使每一方能够确知对方的存在。
  2. 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）。
  3. 能够对运输实体资源（如缓存大小，连接表中的项目等）进行分配。
- TCP连接的建立采用客户服务器方式。
  - 主动发起连接建立的应用进程叫做客户 (client)。
  - 被动等待连接建立的应用进程叫做服务器 (server)。

9. TCP 的运输连接管理

- TCP 建立连接的过程叫做握手。
- 采用三报文握手：在客户和服务器之间交换三个 TCP 报文段，以防止已失效的连接请求报文段突然又传送到了，因而产生 TCP 连接建立错误。

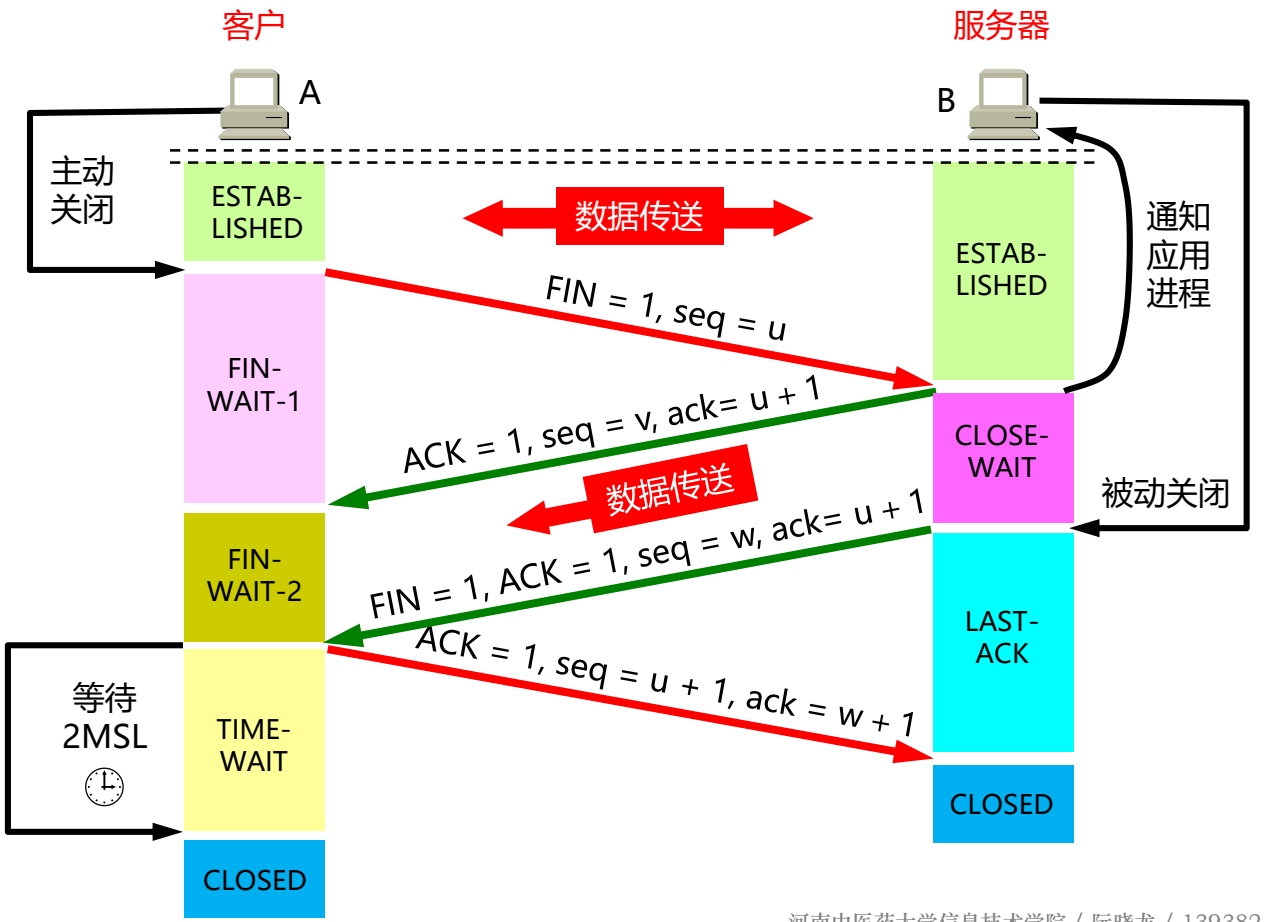
9. TCP 的运输连接管理



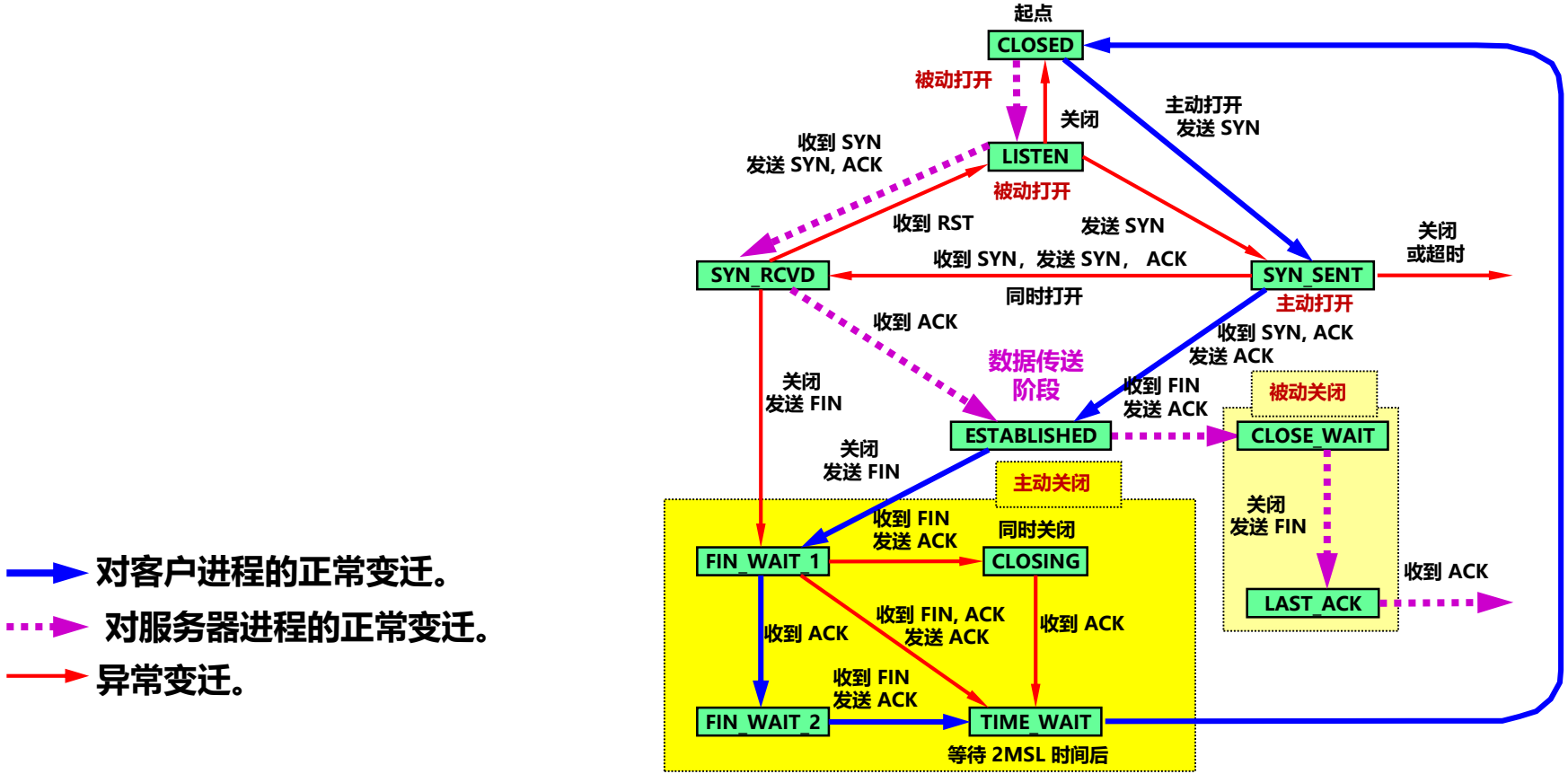
9. TCP 的运输连接管理

- TCP 连接释放过程比较复杂。
- 数据传输结束后，通信的双方都可释放连接。
- TCP 连接释放过程是四报文握手。

9. TCP 的运输连接管理



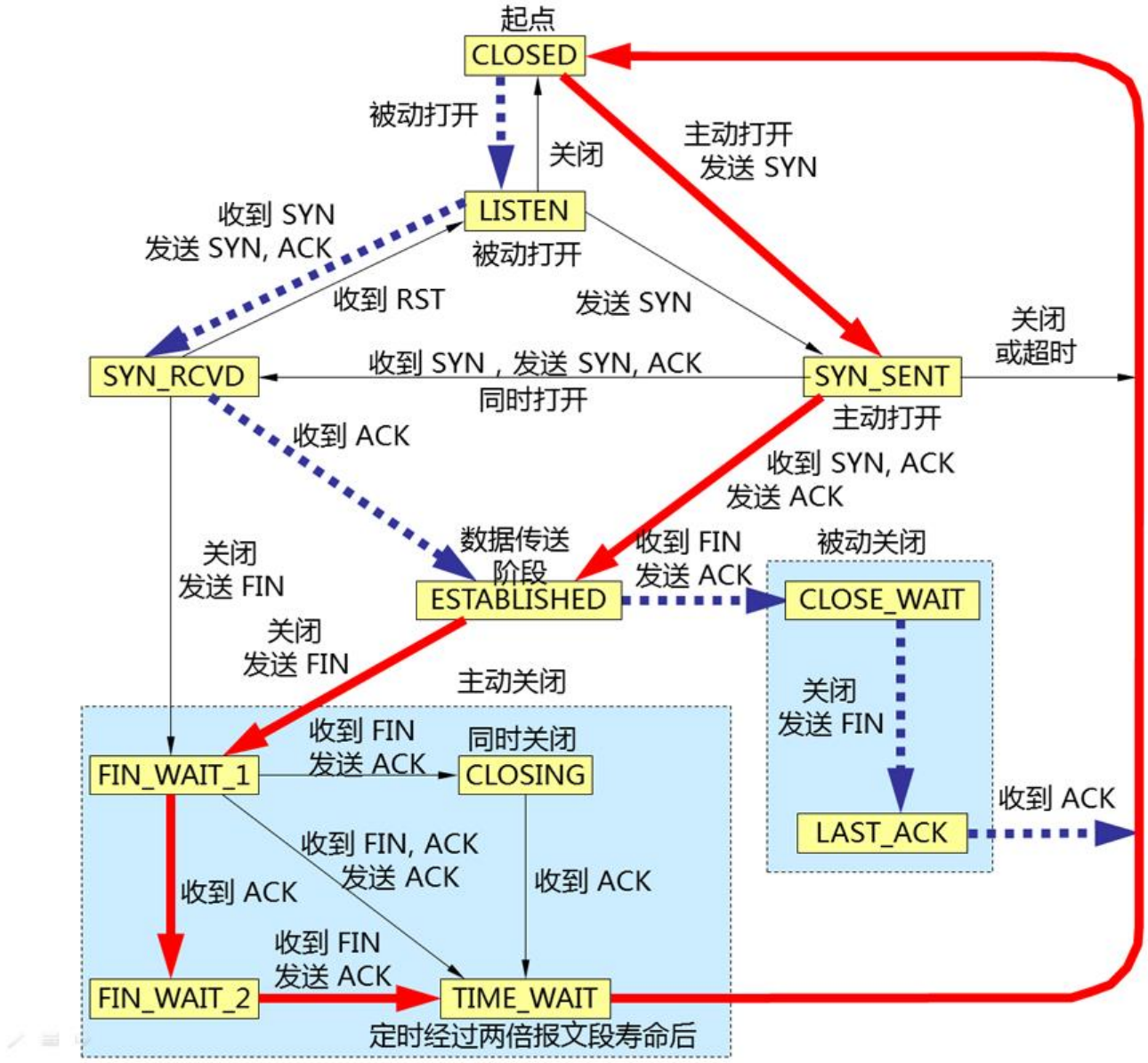
9. TCP 的运输连接管理



对客户进程的正常变迁。

对服务器进程的正常变迁。

异常变迁。


起点
CLOSED
被动打开
关闭
主动打开
发送 SYN
LISTEN
收到 SYN
发送 SYN, ACK
被动打开
收到 RST
发送 SYN
关闭
或超时
SYN_RCVD
收到 SYN，发送 SYN, ACK
同时打开
SYN_SENT
主动打开
收到 ACK
收到 SYN, ACK
发送 ACK
关闭
发送 FIN
数据传送
阶段
收到 FIN
发送 ACK
被动关闭
ESTABLISHED
CLOSE_WAIT
关闭
发送 FIN
主动关闭
收到 FIN
发送 ACK
同时关闭
FIN_WAIT_1
CLOSING
关闭
发送 FIN
收到 ACK
LAST_ACK
收到 FIN, ACK
发送 ACK
收到 ACK
收到 ACK
收到 FIN
发送 ACK
FIN_WAIT_2
TIME_WAIT
定时经过两倍报文段寿命后

Thanks