



# 第6章 传输层

黄敏: [minh@scut.edu.cn](mailto:minh@scut.edu.cn)

华南理工大学软件学院



# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- TCP连接管理 (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)





# 传输协议概述

---

- 传输层在分层网络体系结构中介于网络层和应用层之间，在利用网络层提供服务的基础之上，把数据交付的双方从两台主机扩展到了两台主机上的进程，向进程屏蔽了底层网络实现细节
- 传输层将数据交付给正确的进程，这是网络层所不具备的
- 传输层处于用户视角的最低层，用户并不需要关注网络拓扑、链路类型、路由协议等底层网络细节，如同在和对等方直接通信
- 传输层有两种不同的传输协议，面向连接的TCP和无连接的UDP





# 多路复用/分用\_1

---

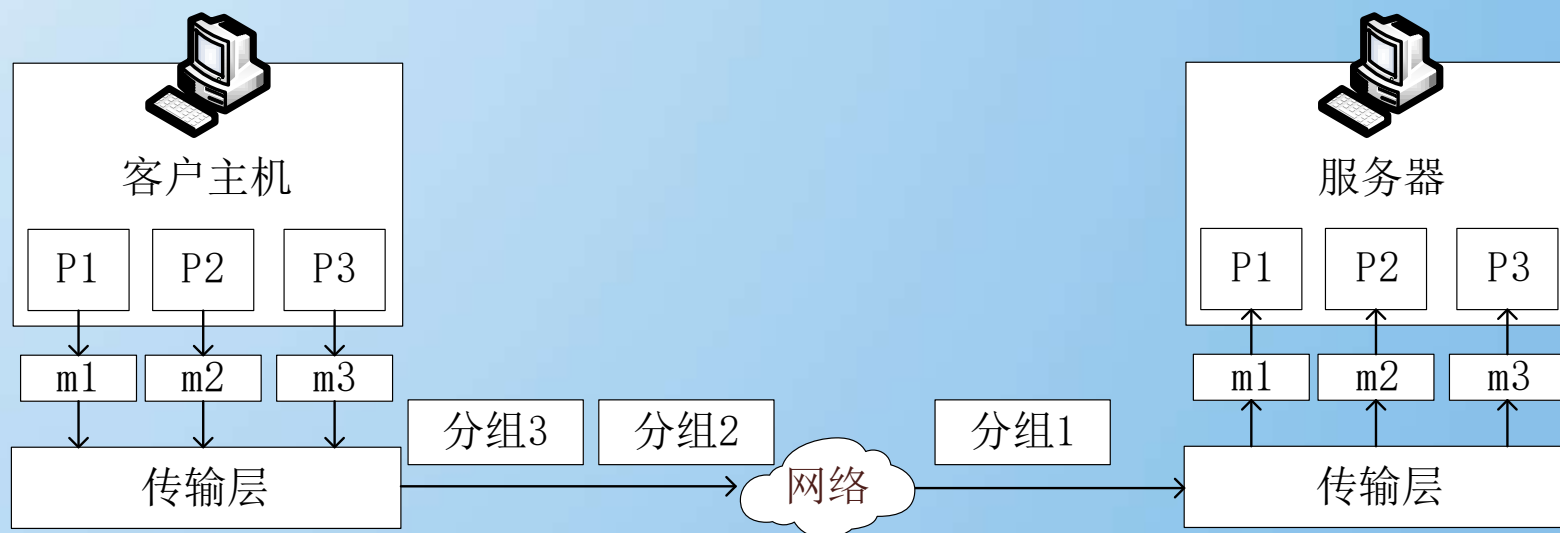
- 传输层向应用层屏蔽了网络拓扑、链路类型、路由协议等底层网络细节，提供了复用和分用以及对数据段的差错检测功能，实现了逻辑通信（logic communication）
  - 复用（multiplexing）：发送方的传输层收到来自不同应用进程的数据后封装传输层协议头部，统一交付给网络层并向协议栈的下层传递
  - 分用（demultiplexing）：接收方的传输层在从网络层收到递交上来的数据，解析完协议头部信息后，将数据正确交付给对应的目的应用进程





# 多路复用/分用\_2

- 传输层为浏览器（P1）、通讯软件（P2）、音乐软件（P3）进程提供多路复用/分用，通信数据有序交付



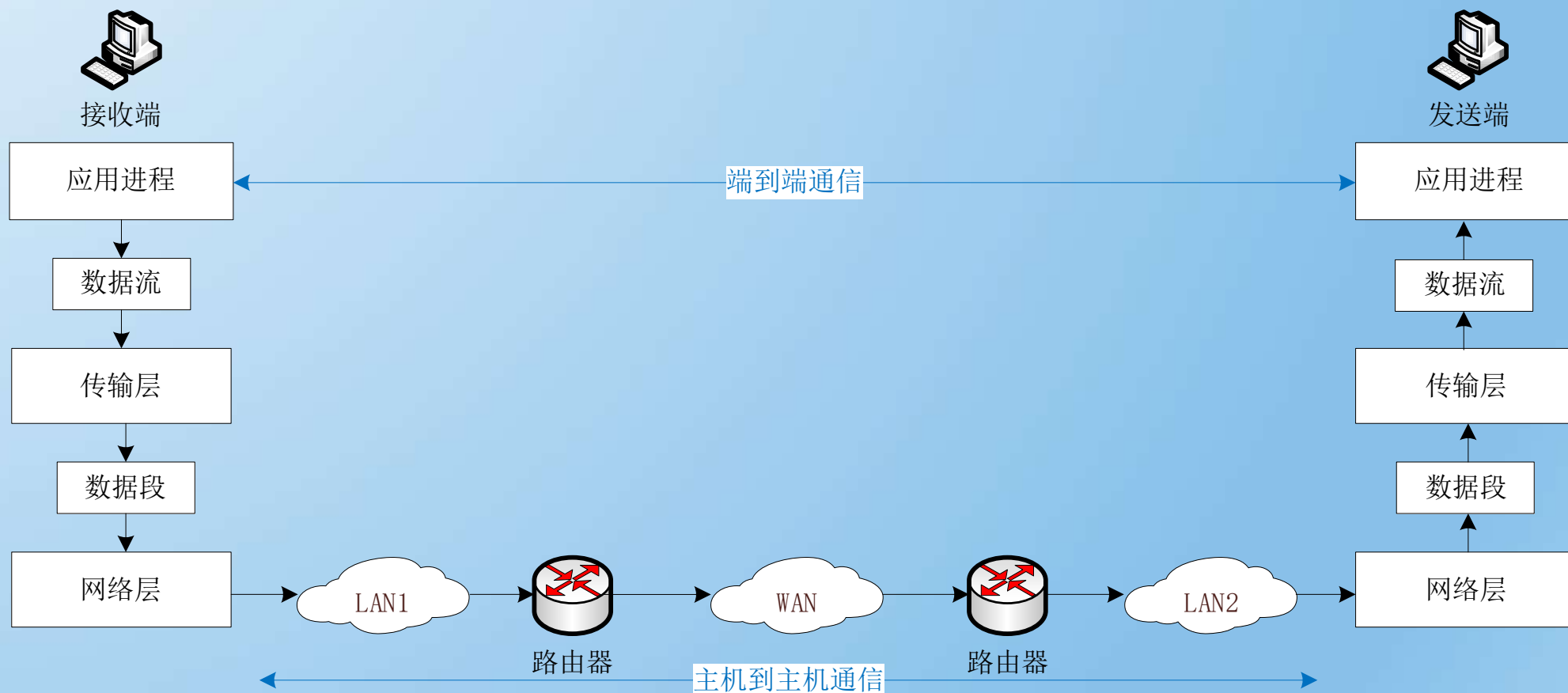
如何标识不同的应用进程？





# 端到端通信

- 网络层提供主机到主机通信，传输层提供端到端通信
- 主机之间通信的实质是主机上应用进程之间的通信





# 如何标识不同的应用进程？

---

## ■ 通信模型

- IP地址：主机的网络层地址

- 端口号：识别一台主机上的应用进程，只具有本地意义

- 通信协议：进程间数据传输使用的协议

- 半相关（half-association）由协议、本地地址、本地端口号组成，唯一标识通信双方中的一个进程

- 全相关（full-association）由协议、本地地址、本地端口号、远地地址、远地端口号组成，标识通信的收发双方





# 端口号分类

- 熟知端口号：0~1023，互联网名称与数字地址分配机构 (ICANN) 固定分配给重要的应用进程，如文件传输、远程登录、电子邮件、Web应用.....

应用程序↵	FTP↵	TELNET↵	SMTP↵	DNS↵	TFTP↵	HTTP↵	SNMP↵	SNMP(trap)↵	HTTPS↵
熟知端口号↵	21↵	23↵	25↵	53↵	69↵	80↵	161↵	162↵	443↵

- 注册端口号：1024~49151，使用需要向ICANN注册，进行统一管理
- 动态端口号：49152~65535，不需要注册，可临时使用







# 套接字

---

- 套接字标识通信端到端中的端点

- 点分十进制IP地址拼接端口号

- 由逗号或分号分隔

套接字socket=(IP地址: 端口号)

- 一对套接字可以标识一条虚连接, 例如TCP连接

TCP连接=(socket<sub>1</sub>, socket<sub>2</sub>)={(IP<sub>1</sub>: port<sub>1</sub>), (IP<sub>2</sub>: port<sub>2</sub>)}





# 练习

---

1、下面哪一项唯一地标识了一个运行的应用程序？

- A. IP地址
- B. 主机
- C. NIC（网卡）
- D. socket

解析：在一台主机上，应用程序是由端口号所标识，但是在全网范围，应用程序由**IP**和端口号共同标识，这既是**socket**套接字。



# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- TCP连接管理 (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)





# UDP的特点与优点

---

- 用户数据报协议（User Datagram Protocol, UDP）提供不可靠的逻辑通信信道，特点与优点包括：
  - 较小的协议头部，数据段解析更快
  - 无需建立连接，减少了传输开销但难以保证可靠性
  - 无拥塞控制，实时性更高
  - 无可靠传输机制，不对数据段确定
  - 面向数据段





# UDP协议数据段格式

- 源端口号：需要回复时设为发送进程端口号，不需回复可设全0
  - 实现复用功能
- 目的端口号：接收主机进程端口号
  - 实现分用功能
- 长度：数据段总长度
- 校验和：差错检测，不使用校验和时校验和字段为全0

2B	2B	2B	2B
源端口号	目的端口号	长度	校验和





# UDP计算校验和使用伪首部

## ■ 伪首部参与校验和计算但不发送

- ❑ 源IP地址
- ❑ 目的IP地址
- ❑ UDP协议字段17
- ❑ UP数据段长度



## ■ 注意UDP伪首部字段与IP首部最后12个字节的相似之处

为什么计算校验和要加上伪首部呢？

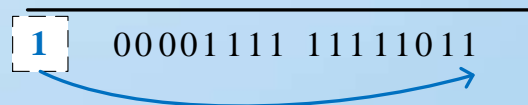




# UDP计算校验和

- 伪首部和UDP数据段按16位（2字节）对齐
- UDP数据部分为奇数字节时补充1个全0字节，但该字节不发送
- 二进制反码求和
- 求和溢出时回卷

10001111 10101010  
10000000 01010000  
1 00001111 11111011



125.216.251.36		
119.75.217.109		
全0	17	9
1024		2048
9		全0
数据	全0	

125.216 → 01111101 11011000  
251.36 → 11111011 00100100  
119.75 → 01110111 01001011  
217.109 → 11011001 01101101  
17 → 00000000 00010001  
9 → 00000000 00001001  
1024 → 0000100 00000000  
2048 → 00001000 00000000  
9 → 00000000 00001001  
数据 → 01010101 01010101

二进制反码运算求和=00101011 00101111  
结果求反码得校验和=11010100 11010000





# 远程过程调用RPC\_1

---

- 远程过程调用：允许本地程序调用远程主机上的过程，通过网络传输请求/应答，隐藏客户机不在本地调用服务的事实
- 主动调用一方为客户，绑定客户存根（client stub）库过程
- 被调用一方为服务器，绑定服务器存根（server stub）库过程
- 为什么采用UDP实现RPC？
  - 实现简单
  - 实时性
  - 过程调用时间点的突发和不确定性



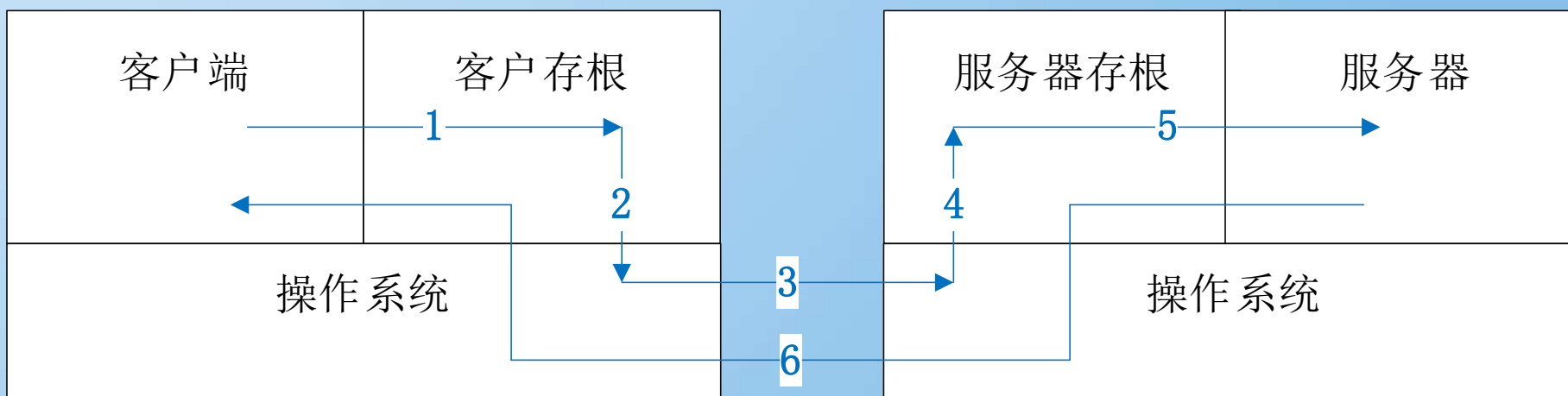




# RPC过程\_1

列集是将内存中的对象变换为适合存储或发送的形式过程，与序列化相似

- 客户通过本地方法调用客户存根
- 客户存根通过列集（marshalling）将参数封装到一个消息中，由操作系统发送该消息
- 操作系统将消息从客户主机发送到服务器主机

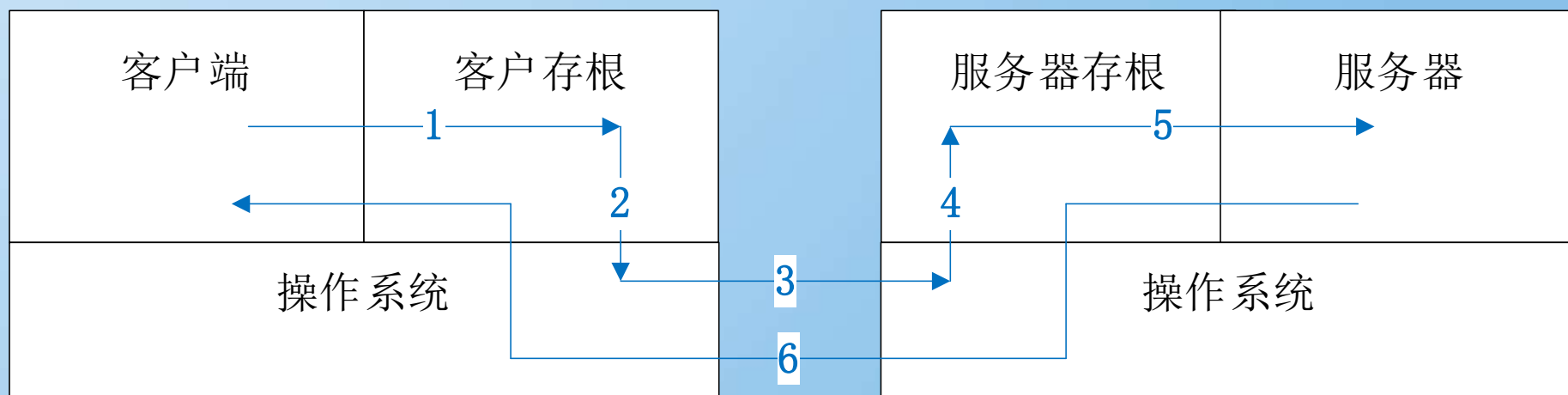




# RPC过程\_2

散集是列集的逆过程

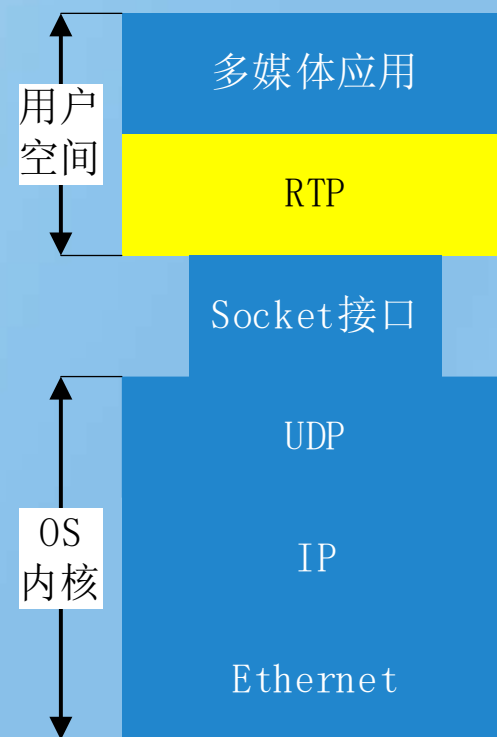
- 服务器操作系统将收到的数据段传递给服务器存根
- 服务器存根通过散集（unmarshaled）得到的参数来调用服务器过程
- 服务器过程执行后的结果按同样的路径传回给客户主机





# 实时传输协议RTP

- 实时传输协议广泛应用于互联网网络电话、网络视频、网络游戏等多媒体应用
- RTP的基本功能是多个实时数据复用到一个UDP数据包流中，通过单播和组播方式传输
- 为什么使用UDP实现RTP?
  - 实时性
  - 多媒体数据复用UDP数据包
  - UDP同时具备单播和组播能力，TCP只能单播





# RTP数据段头部\_1

- V: 版本字段
- P: 数据包被填充至4字节整数倍
- X: 存在扩展头, 要求扩展头第一个字节标识扩展头长度
- CC: 贡献源数量





# RTP数据段头部\_2

- M：由应用解释的字段，例如可以标识视频或音频的开始
- 有效载荷类型：数据部分编码方式
- 序号：数据包被赋予的递增编号，帮助接收方检测是否存在数据包丢失，丢失的数据包不和TCP类似请求重传，实时应用重传过时数据意义不大，而是交由多媒体应用程序处理

TCP序号字段的具体作用将在6.3学习

2b	1b	1b	4b	1b	7b	16b=2B
V	P	X	CC	M	有效载荷类型	序号
时间戳 (4B)						
同步源标识符 (4B)						
贡献源标识符[1] (4B)						
贡献源标识符[2] (4B)						
...						
贡献源标识符[n] (4B)						





# RTP数据段头部\_3

- 时间戳：用于多流同步，例如数字电视节目视频流和音频流同步
- 同步源标识符：指明数据包属于哪个流，在发送方将多流数据复用到一个UDP数据包中，在接收方分用出数据流
- 贡献源标识符：标识负载的贡献源





# 练习

---

1、在发送数据的计算机中，UDP实体从哪里接收数据单元？

- A. 应用层
- B. 传输层
- C. 网络层
- D. 数据链路层

解析：UDP实体参与了封装的一环，从上层接收数据，UDP位于传输层，它的上层就是应用层。





# 练习

---

2、在发送数据的计算机中，UDP实体处理好数据单元，要向哪里传送？

- A. 应用层
- B. 传输层
- C. 网络层
- D. 数据链路层

解析：UDP实体参与了封装的一环，从上层接收数据，切割成合适的长度，封装成数据段之后，向下层传送，它的下层是网络层。







# 练习

3、UDP提供了下面哪个功能？

- A. 进程到进程（**process-to-process**）通信
- B. 主机到主机（**host-to-host**）通信
- C. 端到端可靠数据传输
- D. 所有上述功能

解析：UDP实体产生的数据段，从源端（**end point**）发送到 目的端点（**end point**），端点由**IP**和**Port**两个元素指定，实际上端点唯一绑定了某个应用进程，所以，UDP数据段是从端到端的，即进程到进程的；但是选项“端到端可靠数据传输”选项是错误的，因为UDP提供的是不可靠的段传输。





# 练习

4、为了向应用程序传输用户数据，UDP数据段中需要下面哪种地址？

- A. port（端口号）
- B. 应用
- C. IP地址
- D. MAC地址/物理地址

解析：应用程序的数据接收地址是端点（进程），端点的两个元素，一个是IP地址，一个是端口号，IP地址是网络层的，属于IP协议的内容，只有端口号满足题意。





# 练习

5、UDP能确保下面哪个？

- A. 每个数据段中的序列号
- B. 对发送方的确认
- C. 流控制
- D. 其余都没有

解析：UDP提供无连接的数据段服务，段中无序列号，也不确认和做任何流控，只是简单的端到端数据传输。





# 练习

6、UDP被认为是无连接的传输层协议，主要原因是下面哪一个？

- A. 无确认
- B. 无虚电路
- C. 无可靠措施
- D. 无流控

解析：有连接、无连接的区分主要在于数据传输前是否有建立了虚连接/虚电路。





# 练习

7、UDP数据段中的源端口号定义了下面哪个？

- A. 发送计算机
- B. 接收计算机
- C. 发送计算机的应用程序
- D. 接收计算机的应用程序

解析：源端口号是通信五元组中的一个元素，指定了源机的应用进程，即发送计算机的应用程序。





# 练习

8、下面哪项不是UDP数据段中的字段？

- A. 头部长度的
- B. 源端口
- C. 校验和
- D. 目的端口

解析：源端口和目的端口是UDP数据段中最重要的两个字段，校验和也是UDP的一个字段，只有“头部长度的”选项不是，头部中有个“总长度”





# 练习

9、为什么会存在UDP？用户进程使用原始IP数据包还不够吗？

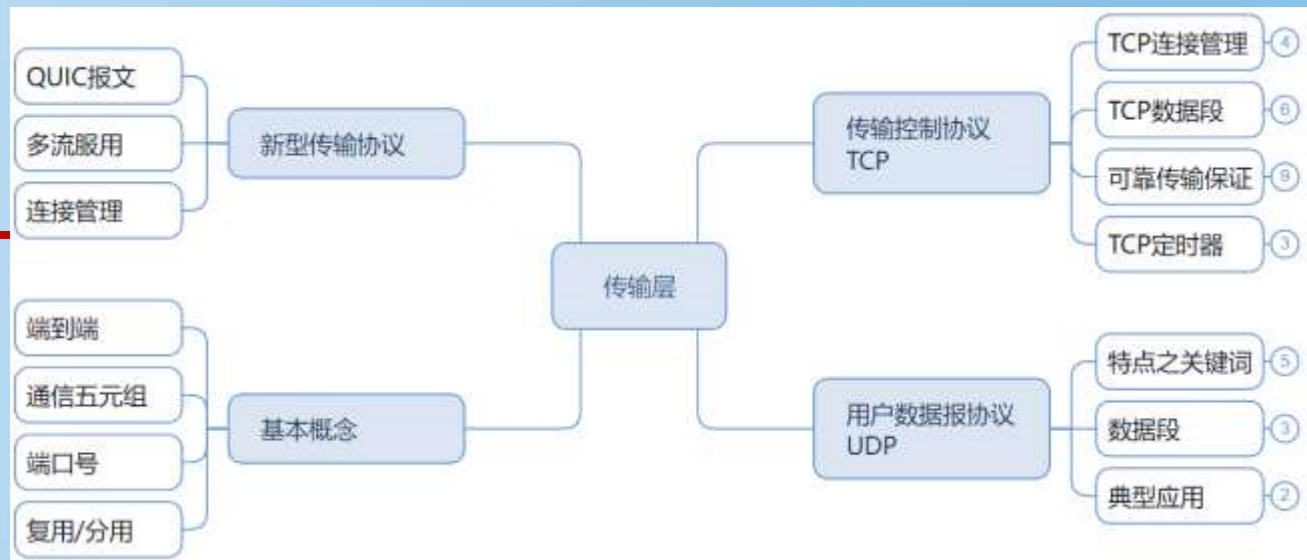
参考答案：

存在UDP是因为UDP里面包含了端口信息。原始的IP只是标识了目标机器，但是目标机器接收到后却不知道当前IP数据包应该交给哪个应用程序进行处理。而UDP中的端口信息则可以让目标机器根据端口选择正确的处理程序。



# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- TCP连接管理 (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)







# TCP概述

---

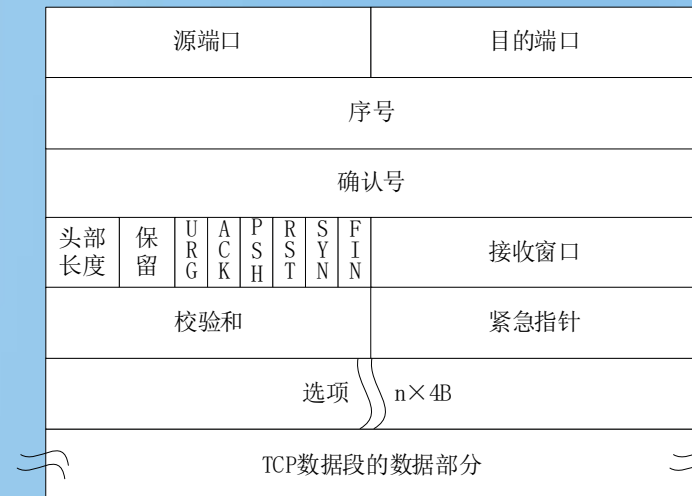
- TCP在不可靠的互联网络上提供可靠的端到端字节流传输
- 面向连接，在两个端系统的TCP实体中保存连接状态信息
  - 虚连接，并非物理连接
- 面向字节流的可靠传输，包括无差错、无丢失、有序数据交付
- 单播全双工通信

通过比较UDP和TCP特点，哪些应用适合使用UDP？哪些适合使用TCP呢？



# TCP协议数据段格式\_1

- 源端口和目的端口：用于分用/复用
- 序号：按字节编号，数据段头部的序号字段等值于数据部分第一个字节序号
- 确认号：接收方采用累积确认
  - 确认号为N代表序号为N-1及之前的字节均已按序到达
- 头部长度的：占4位，以4字节为单位
  - 头部最长60字节（4字节\*15）
  - 其中固定头部占20字节
- URG~FIN为控制位

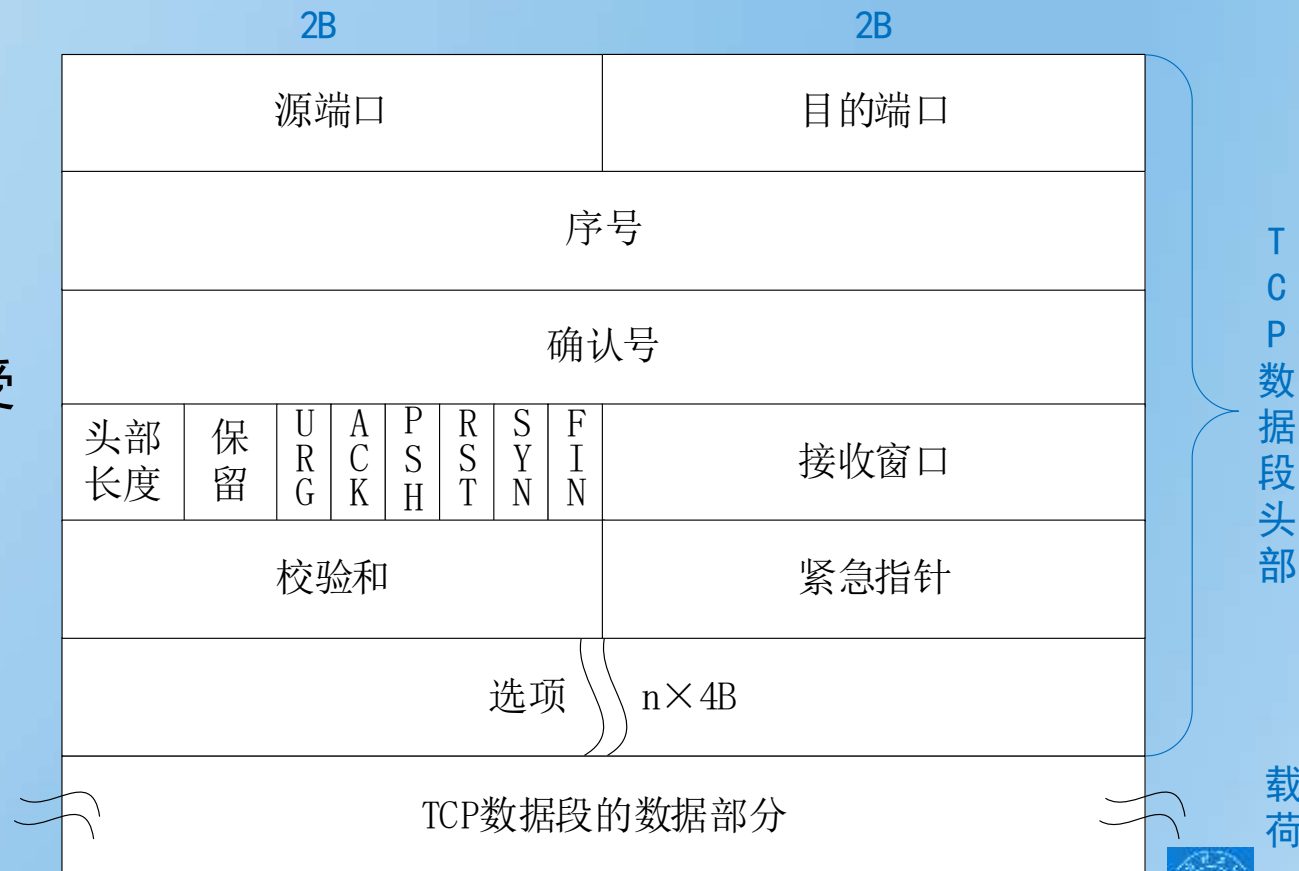




# TCP协议数据段格式\_2

- 紧急URG位：置1代表紧急指针字段有效
- 确认ACK位：置1代表确认号字段有效，建立连接后都需要置1
- 推送PSH位：常用于交互式通信，置1时接收方优先处理该数据包
- 复位RST位：用于连接出错时重置连接
- 同步SYN位：用于建立连接请求和连接接受
- 终止FIN位：用于释放TCP连接
- 接收窗口：用于流量控制

**TCP流量控制将在稍后学习**





# TCP协议数据段格式\_3

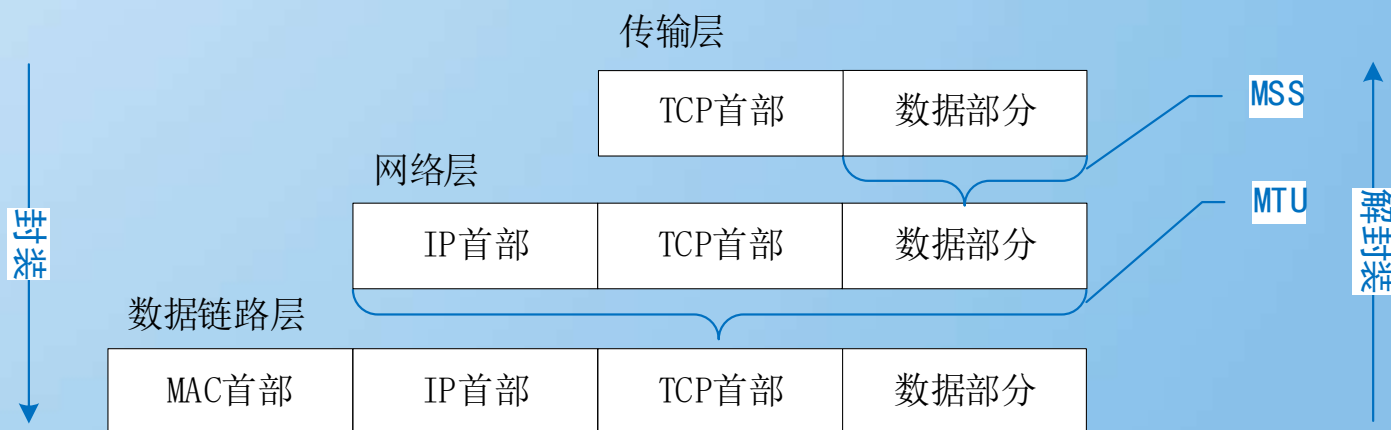
- 校验和：计算方式与UDP计算校验和相同，伪首部协议字段改为6
- 紧急指针：配合URG位使用，标志紧急数据截止位置
- 选项：可选字段，包括最大数据段长度选项、时间戳选项、窗口扩大选项.....





# TCP最大数据段长度选项

- 最大数据段长度（Maximum Segment Size, MSS）是除去TCP头部后的数据部分最大长度
- MSS需要在网络利用率和物理链路帧长度限制之间均衡，选择合适的MSS有利于提高网络利用率并避免数据段分片





# TCP窗口扩大选项

---

- TCP固定头部中接收窗口字段占16位，代表发送窗口最大约为 $2^{16}-1 \approx 64\text{KB}$ （不考虑拥塞窗口）
- 在网络条件允许的情况下，可通过窗口扩大选项中的移位值S（最大14）增大窗口，增大后最大值为

$$2^{16+14}-1 \approx 1\text{GB}$$





# TCP时间戳选项

---

- 时间戳选项的两个主要用途为：
  - 计算往返时间RTT，发送方将当前时间写入时间戳字段，接收方在对应的确认数据段复制一份时间戳值，发送方由此计算RTT
  - 高速传输下区分新旧数据段，TCP数据段按数据字节来编号，并且头部的序号长度为32位，当网络带宽达到2.5Gbit/s时， $2^{32}$ 字节的数据量在15秒之内就会传输完，后面的数据段将重复利用序号，接收方可通过时间戳选项识别出带有相同序号的数据段是新的数据段的还是晚到了的前一批数据段





# TCP可靠数据传输机制\_1

## ■ 可靠传输需要解决的问题

- 数据错误
- 丢失
- 重复
- 失序

## ■ TCP可靠数据传输机制

- 序号
- 确认
- 重传







# TCP可靠数据传输机制\_2

---

## ■ 序号

- 发送方按字节编码，每个字节占一个序号，按字节流传输
- 接收方按序号还原数据段发送顺序

## ■ 确认

- 累积确认，不对数据段逐一确认，只确认按序到达的最后一个数据段
- 通过确认号告知发送方期望收到的下一个按序数据段的序号





# TCP可靠数据传输机制\_3

---

## ■ 重传

### □ 发送方设置发送缓存暂存：

- 允许TCP发送，但还没传输到网络中的数据
- TCP已发送但还未收到确认的数据

### □ 接受方设置接受缓存暂存：

- 允许接收且按序到达，但还未被应用进程读取的数据
- 允许接收但乱序到达的数据

### □ 发送缓存使发送方能够重新传输丢失或错误的数据段





# TCP重传事件\_1

---

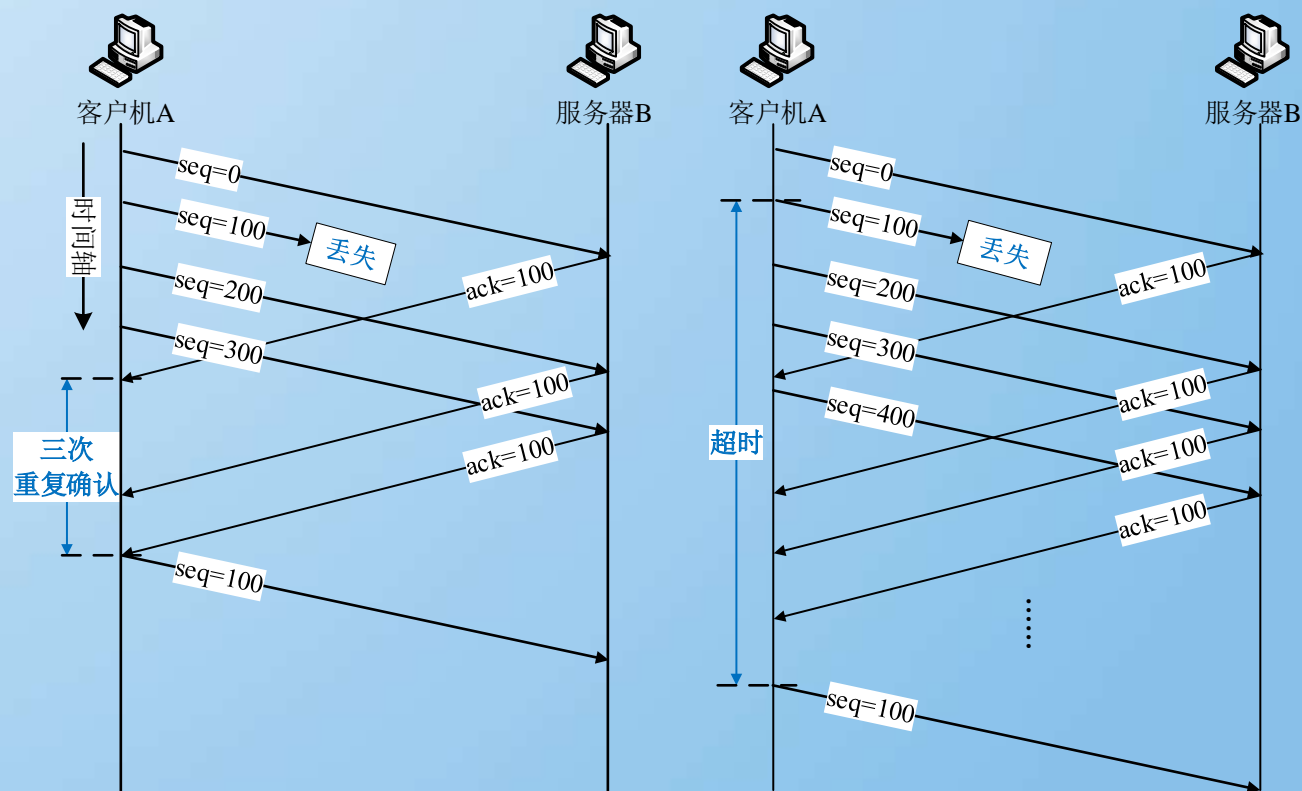
- 有两个事件可以触发TCP进行数据重传
  - 计时器超时重传
    - TCP需要预计数据段往返时间RTT
    - 基于RTT设置时间计时器，若计时器超时前未收到确认数据段，启动重传
  - 三次重复确认重传
    - 若连续三次收到同一个确认数据段，即便计时器还未超时，TCP将更早地启动重传





# TCP重传机制\_2

- 三次重复确认更早启动重传
- 通常认为三次重复确认时网络拥塞状况没有超时严重，因为接收方仍然能收到确认数据段，并不是毫无消息





# TCP基于滑动窗口的流量控制\_1

---

- TCP提供一种基于滑动窗口的流量控制机制来避免发送方速率与接收方应用程序的读取速率不匹配而使得缓存溢出的问题
- TCP提供全双工通信，接收方和发送方可各设置接收缓存和发送缓存
- TCP固定头部中接收窗口字段告知了发送方当前接收方可接收的数据容量，发送窗口应当不大于接收窗口值



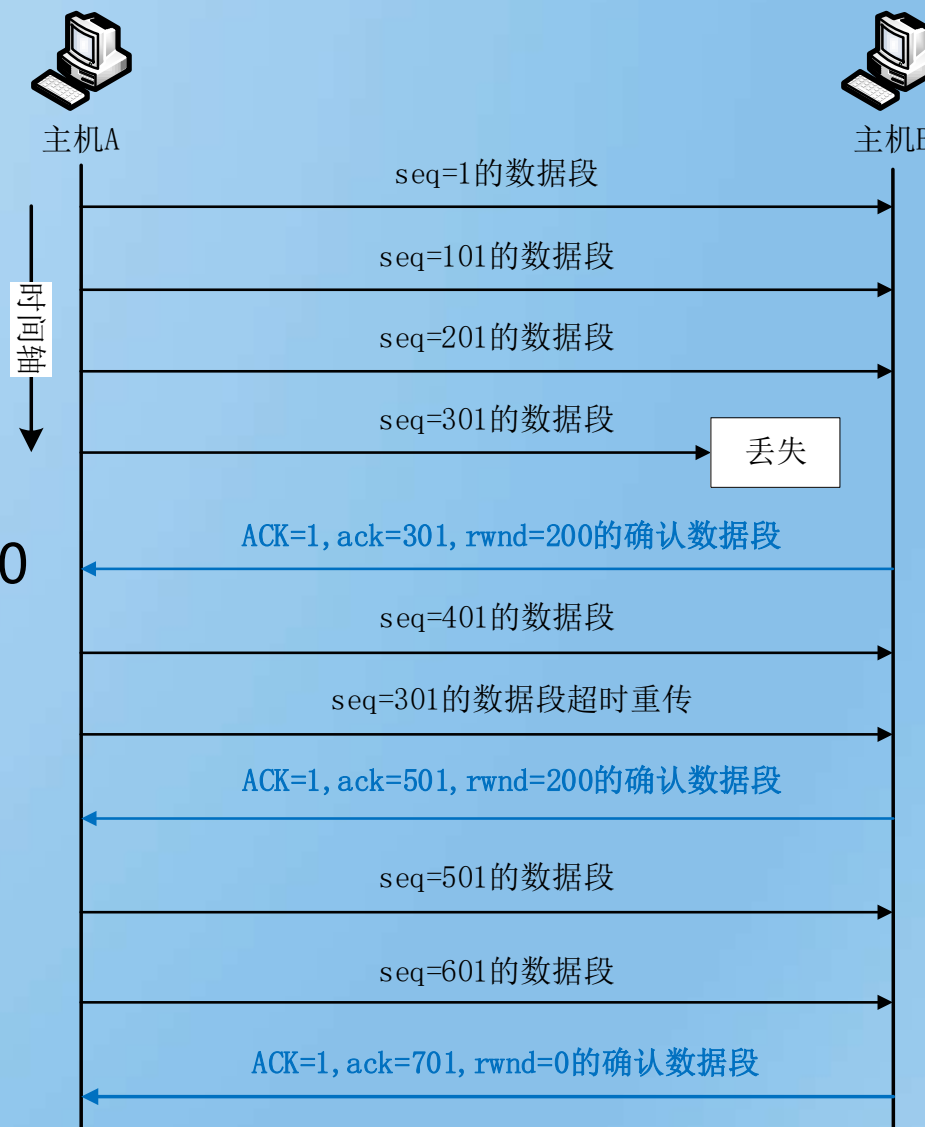


# TCP基于滑动窗口的流量控制\_2

## ■ TCP确认数据段中接收窗口字段标识了滑动窗口大小

### ■ 调整窗口值为200

### ■ 调整窗口值为0





# 糊涂窗口综合征

---

- 糊涂窗口综合征指TCP流量控制实现不良导致网络利用率低下的计算机网络问题
  - IP和TCP固定头部各20字节，若数据部分只有10字节，利用率仅有20%，若数据部分只有1字节，利用率仅有2.4%
- Nagle算法：允许发送程序执行写调用，先收集发送方程序向网络写入的数据，部分数据不直接发送，小数据累积到大数据段中或收到接收方确认数据段才发送
- Clark算法：接收方缓存有足够缓存空间放下一个较大数据段之前，或者至少有一半缓存空间空闲之前，一直宣布窗口为零





# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- **TCP连接管理** (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)







# TCP连接管理

---

- TCP是面向连接的协议，每次数据传输需要包含：连接建立、数据传输、连接释放三个阶段
- 连接建立的目的是允许双方就一些通信参数达成一致（初始报文段序号、最大报文段长度、接收窗口大小等），同时能够对通信实体资源进行分配（收发缓存、主机进程等）
- TCP建立连接采用客户/服务器模式，主动建立连接请求的一方为客户，被请求建立连接的一方为服务器





# TCP三次握手

---

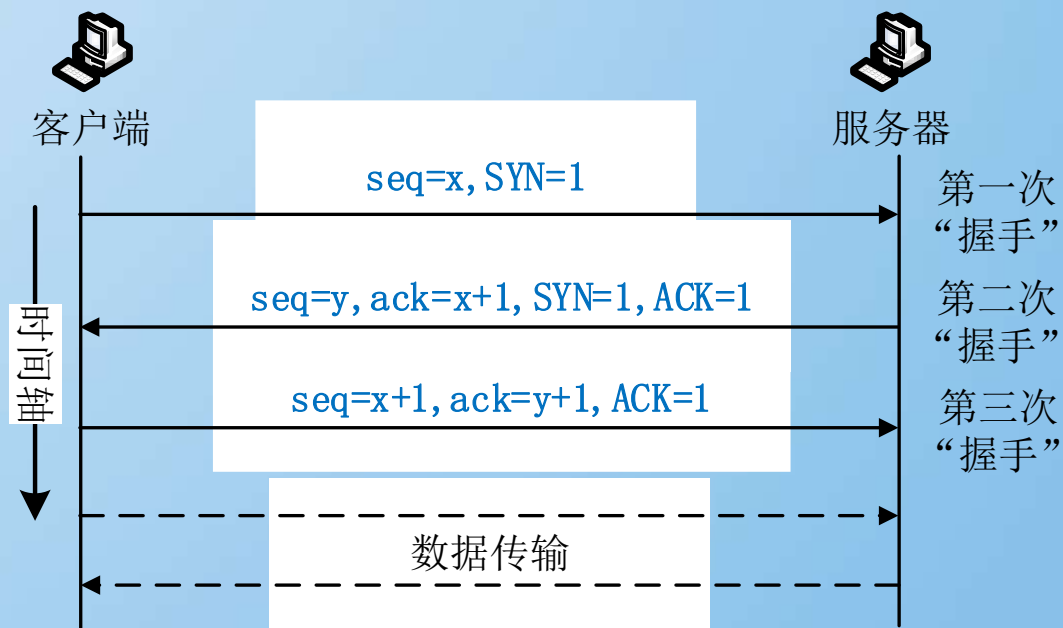




# TCP连接建立\_1

## ■ TCP建立连接通过三次握手

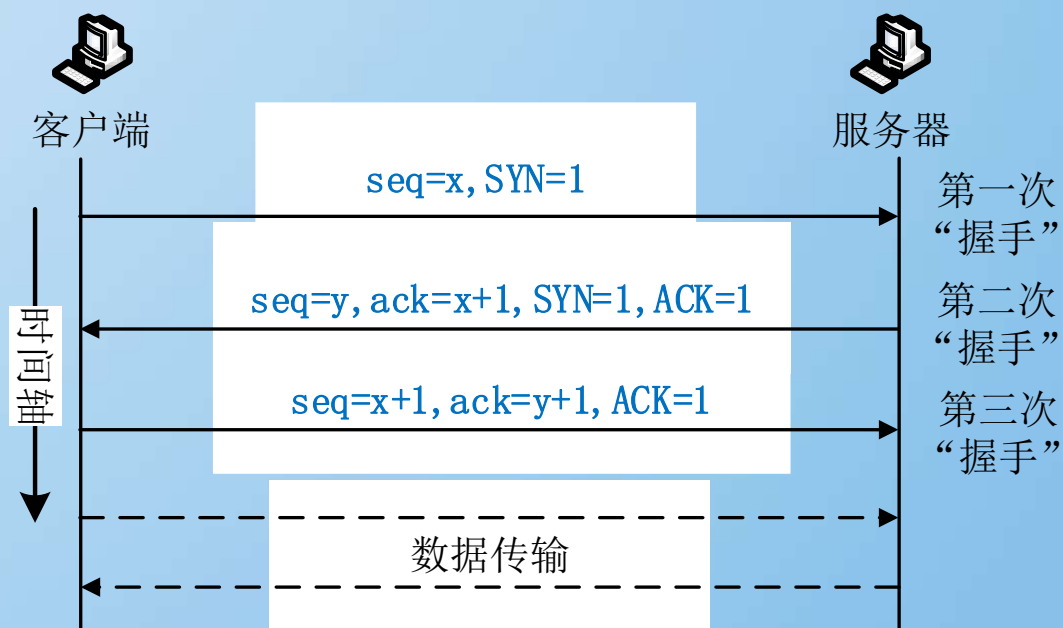
- 第一次握手：TCP客户进程向TCP服务器进程发送连接请求数据段，SYN位置1，初始化序号seq（假设为x），请求数据段不含数据，但占1个字节





# TCP连接建立\_2

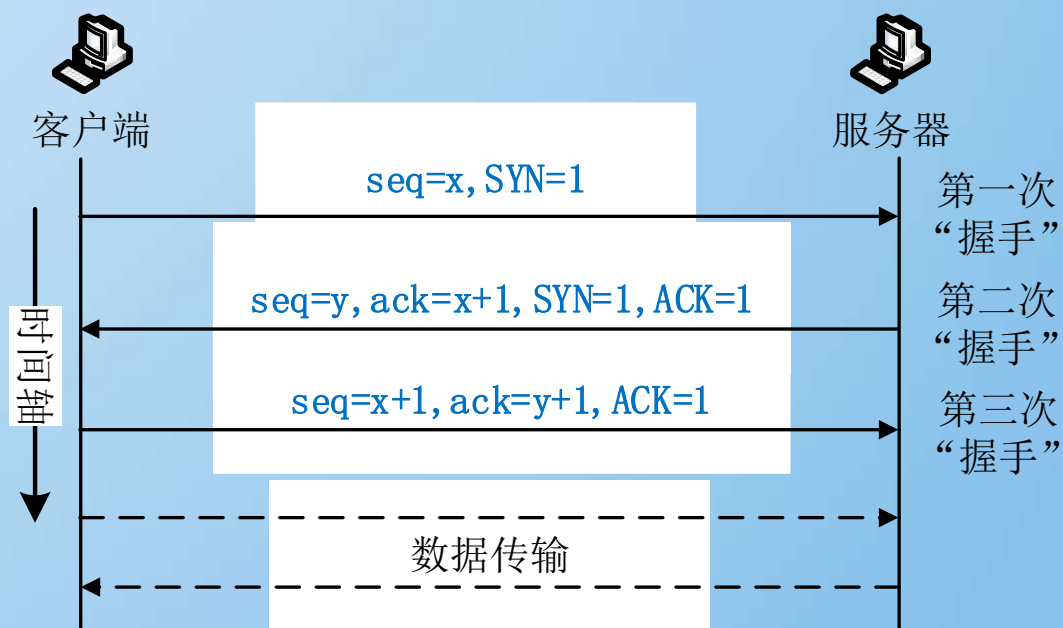
- 第二次握手：TCP服务器进程若接受请求，向客户进程发回确认数据段，初始化序号seq（假设为y），SYN位和ACK位置1，确认号ack为x+1，确认数据段不含数据，但占1个字节，服务进程为TCP连接分配缓存和变量





# TCP连接建立\_3

- 第三次握手：TCP客户进程向TCP服务器进程发回确认数据段，序号seq为 $x+1$ ，确认号ack为 $y+1$ ，该数据段可包含数据，客户端为TCP连接分配缓存和变量





# SYN洪泛攻击

---

- 注意上述过程中，TCP服务器进程在第二次握手阶段就为连接分配资源，导致服务器存在遭受SYN洪泛攻击的风险，恶意用户以客户方式发送大量连接请求数据段，服务器为这些连接分配大量资源后，攻击者又不完成后续的连接建立，导致服务器资源耗尽





# SYN Cookie

---

- SYN Cookie可抵御SYN洪泛攻击，其原理为：服务器收到SYN数据段后，不立即为连接分配资源，而是先将数据段源IP和目的IP地址、端口号和随机生成的秘密数通过散列生成初始TCP序列号（称为Cookie值），对于一个合法用户，他返回的确认号为Cookie+1，服务器将确认数据段的源IP和目的IP地址、端口号以及缓存的秘密数通过散列函数计算得到的结果加1后，假如能和确认数据段的确认号ack一致，此时可以确定客户是合法的，接下来才完成连接的建立





# 练习

1、客户端与服务器开始建立 TCP 连接。客户端发送一个 SYN 数据段，序列号 7000，服务器回应一个 SYN 数据段，序列号为 9000，ACK 确认号 设置为7001。以下哪一个选项最好地描述了客户端向服务器发送的下一个数据段的情况？

- A. SYN 数据段序列号 7001，ACK 号为 9000
- B. 标准数据段（即，SYN 标志未设置）序列号为 7001，ACK 号为 9001
- C. 标准数据段（即，SYN 标志未设置）序列号为 7001，ACK 号为 9000
- D. SYN 数据段序列号 7000，ACK 号为 9001

解析：SYN数据段不是标准的数据段，而是三次握手建立连接信息中的第一次、第二次握手信息，其中的SYN控制位置位（SYN=1），第三次握手信息是标准数据段，SYN=0，其序列号应该是初始序列号+1，即7000+1，确认号应该是对方（服务器）的初始序列号+1，即9000+1。







# 练习

2、如果一个TCP数据段的接收方，向发送方发出了一个确认消息，其中的ACK确认号和窗口尺寸两个字段的值分别是：**ACK=12000**，**WIN=8000**。下列哪一项不是发送方的可以传输的有效的数据段？

- A. 发送方可以传输 2000 字节数据段，**SEQ = 18100**
- B. 发送方可以传输 1500 字节数据段，**SEQ = 18100**
- C. 发送方可以传输 1000 字节数据段，**SEQ = 18000**
- D. 发送方可以传输 2000 字节数据段，**SEQ = 17000**

解析：确认号字段的值表明了期待接收的第一个数据字节的序列号（也就是说，此序列号之前编号的那些字节已经被正确接收了）；而窗口尺寸意味着接收方能够接收的数据段的长度上限值。因为**ACK=12000**，**WIN=8000**，所以，发送方能够发送的字节的序列号必须在**12000~19999**之间。**SEQ=18100**，长度**2000B**的数据段的字节编号（序列号）是**18100~20099**，已经超出了有效序列号的范围。**20099**是这样计算出来的：**18100+2000-1**。





# 练习

3、一个 TCP 发送端发送数据段，每个数据段具有 1200 个字节的有效载荷，接收方使用累计确认。如果发送端按照下面的序列号发送数据段：14200，15400，16600，17800。假设所有的数据段都按顺序收到。如果接收方在收到最后一个数据分组后发送一个 ACK 数据段，那么此数据段的确认号是多少？

- A. 17799
- B. 19000
- C. 17801
- D. 17800

解析：确认号字段的值表示：期待接收的第一个数据字节的序列号（也就是说，此序列号之前编号的那些字节已经被正确接收了）。发送端发送的最后一个数据段的序列号是17800，每个数据段的长度是1200字节，那么这个数据段的最后一个字节的编号是： $17800+1200-1=18999$ 。那么接收方期待接收的下一个字节的编号应该是 $18999+1=19000$ ，即确认号字段的值应该为：19000。





# 练习

4、一个 TCP 发送端发送数据段，每个数据段具有 1000 字节的有效载荷，如果发送端按照下面的序列号发送数据段：： 3000、4000、5000、6000。假设序列号 4000 的数据段丢失，其他数据段都按顺序传输。如果接收方在收到最后一个数据分组后发送一个 ACK 数据段，那么此数据段的确认号是多少？

- A. 6000
- B. 7000
- C. 3999
- D. 4000

解析：4000序列号的数据段丢失，需要重发，所以，接收端直接将确认号的值填写为4000，告诉发方从4000序列号的那个字节开始重发。





# 练习

5、在网络中的TCP端点使用了慢启动，网络状况很好，没有拥塞，端点间的数据往返时间是20ms。如果一个TCP端点在 $t=0$ 时发送1个数据段，那么在时间  $t = 60 \text{ ms}$  到  $t = 80 \text{ ms}$  之间，共可以发送多少个数据段？（假设数据段能够以很小的时延注入，数据段背靠背，可以毫无间隙）

- A. 2
- B. 3
- C. 8
- D. 4

解析：慢启动算法，在达到阈值之前，每成功传送一次（数据抵达，确认回来，一个RTT），拥塞窗口增加1倍，即翻倍。本题中的网络顺畅，意味着都成功传输，没有阈值信息，考虑还没有设置阈值，非线性增长。 $t=0$ 时，第一次发送了1个数据段； $t=20$ ，收到了确认，成功传输，拥塞窗口时可以翻倍为 $1*2=2$ ； $t=40$ 时，拥塞窗口可以翻倍为 $2*2=4$ ； $t=60$ 时，拥塞窗口可以翻倍为 $4*2=8$ ，所以，在60~80毫秒期间，可以发送8个数据段。





# 练习

- 6、当 TCP 端点发送的段中包含 **ACK=10000**（超出初始序列号），以下哪一项是最恰当的表述？
- A. 端点已经接收到 到**9999** 序列号的数据， 下一个预期的序列号是 **10000**
  - B. 由端点接收到的最后一个数据段包含到**10000**的数据并包含序列号**10000**
  - C. 由端点接收到的最后一个数据段包含到**9999**的数据并包含序列号**9999**
  - D. 端点已经接收到 到**10000** 序列号的数据， 下一个预期的序列号是 **10001**

解析：**ACK=10000**，意味着**9999**编号及之前编号的字节都已经正确接收。表明期待发方发送的字节编号从**10000**开始。





# 练习

7、考虑从一个接收端收到的 ACK 数据段序列：第1个ACK 数据段中的ack=3000；第2个ACK 数据段中的ack=4000；第3个ACK 数据段中的ack=5000；第4个ACK 数据段中的ack=5000；第5个ACK 数据段中的ack=5000；第6个ACK 数据段中的ack=5000；第7个ACK 数据段中的ack=5000。假设发送端使用了快速重传。当它重传时，哪一个是由发送端重新传输的数据段？

- A. 收到第6个ACK 数据段后，发送序列号 5000 的数据段
- B. 收到第6个ACK 数据段后，发送序列号 6000 的数据段
- C. 收到第5个ACK 数据段后，发送序列号 5000 的数据段
- D. 收到第7个ACK 数据段后，发送序列号 6000 的数据段

解析：教材6.5.10小节中讲到，快速重传是TCP假设三个重复确认意味着已经丢失一个包，启动重传，而不必等到重传定时器超期。题目中第4个ACK数据段开始，连续4个数据段的确认号都是5000，意味着5000序列号的数据段丢失，根据快速重传的规定，在收到第三个重复确认的时候就启动了重传，重传序列号5000的数据段。





# 练习

8、主机甲和主机乙之间建立一个TCP连接，TCP最大段长度为1000B，若主机甲的当前拥塞窗口为4000B，在主机甲向主机乙连续发送两个最大段后，成功收到主机乙发送的第一段的确认段，确认段中通告的接收窗口为2000B，则此时主机甲还可以向主机乙发送的最大字节数是多少？

- A. 1000B
- B. 2000B
- C. 3000B
- D. 4000B

解析：发送窗口= $\text{Min}\{\text{接收窗口}, \text{拥塞窗口}\} = \text{Min}(2000\text{B}, 4000\text{B}) = 2000\text{B}$ ，现在，甲仅成功地发送了一个段(1000B)，当前拥塞窗口回退到1000B，所以，甲可发的数量应为： $\text{Min}(1000\text{B}, 4000\text{B}) = 1000\text{B}$ 。







# 练习

9、下面哪些端口号是合法的值？

- A. 0
- B. 513
- C. 65535
- D. 上面的所有

解析：0~65535的端口号都是合法的值。







# 练习

10、一个值为1000的确认号（TCP段）意味着下面哪项？

- A. 999字节已经被成功接收
- B. 1000字节已经被成功接收
- C. 1001字节已经被成功接收
- D. 其它三项都不正确

解析：确认号1000意味着它期望对方下一次发送数据段从1000编号的字节开始，并不表明前面收到了多少字节，因为题目没有告诉初始序列号。





# 练习

11、为了防止两个TCP实体之间的连接长时间空闲，启用了下面哪种定时器？

- A. 重传（retransmission）
- B. 持续（persistence）
- C. 保活（keepalive）
- D. 时间等待（time-waited）

解析：保活定时器用于防止一条连接长时间空闲，而这种空闲可能是因为一方已经崩溃离线。通常服务器会启动一个保活定时器，设为2小时，如果定时器超期之前，收到了客户端的数据，这条连接是有效的，如果定时器超期了，仍未收到客户端的数据，服务器会每隔75s发送一个探测包给客户端，仍然未收到任何回音，服务器就认为客户端down掉，关闭TCP连接。





# 练习

12、为了处理零窗口大小（`window size=0`）通告，启用了下面哪种定时器？

- A. 重传（`retransmission`）
- B. 持续（`persistence`）
- C. 保活（`keepalive`）
- D. 时间等待（`time-waited`）

解析：如果一个发方TCP实体收到了收方TCP实体的零窗口（`win=0`）通告，它必须等待收方的再次窗口更新通告，此时，必须同时启动一个持续定时器，以期待收到窗口更新，如果窗口更新丢失导致超时，发方TCP实体会发送也给probe，刺激对方发送窗口更新。所以，这个定时器可以防止双方陷于永久等待的死锁状态。





# 练习

13、最小TCP MTU的总长度是多少？包括TCP和IP的开销，但是不包括数据链路层的开销。

参考答案：

TCP默认有效载荷为536个字节，TCP头开销为20个字节，IP头开销为20个字节。因此最小TCP MTU的总长度是576个字节。





# 练习

14、TCP段的最大有效载荷为65495字节，为什么会选择这样奇怪的数值？

参考答案：

TCP被装载于IP的有效载荷中，IP的有效载荷为65515个字节，而TCP头为20个字节。因此TCP的有效载荷为 $65515 - 20 = 65495$ 字节。





# 练习

15、假设TCP的拥塞窗口被设置为18KB，并且发生了超时，如果接下来的4次和6次突发传输全部成功。试问这两次对应的拥塞窗口将达到多大？假设最大段长为1KB。

参考答案：

超时后，慢启动阈值为9KB，接下来的传输从1KB开始，4次传输全部成功，即1KB，2KB，4KB，8KB；因此拥塞窗口变成8KB。6次传输全部成功，即1KB，2KB，4KB，8KB，9KB，10KB；因此拥塞窗口变成10KB。



# 练习

编号	1	2	3	4	5	6	7	8	9	10
数据	0d	28	00	15	00	5f	a9	06	00	00
编号	11	12	13	14	15	16	17	18	19	20
数据	00	00	70	02	40	00	c0	29	00	00

16、一个TCP的头部字节数据(头部的前20个字节)如表所示，请根据表中数据回答问题：

- 1) 本地端口号是多少？目的端口号是多少？
- 2) 发送的字节序列号是多少？确认号是多少？
- 3) TCP的头部长是多少？
- 4) 使用该TCP连接的应用是什么？该TCP连接的状态是什么？

参考答案：

(1)源端口  $0x0d28=256*13+16*2+8=3368$ ，目的端口 $=1*16+5=21$

(2)序列号Seq= $0x005fa906=5*16^{**}5+15*16^{**}4+10*16^{**}3+9*16^{**}2+6$   
 $=5242880+983040+40960+2304+6=6269190$

确认号ack= $0x00000000=0$

(3)头部长HL= $0x7(4B)=7*4B=28B$

(4)这是ftp应用，目的端口21是著名端口号；这是第一次握手信息，在发起连接建立，因为SYN/ACK=1/0





# 对称释放和非对称释放

---

- 对称释放：通信双方都独立完成连接释放的请求并确认，可以看作完成两个方向的半连接释放
- 非对称释放：只要其中一方完成连接释放请求并确认即可（电话的工作模式，任意一方挂断电话）





# TCP连接释放

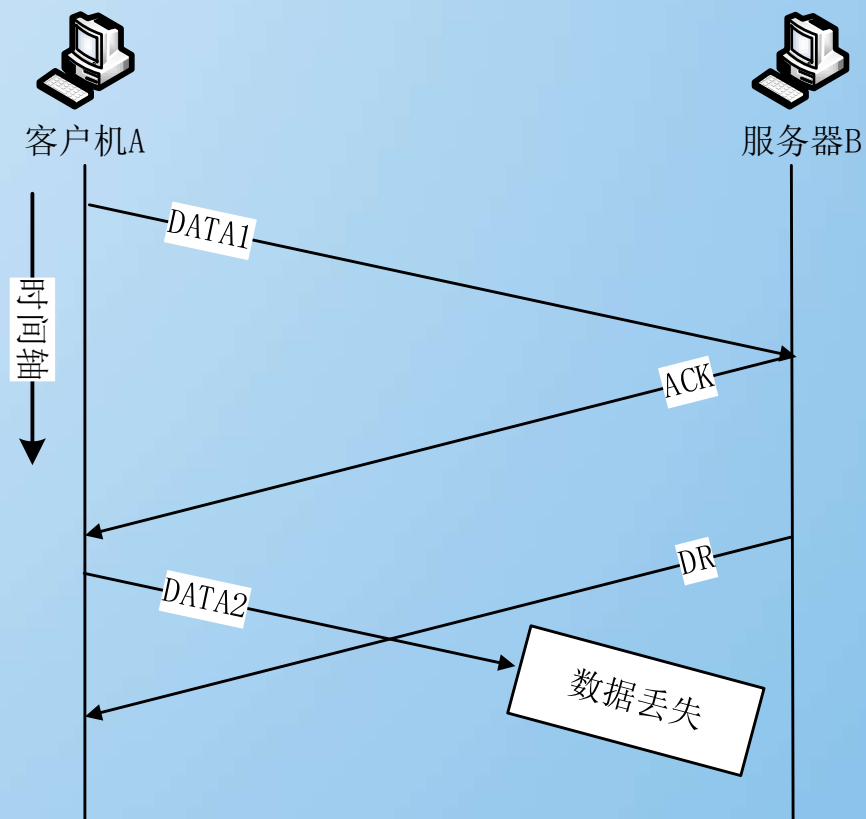
---





# TCP连接释放\_1

## ■ TCP采用对称释放避免数据段丢失



服务器提前Disconnect  
Request导致DATA2丢失

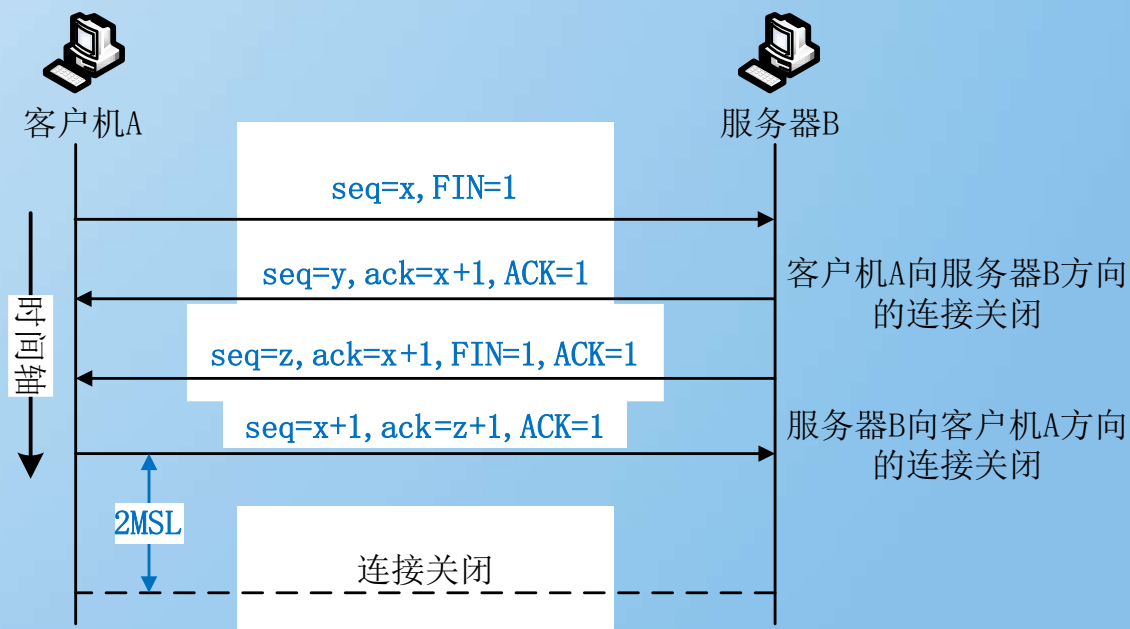




# TCP连接释放\_2

## ■ TCP释放连接通过四次握手

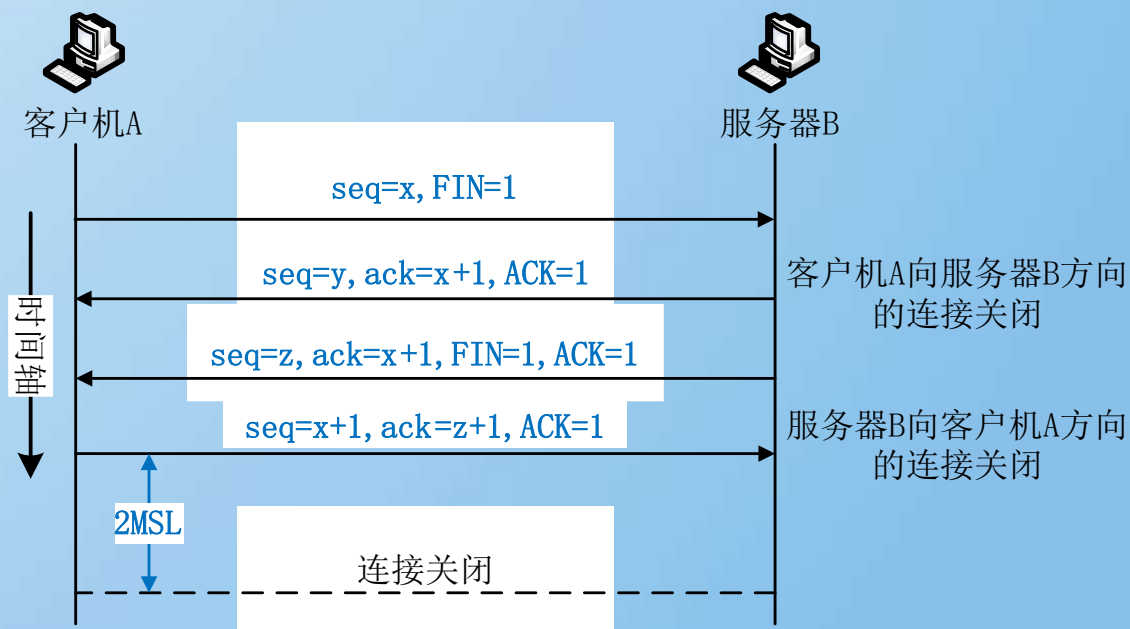
- 第一次握手：客户向服务器方向请求连接释放，FIN位置1，数据部分占1字节





# TCP连接释放\_3

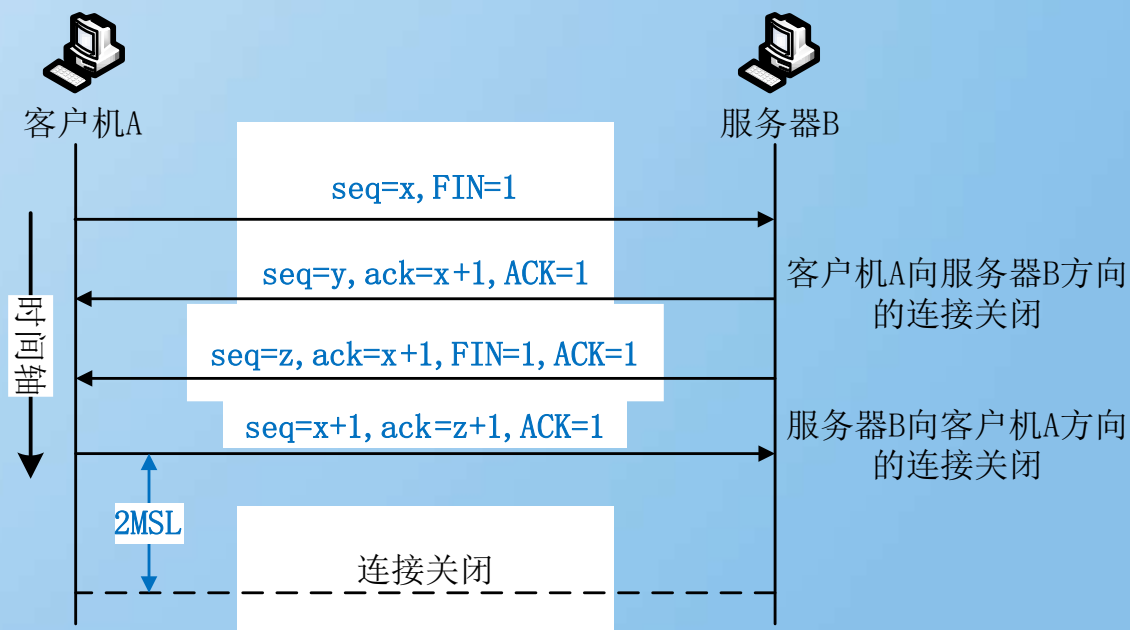
- 第二次握手：服务器对客户的连接释放请求确认，ACK置1，确认号ack为 $x+1$ ，此后客户向服务器方向连接关闭，但服务器仍然可以向客户传输数据





# TCP连接释放\_4

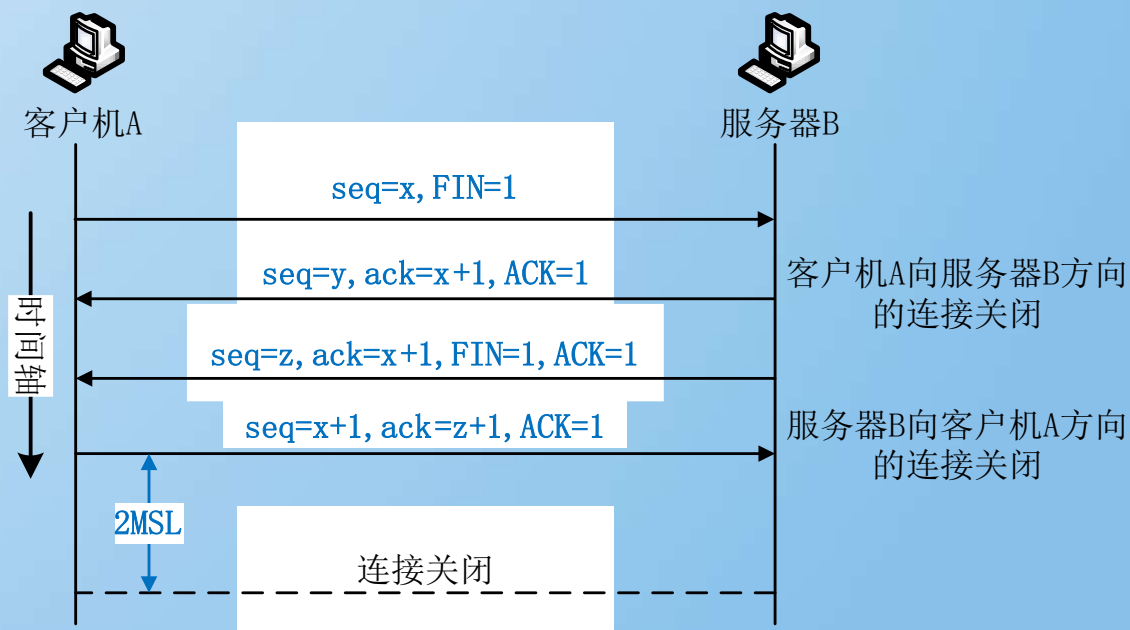
- 第三次握手：服务器向客户请求释放连接，序号字段为z（单向连接释放后服务器可能又发送了部分数据，因此z不一定等于y+1）  
数据段数据部分占1字节





# TCP连接释放\_5

- 第四次握手：客户发回确认数据段，数据段头部序号seq为 $x+1$ ，确认位ACK置1，确认号ack为 $z+1$ ，客户需要等待2倍最长段寿命（MSL）之后才完成客户向服务器方向的连接关闭





# 主动释放连接一方需要等待2MSL

---

- 主动释放连接一方需要等待2MSL，原因在于：
  - 若最后一个确认数据段丢失而导致服务器没能收到确认，需要重传确认数据段
  - 避免本次连接产生的数据段影响下一次连接（不排除由于路由、带宽等各方因素，部分数据段很迟甚至在下一次连接建立后才到达，导致接收一方无法分辨数据段属于哪次连接）因此需要等待2MSL使本次连接所有数据段在中间链路和转发节点上消失





# TCP连接管理状态机\_1

## ■TCP连接管理状态机使用的状态及描述

状态↵	描述↵	↵
CLOSED↵	没有活跃的连接或者挂起↵	↵
LISTEN↵	服务器等待入境呼叫↵	↵
SYN·RCVD↵	到达一个连接请求，等待 ACK↵	↵
SYN·SENT↵	应用已经启动了打开一个连接↵	↵
ESTABLISHED↵	正常的数据传送状态↵	↵
FIN·WAIT1↵	应用没有数据要发了↵	↵
FIN·WAIT2↵	另一端同意释放连接↵	↵
TIME·WAIT↵	等待所有数据包寿终正寝↵	↵
CLOSING↵	两端同时试图关闭连接↵	↵
CLOSE·WAIT↵	另一端已经发送关闭连接↵	↵
LAST·ACK↵	等待所有数据包寿终正寝↵	↵







# TCP连接管理状态机\_2

## ■ TCP客户进程正常变迁:

CLOSED->

SYN-SEND->

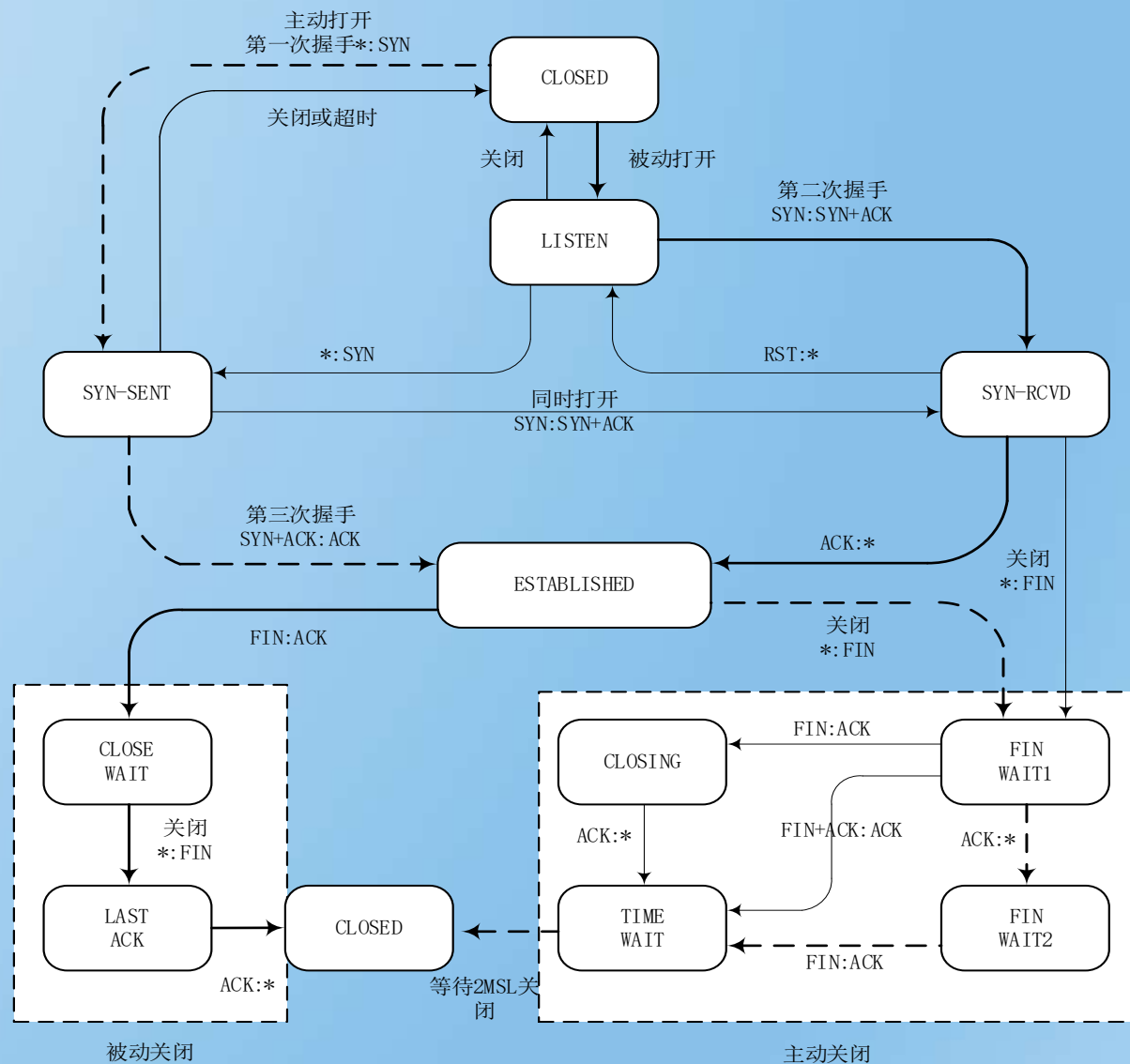
ESTABLISHED->

FIN-WAIT1->

FIN-WAIT2->

TIME-WAIT->

CLOSED





# TCP连接管理状态机\_3

## ■ TCP服务器进程正常变迁:

CLOSED->

LISTEN->

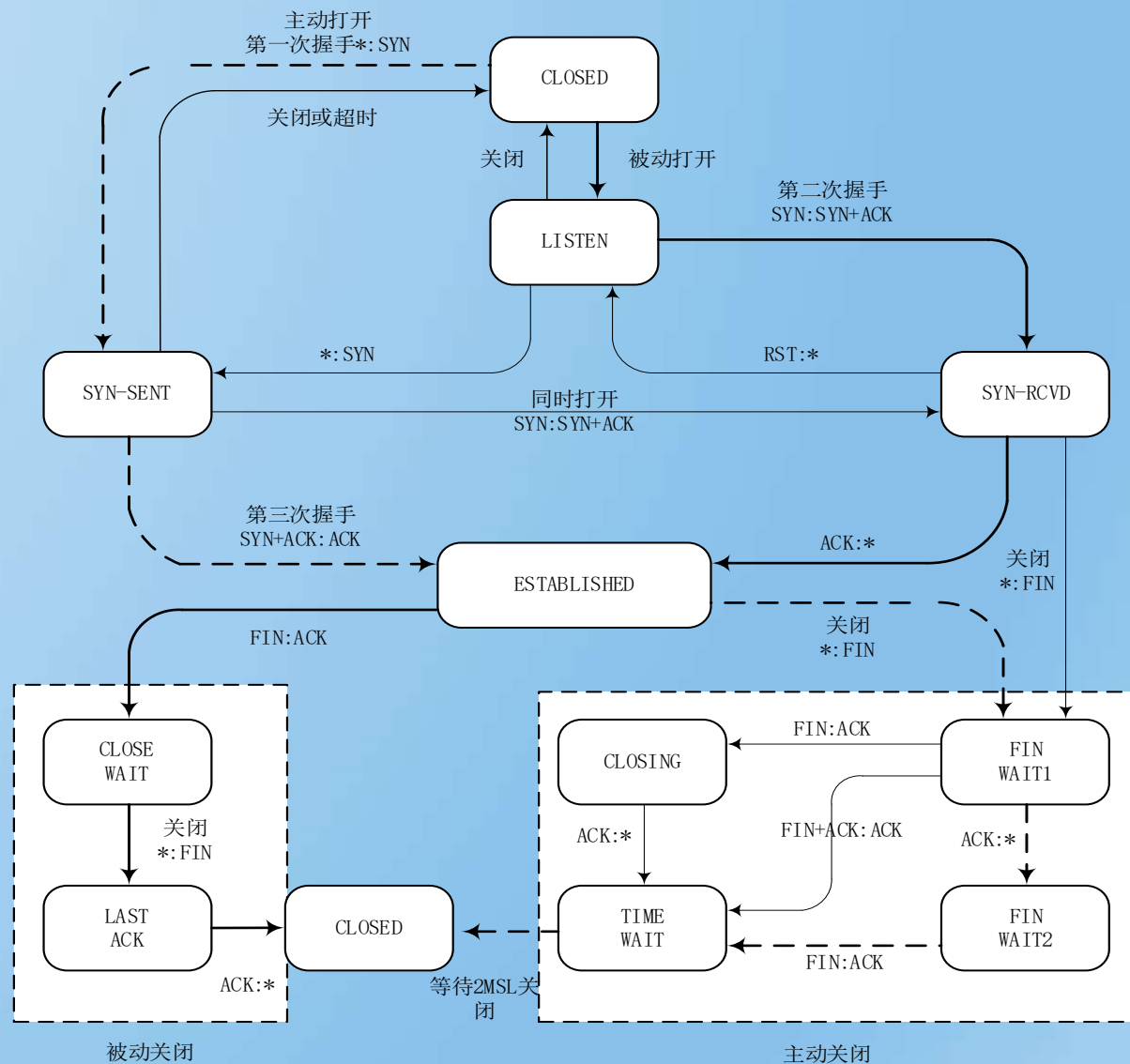
SYN-RCVD->

ESTABLISHED->

CLOSE-WAIT->

LAST-ACK->

CLOSED



# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- TCP连接管理 (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)





# TCP计时器管理

- TCP通过计时器跟踪数据发送情况，解决无限等待和死锁的情况

名称↵	功能↵
重传计时器↵	对数据段发过去和确认回来所需要的往返时间进行预估，设置数据段重传超时时间↵
持续计时器↵	每隔一定时间向对方发送探测数据段，避免发送方和接收方都会处于等待对方下一步操作的过程而陷入死锁↵
保活计时器↵	防止两个 TCP 之间的长期空闲连接，定时发送探测数据段↵
时间等待计时器↵	当连接被关闭后启动，等待足以确保连接上创建和传输的所有数据段都已完全传输完毕或消失的时间↵





# TCP重传时间选择

---

- 重传计时器跟踪发送窗口中已经发送，但还没确认的数据段，在计时器超时时启动重传
- 重传计时器需要对数据段从发送到确认所需的往返时间（RTT）进行预估，超时时间应略大于往返时间
  - 重传计时过长->数据段丢失没能及时启动重传，加大数据段传输时延
  - 重传计时过短->数据段或确认并没有丢失的情况下将启动不必要的重传
- 重传机制设计和实现是TCP最重要同时也最复杂的问题之一





# TCP重传超时时间计算

- 指数加权移动平均 (EWMA) 动态算法

- 平均的往返时间 (SRTT)

$$SRTT = (1 - \alpha)SRTT + \alpha R_{\leftarrow}$$

- R: 往返时延最新采样值

- 往返时间差值 (RTTVAR)

$$RTTVAR = (1 - \beta) RTTVAR + \beta |SRTT - R|_{\leftarrow}$$

- TCP平均重传超时 (RT)

$$RT = SRTT + 4 \times RTTVAR_{\leftarrow}$$





# TCP往返时延更新

---

- 往返时延R的采样面临一个问题：当发送方重发一个超时的数据段后，如果收到一个对应序号的确认，如何判断该确认是对之前超时的数据段还是后来重发的数据段呢？
- Karn改进：重传数据段的往返时延R不再用于RT的更新，数据段每重传一次，就将重传时间增大一倍，直到不再发送重传事件才更新RT值





# TCP流量控制死锁

---

- TCP流量控制死锁：接收缓冲区大小为0时，设置确认数据段头部接收窗口为0，告知发送方暂停发送数据，直到接收方发送确认数据段声明接收窗口不再为0才解除暂停状态。假如声明窗口不再为0的确认数据段丢失，发送方将进入锁定状态无法解除，发送方和接收方都处于等待对方下一步操作而陷入死锁







# TCP持续计时器

---

- 接收窗口为0时，TCP启动持续计时器，当计时器超时，发送方主动发送一个探测数据段来触发接收方重新发送确认
- 探测数据段只包含一个字节数据，占一个序号，但该序号不被接收方考虑，后续数据段计算序号可以忽略该探测数据段序号
- 持续计时器超时时间间隔加倍增大，达到阈值（通常为60s）后每隔阈值时间发送探测数据段，直到接收方发回窗口值不为0的确认数据段





# TCP保活计时器

---

- 保活计时器被用于防止TCP实体之间的长期空闲连接
  - 例如，客户和服务端建立连接后，客户机出现故障，服务器不再收到客户发送的数据段，但又不知道发生什么而一直处于等待状态，白白浪费通信资源
- 保活计时器在服务器每收到一次客户数据段后重置，当计时器超时还没有收到数据段时，服务器主动发送一个探测数据段，并以预设的间隔（通常75s）持续发送，当连续发送10个探测数据段后仍没有收到响应，服务器认为客户机出现故障，主动关闭连接





# TCP时间等待计时器

---

- 时间等待计时器用于TCP连接释放后确保本次连接创建和传输的数据段在中间链路和转发节点上全部消失
- TCP释放连接第四次握手发送最后一个确认数据段后，主动关闭连接一方启动时间等待计时器，连接保持2MSL时间后才关闭
- 最长段寿命（MSL）：任意数据段在被丢弃前在网络中存在的时间，通常为30秒、1分钟、2分钟等





# 练习

---

1、TCP连接释放阶段，启用了下面哪种定时器？

- A. 重传（retransmission）
- B. 持续（persistence）
- C. 保活（keepalive）
- D. 时间等待（time-waited）

解析：时间等待定时器用于连接释放阶段，在time-wait状态启动该定时器，超时后，TCP连接关闭。





# 练习

---

2、下面哪个定时器追踪数据段发送和确认的接收之间的时间？

- A. 重传（retransmission）
- B. 持续（persistence）
- C. 保活（keepalive）
- D. 时间等待（time-waited）

解析：发方发送一个数据段，会启动一个重传定时器，期望超时之前收到对方的确认，如果超时未收到确认，超时会触发重新发送。





# 练习

---

3、建立和释放TCP连接分别需要几次握手信息？

- A. 一次、两次
- B. 两次、三次
- C. 三次、三次
- D. 三次、四次

解析：三次握手建立TCP连接；而释放TCP连接相当于释放两根单向连接，**FIN+ACK**一来一回释放一个方向的连接；再进行**FIN+ACK**释放反方向的连接，此时TCP连接才算关闭，所以，释放连接需要4次握手。





# 练习

---

4、一个服务器的TCP实体，处于下面哪种状态是数据传输状态，即可以接收也可以发送数据？

- A. ESTABLISHED
- B. LISTEN
- C. TRANSFER
- D. OPEN

解析：TRANSFER和OPEN不是TCP实体的状态，LISTEN状态用于接收客户端的请求，只有ESTABLISHED状态是建立好了连接之后的状态，可以在此状态下收发数据。





# 练习

---

5、当下面的哪个定时器超时的时候，发方TCP实体会发出一个叫probe的特殊数据段？

- A. 重传（retransmission）
- B. 持续（persistence）
- C. 保活（keepalive）
- D. 时间等待（time-waited）

解析：持续定时器和保活定时器都会发送probe探测数据段，但是保活定时器超时的probe是服务器端TCP实体发出的，只有持续定时器超时的probe是数据发送方TCP实体发出的。





# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- TCP连接管理 (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)





# 拥塞控制原理

---

- 网络拥塞 (network congestion) 是指分组交换网络中需要传输的数据段数量太多时, 由于网络转发节点的资源有限而造成网络传输能力下降的情况
  - 网络资源: 链路带宽、节点存储容量.....
  - 网络资源需求 > 网络承受能力





# 开环控制和闭环控制

---

- 开环控制在设计网络时预先考虑所有可能导致堵塞的因素，在设计阶段尽可能地避免网络实现后发生堵塞
- 闭环控制基于反馈环路的概念，需考虑以下几种措施：
  - 监测网络系统以检测何时何处出现堵塞 <--> 晚高峰交通堵塞
  - 将堵塞信息传送到可采取行动解决堵塞的地方 <--> 通知交警
  - 调整网络系统的运行以解决出现的问题 <--> 交警指挥交通





# 拥塞控制与流量控制的异同点

---

- 流量控制是为了防止发送方的发送速率超过接收方的接收能力而对发送方的发送行为进行限制，是一种点对点的通信量的控制
- 拥塞控制是为了防止注入网络的数据量超过网络中的路由器和链路可以承受的能力而增加的一种端到端的通信控制
- 两者在实现上最终都体现为对发送方发送速率限制





# 拥塞窗口

---

- 发送方维持一个状态变量拥塞窗口cwnd。拥塞窗口随网络的拥塞程度动态变化，为尽可能地利用网络资源，发送窗口应当接近拥塞窗口
- 同时考虑流量控制和拥塞控制时，发送窗口（Window\_send）等于接收窗口（rwnd）和拥塞窗口（cwnd）的较小值

$$Windows\_Send = \min (rwnd, cwnd) \leftarrow$$





# 如何检测拥塞？

---

- TCP发送方判断网络拥塞依据于：
  - 重传计时器超时的情况下，发送方认为网络拥塞导致了数据段或确认的丢失，并且拥塞程度比较严重
  - 收到三次重复确认的情况下，发送方认为由于网络拥塞导致某个数据段丢失，但因为还能收到对于其他数据段的确认数据段，则网络拥塞还不算很严重





# 何时调整拥塞窗口？

---

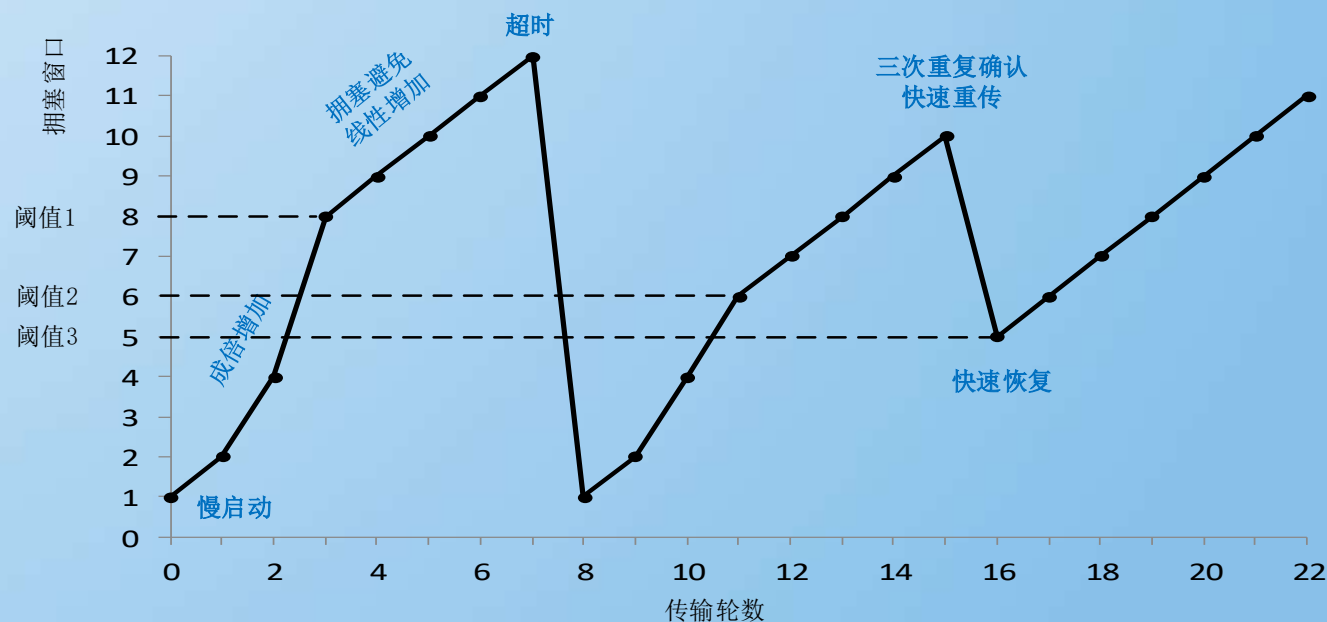
- TCP每隔一个传输轮次调整一次拥塞窗口
- 传输轮次指从把拥塞窗口内的数据段全都发送，并且接收到所有发送数据段的确认数据段
- 传输轮次值大致等于往返时间RTT，但传输轮次将窗口内的数据段看作一个整体看待





# 慢启动和拥塞避免\_1

- 发送方刚准备发送数据，尚不清楚网络当前的负荷情况，因此以尝试的方式来发送，先发送少一点的数据（1MSS），如果能顺利收到确认，则认为网络的状况比较好而逐步加大拥塞窗口
- 成倍增加阶段每隔一个传输轮次将拥塞窗口加倍

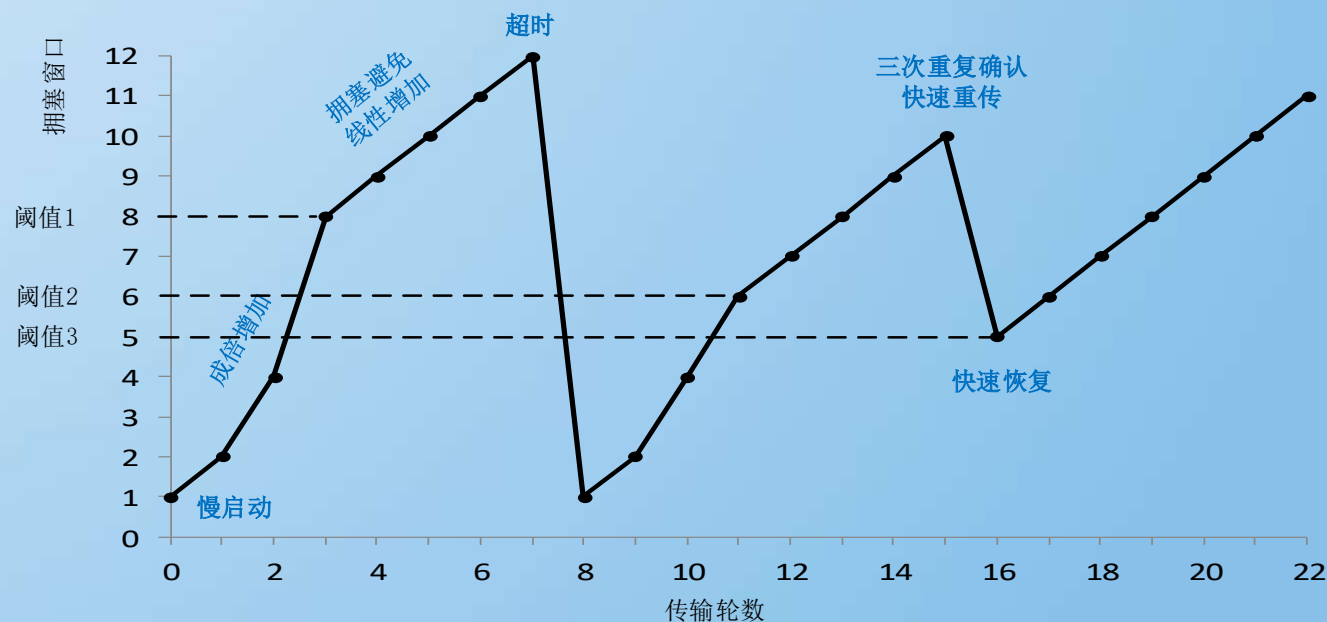






# 慢启动和拥塞避免\_2

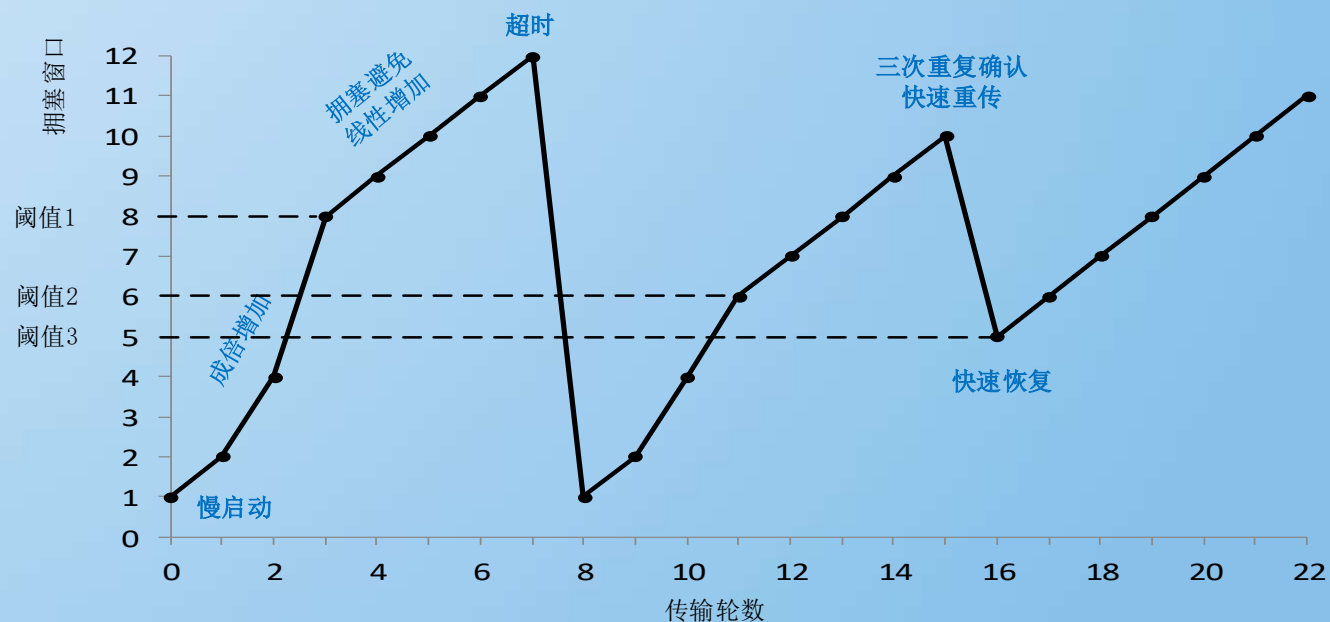
- 拥塞窗口值达到阈值后，采取谨慎一点的态度，不再以成倍的方式增大窗口值了，而是以线性方式增长，通过放缓增长速率以减少拥塞发生的可能
- 拥塞避免阶段每经过一个传输轮次就把拥塞窗口值加1MSS





# 重传计时器超时阶段

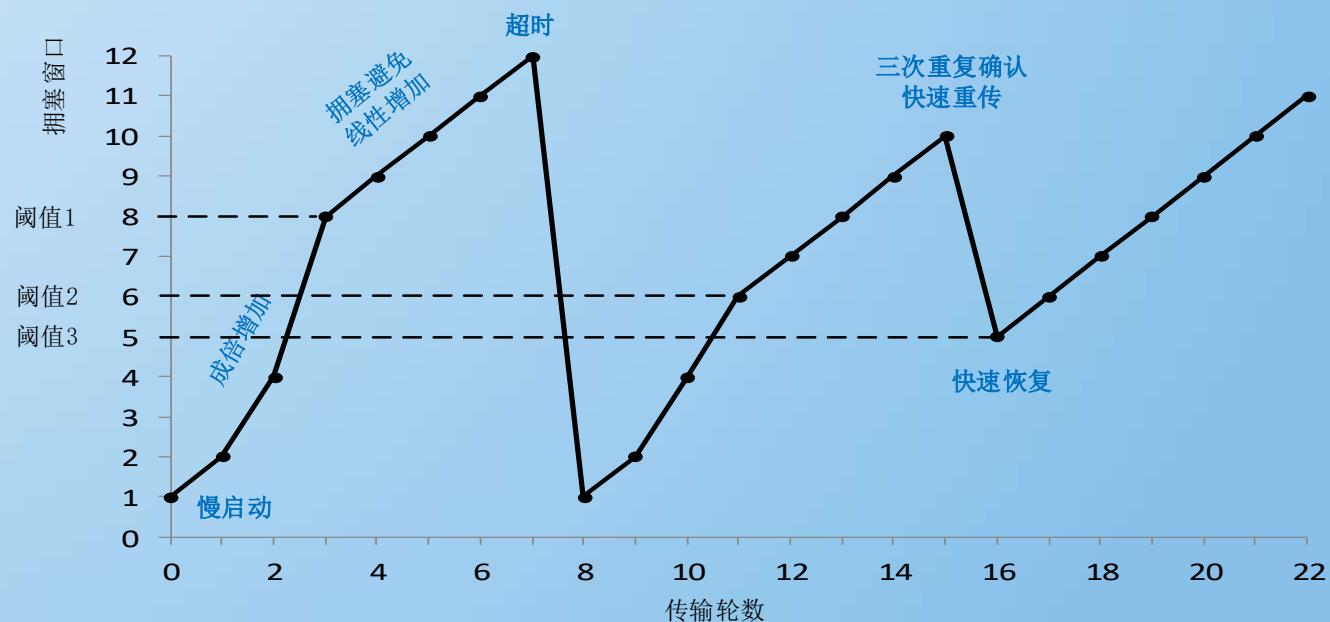
- 重传计时器超时，阈值 $ssthresh$ 设置为当前拥塞窗口值一半，将拥塞窗口 $cwnd$ 重新设置为1MSS，发送方重新进入慢启动阶段





# 快速重传和快速恢复阶段

- 连续收到三个重复确认时，立即重传数据段而不用等待计时器超时，将阈值 $ssthresh$ 设置为当前拥塞窗口值一半
- 快速恢复阶段将拥塞窗口 $cwnd$ 重新设置为调整后的阈值，从而跳过慢启动阶段

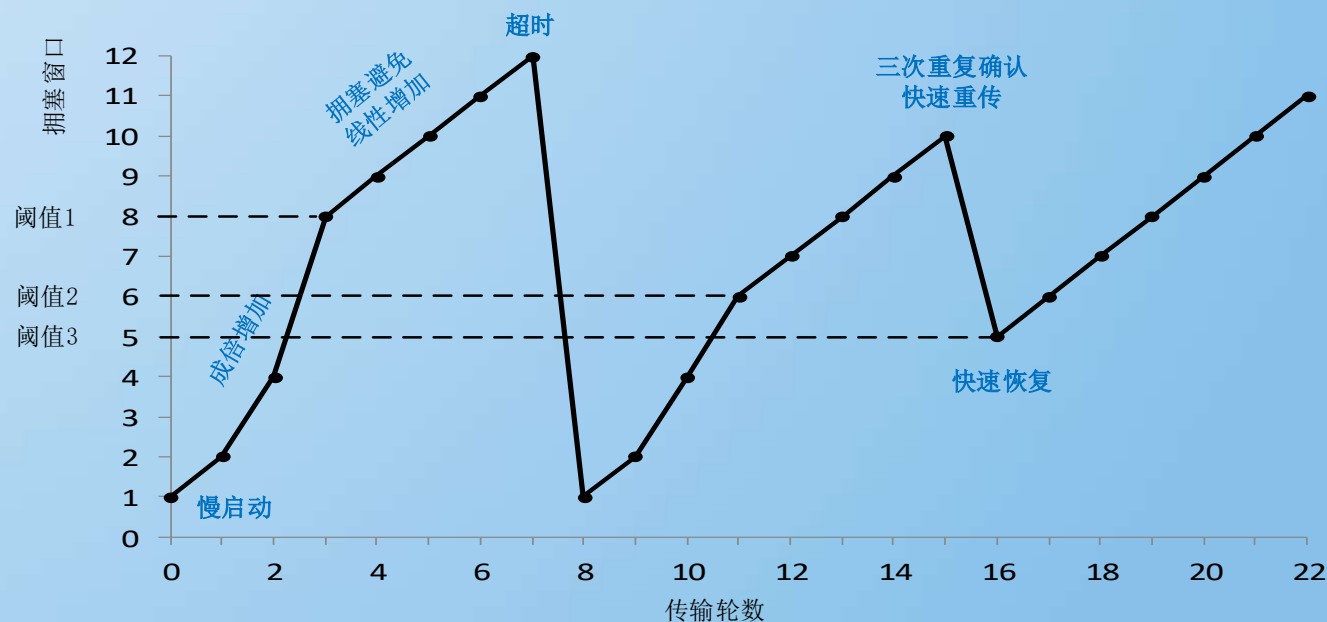




# 加法增、乘法减

## ■ TCP拥塞控制机制总结来说采用加法增、乘法减 (AIMD)

- 加法增：拥塞避免阶段拥塞窗口以线性方式缓慢增大
- 乘法减：只要出现超时或3次重复确认就将阈值减小一半





# 练习

1、UDP和TCP都是哪一层的协议？

- A. 物理层
- B. 数据链路层
- C. 网络层
- D. 传输层

解析：传输层只有两个主流协议，轻量级的用户数据包协议**UDP**和重量级协议：传输控制协议**TCP**。





# 练习

2、客户端程序由下面哪项定义？

- A. 临时端口号
- B. IP地址
- C. 知名端口号
- D. 物理地址

解析：客户端程序由端口号来定义，但知名端口号（0-1023）通常用于知名应用的服务端程序，而客户端程序通常使用临时端口号，或成动态端口号，49152~65535。





# 练习

3、服务端程序由下面哪项定义？

- A. 临时端口号
- B. IP地址
- C. 知名端口号
- D. 物理地址

解析：服务端程序由端口号来定义，而且是知名端口号（0-1023）；而客户端程序通常使用临时端口号，或成动态端口号，49152~65535。





# 练习

4、IP和TCP分别负责什么范围的通信？

- A. 主机到主机；进程到进程
- B. 进程到进程；主机到主机
- C. 进程到进程；网络到网络
- D. 网络到网络；进程到进程

解析：IP的作用范围是主机到主机，即数据分组从源IP到目的IP；而传输层TCP的作用范围是进程到进程（端到端），即数据段从源端点到目的端点。







# 练习

5、一台主机可以被什么地址标识？而一台主机上的运行进程可以被什么标识？

- A. IP地址；端口号
- B. 端口号；IP地址
- C. IP地址；主机地址
- D. 一个IP地址，一个著名端口

解析：一台主机由**IP**地址所标识，而应用进程是被端口号所标识，二者共同构成了通信五元组中的两个元素。”主机地址“是模糊，所以该选项不正确；进程不一定被著名端口号所标识，比如客户端进程是被动态临时的端口号标识，所以“著名端口号”选项不正确。





# 练习

6、如果一个传输数据的数据段携带了确认，这被称为什么？

- A. Backpacking（背包旅行）
- B. Piggybacking（背负式装运）
- C. Piggypacking（背负式封装）
- D. mother's helper（妈妈帮手）

解析：将确认搭载到数据段中，节约了单独封装所需要的开销，这种确认叫做捎带确认（piggybacking）





# 练习

7、下面哪个字段用作错误检测？

- A. 紧急指针
- B. 校验和
- C. 序列号
- D. 确认号

解析：校验和用于检查错误。紧急指针用于表明紧急数据的位置；序列号表明数据段中第一个字节的编号；确认号表明期待对方从这个编号的字节开始发送。





# 练习

8、为了阻止因为接收方极端低效地处理数据而带来的傻瓜窗口综合症，可以采用下面哪项来解决？

- A. Clark解决方案
- B. Nagle算法
- C. 被延迟的确认
- D. Clark解决方案和Nagle算法

解析：Clark解决方案是禁止接收端发送1个字节的窗口更新，相反，它强制接收端必须等待一段时间，累积到一定数量的空间之后，才更新窗口给对方。Nagle算法试图解决每次向TCP实体传送1个字节而引起的问题，所以它是发方的解决方案。题干描述的是收方引起的问题，应该由收方的Clark解决方案来解决。





# 练习

9、为了阻止因为发送方极低速地发送数据而带来的傻瓜窗口综合症，可以采用下面哪项来解决？

- A. Clark解决方案
- B. Nagle算法
- C. 被延迟的确认
- D. Clark解决方案和Nagle算法

解析：Clark解决方案是禁止接收端发送1个字节的窗口更新，相反，它强制接收端必须等待一段时间，累积到一定数量的空间之后，才更新窗口给对方。Nagle算法试图解决每次向TCP实体传送1个字节而引起的问题，所以它是发方的解决方案。题干描述的是发送方引起的问题，应该由发送方的Nagle算法来解决。





# 主要内容

- 传输协议概述 (6.1)
- UDP (6.2)
- TCP概述 (6.3)
- TCP连接管理 (6.4)
- TCP计时器管理 (6.5)
- TCP拥塞控制 (6.6)
- 新型传输协议QUIC (6.7)





# QUIC概述

- 快速UDP因特网连接（QUIC）是一种基于UDP的多流复用加密传输协议
- 设计QUIC的主要目的是最大限度地减少延迟、提高传输性能并提供默认使用TLS 1.3的安全通信

应用层	HTTP/2		HTTP over QUIC
网络安全	TLS		QUIC TLS 1.3
传输层	TCP		TCP-like拥塞控制 丢包检测、重传 UDP
网络层	IP		





# QUIC发展历程

---

- 2012年，Google工程师们设计出了QUIC协议的原始协议gQUIC。
- 2013年，Google公开了gQUIC协议并将其提交给IETF进行审议。
- 2015年6月，QUIC的Internet Draft提交到IETF进行标准化，同年成立了QUIC工作组。
- 2016年11月，国际互联网工程任务组(IETF)召开的第一次QUIC工作组会议开启了QUIC的标准化过程。
- 2018年10月，IETF的HTTP工作组和QUIC工作组联合声明了HTTP/3（即运行在QUIC之上的HTTP协议），但未完全确定其具体标准。
- 2021年5月，IETF宣布了QUIC的标准RFC 9000，该标准同时支持RFC 8999，RFC 9001和RFC 9002，至此形成了QUIC协议的完整标准。







# TCP协议存在不足

---

- 连接管理和可靠性机制带来的时间开销大。TCP连接建立和释放需要三次握手和两次对称释放，消耗较多的RTT时延。同时网络拥塞控制和可靠性传输机制会影响数据段的实时传递，降低网络传输效率
- TCP协议升级困难。TCP协议的实现在操作系统内核中，协议升级意味着升级操作系统及相关硬件，很难保证所有设备都能进行升级。同时，网络中间设备例如防火墙、NAT网关都对TCP数据段的通过对其头部做出相应修改，基于TCP的网络协议升级考虑因素较多，造成了升级TCP的困难





# QUIC性能优势

---

- QUIC是绕开升级TCP而直接设计的新型传输协议
  - QUIC降低了建立连接延迟，传输握手和交换密钥同时进行
  - QUIC提供了可拔插的拥塞控制，允许用户根据实际需求指定拥塞控制方法和调整参数
  - QUIC解决了TCP队头拥塞问题，降低等待数据段重传的阻塞延时
  - QUIC实现连接迁移，在客户端网络环境发生变化的情况下，无需重新建立连接，维持网络应用数据传输不中断、卡顿





# QUIC报文

---

- QUIC报文分为常规报文和特殊报文

- 常规报文

  - 帧报文

  - FEC报文

- 特殊报文

  - 版本协商报文

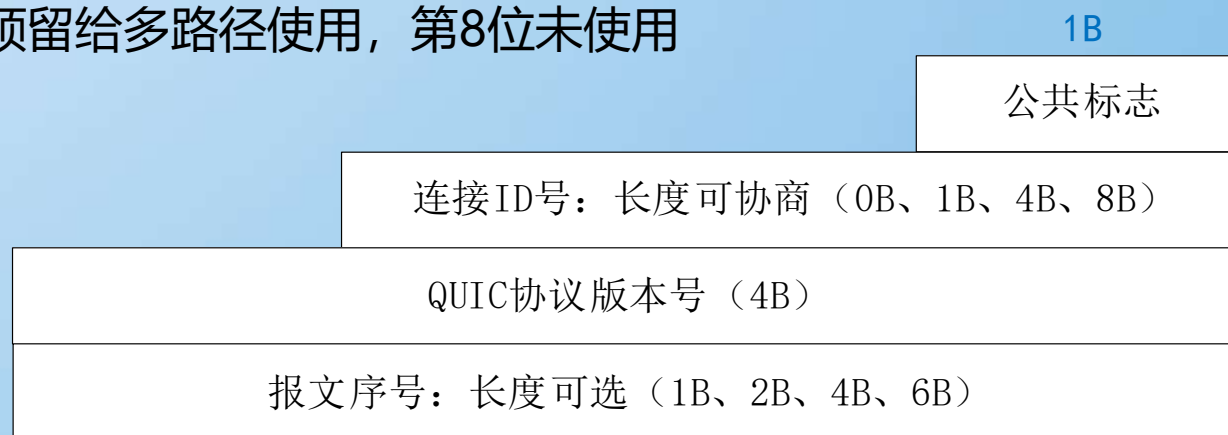
  - 公共重置报文





# QUIC公共头部\_1

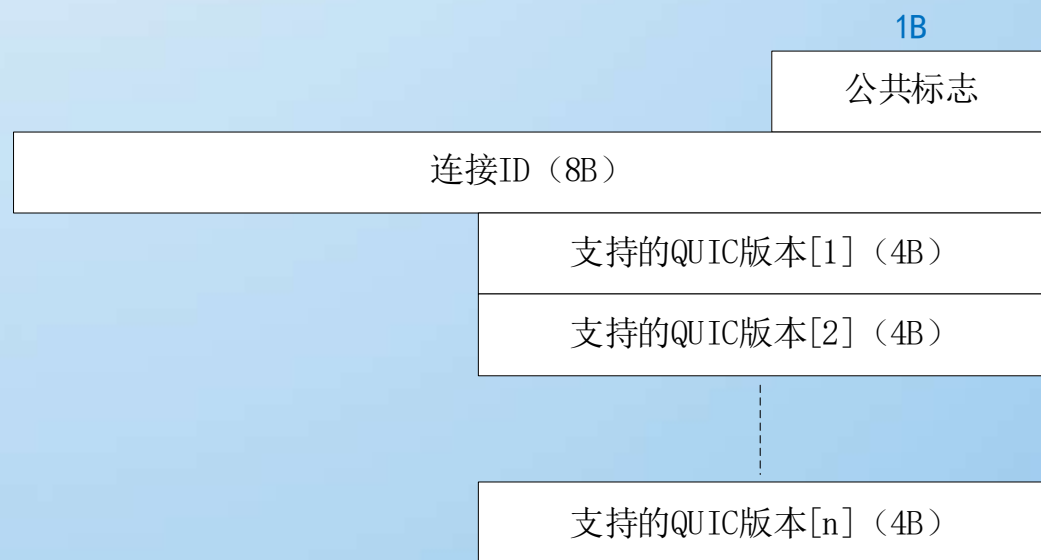
- 公共头部是常规报文和特殊报文都具有的固定字段，第1个字节为公共标志
- 公共标志第1位（最右，Bit0）为版本标识
  - 若客户端发送，且版本标识为1，则该头部包含QUIC版本号
  - 若服务器发送，且版本标识为1，该报文为版本协商报文
- 公共标志第2位为公共重置标识，为1代表报文为公共重置报文
- 公共标志第3、4位代表连接ID号长度，第5、6位代表报文序号字节长度，2位比特可表示4种长度
- 公共标志第7位预留给多路径使用，第8位未使用



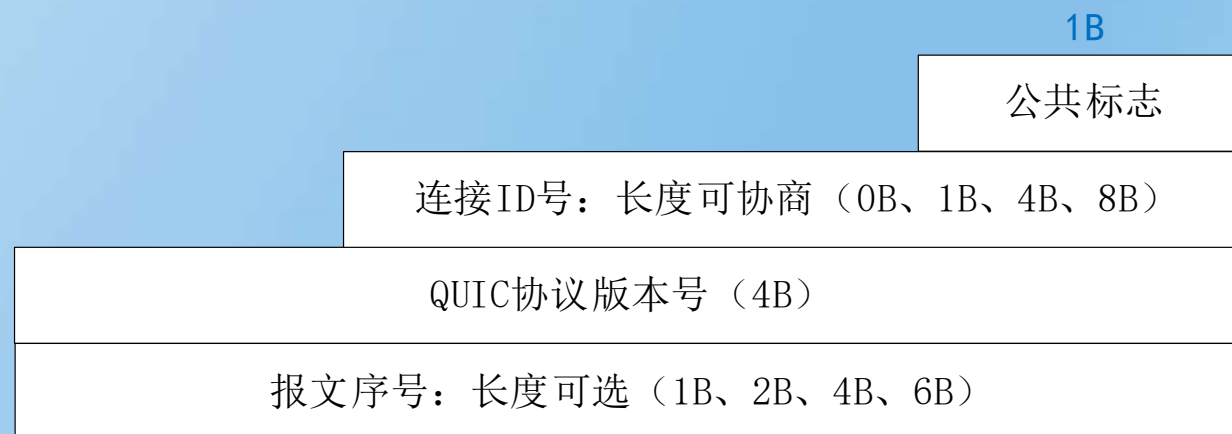


# QUIC公共头部\_2

- 连接ID是客户端和服务端识别连接的标识，长度可协商
- QUIC协议版本号需要在建立连接过程中协商，客户端将公共标志Bit0置1，并填写期望的版本号，若服务器对期望版本号不支持，需要在响应报文中将Bit0置1，并列0至多个服务端可支持的QUIC版本，且该字段后不能增加其他报文
- 报文序号长度可协商，各报文占据一个序号



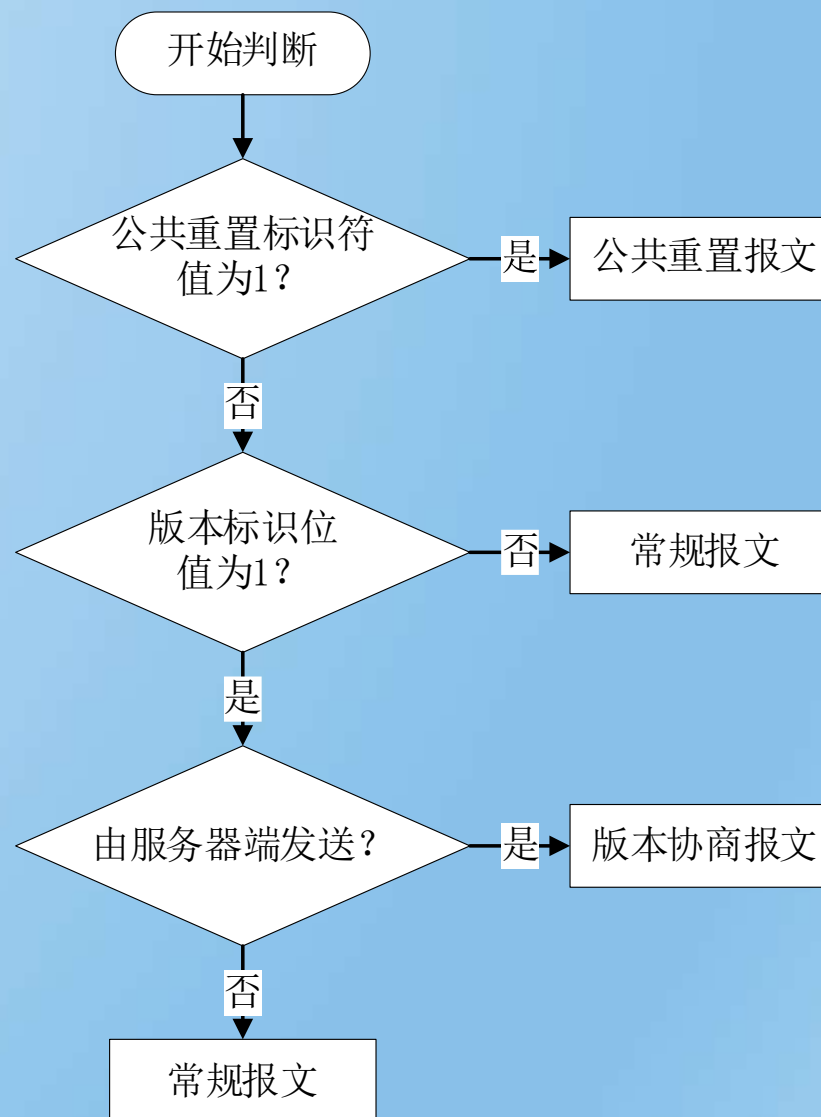
版本协商报文





# QUIC报文类型判断

- 根据公共标志以及报文传输方向，可判断报文类型
- 若公共重置标识符为1
  - 公共重置报文
- 否则，若版本标识不为1
  - 常规报文
- 否则，若来自服务端
  - 版本协商报文
- 否则为常规报文





# QUIC常规报文段

- 常规报文段的剩余部分由0至多个帧填充，每个帧通过类型字段标识紧跟其后的载荷类型
- 常规帧类型：PADDING、RST\_STREAM、CONNECTION\_CLOSE、GOAWAY、WINDOW\_UPDATE、BLOCKED、STOP\_WAITING、PING
- 特殊帧类型：STREAM、ACK、CONGESTION\_FEEDBACK





# QUIC流帧

- 流帧是QUIC中使用较多的帧类型，用于传输应用数据
- 类型最高位（Bit7）固定为1，代表流帧
- 类型Bit6位为1是代表终止位有效，发送方已完成此流上的发送，希望将连接设置为半关闭
- 类型Bit5位代表数据长度字段长度
- 类型Bit4、Bit3、Bit2位代表流偏移量字段长度
- 类型Bit1、Bit0位代表流ID字段长度

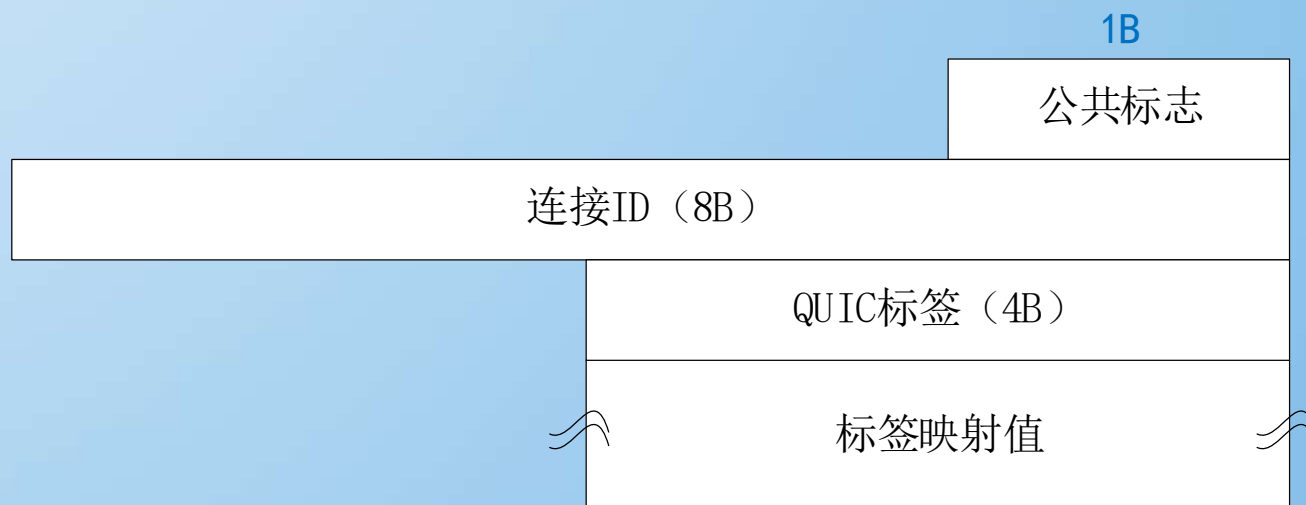






# QUIC公共重置报文

- 公共重置报文用于断开连接的服务器在其路由信息发生改变需要恢复连接或者重启时使用
- 标签映射值包括：
  - 公共重置随机数证明
  - 被拒绝的报文段序号
  - 客户端地址





# TCP队头阻塞问题

---

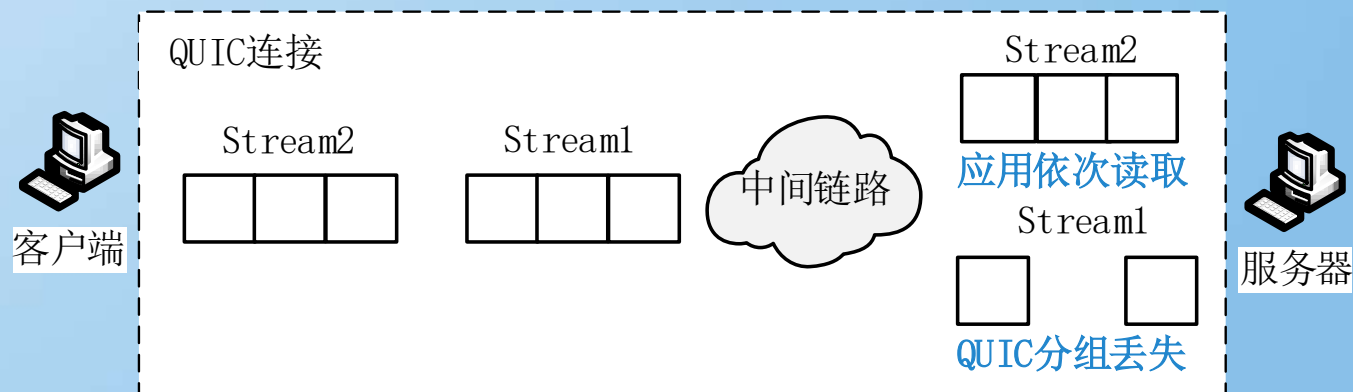
- 试考虑以下场景：TCP数据传输过程中，一旦某个数据段丢失，即便后续数据段被成功接收，也会被操作系统缓存在内核缓冲区中，无法被应用层读取，发送方也将因为没能收到确认消息而停止移动发送窗口，这一僵局将维持到数据段丢失被检测到并成功重传
- TCP队头阻塞：丢失的数据段将会推迟后续数据段被读取和发送的时间





# 多流复用

- “流”的产生是为了实现复用，在单个连接上同时传输多种业务数据，每种业务数据在一个流内处理完成，从而避免重复建立和销毁连接的时间开销
- 多流复用也有利于解决队头阻塞问题，流之间逻辑独立，Stream1的数据段丢失不影响Stream2完整数据交付，接收端根据流ID分用不同流数据





# QUIC序号与TCP序号差异

---

- QUIC虽基于UDP，但为实现可靠传输而在公共头部设计报文序号字段，QUIC的报文序号为严格递增型，当报文序号为X的数据段丢失时，重传数据段的报文序号不等于X，而是 $X+Y$  ( $Y \geq 1$ )
- 单调严格递增的报文序号可以实现乱序确认，发送方不必严格等待按序确认报文的到达才移动发送窗口，只要有新的报文段确认到来，发送窗口就可以继续滑动，当出现报文段X丢失时，将该报文段按照最新的可用序号 $X+Y$ 重传
- 接收方如何还原数据段的原始顺序呢？
  - 通过帧报文头部的流ID识别数据属于哪个流
  - 通过流偏移量字段标识当前帧在所属流中的字节偏移量，从而保证报文正确性





# QUIC连接管理\_1

---

- QUIC将传输握手和密钥交换（借助Diffie-Hellman算法）相结合来尽可能地降低连接延迟
- TCP完成握手需要1RTT，采用传输层安全协议（TLS）交换密钥需要1~2RTT
- QUIC首次连接需要1RTT，重复连接需要0RTT
- QUIC对于频繁建立和断开连接的移动应用十分具有吸引力





# QUIC连接管理\_2

---

- 客户端没有服务器的相关配置信息时，通过以下过程与服务器建立连接。
  - 客户端向服务器发送Inchoate CHLO消息请求建立连接，尝试获取服务器配置参数
  - 服务器收到Inchoate CHLO后回复REJ消息，其中包含了：服务器公开值（long-term Diffie-Hellman public value）、认证服务器的证书链、使用链叶节点的私钥加密的服务器配置签名等
  - 客户端提取REJ消息中的服务器配置参数，通过证书验证服务器身份，通过本地的密钥（ephemeral Diffie-Hellman private key）和解析出的服务器公开值生成初始密钥，将客户端公开值（ephemeral Diffie-Hellman public value）携带在complete CHLO消息中发送给服务器





# QUIC连接管理\_3

- ❑ 客户端发送完complete CHLO消息后，不需要等待该消息的响应，可通过服务器公开值与本地密钥生成初始密钥，基于初始密钥发送加密后的请求，因此从开始建立连接到发送第一个携带应用数据的请求只经历了1RTT
  - ❑ 服务器收到complete CHLO消息，解析出客户端公开值，结合本地的密钥生成同样的一个初始密钥，并向客户端回复用初始密钥加密的SHLO消息，其中包含一个临时随机数（ephemeral public value），用于客户端生成会话密钥，服务器在发送完SHLO消息后，后续的响应都将使用会话密钥加密，初始密钥不再被使用
  - ❑ 客户端收到SHLO消息，利用服务端配置信息和临时随机数计算出会话密钥，此后的请求都将使用会话密钥进行加密
  - ❑ 客户端和服务端之间，利用会话密钥进行加密通信
- 客户端缓存有服务器配置信息时，建立连接将从第3步开始，从开始建立连接到发送第一条应用请求经历了0RTT





# QUIC连接迁移

---

- TCP协议通过一对套接字或者四元组（源IP、源端口号、目标IP、目标端口号）来标识每个连接，通信主机因移动而带来的IP地址和端口号变化使得连续经常需要断开和重新连接
- QUIC的连接标识为公共头部的连接ID字段，在通信主机发生移动时维持连接ID不变，通过控制块对源IP和源端口号进行更新







# 本章小结\_1

---

- 传输层多路复用/分用功能，端到端数据段传输概念，端口号功能，端口的分类，常见的熟知端口号，套接字组成要素
- 面向无连接的服务，UDP的概念及特点，协议数据段格式，校验和计算
- UDP的典型应用，远程过程调用和实时传输协议原理
- 面向连接的服务，TCP的概念及特点，服务模式、协议数据段格式，TCP可靠数据传输机制，TCP基于滑动窗口的流量控制机制





## 本章小结\_2

---

- TCP连接管理，三次握手建立连接和释放连接的过程和必要性，TCP连接管理状态机
- TCP计时器管理，重传计时器、持续计时器、保活计时器和时间等待计时器功能，TCP重传时间估计，Karn平均往返时间算法
- TCP拥塞控制原理和机制，慢启动、快速重传、快速恢复和拥塞避免阶段工作原理，AIMD算法主要思想
- TCP协议面临的主要技术问题，QUIC优化思路、数据段格式、多流复用和连接管理方式





# 本章对应的中英文术语\_1

---

- Logic Communication:逻辑通信, multiplexing:复用, demultiplexing:分用
- half-association:半相关, full-association:全相关
- 互联网名称与数字地址分配机构 ( Internet Corporation for Assigned Names and Numbers, ICANN )
- socket:套接字
- 用户数据报协议 (User Datagram Protocol, UDP)
- 远程过程调用 (Remote Procedure Call, RPC)
- 实时传输协议 (Real-time Transport Protocol, RTP)





# 本章对应的中英文术语\_2

---

- 传输控制协议 (Transmission Control Protocol, TCP)
- 最大数据段长度 (Maximum Segment Size, MSS)
- 数据段往返时间 (Round trip time, RTT)
- 最长段寿命 (Maximum Segment Lifetime, MSL)
- 网络拥塞 (network congestion)
- 快速UDP因特网连接 (Quick UDP Internet Connection, QUIC)





# 致谢和声明

---

- 本课程课件中的绝大部分插图来自于
  - 袁华，王昊翔，黄敏主编《深入理解计算机网络》，清华大学出版社
- 其余的部分素材来自于
  - 华为智能基座课程
  - 思科网络技术学院教程
  - 网络上搜到的其它公开资料
- 特别对资料的各提供机构和个人表示诚挚的感谢！
- 引用的素材，仅用于课程学习，如果有任何问题，请与我们联系！

