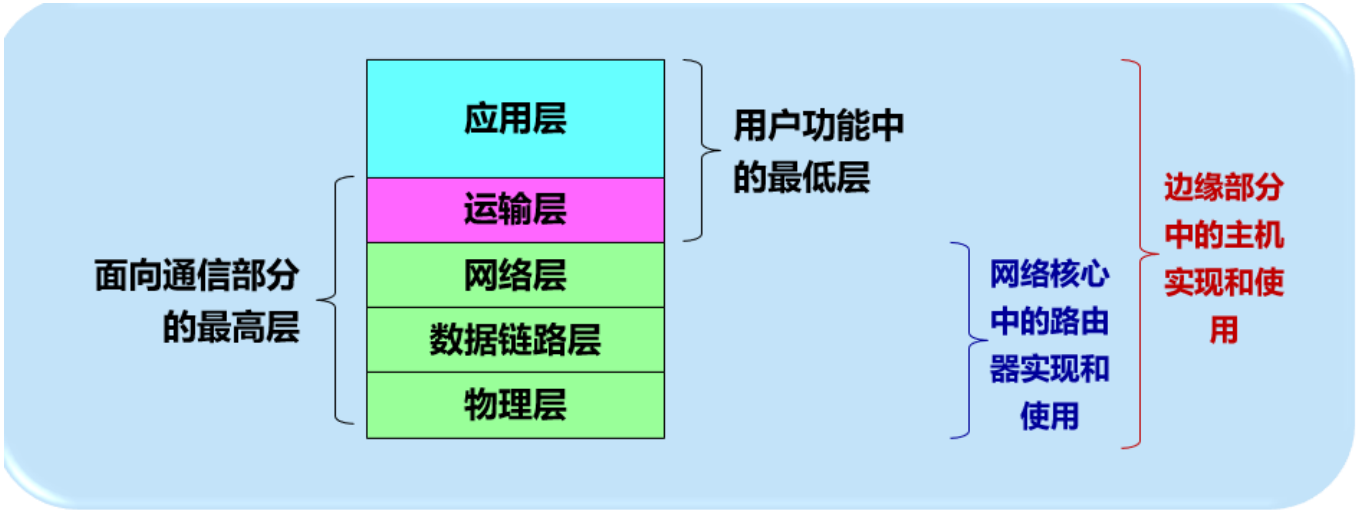
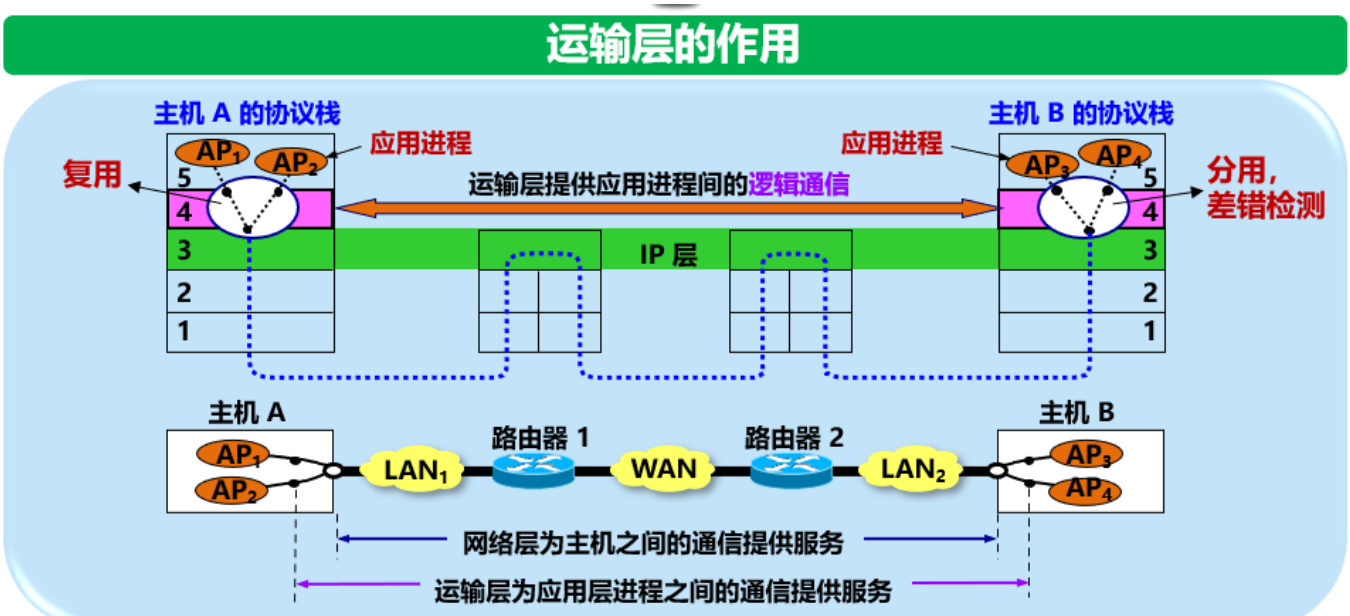
运输层基本概念.excalidraw

TCP&UDP.excalidraw



这个是什么意思



上面复用和分用体现（4层时候）

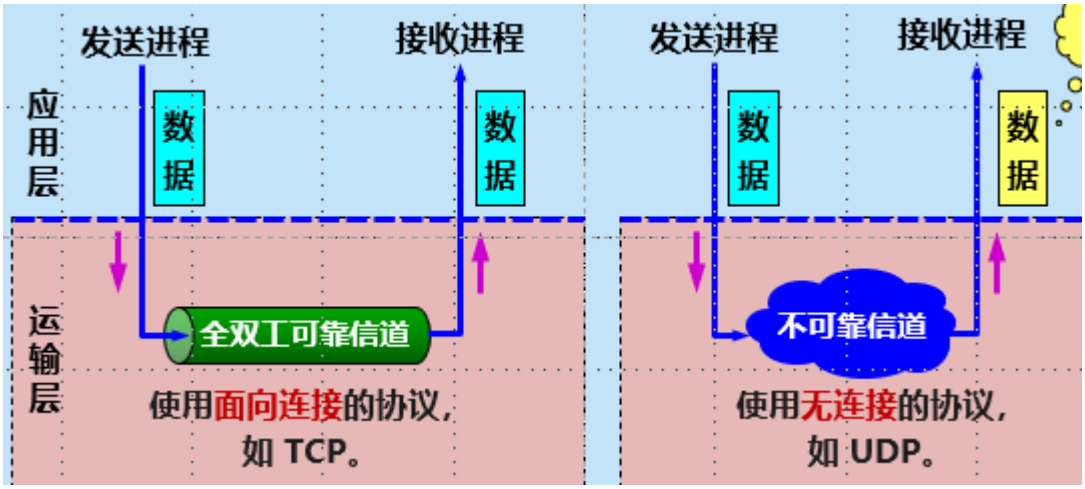
复用时候5把协议分别给4，分用4根据协议分别拆给5的两个程序

- **复用：应用进程都可以通过运输层再传送到 IP 层（网络层）。**
- **分用：运输层从 IP 层收到发送给应用进程的数据后，必须分别交付给指明的各应用进程。** **如何指明各应用进程?**

协议

运输层UDP TCP

UDP不需要发送链接，不需要接受确认



传输的报文

TCP 传送的数据单位协议是 TCP 报文段 (segment)。

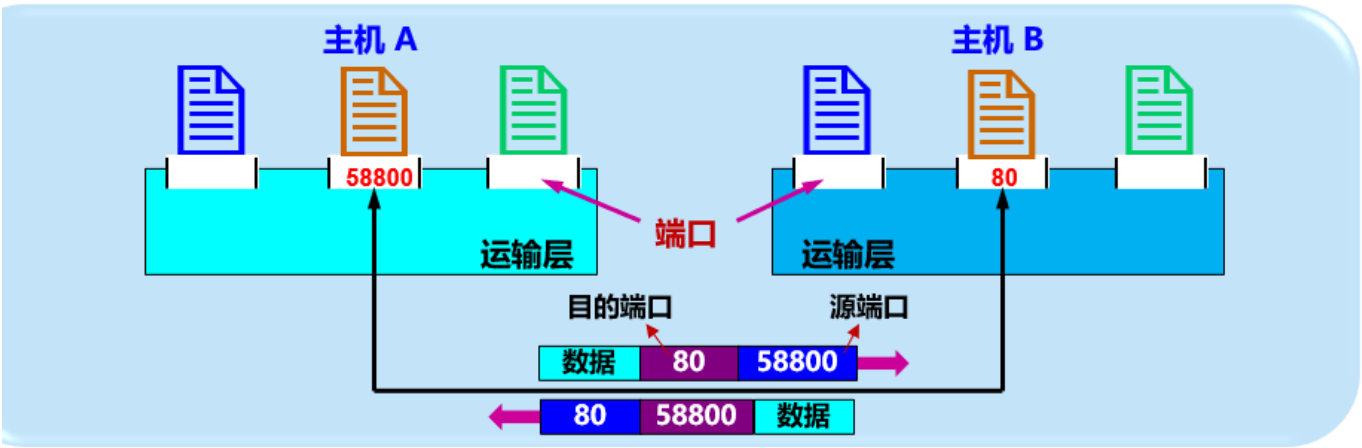
UDP 传送的数据单位协议是 UDP 报文或用户数据报。

程序：硬盘的，放在堆里面的代码

进程：栈里面的运行的

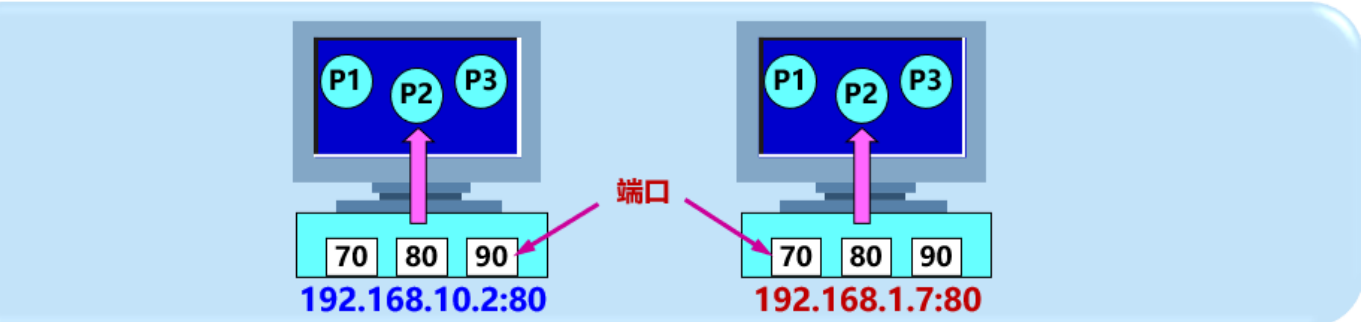
DNS将域名解析IP地址，但是我们不知道要给哪个进程

解决方法：在运输层使用协议端口号 (protocol port number)，或通常简称为端口 (port)。把端口设为通信的抽象终点。

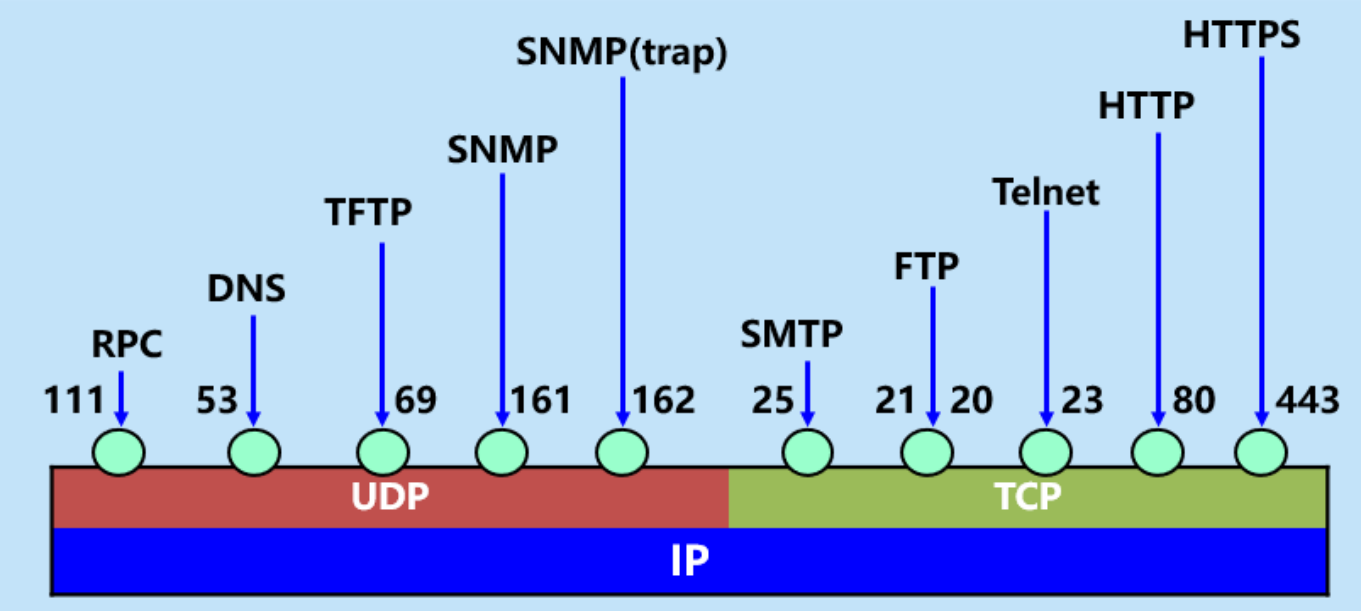


虚拟的端口，一个进程对应一个端口

端口用一个 16 位端口号进行标志，允许有 65,535 个不同的端口号。
端口号只具有本地意义，只是为了标志本计算机应用层中的各进程。
在互联网中，不同计算机的相同端口号没有联系。



注意区分IP地址，形式非常像
计算机通信，需要IP+端口号



很好这个要背

上网页时候，个人自己主机随机创建的临时端口号

54.某客户使用 UDP 将数据发送给一服务器，数据共 16 字节。试计算在运输层的传输效率（有用字节与总字节之比）。

解：UDP的数据报总长度=16+8=24字节

传输效率=(16/24)×100%=66.7%

55.重做 54，但在 IP 层计算传输效率。假定 IP 首部无选项。

解：IP数据报总长度=24+20=44字节

传输效率=(16/44)×100%=32.4%

56.重做 54，但在数据链路层计算传输效率。假定 IP 首部无选项，在数据链路层使用以太网。

解：以太网有 14 字节的首部，4 字节的尾部（FCS 字段），发送以太网的帧之前还有 8 字节的前同步码。但其数据字段的最小长度是 46 字节，而我们的 IP 数据报仅有 44 字节，因此还必须加上 2 字节的填充。这样，以太网的总长度 = 14 + 4 + 8 + 2 + 44 = 72 字节。

传输效率=(16/72)×100%=22.2%

太经典太全面！！！

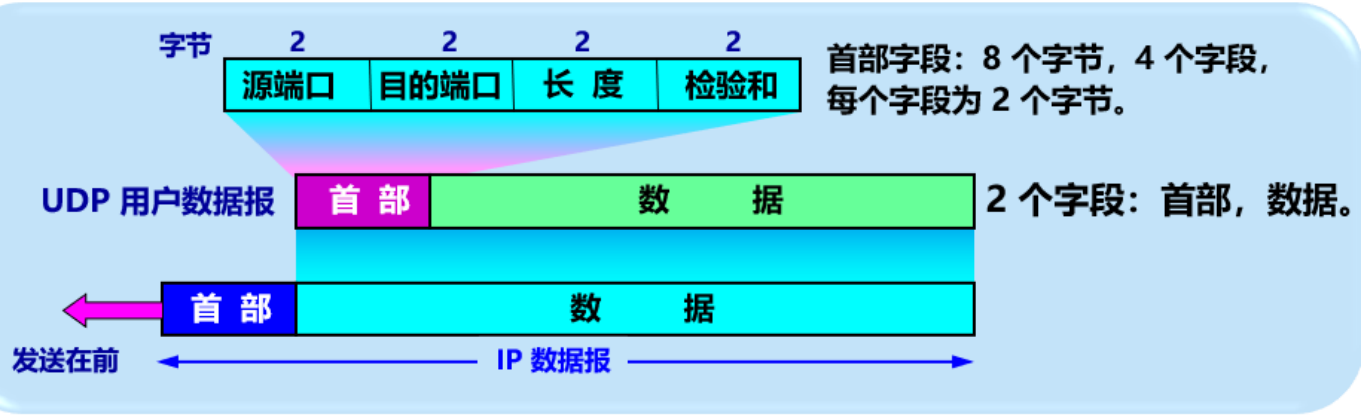
UDP

1. **无连接。发送数据之前不需要建立连接。**
2. **使用尽最大努力交付。即不保证可靠交付。**
3. **面向报文。UDP 一次传送和交付一个完整的报文。**
4. **没有拥塞控制。网络出现的拥塞不会使源主机的发送速率降低。很适合多媒体通信的要求。**
5. **支持一对一、一对多、多对一、多对多等交互通信。**
6. **首部开销小，只有 8 个字节。**

UDP 通信的特点：简单方便，但不可靠。

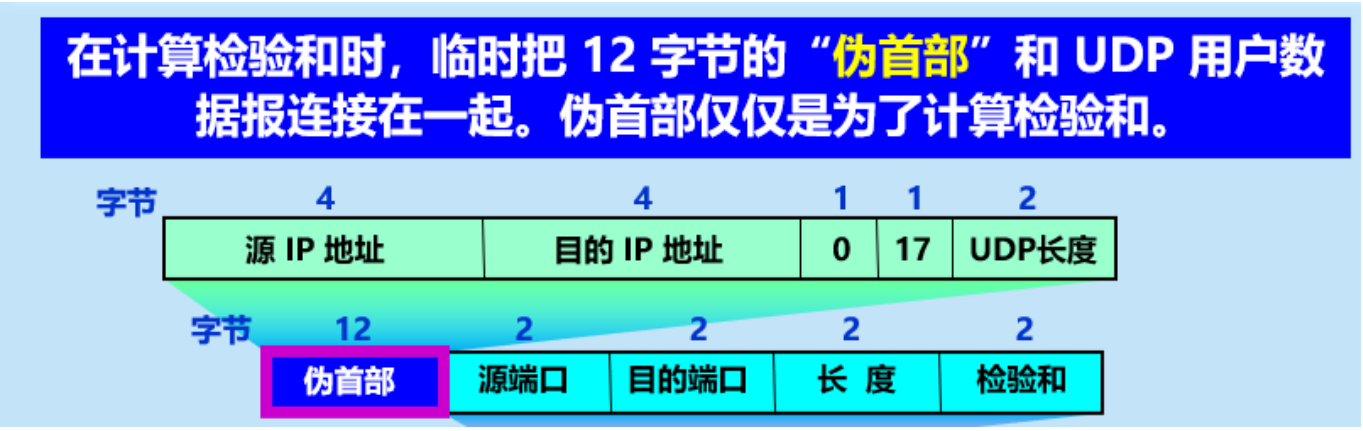
3. 面向报文（TCP面向字节流）：只是加一个头部，不出现分片
5. 多个方式通信都可以
6. 只有8个字节（记牢）

5.2.2 UDP 的首部格式



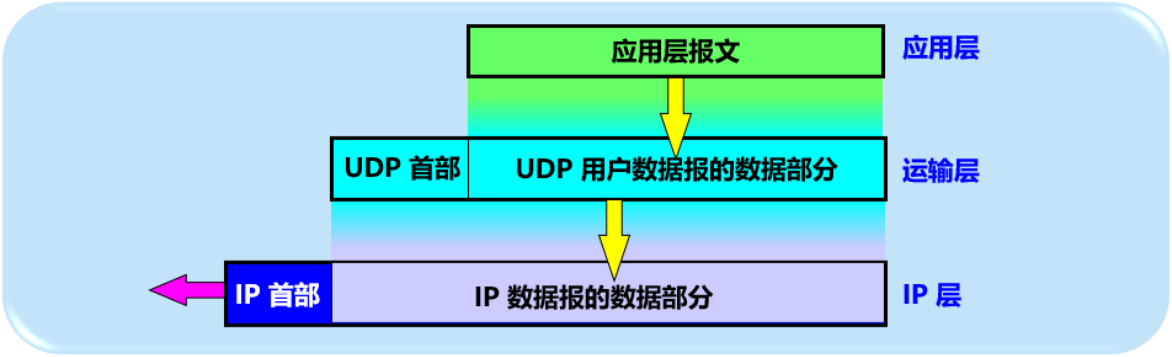
(1) 源端口：源端口号。在需要对方回信时选用。不需要时可用全 0。
(2) 目的端口：目的端口号。终点交付报文时必须使用。
(3) 长度：UDP 用户数据报的长度，其最小值是 8（仅有首部）。
(4) 检验和：检测 UDP 用户数据报在传输中是否有错。有错就丢弃。

UDP校验和包含数据
UDP的端口号不正确（不存在该端口的应用进程）
ICMP只有终点不可达



- 应用场景
  我需要实时的看到你的图像跟声音，至于中间丢一帧什么的完全不重要。

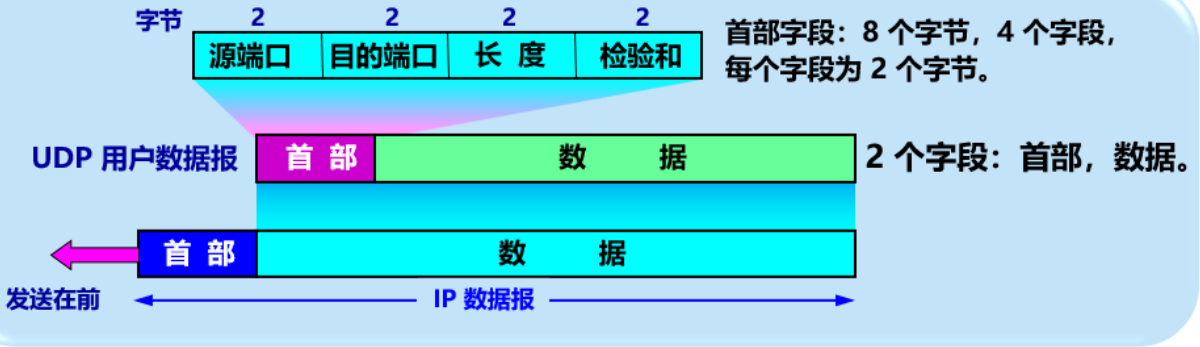
UDP 是面向报文的



- **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。

重点

5.2.2 UDP 的首部格式

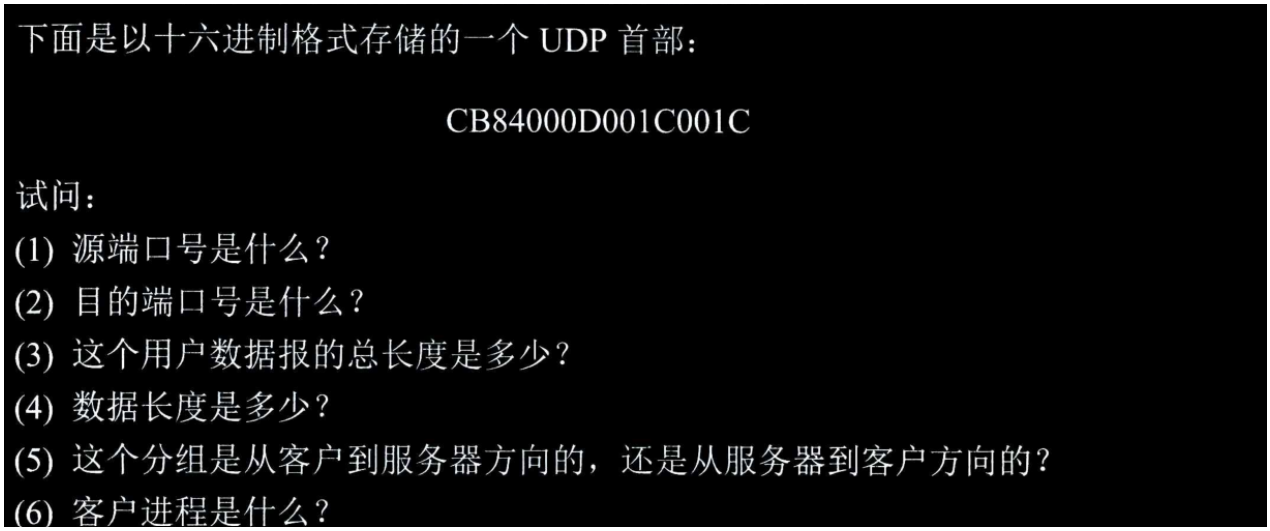


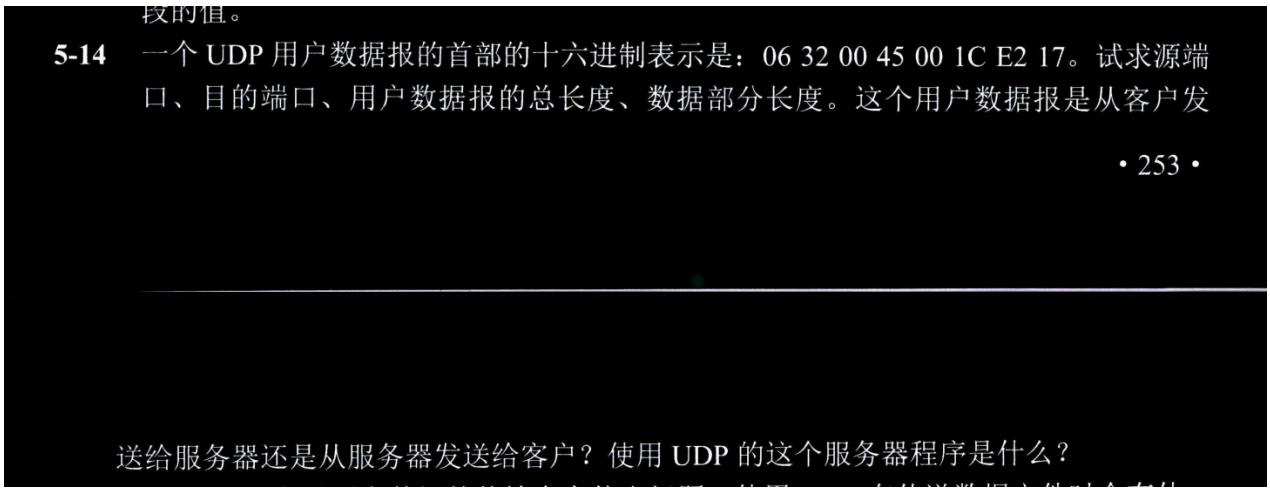
(1) **源端口**：源端口号。在需要对方回信时选用。不需要时可用全 0。

(2) **目的端口**：目的端口号。终点交付报文时必须使用。

(3) **长度**：UDP 用户数据报的长度，其最小值是 8（仅有首部）。

(4) **检验和**：检测 UDP 用户数据报在传输中是否有错。有错就丢弃。





TCP

每个字节都需要编号（虽然他是面向字节，但是还是需要编号）
快递栈作为缓存区

套接字（表示TCP的链接）
网络编程第一步：把套接字搞定

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：

$$\text{TCP 连接} ::= \{socket_1, socket_2\} = \{(IP_1: port_1), (IP_2: port_2)\} \quad (5\text{-}2)$$

IP提供不可靠服务，TCP提供可靠的服务
老师说听懂了吗？？？

成功接收的人会说听懂了
不成功接收：不吭声（老师不知道的）

弊端：接收方没有否定确认，只有肯定确认

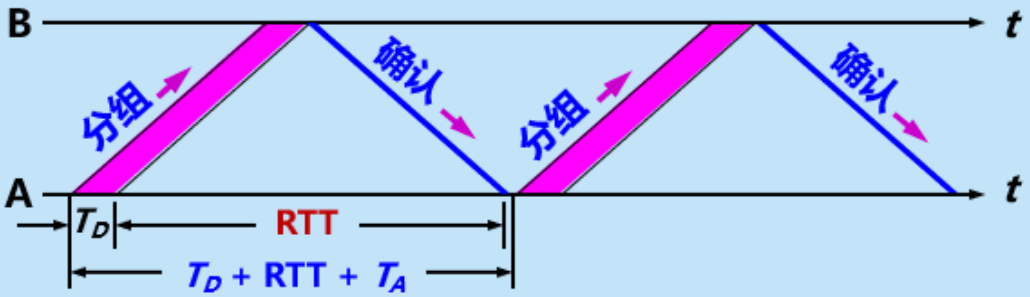
2. 出现差错

- **问题：A 如何知道 B 是否正确收到了 M_1 呢?**
- **解决方法：超时重传**
  1. **A 为每一个已发送的分组设置一个超时计时器。**
  2. **A 只要在超时计时器到期之前收到了相应的确认，就撤销该超时计时器，继续发送下一个分组 M_2 。**
  3. **若 A 在超时计时器规定时间内没有收到 B 的确认，就认为分组错误或丢失，就重发该分组。**

发送方发完数据就会启动计时器

为什么会考信道利用率啊

4. 信道利用率



信道利用率 $$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A} \qquad (5\text{-}3)$$

当往返时间 RTT 远大于分组发送时间 T_D 时，信道的利用率会非常低。

优点：简单。缺点：信道利用率太低。

发送的时间/发送+传输+接收时间=信道利用率

31.通信信道带宽为 1 Gb/s，端到端时延为 10 ms。TCP 的发送窗口为 65535 字节。试问:可能达到的最大吞吐量是多少?信道的利用率是多少?

解：吞吐量=发送数据/时间

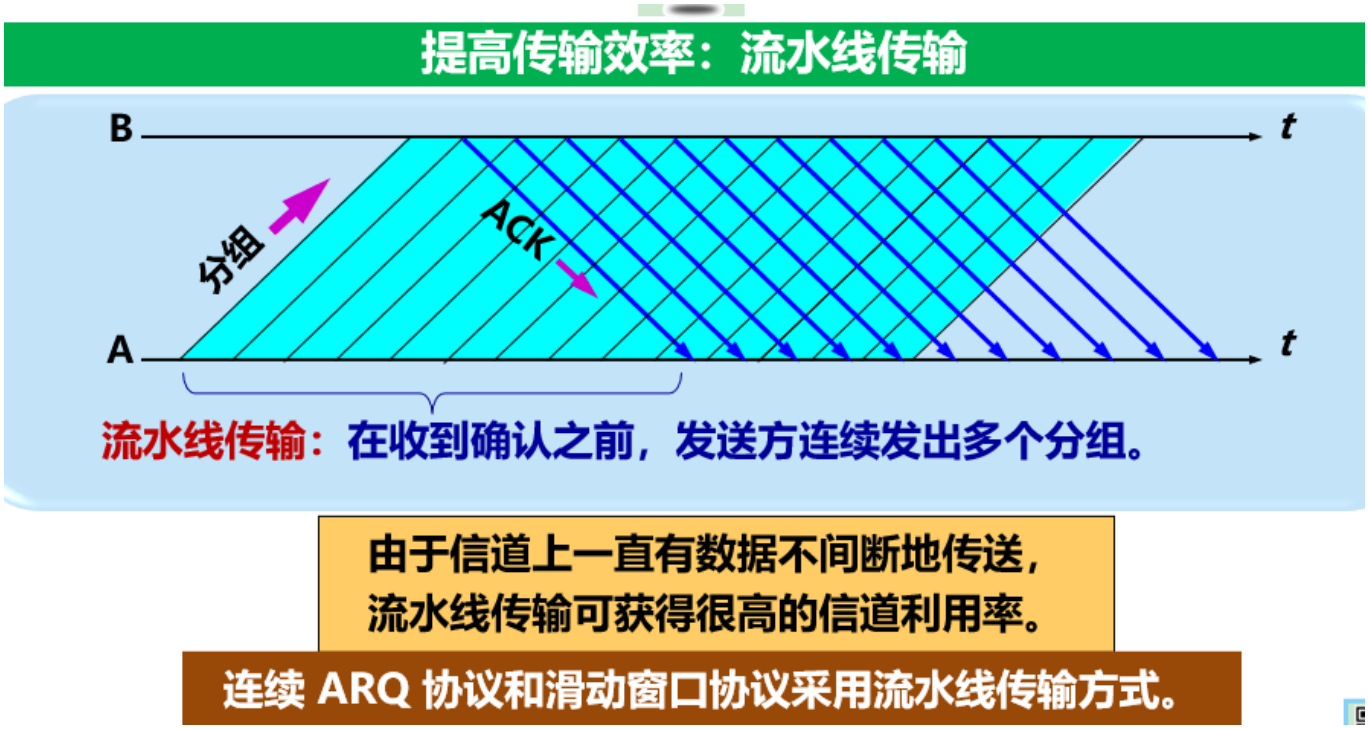
发送数据最大=65535×8=524280bit。

时间=发送时延+往返时延=524280bit/(1Gbit/s)+20ms=20.524ms。

最大吞吐量=524280bit/20.524ms=25.5Mbit/s。

信道利用率=吞吐量/带宽=(25.5Mbit/s)/(1Gbit/s)×100%=2.55%。

这个题目完美阐释



老师说出一连串话，然后到下课时候说，听懂了吗？？？

ARQ

发送缓存保留一个副本（这个缓存就是一个窗口）

发送窗口滑动，它在删除缓存（有怀疑这是一个数组底层结构）

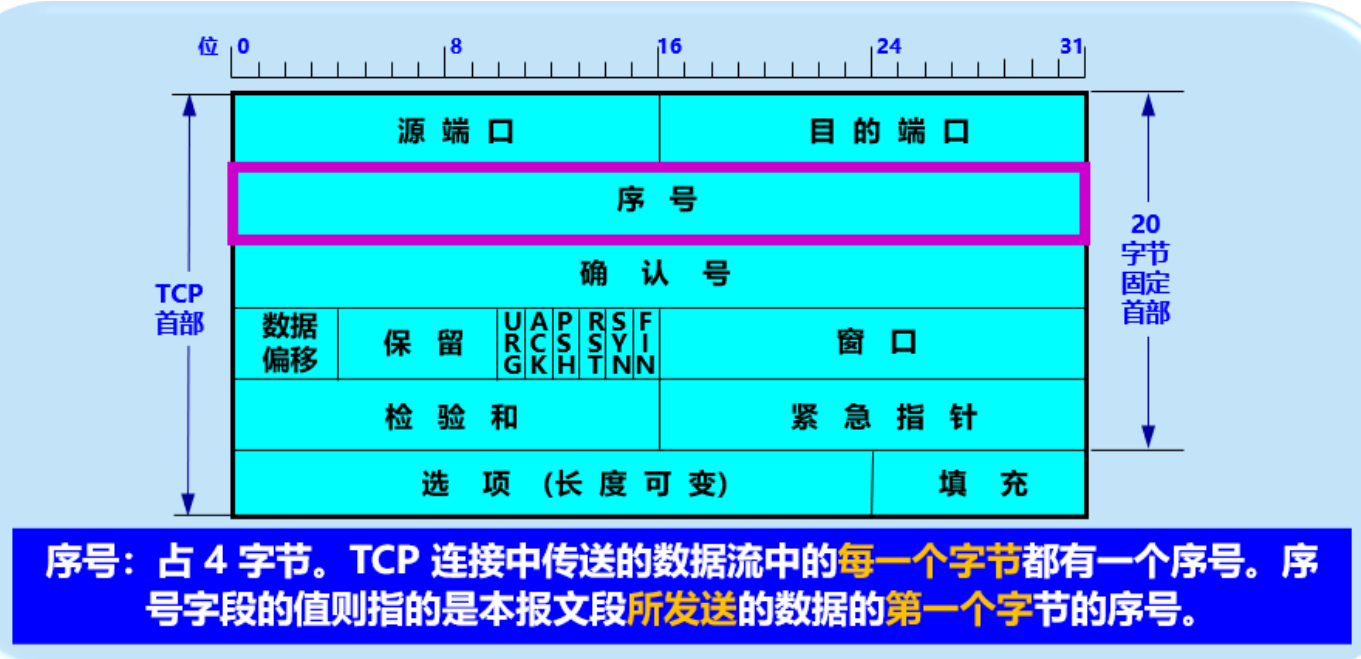
5.4.2 连续 ARQ 协议

- **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。**

对分组三确认时候，说明分组一分组二已经确认

这个滑动我好像见过（窗口的空间不能改变）

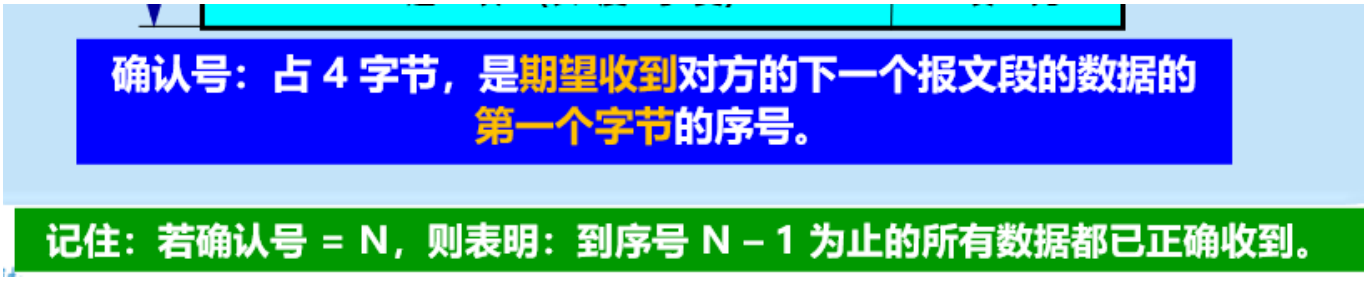
发送一系列字节流（里面包含很多报文段）



注意这个是全双工通信：发送方填写序号，不写确认号(ACK设置为0)，接收方成功接收到的情况下填写确认号（ACK设置成1），示意下一页（随后发送方把确认号填过来）

特殊情况下，老师叫某个同学过来，同学一般只用过来即可，结果有同学有话要说，连人带电脑（数据）一起过来了

如果还需要说一些话：还需要填序号



N-1发送完之后，把下一个序号设置为N（老师，PPT翻到第N页）

如果ACK的值是1，我才会解释确认号

题目

5-23 主机 A 向主机 B 连续发送了两个 TCP 报文段，其序号分别是 70 和 100。试问：

(1) 第一个报文段携带了多少字节的数据？

(2) 主机 B 收到第一个报文段后发回的确认中的确认号应当是多少？

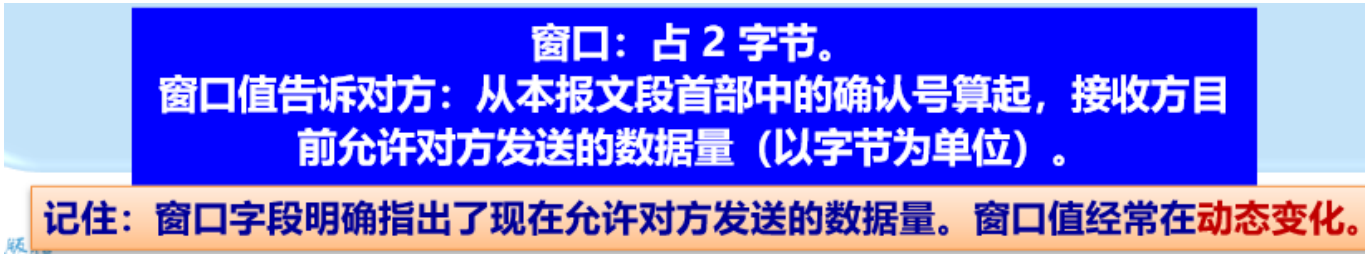
(3) 如果 B 收到第二个报文段后发回的确认中的确认号是 180，试问 A 发送的第二个报文段中的数据有多少字节？

(4) 如果 A 发送的第一个报文段丢失了，但第二个报文段到达了 B。B 在第二个报文段到达后向 A 发送确认。试问这个确认号应为多少？

确认号和缓存区删除有关，如果前面的东西丢了，优先发送前面的确认号
接收确认号时候---发送方把发送序号改成确认号，同时清空之前的发送缓存

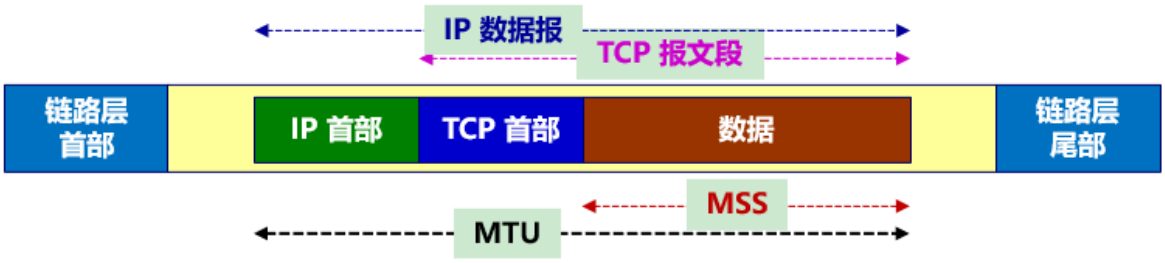
RST恢复出厂设置（然后从最开始还是需要同步）

接收方会根据现状跟发送方说，自己能够一次能够接收多少数据量



这是接收窗口，接收窗口每时每刻不一样，这是显示接收方缓存空间（当然最开始时一个默认值）

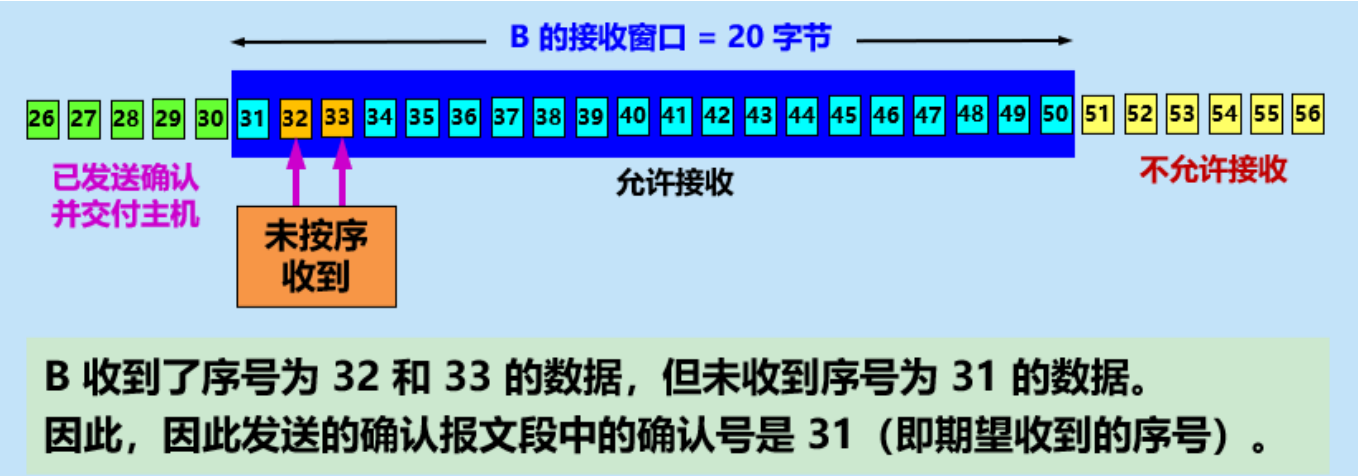
最大报文段长度 MSS（Maximum Segment Size）是每个 TCP 报文段中的数据字段的最大长度。
与接收窗口值没有关系。



注意：这是TCP报文段，里面的数据段

发送窗口：在没有收到确认的情况下，发送方可以连续把窗口内的数据全部发送出去。凡是已经发送过的数据，在未收到确认之前都必须暂时保留，以便在超时重传时使用。
接收窗口：只允许接收落入窗口内的数据。

注意：发送一个成功之后，立刻启动计时器

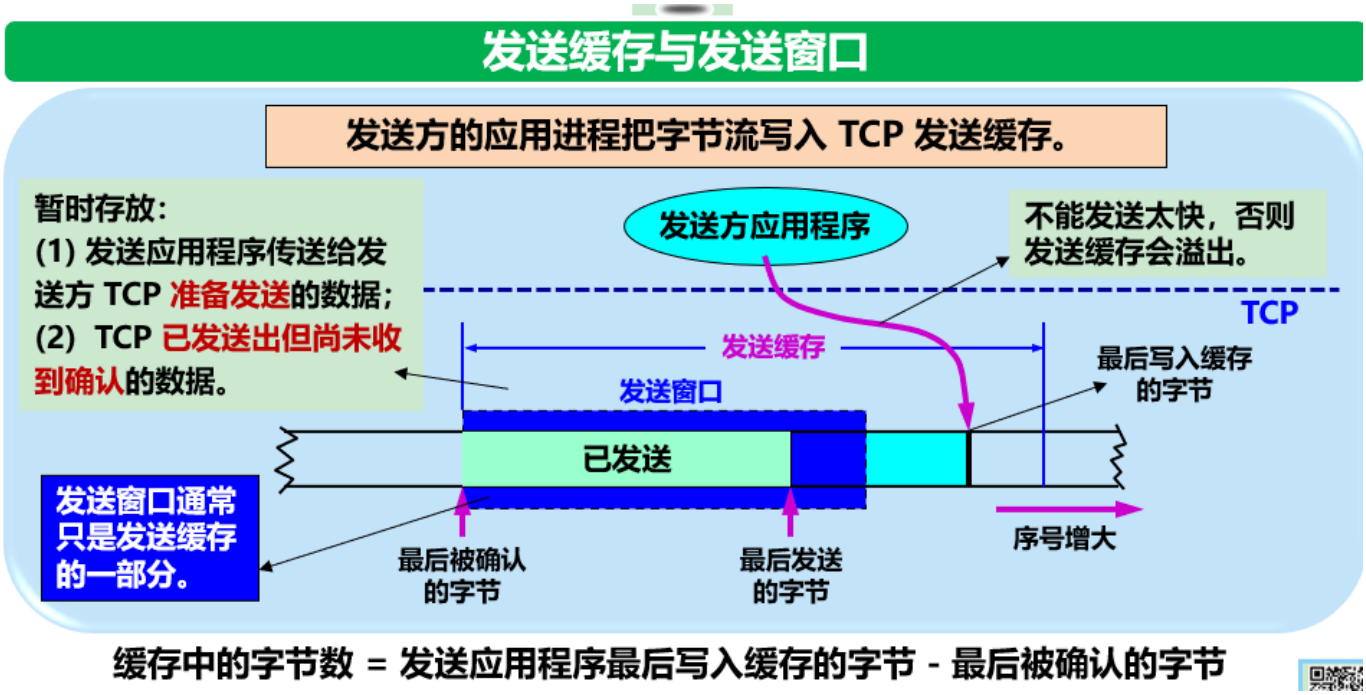


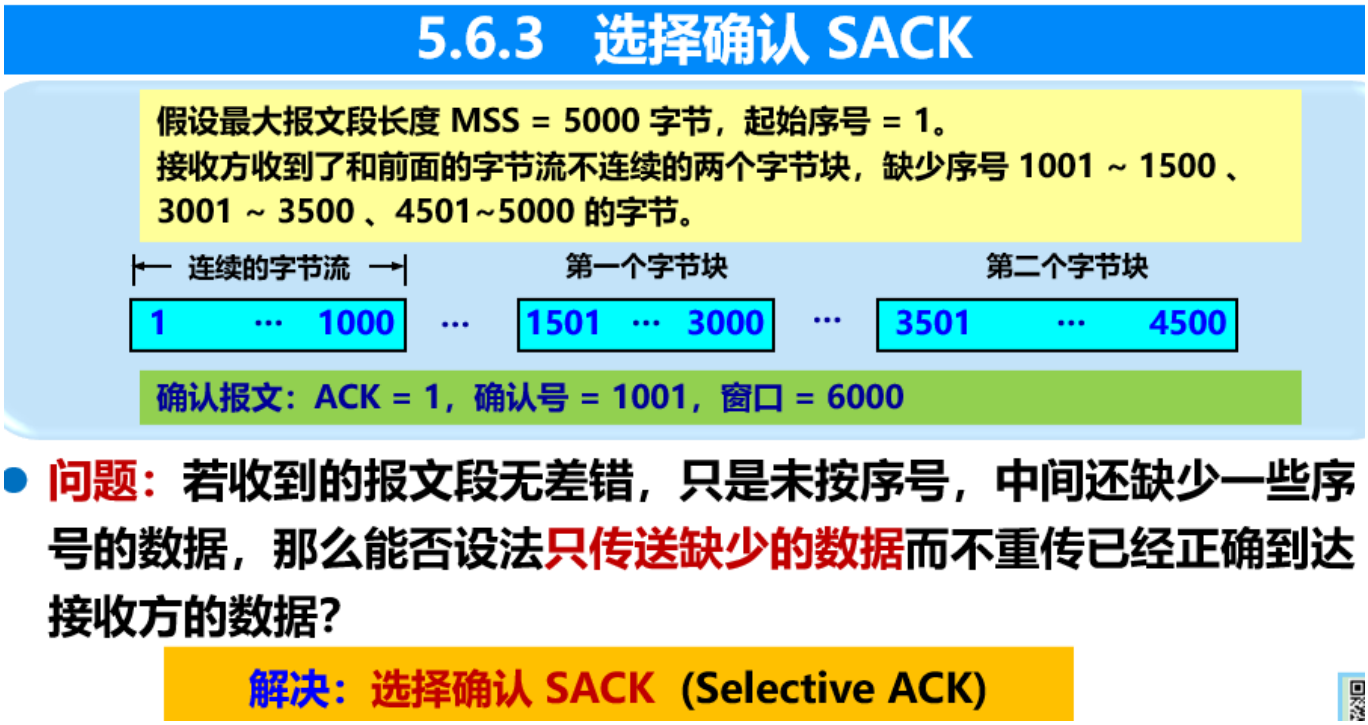
如果接收窗口缓存区只有一个，那么以后都需要重传

接收的空间变大：把数据交给上层(应用层)之后就可以腾出缓存区
接收的空间变大：接收方确实收到了，存入缓存区，并且发回含有确认号

的东西

发送缓存!=发送窗口





- **问题：若收到的报文段无差错，只是未按序号，中间还缺少一些序号的数据，那么能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?**

解决：选择确认 SACK (Selective ACK)

之前那个题目就是SACK的重发协议（预估后面跳着发的都保存了）

还有一个出错重发，那么之后从出错点开始，之后全部重发（预估后面跳着发的都丢弃了）

问题是：SACK到底用美元用上去，在之前的题目里面UDP

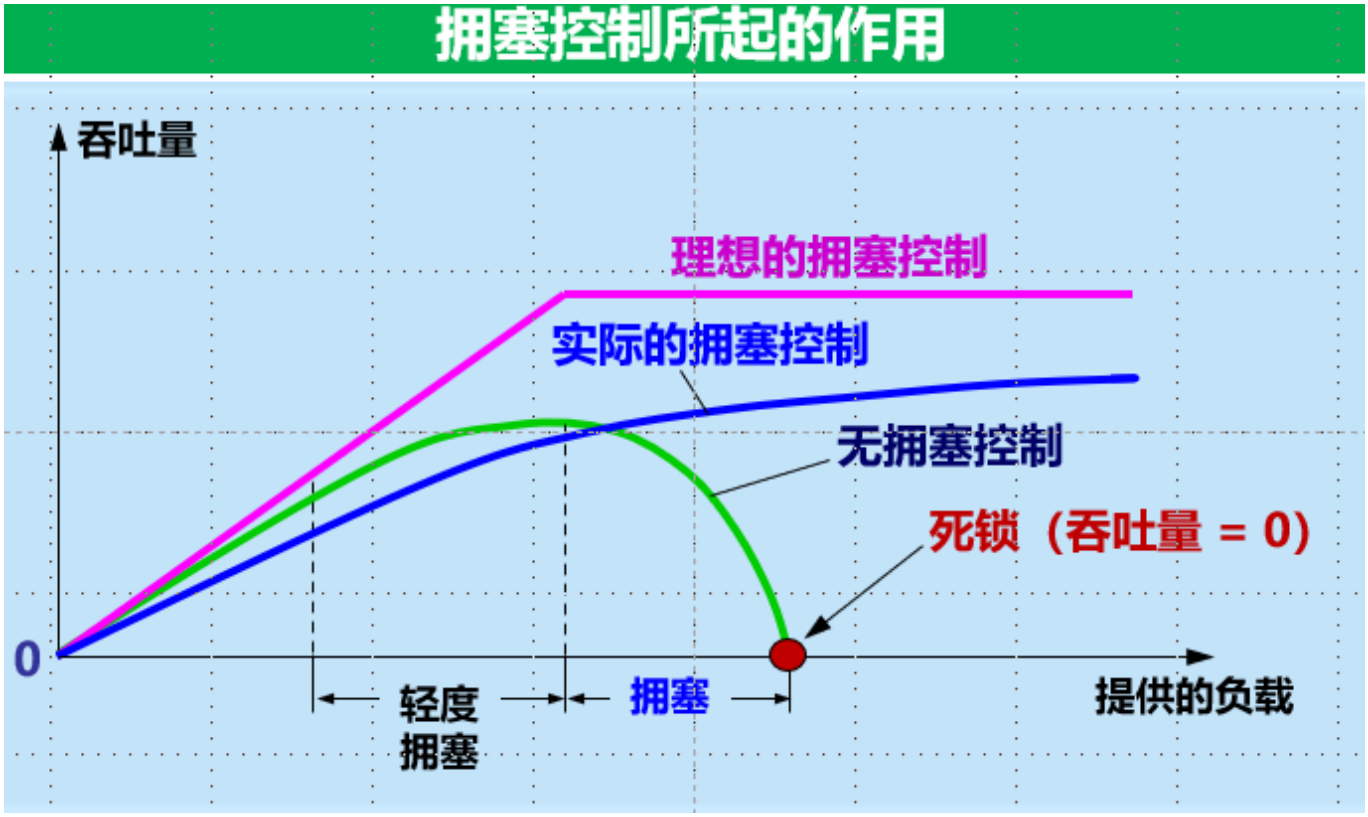
出错全部重发：可以理解为接收窗口大小是1？？？
选择重发

个人理解是在接收到回应之前，发送方还在继续发送
rwnd只能确定接收方还有多少空余，但是发送方的发送窗口并不知道有多大！！！

拥塞控制（考试占10分）

在讨论这个问题时候（设置接收窗口无穷大）

网络在人多时候，明显网络变慢了
针对拥塞窗口（网络运输承受能力）



这个图起着什么作用

分组的丢失是网络发生拥塞的征兆，而不是原因。

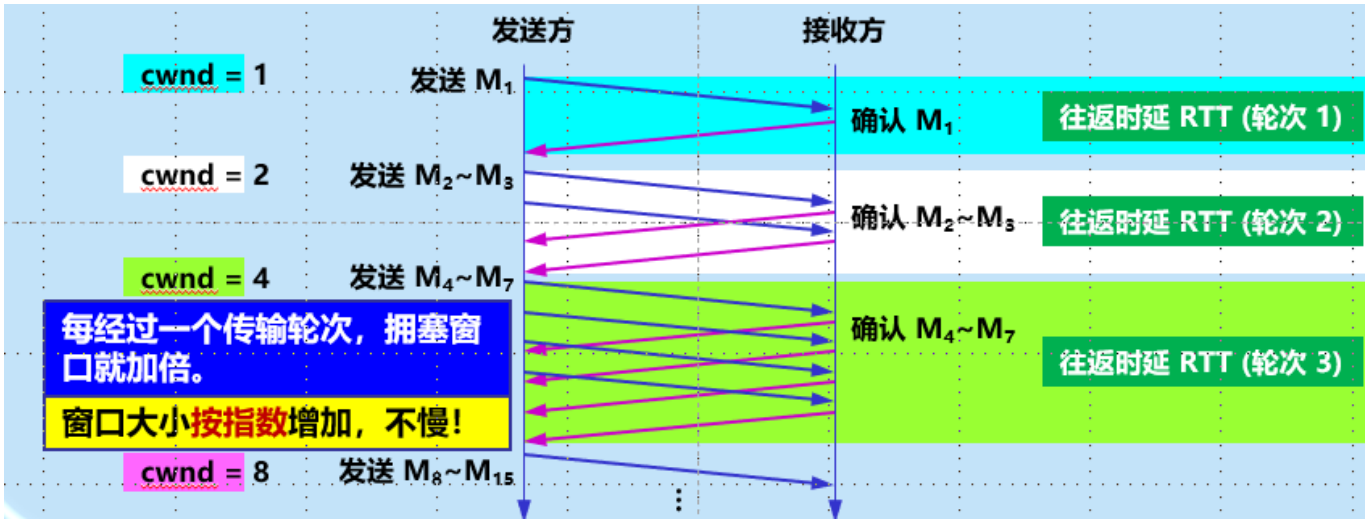
注意开环/闭环控制（反馈）

10分大题：拥塞控制的算法（闭环控制）
拥塞窗口cwd（发送窗口时拥塞窗口和接受窗口取最小值的）

捎带确认：等待合适时机，把自己需要发的东西连带确认信息一起发送出去（时间延长了）

慢开始 (slow-start)

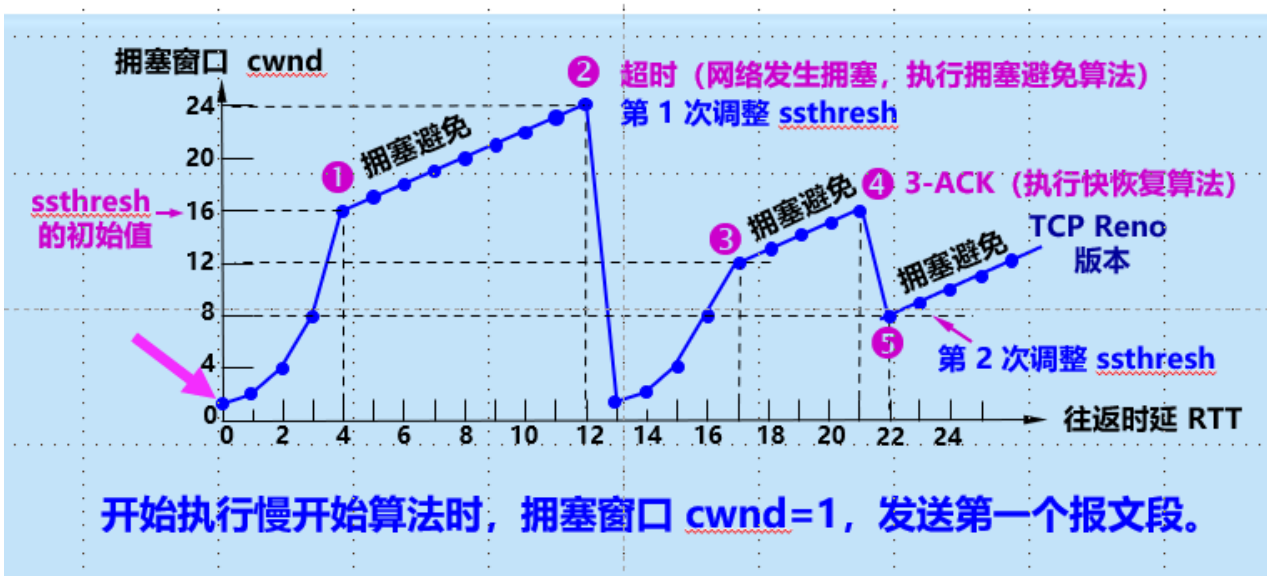
cwnd=1 1个mss(最大报文段)



慢开始门限

1. 当 cwnd < ssthresh 时，使用慢开始算法。
2. 当 cwnd > ssthresh 时，停止使用慢开始算法，改用拥塞避免算法。
3. 当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

- 拥塞避免 (congestion avoidance)
- 快重传 (fast retransmit)
- 快恢复 (fast recovery)



最开始是慢开始，慢开始是指数型的，达到慢开始门限时候，执行拥

> 塞避免
> 如果超时了，立刻归为1，重新慢开始，把ssthresh除以2
> 如果收到三个重复确认，为特殊情况，只用达到ssthresh即可，之后开始拥塞控制

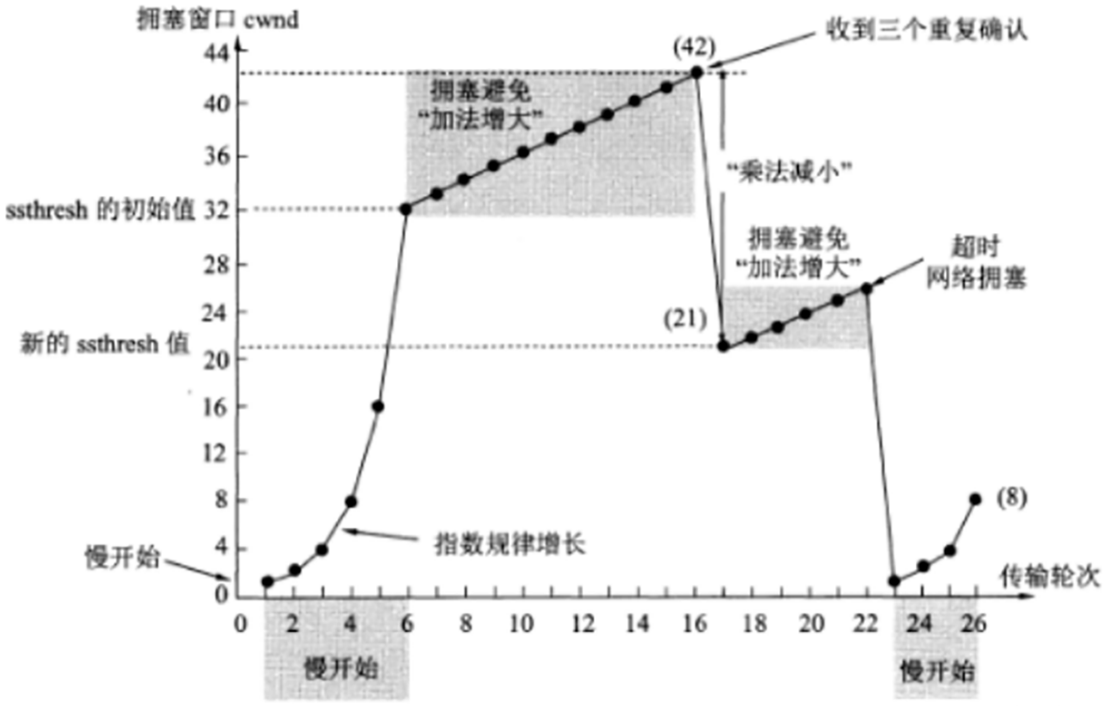
5-39 TCP 的拥塞窗口 cwnd 大小与 RTT 的关系如下所示：

cwnd	1	2	4	8	16	32	33	34	35	36	37	38	39
RTT	1	2	3	4	5	6	7	8	9	10	11	12	13
cwnd	40	41	42	21	22	23	24	25	26	1	2	4	8
RTT	14	15	16	17	18	19	20	21	22	23	24	25	26

(1) 试画出如图 5-25 所示的拥塞窗口与 RTT 的关系曲线。
(2) 指明 TCP 工作在慢开始阶段的时间间隔。
(3) 指明 TCP 工作在拥塞避免阶段的时间间隔。
(4) 在 RTT = 16 和 RTT = 22 之后发送方是通过收到三个重复的确认还是通过超时检测到丢失了报文段？
(5) 在 RTT = 1、RTT = 18 和 RTT = 24 时，门限 ssthresh 分别被设置为多大？
(6) 在 RTT 等于多少时发送出第 70 个报文段？
(7) 假定在 RTT = 26 之后收到了三个重复的确认，因而检测出了报文段的丢失，那么拥塞窗口 cwnd 和门限 ssthresh 应设置为多大？

考试的考法

快开始块恢复时一起的吗？？？



（2）慢开始时间间隔：[1, 6] 和 [23, 26]
（3）拥塞避免时间间隔：[6, 16] 和 [17, 22]
（4）在第 16 轮次之后发送方通过收到三个重复的确认，检测到丢失了报文段，因为题目给出，下一个轮次的拥塞窗口减半了。在第 22 轮次之后发送方通过超时，检测到丢失了报文段，因为题目给出，下一个轮次的拥塞窗口下降到 1了。
（5）在第 1 轮次发送时，门限 ssthresh 被设置为 32，因为从第 6 轮次起就进入了拥塞避免状态，拥塞窗口每个轮次加 1。
在第 18 轮次发送时，门限 ssthresh 被设置为发生拥塞时拥塞窗口 42 的一半，即 21。
在第 24 轮次发送时，门限 ssthresh 被设置为发生拥塞时拥塞窗口 26 的一半，即 13。
（6）第 1 轮次发送报文段 1。（cwnd = 1）
第 2 轮次发送报文段 2， 3。（cwnd = 2）
第 3 轮次发送报文段 4 ~ 7。（cwnd = 4）
第 4 轮次发送报文段 8 ~ 15。（cwnd = 8）
第 5 轮次发送报文段 16 ~ 31。（cwnd = 16）
第 6 轮次发送报文段 32 ~ 63。（cwnd = 32）
第 7 轮次发送报文段 64 ~ 96。（cwnd = 33）
因此第 70 报文段在第 7 轮次发送出。
（7）检测出了报文段的丢失时拥塞窗口 cwnd 是 8，因此拥塞窗口 cwnd 的数值应当减半，等于 4，而门限 ssthresh 应设置为检测出报文段丢失时的拥塞窗口 8 的一半，即 4。

5.8.3不要求掌握

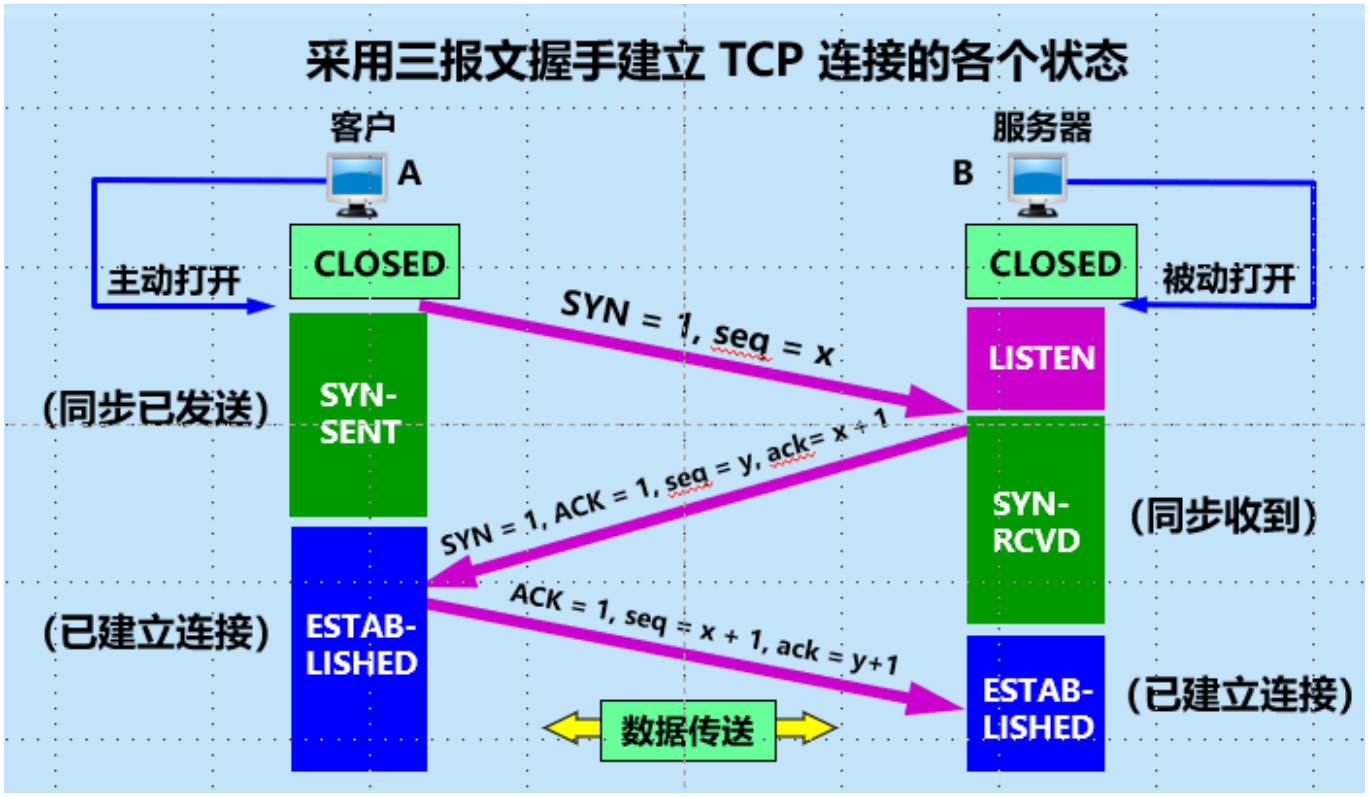
考点：画图：大小写不要写错了！！！

40.TCP 在进行流量控制时是以分组的丢失作为产生拥塞的标志。有没有不是因拥塞而引起的分组丢失的情况?如有，请举出三种情况。
答：不是因为拥塞而引起分组丢失的情况是有的，举例如下：
① 当 IP 数据报在传输过程中需要分片，但其中一个数据报片未能及时到达终点，而终点组装 IP 数据报已超时，因而只能丢弃该数据报。
② IP 数据报已经到达终点，但终点的缓存没有足够的空间存放此数据报。
③ 数据报在转发过程中经过一个局域网的网桥，但网桥在转发该数据报的帧时没有足够的储存空间而只好丢弃。

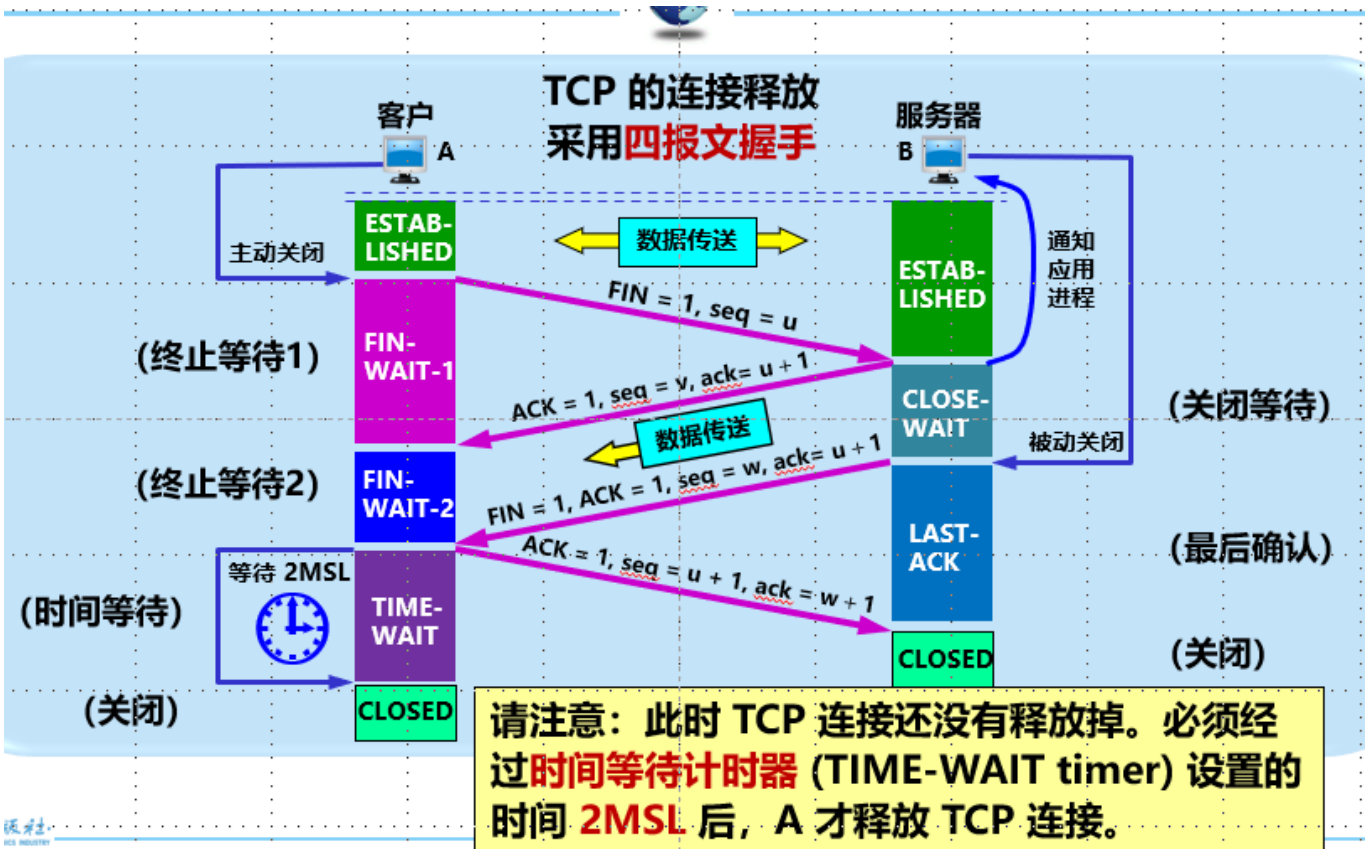
这个不要求掌握，只是个人兴趣

建立链接---三报文握手

两次握手原因：请求+响应之后，万一客户端不说话了。。。白等着（可能是过时的报文）



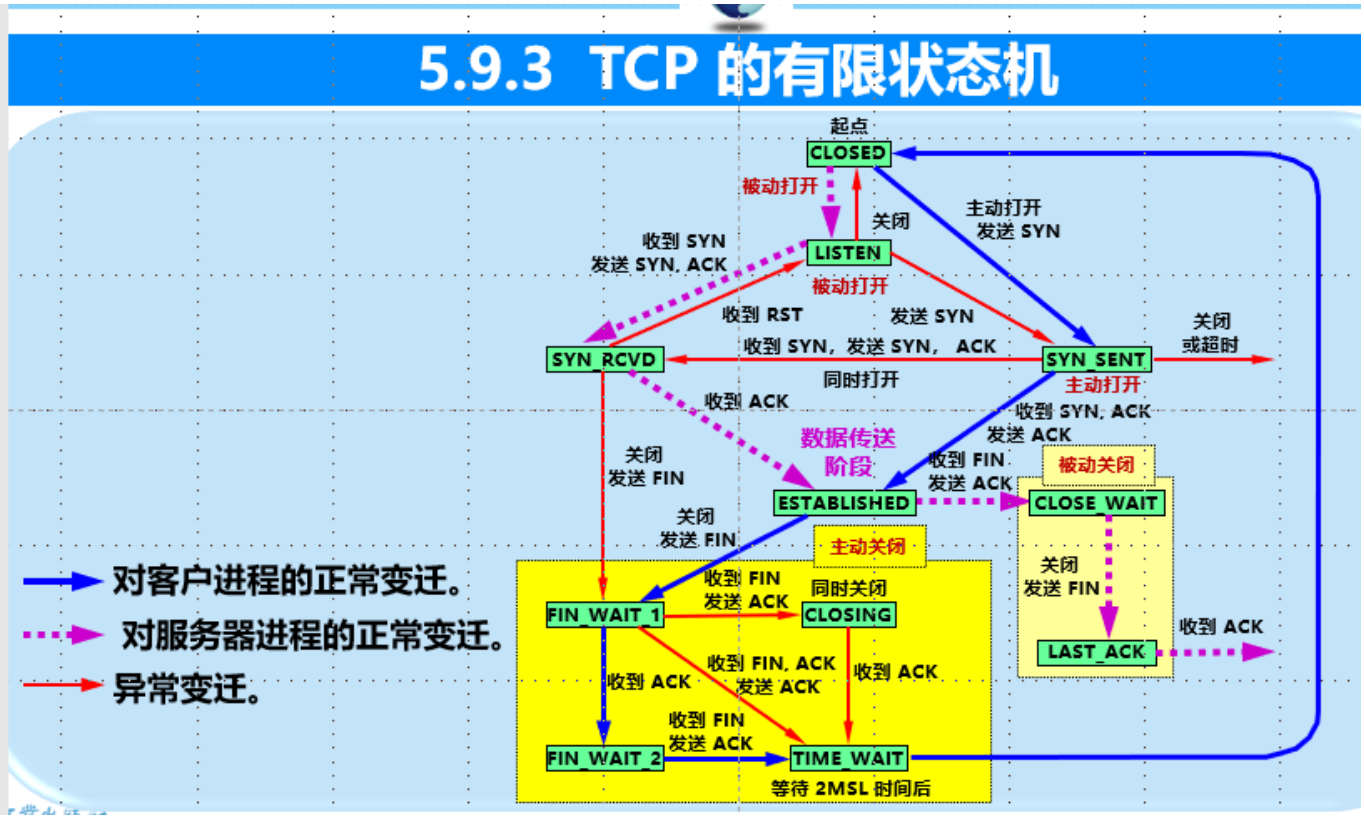
断开链接---四报文握手



第三个：我说完了

第四个:好的那么你 挂电话把

总结


5.9.3 TCP 的有限状态机
起点
CLOSED
被动打开
关闭
主动打开
发送 SYN
收到 SYN
发送 SYN, ACK
LISTEN
被动打开
收到 RST
发送 SYN
关闭
或超时
收到 SYN, 发送 SYN, ACK
SYN_RCVD
SYN_SENT
同时打开
主动打开
收到 ACK
收到 SYN, ACK
发送 ACK
数据传送
阶段
关闭
发送 FIN
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 ACK
收到 FIN, ACK
发送 ACK
收到 ACK
收到 FIN
发送 ACK
FIN_WAIT_2
TIME_WAIT
等待 2MSL 时间后
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。

呵呵呵。。。