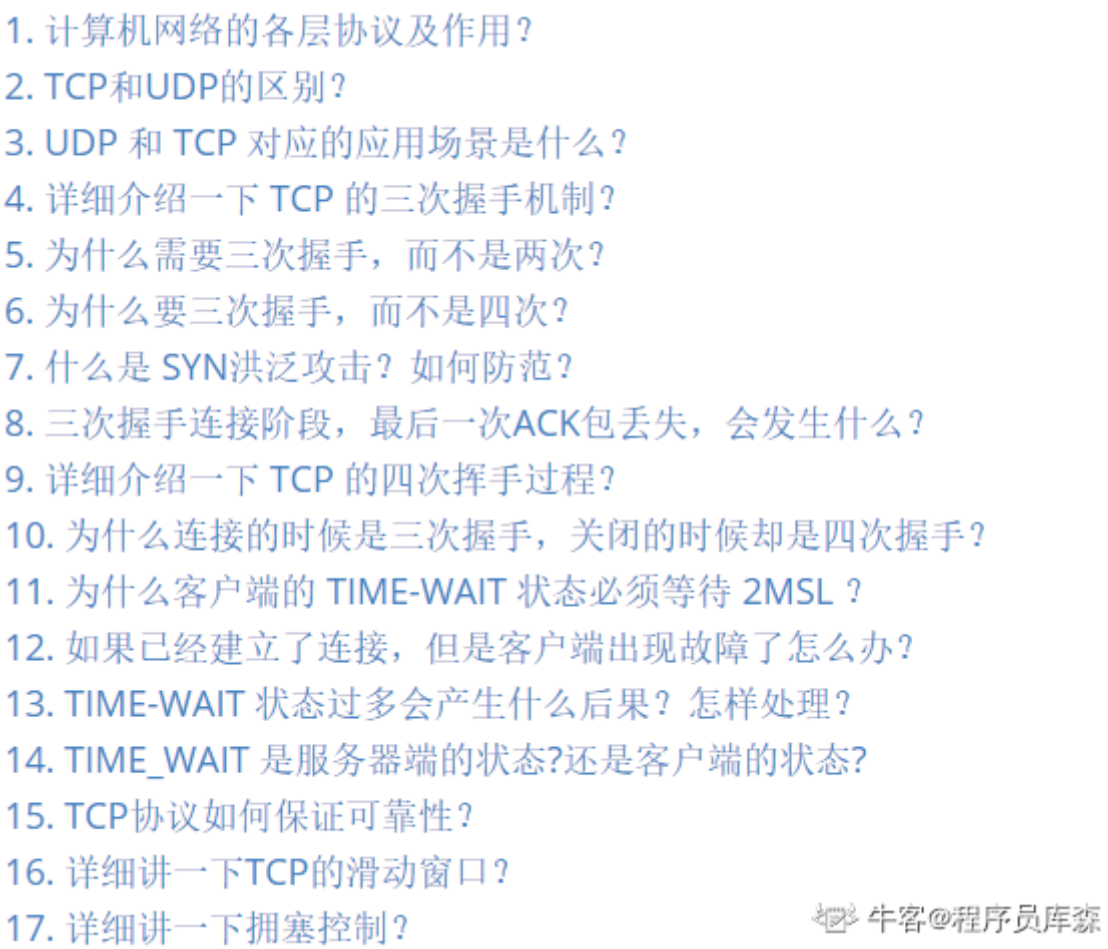
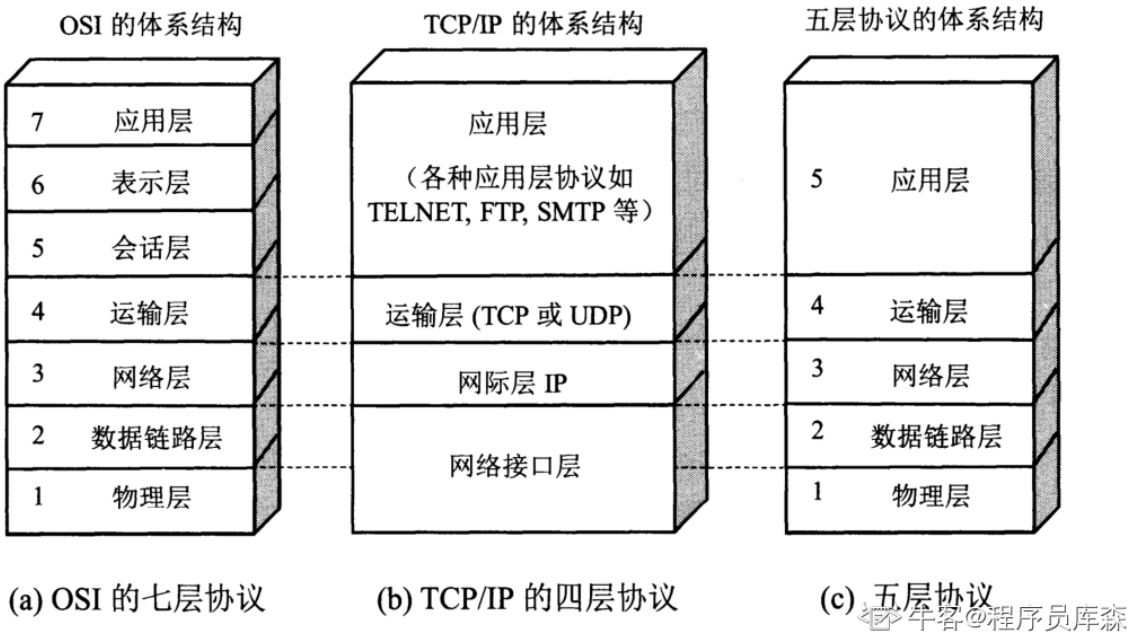
计算机网络常见问题



1. 计算机网络的各层协议及作用?

计算机网络体系可以大致分为一下三种，OSI七层模型、TCP/IP四层模型和五层模型。

- OSI七层模型：大而全，但是比较复杂、而且是先有了理论模型，没有实际应用。
- TCP/IP四层模型：是由实际应用发展总结出来的，从实质上讲，TCP/IP只有最上面三层，最下面一层没有什么具体内容，TCP/IP参考模型没有真正描述这一层的实现。
- 五层模型：五层模型只出现在计算机网络教学过程中，这是对七层模型和四层模型的一个折中，既简洁又能将概念阐述清楚。



七层网络体系结构各层的主要功能：

- 应用层：为应用程序提供交互服务。在互联网中的应用层协议很多，如域名系统DNS，支持万维网应用的HTTP协议，支持电子邮件的SMTP协议等。
- 表示层：主要负责数据格式的转换，如加密解密、转换翻译、压缩解压缩等。
- 会话层：负责在网络中的两节点之间建立、维持和终止通信，如服务器验证用户登录便是由会话层完成的。
- 运输层：有时也译为传输层，向主机进程提供通用的数据传输服务。该层主要有以下两种协议：
  - TCP：提供面向连接的、可靠的数据传输服务；
  - UDP：提供无连接的、尽最大努力的数据传输服务，但不保证数据传输的可靠性。
- 网络层：选择合适的路由和交换结点，确保数据及时传送。主要包括IP协议。
- 数据链路层：数据链路层通常简称为链路层。将网络层传下来的IP数据包组装成帧，并再相邻节点的链路上传送帧。
- 物理层：实现相邻节点间比特流的透明传输，尽可能屏蔽传输介质和通信手段的差异。

2. TCP和UDP的区别?

对比如下：

| | UDP | TCP |

是否连接 无连接 面向连接

是否可靠 不可靠传输，不使用流量控制和拥塞控制 可靠传输，使用流量控制和拥塞控制

是否有序 无序 有序，消息在传输过程中可能会乱序，TCP 会重新排序

传输速度 快 慢

连接对象个数 支持一对一，一对多，多对一和多对多交互通信 只能是一对一通信

传输方式 面向报文 面向字节流

首部开销 首部开销小，仅8字节 首部最小20字节，最大60字节

适用场景 适用于实时应用（IP电话、视频会议、直播等） 适用于要求可靠传输的应用，例如文件传输

总结：

TCP 用于在传输层有必要实现可靠传输的情况，UDP 用于对高速传输和实时性有较高要求的通信。TCP 和 UDP 应该根据应用目的按需使用。

3. UDP 和 TCP 对应的应用场景是什么?

TCP 是面向连接，能保证数据的可靠性交付，因此经常用于：

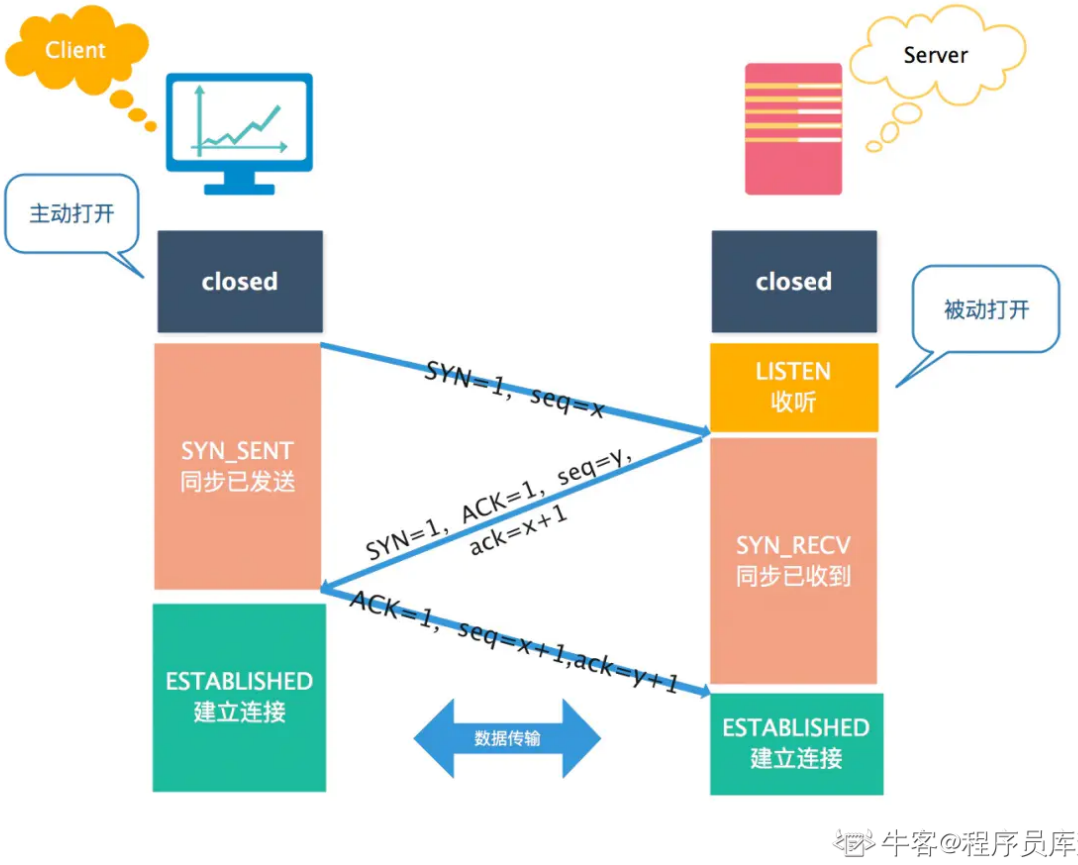
- FTP文件传输
- HTTP / HTTPS

UDP 面向无连接，它可以随时发送数据，再加上UDP本身的处理既简单又高效，因此经常用于：

- 包总量较少的通信，如 DNS 、SNMP等
- 视频、音频等多媒体通信
- 广播通信

应用层协议	应用	传输层协议
SMTP	电子邮件	TCP
TELNET	远程终端接入	
HTTP	万维网	
FTP	文件传输	
DNS	域名转换	UDP
TFTP	文件传输	
SNMP	网络管理	
NFS	远程文件服务器	

4. 详细介绍一下 TCP 的三次握手机制?



> 图片来自：https://juejin.cn/post/6844904005315854343

三次握手机制：

- 第一次握手：客户端请求建立连接，向服务端发送一个**同步报文**（SYN=1），同时选择一个随机数 seq = x 作为**初始序列号**，并进入SYN_SENT状态，等待服务器确认。
- 第二次握手：：服务端收到连接请求报文后，如果同意建立连接，则向客户端发送**同步确认报文**（SYN=1，ACK=1），确认号为 ack = x + 1，同时选择一个随机数 seq = y 作为初始序列号，此时服务器进入SYN_RECV状态。
- 第三次握手：客户端收到服务端的确认后，向服务端发送一个**确认报文**（ACK=1），确认号为 ack = y + 1，序列号为 seq = x + 1，客户端和服务器进入ESTABLISHED状态，完成三次握手。

理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。

5. 为什么需要三次握手，而不是两次?

主要有三个原因：

1. 防止已过期的连接请求报文突然又传送到服务器，因而产生错误和资源浪费。

   在双方两次握手即可建立连接的情况下，假设客户端发送 A 报文段请求建立连接，由于网络原因造成 A 暂时无法到达服务器，服务器接收不到请求报文段就不会返回确认报文段。

   客户端在长时间得不到应答的情况下重新发送请求报文段 B，这次 B 顺利到达服务器，服务器随即返回确认报文并进入 ESTABLISHED 状态，客户端在收到 确认报文后也进入 ESTABLISHED 状态，双方建立连接并传输数据，之后正常断开连接。

   此时姗姗来迟的 A 报文段才到达服务器，服务器随即返回确认报文并进入 ESTABLISHED 状态，但是已经进入 CLOSED 状态的客户端无法再接受确认报文段，更无法进入 ESTABLISHED 状态，这将导致服务器长时间单方面等待，造成资源浪费。

2. 三次握手才能让双方均确认自己和对方的发送和接收能力都正常。

第一次握手：客户端只是发送处请求报文段，什么都无法确认，而服务器可以确认自己的接收能力和对方的发送能力正常；

第二次握手：客户端可以确认自己发送能力和接收能力正常，对方发送能力和接收能力正常；

第三次握手：服务器可以确认自己发送能力和接收能力正常，对方发送能力和接收能力正常；

可见三次握手才能让双方都确认自己和对方的发送和接收能力全部正常，这样就可以愉快地进行通信了。

3. 告知对方自己的初始序号值，并确认收到对方的初始序号值。

   TCP 实现了可靠的数据传输，原因之一就是 TCP 报文段中维护了序号字段和确认序号字段，通过这两个字段双方都可以知道在自己发出的数据中，哪些是已经被对方确认接收的。这两个字段的值会在初始序号值得基础递增，如果是两次握手，只有发起方的初始序号可以得到确认，而另一方的初始序号则得不到确认。

6. 为什么要三次握手，而不是四次?

因为三次握手已经可以确认双方的发送接收能力正常，双方都知道彼此已经准备好，而且也可以完成对双方初始序号值得确认，也就无需再第四次握手了。

- 第一次握手：服务端确认“自己收、客户端发”报文功能正常。
- 第二次握手：客户端确认“自己发、自己收、服务端收、客户端发”报文功能正常，客户端认为连接已建立。
- 第三次握手：服务端确认“自己发、客户端收”报文功能正常，此时双方均建立连接，可以正常通信。

7. 什么是 SYN洪泛攻击？如何防范?

SYN洪泛攻击属于 DOS 攻击的一种，它利用 TCP 协议缺陷，通过发送大量的半连接请求，耗费 CPU 和内存资源。

原理：

- 在三次握手过程中，服务器发送 [SYN/ACK] 包（第二个包）之后、收到客户端的 [ACK] 包（第三个包）之前的 TCP 连接称为半连接（half-open connect），此时服务器处于 SYN_RECV（等待客户端响应）状态。如果接收到客户端的 [ACK]，则 TCP 连接成功，如果未接受到，则会**不断重发请求**直至成功。
- SYN 攻击的攻击者在短时间内**伪造大量不存在的 IP 地址**，向服务器不断地发送 [SYN] 包，服务器回复 [SYN/ACK] 包，并等待客户的确认。由于源地址是不存在的，服务器需要不断的重发直至超时。
- 这些伪造的 [SYN] 包将长时间占用未连接队列，影响了正常的 SYN，导致目标系统运行缓慢、网络堵塞甚至系统瘫痪。

检测：当在服务器上看到大量的半连接状态时，特别是源 IP 地址是随机的，基本上可以断定这是一次 SYN 攻击。

防范：

- 通过***、路由器等过滤网关防护。
- 通过加固 TCP/IP 协议栈防范，如增加最大半连接数，缩短超时时间。
- SYN cookies技术。SYN Cookies 是对 TCP 服务器端的三次握手做一些修改，专门用来防范 SYN 洪泛攻击的一种手段。

8. 三次握手连接阶段，最后一次ACK包丢失，会发生什么?

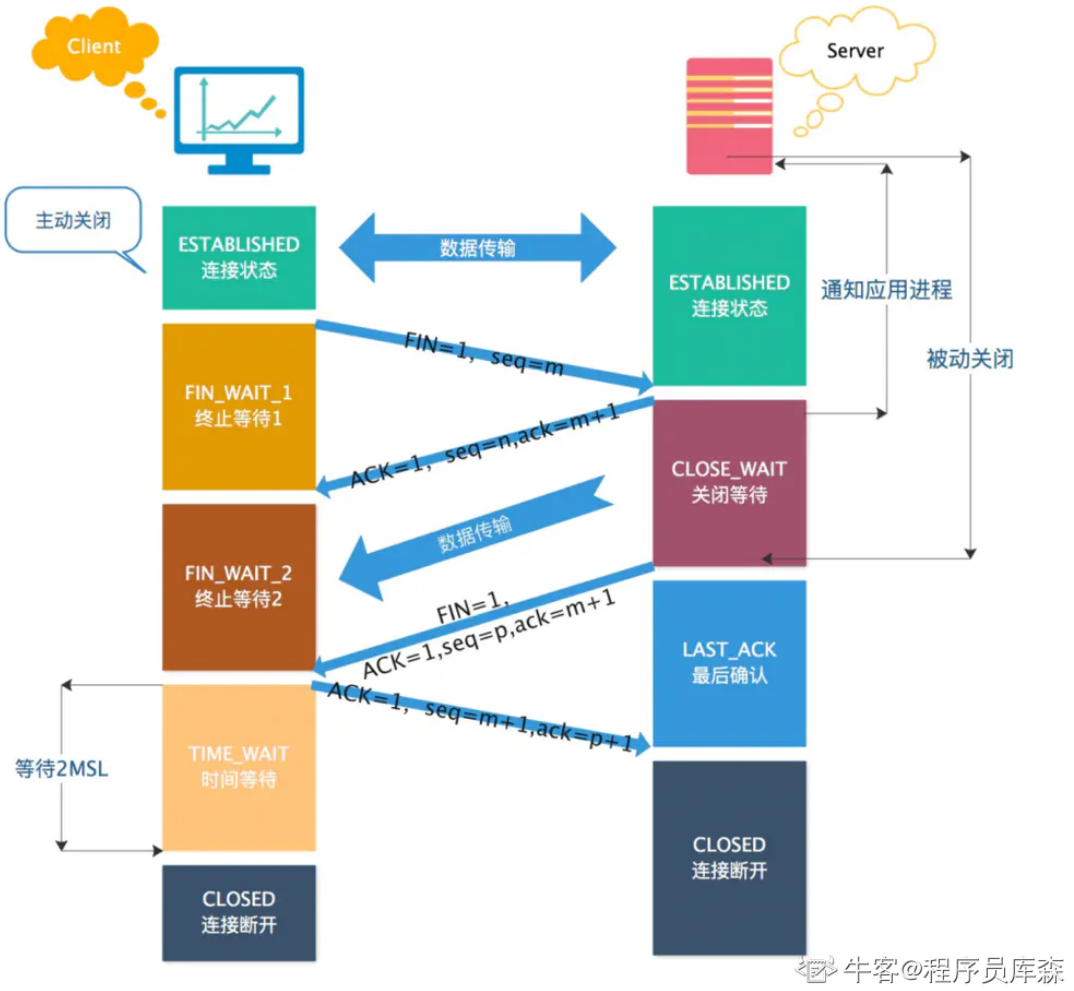
服务端：

- 第三次的ACK在网络中丢失，那么服务端该TCP连接的状态为SYN_RECV,并且会根据 TCP的超时重传机制，会等待3秒、6秒、12秒后重新发送SYN+ACK包，以便客户端重新发送ACK包。
- 如果重发指定次数之后，仍然未收到 客户端的ACK应答，那么一段时间后，服务端自动关闭这个连接。

客户端：

客户端认为这个连接已经建立，如果客户端向服务端发送数据，服务端将以RST包（Reset，标示复位，用于异常的关闭连接）响应。此时，客户端知道第三次握手失败。

9. 详细介绍一下 TCP 的四次挥手过程?



> 图片来源：https://juejin.im/post/5ddd1f30e51d4532c42c5abe

- 第一次挥手：客户端向服务端发送连接释放报文（FIN=1，ACK=1），主动关闭连接，同时等待服务端的确认。
  - 序列号 seq = u，即客户端上次发送的报文的最后一个字节的序号 + 1
  - 确认号 ack = k, 即服务端上次发送的报文的最后一个字节的序号 + 1
- 第二次挥手：服务端收到连接释放报文后，立即发出**确认报文**（ACK=1），序列号 seq = k，确认号 ack = u + 1。

  这时 TCP 连接处于半关闭状态，即客户端到服务端的连接已经释放了，但是服务端到客户端的连接还未释放。这表示客户端已经没有数据发送了，但是服务端可能还要给客户端发送数据。
- 第三次挥手：服务端向客户端发送连接释放报文（FIN=1，ACK=1），主动关闭连接，同时等待 A 的确认。
  - 序列号 seq = w，即服务端上次发送的报文的最后一个字节的序号 + 1。
  - 确认号 ack = u + 1，与第二次挥手相同，因为这段时间客户端没有发送数据

- 第四次挥手：客户端收到服务端的连接释放报文后，立即发出**确认报文**（ACK=1），序列号 seq = u + 1，确认号为 ack = w + 1。

  此时，客户端就进入了 TIME-WAIT 状态。注意此时客户端到 TCP 连接还没有释放，必须经过 2*MSL（最长报文段寿命）的时间后，才进入 CLOSED 状态。而服务端只要收到客户端发出的确认，就立即进入 CLOSED 状态。可以看到，服务端结束 TCP 连接的时间要比客户端早一些。

10. 为什么连接的时候是三次握手，关闭的时候却是四次握手?

服务器在收到客户端的 FIN 报文段后，可能还有一些数据要传输，所以不能马上关闭连接，但是会做出应答，返回 ACK 报文段.

接下来可能会继续发送数据，在数据发送完后，服务器会向客户单发送 FIN 报文，表示数据已经发送完毕，请求关闭连接。服务器的**ACK和FIN一般都会分开发送**，从而导致多了一次，因此一共需要四次挥手。

11. 为什么客户端的 TIME-WAIT 状态必须等待 2MSL ?

主要有两个原因：

1. 确保 ACK 报文能够到达服务端，从而使服务端正常关闭连接。

   第四次挥手时，客户端第四次挥手的 ACK 报文不一定会到达服务端。服务端会超时重传 FIN/ACK 报文，此时如果客户端已经断开了连接，那么就无法响应服务端的二次请求，这样服务端迟迟收不到 FIN/ACK 报文的确认，就无法正常断开连接。

   MSL 是报文段在网络上存活的最长时间。客户端等待 2MSL 时间，即「客户端 ACK 报文 1MSL 超时 + 服务端 FIN 报文 1MSL 传输」，就能够收到服务端重传的 FIN/ACK 报文，然后客户端重传一次 ACK 报文，并重新启动 2MSL 计时器。如此保证服务端能够正常关闭。

   如果服务端重发的 FIN 没有成功地在 2MSL 时间里传给客户端，服务端则会继续超时重试直到断开连接。

2. 防止已失效的连接请求报文段出现在之后的连接中。

   TCP 要求在 2MSL 内不使用相同的序列号。客户端在发送完最后一个 ACK 报文段后，再经过时间 2MSL，就可以保证本连接持续的时间内产生的所有报文段都从网络中消失。这样就可以使下一个连接中不会出现这种旧的连接请求报文段。或者即使收到这些过时的报文，也可以不处理它。

12. 如果已经建立了连接，但是客户端出现故障了怎么办?

或者说，如果三次握手阶段、四次挥手阶段的包丢失了怎么办？如“服务端重发 FIN丢失”的问题。

简而言之，通过**定时器 + 超时重试机制**，尝试获取确认，直到最后会自动断开连接。

具体而言，TCP 设有一个保活计时器。服务器每收到一次客户端的数据，都会重新复位这个计时器，时间通常是设置为 2 小时。若 2 小时还没有收到客户端的任何数据，服务器就开始重试：每隔 75 分钟发送一个探测报文段，若一连发送 10 个探测报文后客户端依然没有回应，那么服务器就认为连接已经断开了。

13. TIME-WAIT 状态过多会产生什么后果？怎样处理?

从服务器来讲，短时间内关闭了大量的Client连接，就会造成服务器上出现大量的TIME_WAIT连接，严重消耗着服务器的资源，此时部分客户端就会显示连接不上。

从客户端来讲，客户端TIME_WAIT过多，就会导致端口资源被占用，因为端口就65536个，被占满就会导致无法创建新的连接。

解决办法：

- 服务器可以设置 SO_REUSEADDR 套接字选项来避免 TIME_WAIT状态，此套接字选项告诉内核，即使此端口正忙（处于 TIME_WAIT状态），也请继续并重用它。
- 调整系统内核参数，修改/etc/sysctl.conf文件，即修改net.ipv4.tcp_tw_reuse 和 tcp_timestamps

  `net.ipv4.tcp_tw_reuse = ``1` 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为``0``，表示关闭； `net.ipv4.tcp_tw_recycle = ``1` 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为``0``，表示关闭。
- 强制关闭，发送 RST 包越过TIME_WAIT状态，直接进入CLOSED状态。

14. TIME_WAIT 是服务器端的状态?还是客户端的状态?

TIME_WAIT 是主动断开连接的一方会进入的状态，一般情况下，都是客户端所处的状态;服务器端一般设置不主动关闭连接。

TIME_WAIT 需要等待 2MSL，在大量短连接的情况下，TIME_WAIT会太多，这也会消耗很多系统资源。对于服务器来说，在 HTTP 协议里指定 KeepAlive（浏览器重用一个 TCP 连接来处理多个 HTTP 请求），由浏览器来主动断开连接，可以一定程度上减少服务器的这个问题。

15. TCP协议如何保证可靠性?

TCP主要提供了检验和、序列号/确认应答、超时重传、滑动窗口、拥塞控制和 流量控制等方法实现了可靠性传输。

- 检验和：通过检验和的方式，接收端可以检测出来数据是否有差错和异常，假如有差错就会直接丢弃TCP段，重新发送。
- 序列号/确认应答：

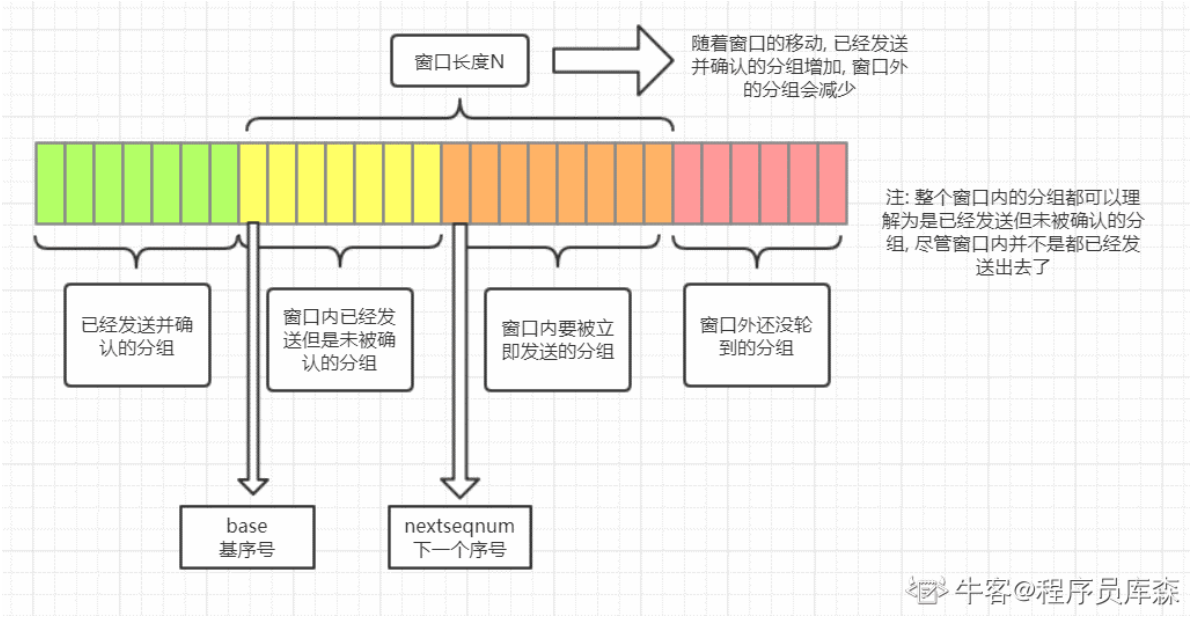
  序列号的作用不仅仅是应答的作用，有了序列号能够将接收到的数据根据序列号排序，并且去掉重复序列号的数据。

  TCP传输的过程中，每次接收方收到数据后，都会对传输方进行确认应答。也就是发送ACK报文，这个ACK报文当中带有对应的确认序列号，告诉发送方，接收到了哪些数据，下一次的数据从哪里发。
- 滑动窗口：滑动窗口既提高了报文传输的效率，也避免了发送方发送过多的数据而导致接收方无法正常处理的异常。
- 超时重传：超时重传是指发送出去的数据包到接收到确认包之间的时间，如果超过了这个时间会被认为是丢包了，需要重传。最大超时时间是动态计算的。
- 拥塞控制：在数据传输过程中，可能由于网络状态的问题，造成网络拥堵，此时引入拥塞控制机制，在保证TCP可靠性的同时，提高性能。
- 流量控制：如果主机A 一直向主机B发送数据，不考虑主机B的接受能力，则可能导致主机B的接受缓冲区满了而无法再接受数据，从而会导致大量的数据丢包，引发重传机制。而在重传的过程中，若主机B的接收缓冲区情况仍未好转，则会将大量的时间浪费在重传数据上，降低传送数据的效率。所以引入流量控制机制，主机B通过告诉主机A自己接收缓冲区的大小，来使主机A控制发送的数据量。流量控制与TCP协议报头中的窗口大小有关。

16. 详细讲一下TCP的滑动窗口?

在进行数据传输时，如果传输的数据比较大，就需要拆分为多个数据包进行发送。TCP 协议需要对数据进行确认后，才可以发送下一个数据包。这样一来，就会在等待确认应答包环节浪费时间。

为了避免这种情况，TCP引入了窗口概念。窗口大小指的是不需要等待确认应答包而可以继续发送数据包的最大值。



从上面的图可以看到滑动窗口左边的是已发送并且被确认的分组，滑动窗口右边是还没有轮到的分组。

滑动窗口里面也分为两块，一块是已经发送但是未被确认的分组，另一块是窗口内等待发送的分组。随着已发送的分组不断被确认，窗口内等待发送的分组也会不断被发送。整个窗口就会往右移动，让还没轮到的分组进入窗口内。

可以看到滑动窗口起到了一个限流的作用，也就是说当前滑动窗口的大小决定了当前 TCP 发送包的速率，而滑动窗口的大小取决于拥塞控制窗口和流量控制窗口的两者间的最小值。

17. 详细讲一下拥塞控制?

TCP 一共使用了四种算法来实现拥塞控制：

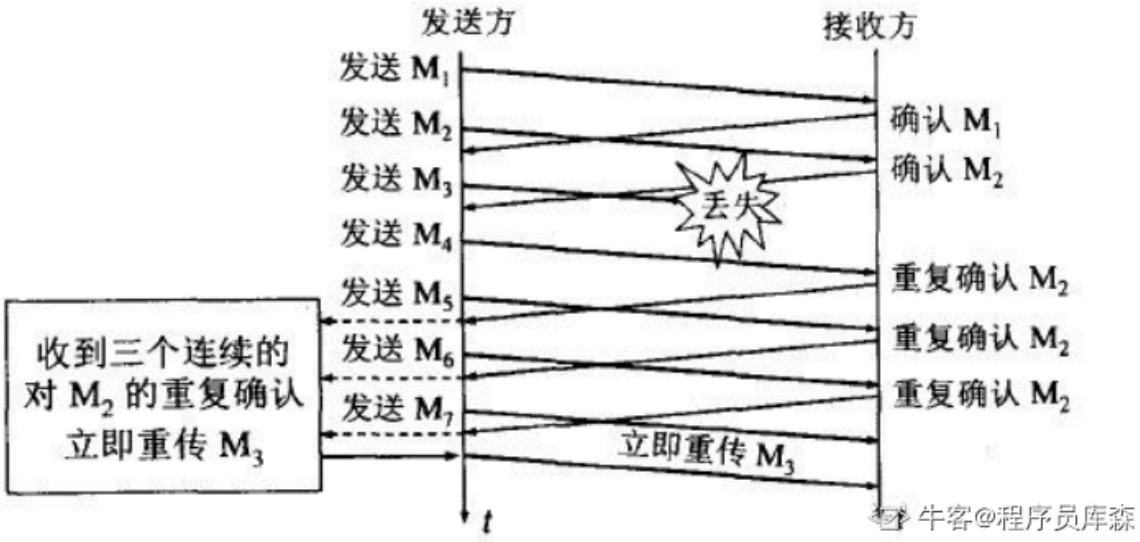
- 慢开始 (slow-start)；
- 拥塞避免 (congestion avoidance)；
- 快速重传 (fast retransmit)；
- 快速恢复 (fast recovery)。

发送方维持一个叫做拥塞窗口cwnd（congestion window）的状态变量。当cwndssthresh时，改用拥塞避免算法。

慢开始：不要一开始就发送大量的数据，由小到大逐渐增加拥塞窗口的大小。

拥塞避免：拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1而不是加倍。这样拥塞窗口按线性规律缓慢增长。

快重传：我们可以剔除一些不必要的拥塞报文，提高网络吞吐量。比如接收方在**收到一个失序的报文段后就立即发出重复确认，而不要等到自己发送数据时捎带确认。**快重传规定：发送方只要**一连收到三个**重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。



快恢复：主要是配合快重传。当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半（为了预防网络发生拥塞），但**接下来并不执行慢开始算法**，因为如果网络出现拥塞的话就不会收到好几个重复的确认，收到三个重复确认说明网络状况还可以。

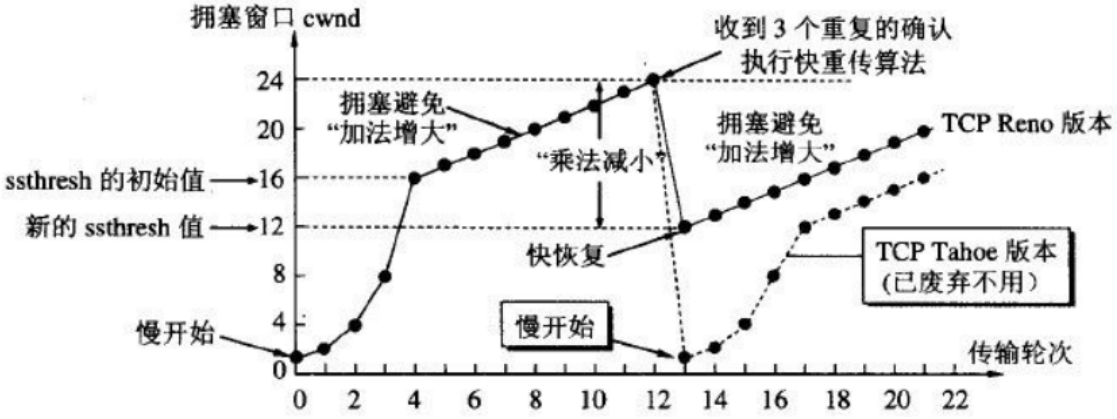


图 5-27　从连续收到三个重复的确认转入拥塞避免

End