## OSI七层网络模型



## TCP三次握手过程，为什么需要三次？

1. 首先Client向Server发送请求报文段，同时同步自己的SYN(x)，Client进入SYN\_SENT状态。
2. Server接收到Client的请求报文段，返回CLient自己的seq(y)及ack(x+1)，Server进入SYN\_REVD状态。
3. CLinet收到Server的SYN+ACK包，向服务器发送一个序列号seq(x+1)，确认号为ack(y+1)，此包发送完毕，Client和Server进入ESTABLISHED(TCP连接成功)状态，完成三次握手。

需要三次的原因：防止已失效的报文段出现在本连接中。

## TCP四次挥手的过程

1. 客户端发送断开TCP连接请求的报文，其中报文中包含seq序列号，是由发送端随机生成的，并且还将报文中的FIN字段置为1，表示需要断开TCP连接。（FIN=1，seq=x，x由客户端随机生成）
2. 服务端会回复客户端发送的TCP断开请求报文，其包含seq序列号，是由回复端随机生成的，而且会产生ACK字段，ACK字段数值是在客户端发过来的seq序列号基础上加1进行回复，以便客户端收到信息时，知晓自己的TCP断开请求已经得到验证。（FIN=1，ACK=x+1，seq=y，y由服务端随机生成）
3. 服务端在回复完客户端的TCP断开请求后，不会马上进行TCP连接的断开，服务端会先确保断开前，所有传输到A的数据是否已经传输完毕，一旦确认传输数据完毕，就会将回复报文的FIN字段置1，并且产生随机seq序列号。（FIN=1，ACK=x+1，seq=z，z由服务端随机生成）
4. 客户端收到服务端的TCP断开请求后，会回复服务端的断开请求，包含随机生成的seq字段和ACK字段，ACK字段会在服务端的TCP断开请求的seq基础上加1，从而完成服务端请求的验证回复。（FIN=1，ACK=z+1，seq=h，h为客户端随机生成）

为什么TCP协议终止链接要四次？

1、当客户端确认发送完数据且知道服务器已经接收完了，想要关闭发送数据口（当然确认信号还是可以发），就会发FIN给服务器。

2、服务器收到客户端发送的FIN，表示收到了，就会发送ACK回复。

3、但这时候服务器可能还在发送数据，没有想要关闭数据口的意思，所以服务器的FIN与ACK不是同时发送的，而是等到服务器数据发送完了，才会发送FIN给客户端。

4、客户端收到服务器发来的FIN，知道服务器的数据也发送完了，回复ACK， 客户端等待2MSL以后，没有收到服务器传来的任何消息，知道服务器已经收到自己的ACK了，客户端就关闭链接，服务器也关闭链接了。

5）2MSL意义：

1、保证最后一次握手报文能到B，能进行超时重传。

2、2MSL后，这次连接的所有报文都会消失，不会影响下一次连接。

## TIME\_WAIT的意义，为什么等于2MSL？

1. 为什么需要TIME\_WAIT状态？

假设最后的ACK丢失，server将重发FIN，client必须维护TCP状态信息以便可以重发最后的ACK，否则将会发送RST，结果server认为发生错误。TCP实现必须可靠地终止连接的两个方向，所以client必须进入TIME\_WAIT状态。

此外，考虑一种情况，TCP实现可能面临着先后两个相同的五元组。如果前一个连接处于TIME\_WAIT状态，而允许另一个拥有相同五元组连接出现，可能处理TCP报文时，两个连接互相干扰。所以使用SO\_REUSEADDR选项就需要考虑这种情况。

2. 什么是2MSL

MSL是Maximum Segment Lifetime,译为“报文最大生存时间”，他是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。

因为TCP报文（segment）是IP数据报（datagram）的数据部分，而IP头中有一个TTL域，TTL是time to live的缩写，中文可以译为“生存时间”，这个生存时间是由源主机设置初始值但不是存的具体时间，而是存储了一个IP数据报可以经过的最大路由数，每经过一个处理他的路由器此值就减1，当此值为0则数据报将被丢弃，同时发送ICMP报文通知源主机。

RFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等

2MSL即两倍的MSL，TCP的TIME\_WAIT状态也称为2MSL等待状态，当TCP的一端发起主动关闭，在发出最后一个ACK包后，即第3次握手完成后发送了第四次握手的ACK包后就进入了TIME\_WAIT状态，必须在此状态上停留两倍的MSL时间。

等待2MSL时间主要目的是怕最后一个ACK包对方没收到，那么对方在超时后将重发第三次握手的FIN包，主动关闭端接到重发的FIN包后可以再发一个ACK应答包。

在TIME\_WAIT状态时两端的端口不能使用，要等到2MSL时间结束才可继续使用。

当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。不过在实际应用中可以通过设置SO\_REUSEADDR选项达到不必等待2MSL时间结束再使用此端口。

## TCP、UDP的区别？服务器和客户端建立的过程？

* TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能顺序地从一端传到另一端。  
  UDP---用户数据报协议，是一个简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，不保证数据按顺序传递，故而传输速度很快。

UDP编程的服务器端一般步骤

1. 创建一个socket，用函数socket()；
2. 设置socket属性，用函数setsockopt();\* 可选
3. 绑定IP地址、端口等信息到socket上，用函数bind();
4. 循环接收数据，用函数recvfrom();
5. 关闭网络连接；

UDP编程的客户端一般步骤是

1. 创建一个socket，用函数socket()；
2. 设置socket属性，用函数setsockopt();\* 可选
3. 设置对方的IP地址和端口等属性;
4. 发送数据，用函数sendto();
5. 关闭网络连接；

TCP编程的服务器端一般步骤是

1. 创建一个socket，用函数socket()；
2. 设置socket属性，用函数setsockopt(); \* 可选
3. 绑定IP地址、端口等信息到socket上，用函数bind();
4. 开启监听，用函数listen()；
5. 接收客户端上来的连接，用函数accept()；
6. 收发数据，用函数send()和recv()，或者read()和write();
7. 关闭网络连接；
8. 关闭监听；

TCP编程的客户端一般步骤是

1. 创建一个socket，用函数socket()；
2. 设置socket属性，用函数setsockopt();\* 可选
3. 设置要连接的对方的IP地址和端口等属性；
4. 连接服务器，用函数connect()；
5. 收发数据，用函数send()和recv()，或者read()和write();
6. 关闭网络连接；

## TCP与UDP的概念，相互的区别及优劣。

**TCP（Transmission Control Protocol）的概念**

TCP是一种面向连接的，提供可靠交付服务和全双工通信的，基于字节流的端到端的传输层通信协议。

TCP在传输数据之前必须先建立连接，数据传输结束后要释放连接。

每一条TCP连接只能有2个端点，故TCP不提供广播或多播服务。

TCP提供可靠交付，通过TCP连接传输的数据，无差错、不丢失、不重复、并且按序到达。

TCP是面向字节流的。虽然应用进程和TCP的交互是一次一个数据块(大小不等），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并不知道所传输的字节流的含义。

**UDP（UserDatagram Protocol）的概念**

UDP是一种无连接的，尽最大努力交付的，基于报文的端到端的传输层通信协议。

UDP，在发送数据之前不需要建立连接

UDP不保证可靠交付，主机不需要位置复杂的连接状态

UDP是面向报文的。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的的边界，即应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。在接收端，UDP一次交付一个完整的报文。

UDP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。

UDP支持一对一、一对多、多对一和多对多的交互通信。

UDP的首部开销小，只有8个字节，比TCP的20个字节的首部要短。

**区别**

TCP协议面向连接，UDP协议面向非连接

TCP协议传输速度慢，UDP协议传输速度快

TCP协议保证数据顺序，UDP协议不保证

TCP协议保证数据正确性，UDP协议可能丢包

TCP协议对系统资源要求多，UDP协议要求少

**使用情况**

TCP协议适用于对效率要求相对低，但对准确性要求相对高的场景下，或者是有一种连接概念的场景下；而UDP协议适用于对效率要求相对高，对准确性要求相对低的场景。

很多文章都说TCP协议可靠，UDP协议不可靠！为什么前者可靠，后者不可靠呢？既然UDP协议不可靠，为什么还要使用它呢？所谓的TCP协议是面向连接的协议，面向连接是什么呢？TCP和UDP都是传输层的协议！从编程的角度看，就是两个模块（模块就是代码的集合，一系列代码的组合提供相应的功能！模块化最终目的就是：分工协作！模块化好处：便于扩展开发以及维护！）。

TCP协议和UDP协议为什么会共存?

1）大家要知道，一种物理线路，单位时间内，能够创建的“虚拟信道”是有限的！

2）使用TCP协议传输数据，当数据从A端传到B端后，B端会发送一个确认包（ACK包）给A端，告知A端数据我已收到！UDP协议就没有这种确认机制！这就是为什么说TCP协议可靠，UDP协议不可靠.

QQ普通会员就是使用的UDP协议进行传输数据！既然UDP协议自身没有确认机制，这个工作可以交给应用层的进程来完成（QQ）！大家使用QQ的时候，感觉出错的几率还是非常小吧！当然，把这个确认工作完全交给QQ自身来做，就直接导致了，QQ软件体积增大！

有些应用，对数据传输可靠性要求非常高，例如大家浏览网页，通过网页注册帐号、转帐等服务，这是不容许出错的，使用TCP协议能把出错的可能性降到最低（当然，网络自身很糟糕，TCP协议也没办法）。但是，提供这种可靠服务，会加大网络带宽的开销，因为“虚拟信道”是持续存在的，同时网络中还会出现大量的ACK和FIN包！

因此，鱼和熊掌不可兼得，需根据实际情况选择传输协议.TCP协议提供了可靠的数据传输,但是其拥塞控制、数据校验、重传机制的网络开销很大,不适合实时通信,所以选择开销很小的UDP协议来传输数据。

## 滑动窗口协议与停止等待协议的区别

滑动窗口协议中，允许发送方发送多个分组（当有多个分组可用时）而不需等待确认，但它受限于在流水线中为未确认的分组数不能超过某个最大允许数N。滑动窗口协议是TCP使用的一种流量控制方法，此协议能够加速数据的传输。只有在接收窗口向前滑动时（与此同时也发送了确认），发送窗口才有可能向前滑动。

收发两端的窗口按照以上规律不断地向前滑动，因此这种协议称为滑动窗口协议。

当发送窗口和接收窗口的大小都等于1时，就是停止等待协议。

## socket中的close是一次就关闭的吗？半关闭状态是怎么产生的？

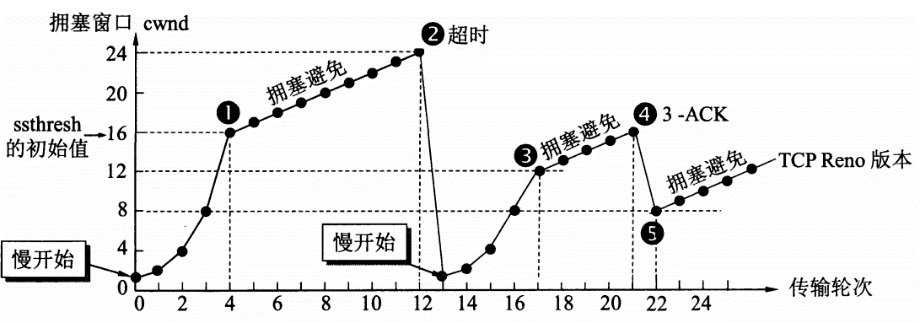
不是，当A发送给B控制FIN的时候，A到B这个方向的连接就关闭了，这个时候处于半关闭的状态，但是B到A这个方向的连接并没有关闭，因为B要等到将数据全部发送完毕之后才会发送FIN给A。

## [TCP拥塞控制](http://note.youdao.com/noteshare?id=3f6c7666f0396afa8214e45ae059b45d&sub=4919D6D86DF04928956A01DB93369B5F)

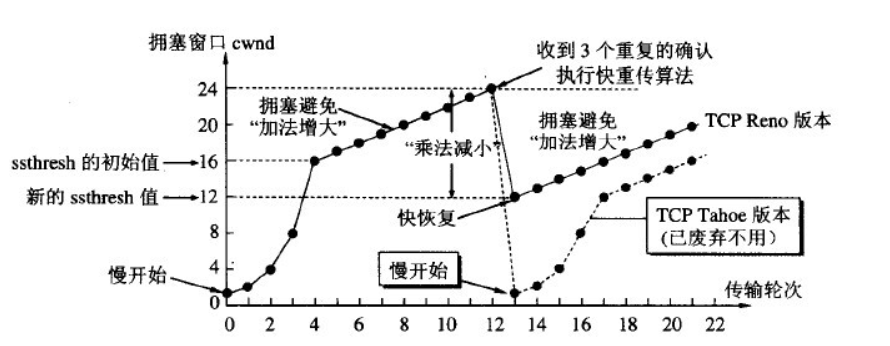
#### 1.TCP拥塞控制方法

网络拥塞的条件：\sum{\text{对资源的需求}}>{\text{可用资源}}∑对资源的需求>可用资源  
拥塞控制的四种算法：**慢开始、拥塞避免、快重传、快恢复。**

发送方维持一个叫做**拥塞窗口(cwnd)** 的状态变量，拥塞窗口的大小取决于网络的拥塞程度，并且动态的变化着，发送方让自己的发送窗口等于拥塞窗口。为了防止拥塞窗口增长过大引起网络阻塞，还需要设置一个**慢开始门限(ssthresh)** 状态变量。

**慢开始算法思路**：有小到大逐渐增大发送窗口，也就是从小到大逐渐增大拥塞窗口。当然收到单个确认但此确认多个数据报的时候就加相应的数值。每经过一个传输轮次，拥塞窗口就加倍。  
**拥塞避免算法思路**：让cwmd缓慢的增大，每经过一个往返时间RTT之后就把发送方的拥塞窗口cwnd加1，而不是像慢开始阶段那样加倍增长。 

**快重传算法思路**：发送方只要一连收到3个重复确认，就知道接收方确实没有收到下一个报文段，因而需要立即进行重传，这样就不会出现超时，发送方也不会认为出现了网络拥塞。  
**快恢复算法思路**：配合快重传使用，不执行慢开始算法。将ssthresh调整为当前cwnd的一半，随后 将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。



#### 2.随机早期检测RED

实际上网络层的策略对拥塞避免算法影响最大的就是路由器的丢弃策略。实现RED需要维持两个参数，即队列长度最小门限和最大门限。每当一个分组到达时，RED就按照规定的算法先计算当前队列的平均队列长度。  
（1）若平均队列长度小于最小门限，则把新到达的分组放入队列进行排队  
（2）若平均队列长度超过最大门限，则把新到达的分组丢弃  
（3）若平均队列长度在最小门限和最大门限之间，则按照某一丢弃概率p把新到达的分组丢弃

## TCP流量控制,采用滑动窗口会用什么问题？

* 流量控制是为了让发送方的发送速率不要太快，要让接收方来得及接收。  
  Nagle算法：①当发送方收到对第一个数据字符的确认后，再把发送缓存中的所有数据组装成一个报文段发送出去，同时继续对随后到到达的数据进行缓存。②当到达的数据已达到发送窗口大小的一半或已达到报文段的长度的时候就立即发送一个报文段。
* 糊涂窗口综合征：就是由于发送端和接收端上的处理不一致，导致网络上产生很多的小包，结果报文段包含了一个大大的头部，携带数据很少。数据传输效率低。处理方法是等待窗口大小满足一定的条件之后(能够接收一个最大报文，或者缓冲区的一半)，再来发送窗口通告，这样就不会产生小报文。

滑动窗口机制为端到端设备间的数据传输提供了可靠的流量控制机制。然而，它只能在源端设备和目的端设备起作用，当网络中间设备（例如路由器等）发生拥塞时，滑动窗口机制将不起作用。

## 拥塞控制和流量控制的区别？

1. 拥塞控制就是防止过多的数据注入到网络中，是一个全局性的过程，涉及到所有的主机、路由器这样可以使网络中的路由器或链路不会过载。
2. 流量控制往往是点对点通信量的控制，是一个端到端的问题，流量控制要做的是抑制发送端发送数据的速率，以便接收端来得及接收。

## TCP怎么保证可靠性？

1. 应用数据被分割成TCP认为最适合发送的数据块
2. 数据到达接收方，接收方需要发出一个确认应答，表示已经收到该数据段，并且确认序号会说明了它下一次需要接收的数据序列号。如果发送发迟迟未收到确认应答，那么可能是发送的数据丢失，也可能是确认应答丢失，这时发送方在等待一定时间后会进行重传。这个时间一般是2\*RTT(报文段往返时间）+一个偏差值。
3. 校验和：TCP将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP将丢弃这个报文段和不确认收到此报文段
4. TCP的接收端会丢弃重复的数据
5. 流量控制：让发送方的发送速率不要太快，要让接收方来得及接收
6. 拥塞控制：当网络拥塞时，减少数据的发送

## 数字证书是什么，里面都包含那些内容

1. 概念：

数字证书是数字证书在一个身份和该身份的持有者所拥有的公/私钥对之间建立了一种联系，由认证中心（CA）或者认证中心的下级认证中心颁发的。根证书是认证中心与用户建立信任关系的基础。在用户使用数字证书之前必须首先下载和安装。

认证中心是一家能向用户签发数字证书以确认用户身份的管理机构。为了防止数字凭证的伪造，认证中心的公共密钥必须是可靠的，认证中心必须公布其公共密钥或由更高级别的认证中心提供一个电子凭证来证明其公共密钥的有效性，后一种方法导致了多级别认证中心的出现。

1. 数字证书颁发过程：

数字证书颁发过程如下：用户产生了自己的密钥对，并将公共密钥及部分个人身份信息传送给一家认证中心。认证中心在核实身份后，将执行一些必要的步骤，以确信请求确实由用户发送而来，然后，认证中心将发给用户一个数字证书，该证书内附了用户和他的密钥等信息，同时还附有对认证中心公共密钥加以确认的数字证书。当用户想证明其公开密钥的合法性时，就可以提供这一数字证书。

1. 内容：
2. 数字证书的格式普遍采用的是X.509V3国际标准，一个标准的X.509数字证书包含以下一些内容：
3. 证书的版本信息；
4. 证书的序列号，每个证书都有一个唯一的证书序列号；
5. 证书所使用的签名算法；
6. 证书的发行机构名称，命名规则一般采用X.500格式；
7. 证书的有效期，通用的证书一般采用UTC时间格式；
8. 证书所有人的名称，命名规则一般采用X.500格式；
9. 证书所有人的公开密钥；
10. 证书发行者对证书的签名。

## http协议与TCP协议的联系

TPC协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据。  
Http协议是建立在TCP协议基础之上的，当浏览器需要从服务器获取网页数据的时候，会发出一次Http请求。Http会通过TCP建立起一个到服务器的连接通道，当本次请求需要的数据完毕后，Http会立即将TCP连接断开，这个过程是很短的。所以Http连接是一种短连接，是一种无状态的连接。

* [**http/1.0和http/1.1的区别？**](https://www.jianshu.com/p/95a521b006a8)

1. http1.1提供永久性连接，即1.0使用非持久连接
2. http1.1增加host头
3. http1.1还提供了身份认证，状态管理和cache缓存机制等相关的请求头和响应头。

## http请求方法有哪些？get和post的区别

1. OPTIONS  
   返回服务器针对特定资源所支持的HTTP请求方法，也可以利用向web服务器发送‘\*’的请求来测试服务器的功能性
2. HEAD  
   向服务器索与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以再不必传输整个响应内容的情况下，就可以获取包含在响应小消息头中的元信息。
3. GET  
   向特定的资源发出请求。注意：GET方法不应当被用于产生“副作用”的操作中，例如在Web Application中，其中一个原因是GET可能会被网络蜘蛛等随意访问。Loadrunner中对应get请求函数：web\_link和web\_url
4. POST  
   向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 Loadrunner中对应POST请求函数：web\_submit\_data,web\_submit\_form
5. PUT  
   向指定资源位置上传其最新内容
6. DELETE  
   请求服务器删除Request-URL所标识的资源
7. TRACE  
   回显服务器收到的请求，主要用于测试或诊断
8. CONNECT  
   HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

(1)根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的  
(2)根据HTTP规范，POST表示可能修改变服务器上的资源的请求

对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）

区别：

1、get参数通过url传递，post放在request body中。

2、get请求在url中传递的参数是有长度限制的，而post没有。

3、get比post更不安全，因为参数直接暴露在url中，所以不能用来传递敏感信息。

4、get请求只能进行url编码，而post支持多种编码方式。

5、get请求会浏览器主动cache，而post支持多种编码方式。

6、get请求参数会被完整保留在浏览历史记录里，而post中的参数不会被保留。

7、GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

8、GET产生一个TCP数据包；POST产生两个TCP数据包。

9、GET 请求用于获取资源，无副作用，所以是幂等的, POST 不具备幂等性，因为 POST 请求每次都会创建一个文件

## http和https的区别？由http升级到https需要哪些操作？

HTTP 指的是超文本传输协议，https 指的是超文本传输安全协议。HTTPS 就是将 HTTP 中的传输内容进行了加密，然后通过可靠的连接，传输到对方的机器上。加密的协议是 TLS,其前身是 SSL。

HTTP协议和HTTPS协议区别如下：

1）HTTP协议是以明文的方式在网络中传输数据，而HTTPS协议传输的数据则是经过TLS加密后的，HTTPS具有更高的安全性

2）HTTPS在TCP三次握手阶段之后，还需要进行SSL 的handshake，协商加密使用的对称加密密钥

3）HTTPS协议需要服务端申请证书，浏览器端安装对应的根证书

4）HTTP协议端口是80，HTTPS协议端口是443

HTTPS优点：

HTTPS传输数据过程中使用密钥进行加密，所以安全性更高

HTTPS协议可以认证用户和服务器，确保数据发送到正确的用户和服务器

HTTPS缺点：

HTTPS握手阶段延时较高：由于在进行HTTP会话之前还需要进行SSL握手，因此HTTPS协议握手阶段延时增加

HTTPS部署成本高：一方面HTTPS协议需要使用证书来验证自身的安全性，所以需要购买CA证书；另一方面由于采用HTTPS协议需要进行加解密的计算，占用CPU资源较多，需要的服务器配置或数目高

## http中浏览器一个URL的流程，这个过程中浏览器做些什么，URL包括哪三个部分？

1）HTTP协议：

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW:World Wide Web）服务器传输超文本到本地浏览器的传送协议。

HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件，图片文件，查询结果等）。

HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。

2）HTTP协议特点

1、简单快速：

客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

2、灵活：

HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

3、无连接：

无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

4、无状态：

HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

5、支持B/S及C/S模式。

6、默认端口80

7、基于TCP协议

3）HTTP过程概述：

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

HTTP 请求/响应的步骤如下：

1、客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，[http://www.baidu.com](http://www.baidu.com/)。

2、发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

3、服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

4、释放连接TCP连接

若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;

5、客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

4、举例：

在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址；

2、解析出 IP 地址后，根据该 IP 地址和默认端口80，和服务器建立TCP连接；

3、浏览器发出读取文件（URL中域名后面部分对应的文件）的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器；

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器；

5、释放 TCP连接；

6、浏览器将该 html 文本并显示内容；

URL包括三部分：①协议（或称为服务方式）；②存有该资源的主机IP地址（有时也包括端口号）；③主机资源的具体地址，如目录和文件名等。

## 一个机器能使用的端口号上限是多少？为什么？可以改变吗？如果想要用的端口超过这个限制怎么办？

## 对称加密和非对称加密

对称加密(也叫私钥加密)指加密和解密使用相同密钥的加密算法。有时又叫传统密码算法，就是加密密钥能够从解密密钥中推算出来，同时解密密钥也可以从加密密钥中推算出来。  
非对称加密算法需要两个密钥:公开密钥(publickey)和私有密钥(privatekey)。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密;如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。

## [TCP/IP分片粘包过程](https://blog.csdn.net/qq513036862/article/details/53640317)

1. 正常情况：如果Socket Client 发送的数据包，在Socket Server端也是一个一个完整接收的，那个就不会出现粘包和分包情况，数据正常读取。
2. 粘包情况：Socket Client发送的数据包，在客户端发送和服务器接收的情况下都有可能发送，因为客户端发送的数据都是发送的一个缓冲buffer，然后由缓冲buffer最后刷到数据链路层的，那么就有可能把数据包2的一部分数据结合数据包1的全部被一起发送出去了，这样在服务器端就有可能出现这样的情况，导致读取的数据包包含了数据包2的一部分数据，这就产生粘包，当然也有可能把数据包1和数据包2全部读取出来。
3. 分包情况：意思就是把数据包2或者数据包1都有可能被分开一部分发送出去，接着发另外的部分，在服务器端有可能一次读取操作只读到一个完整数据包的一部分。
4. 在数据包发送的情况下，有可能后面的数据包分开成2个或者多个，但是最前面的部分包，黏住在前面的一个完整或者部分包的后面，也就是粘包和分包同时产生了。

## [有没有抓过TCP包，描述一下](https://blog.csdn.net/hebbely/article/details/54424823)

## ARP协议的用途及算法、在哪一层上会使用ARP？

ARP (Address Resolution Protocol) 是个地址解析协议。最直白的说法是：在IP以太网中，当一个上层协议要发包时，有了该节点的IP地址，ARP就能提供该节点的MAC地址。

为什么要有ARP？

OSI 模式把网络工作分为七层，彼此不直接打交道，只通过接口(layre interface). IP地址在第三层, MAC地址在第二层。协议在发生数据包时，首先要封装第三层（IP地址）和第二层 （MAC地址）的报头, 但协议只知道目的节点的IP地址，不知道其物理地址，又不能跨第二、三层，所以得用ARP的服务。

详细说明：

Ø  在网络通讯时，源主机的应用程序知道目的主机的IP地址和端口号，却不知道目的主机的硬件地址，而数据包首先是被网卡接收到再去处理上层协议的，如果接收到的数据包的硬件地址与本机不符，则直接丢弃。因此在通讯前必须获得目的主机的硬件地址。ARP协议就起到这个作用

Ø  当一台主机把以太网数据帧发送到位于同一局域网上的另一台主机时，是根据 48位的以太网地址来确定目的接口的，设备驱动程序从不检查 IP数据报中的目的IP地址。ARP（地址解析）模块的功能为这两种不同的地址形式提供映射：32位的 IP地址和 48位的以太网地址

ARP协议的用途：解决同一个局域网内主机或路由器的IP地址和MAC地址的映射问题。

算法：在主机的ARP高速缓存中应存放一个从IP地址到MAC地址的映射表，并且这个映射表还经常动态更新（新增或超时删除）。

在网络层会使用ARP

## ICMP协议

ICMP的全称是 Internet Control Message Protocol。从技术角度来说，ICMP就是一个“错误侦测与回报机制”，其目的就是让我们能够检测网路的连线状况﹐也能确保连线的准确性。用IP数据包发送，由于IP数据报只检查首部，不检查数据，不保证安全（TCP UDP 都检查）其功能主要有：

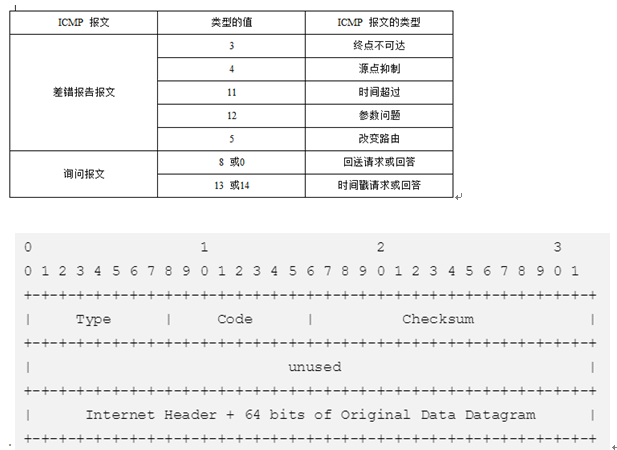
侦测远端主机是否存在

建立及维护路由资料

重导数据传送路径

数据流量控制

ICMP在沟通之中，主要是通过不同类型的报文让主机或路由器识别不同的链路传输情况。常用的ICMP报文类型如下



## 知道各个层使用的是哪个数据交换设备。（交换机、路由器、网关）

物理层用到的设备是集线器和中继器

数据链路层用到的设备是交换机和网桥

网络层用到的设备是路由器

应用层用到的设备是网关

## [介绍一下ping的过程，分别用到了那些协议？](https://blog.csdn.net/fd8559350/article/details/52135571)

Ping（Packet InterNet Groper）分组网间探测是ICMP的一个重要应用，用来测试两个主机之间的连通性。Ping使用了ICMP回送请求与回送回答报文。Ping是应用层直接使用网络层ICMP的一个例子。它没有通过运输层的TCP或UDP。

    实现原理为向目的主机发送4个32字节长的ICMP回送请求报文，若目的主机正常工作并且响应了该ICMP回送请求报文，就将发回ICMP回送回答报文。最后可得出的统计结果为目的IP地址，发送的，收到的和丢失的分组数，及往返时间的最小值、最大值和平均值。

Ping命令格式为ping hostname 此hostname即为要测试连通性的主机名或它的IP地址。

Ping工作过程——

    假定主机A的IP地址是192.168.1.1，主机B的IP地址是192.168.1.2，都在同一子网内，则当你在主机A上运行“Ping 192.168.1.2”后，都发生了些什么呢?

首先，Ping命令会构建一个固定格式的ICMP请求数据包，然后由ICMP协议将这个数据包连同地址“192.168.1.2”一起交给IP层协议（和ICMP一样，实际上是一组后台运行的进程），IP层协议将以地址“192.168.1.2”作为目的地址，本机IP地址作为源地址，加上一些其他的控制信息，构建一个IP数据包，并在一个映射表中查找出IP地址192.168.1.2所对应的物理地址（也叫MAC地址，熟悉网卡配置的朋友不会陌生，这是数据链路层协议构建数据链路层的传输单元——帧所必需的），一并交给数据链路层。后者构建一个数据帧，目的地址是IP层传过来的物理地址，源地址则是本机的物理地址，还要附加上一些控制信息，依据以太网的介质访问规则，将它们传送出去。

    其中映射表由ARP实现。ARP(Address Resolution Protocol)是地址解析协议,是一种将IP地址转化成物理地址的协议。ARP具体说来就是将网络层（IP层，也就是相当于OSI的第三层）地址解析为数据连接层（MAC层，也就是相当于OSI的第二层）的MAC地址。

    主机B收到这个数据帧后，先检查它的目的地址，并和本机的物理地址对比，如符合，则接收；否则丢弃。接收后检查该数据帧，将IP数据包从帧中提取出来，交给本机的IP层协议。同样，IP层检查后，将有用的信息提取后交给ICMP协议，后者处理后，马上构建一个ICMP应答包，发送给主机A，其过程和主机A发送ICMP请求包到主机B一模一样。

即先由IP地址，在网络层传输，然后再根据mac地址由数据链路层传送到目的主机

UDP-ICMP（internet 控制报文协议）-ARP（地址解析的协议，mac地址）-OSPF（开放最短路径优先，内部网关协议）

## 服务器攻击（DDos攻击）

关于Ddos：攻击者向互联网某一个服务器不停发送大量分组，使该服务器无法提供正常服务。

1. SYN Flood：利用TCP协议的原理，这种攻击方法是经典最有效的DDOS方法，可通杀各种系统的网络服务，主要是通过向受害主机发送大量伪造源IP和源端口的SYN或ACK 包，导致主机的缓存资源被耗尽或忙于发送回应包而造成拒绝服务。TCP通道在建立以前，需要三次握手：  
a. 客户端发送一个包含SYN标志的TCP报文， 同步报文指明客户端所需要的端口号和TCP连接的初始序列号  
b. 服务器收到SYN报文之后，返回一个SYN+ ACK报文，表示客户端请求被接受，TCP初始序列号加1  
c.客户端也返回一个确认报文ACK给服务器，同样TCP序列号加1  
d. 如果服务器端没有收到客户端的确认报文ACK，则处于等待状态，将该客户IP加入等待队列，然后轮训发送SYN+ACK报文  
所以攻击者可以通过伪造大量的TCP握手请求，耗尽服务器端的资源。

2. HTTP Flood：针对系统的每个Web页面，或者资源，或者Rest API，用大量肉鸡，发送大量http request。这种攻击主要是针对存在ASP、JSP、PHP、CGI等脚本程序，并调用MSSQLServer、MySQLServer、Oracle等数据库的网站系统而设计的，特征是和服务器建立正常的TCP连接，并不断的向脚本程序提交查询、列表等大量耗费数据库资源的调用，典型的以小博大的攻击方法。缺点是对付只有静态页面的网站效果会大打折扣。

3. 慢速攻击：Http协议中规定，HttpRequest以\r\n\r\n结尾来表示客户端发送结束。攻击者打开一个Http 1.1的连接，将Connection设置为Keep-Alive， 保持和服务器的TCP长连接。然后始终不发送\r\n\r\n， 每隔几分钟写入一些无意义的数据流， 拖死机器。

4. P2P攻击：每当网络上出现一个热门事件，比如XX门， 精心制作一个种子， 里面包含正确的文件下载， 同时也包括攻击目标服务器的IP。这样，当很多人下载的时候， 会无意中发起对目标服务器的TCP连接。

DDOS攻击现象判定方法

1.SYN类攻击判断：A.CPU占用很高；B.网络连接状态：netstat –na,若观察到大量的SYN\_RECEIVED的连接状态；C.网线插上后，服务器立即凝固无法操作，拔出后有时可以恢复，有时候需要重新启动机器才可恢复。  
2.CC类攻击判断：A.网站出现service unavailable提示；B.CPU占用率很高；C.网络连接状态：netstat –na,若观察到大量的ESTABLISHED的连接状态 单个IP高达几十条甚至上百条；D.用户无法访问网站页面或打开过程非常缓慢,软重启后短期内恢复正常,几分钟后又无法访问。  
3.UDP类攻击判断：A.观察网卡状况 每秒接受大量的数据包；B.网络状态：netstat –na TCP信息正常。  
4.TCP洪水攻击判断：A.CPU占用很高；B.netstat –na,若观察到大量的ESTABLISHED的连接状态 单个IP高达几十条甚至上百条

DDoS攻击防御方法：

1. 过滤不必要的服务和端口：可以使用Inexpress、Express、Forwarding等工具来过滤不必要的服务和端口，即在路由器上过滤假IP。比如Cisco公司的CEF(Cisco Express Forwarding)可以针对封包Source IP和Routing Table做比较，并加以过滤。只开放服务端口成为目前很多服务器的流行做法，例如WWW服务器那么只开放80而将其他所有端口关闭或在防火墙上做阻止策略。  
2. 异常流量的清洗过滤：通过DDOS硬件防火墙对异常流量的清洗过滤，通过数据包的规则过滤、数据流指纹检测过滤、及数据包内容定制过滤等顶尖技术能准确判断外来访问流量是否正常，进一步将异常流量禁止过滤。单台负载每秒可防御800-927万个syn攻击包。  
3. 分布式集群防御：这是目前网络安全界防御大规模DDOS攻击的最有效办法。分布式集群防御的特点是在每个节点服务器配置多个IP地址（负载均衡），并且每个节点能承受不低于10G的DDOS攻击，如一个节点受攻击无法提供服务，系统将会根据优先级设置自动切换另一个节点，并将攻击者的数据包全部返回发送点，使攻击源成为瘫痪状态，从更为深度的安全防护角度去影响企业的安全执行决策。  
4. 高防智能DNS解析：高智能DNS解析系统与DDOS防御系统的完美结合，为企业提供对抗新兴安全威胁的超级检测功能。它颠覆了传统一个域名对应一个镜像的做法，智能根据用户的上网路线将DNS解析请求解析到用户所属网络的服务器。同时智能DNS解析系统还有宕机检测功能，随时可将瘫痪的服务器IP智能更换成正常服务器IP，为企业的网络保持一个永不宕机的服务状态

## 客户端服务器通信死锁如何处理?

1. 一旦RecvQ已满，TCP流控制机制就会产生作用（使用流控制机制的目的是为了保证发送者不会传输太多数据，从而超出了接收系统的处理能力），它将阻止传输发送端主机的SendQ中的任何数据，直到接收者调用输入流的read（）方法将RecvQ中的数据移除一部分到Delivered中，从而腾出了空间。发送端可以持续地写出数据，直到SendQ队列被填满，如果SendQ队列已满时调用输出流的write（）方法，则会阻塞等待，直到有一些字节被传输到RecvQ队列中，如果此时RecvQ队列也被填满了，所有的操作都将停止，直到接收端调用了输入流的read（）方法将一些字节传输到了Delivered队列中。

引出问题：

       我们假设SendQ队列和RecvQ队列的大小分别为SQS和RQS。将一个大小为n的字节数组传递给发送端write（）方法调用，其中n > SQS，直到有至少n-SQS字节的数据传递到接收端主机的RecvQ队列后，该方法才返回。如果n的大小超过了SQS+RQS，write（）方法将在接收端从输入流读取了至少n-(SQS+RQS)字节后才会返回。如果接收端没有调用read（）方法，[大数据](http://lib.csdn.net/base/hadoop)量的发送是无法成功的。特别是连接的两端同时分别调用它的输出流的write（）方法，而他们的缓冲区大小又大于SQS+RQS时，将会发生死锁：两个write操作都不能完成，两个程序都将永远保持阻塞状态。

1. 下面考虑一个具体的例子，即主机A上的程序和主机B上的程序之间的TCP连接。假设A和B上的SQS和RQS都是500字节，下图展示了两个程序试图同时发送1500字节时的情况。主机A上的程序中的前500字节已经传输到另一端，另外500字节已经复制到了主机A的SendQ队列中，余下的500字节则无法发送，write（）方法将无法返回，直到主机B上程序的RecvQ队列有空间空出来，然而不幸的是B上的程序也遇到了同样的情况，而二者都没有及时调用read（）方法从自己的RecvQ队列中读取数据到Delivered队列中。因此，两个程序的write（）方法调用都永远无法返回，产生死锁。因此，在写程序时，要仔细设计协议，以避免在两个方向上传输大量数据时产生死锁
2. 如何解决这个问题呢？造成死锁产生的原因是因为客户端在发送数据的同时，没有及时读取反馈回来的数据，从而使数据都阻塞在了底层的传输队列中。

   方案一是在编写客户端程序时，使客户端一边循环调用输出流的read（）方法向服务端发送数据，一边循环调用输入流的read（）方法读取从服务端反馈回来的数据，但这也不能完全保证不会产生死锁。

   更好的解决方案是在不同的线程中执行客户端的write循环和read循环。一个线程从文件中反复读取未压缩的字节并将其发送给服务器，直到文件的结尾，然后调用该套接字的shutdownOutput（）方法。另一个线程从服务端的输入流中不断读取压缩后的字节，并将其写入输出文件，直到到达了输入流的结尾（服务器关闭了套接字）。这样，便可以实现一边发送，一边读取，而且如果一个线程阻塞了，另一个线程仍然可以独立执行。这样我们可以对客户端代码进行简单的修改，将SendByes（）方法调用放到一个线程中：

## 一个ip配置多个域名，靠什么识别？

http请求中的host域，或者是端口号识别，不同端口号指向不同的网站处理

## 对路由协议的了解与介绍

RIP协议是一种内部网关协议（IGP），是一种动态路由选择协议，用于自治系统（AS）内的路由信息的传递。RIP协议基于距离矢量算法（DistanceVectorAlgorithms），使用“跳数”(即metric)来衡量到达目标地址的路由距离。这种协议的路由器只关心自己周围的世界，只与自己相邻的路由器交换信息，范围限制在15跳(15度)之内。

工作原理：RIP通过UDP报文来交换相邻的路由信息，每30秒发送一次路由信息更新。RIP提供跳跃计数(hopcount)作为尺度来衡量路由距离，跳跃计数是一个包到达目标所必须经过的路由器的数目。如果到相同目标有二个不等速或不同带宽的路由器，但跳跃计数相同，则RIP认为两个路由是等距离的。RIP最多支持的跳数为15，即在源和目的网间所要经过的最多路由器的数目为15，跳数16表示不可达。

缺点：当网络出现故障时，要经过比较长的时间才能将此信息传送到所有的路由器。以图2为例，设三个路由器都已经建立了各自的路由表，现在路由器R1和网1的连接线路短开。路由器R1发现后，将到网1的距离改为16，并将此信息发给路由器R2。由于路由器R3发给R2的信息是：“到网1经过R2距离为2”，R2误认为经过R3可到达网1，于是R2将此项目更新为“到网1经过R3距离为3”，发给R3。R3再发给R2信息：“到网1经过肉距离为4”。这样一直到距离增大到16时，R2和R3才知道网1是不可达的。RIP协议的这一特点叫做：好消息传播得快，而坏消息传播得慢。像这种网络出故障的传播时间往往需要较长的时间，这是RIP的一个主要缺点

优点：实现简单，开销小，适用小规模网络

OSPF协议是开放最短路径优先协议。

OSPF协议是“开放式最短路径优先(Open Shortest Path First)”的缩写，属于链路状态路由协议。OSPF提出了“区域（area）”的概念，每个区域中所有路由器维护着一个相同的链路状态数据库 （LSDB）。区域又分为骨干区域（骨干区域的编号必须为0）和非骨干区域（非0编号区域），如果一个运行OSPF的网络只存在单一区域，则该区域可以是 骨干区域或者非骨干区域。如果该网络存在多个区域，那么必须存在骨干区域，并且所有非骨干区域必须和骨干区域直接相连。OSPF利用所维护的链路状态数据 库，通过最短生成树算法（[SPF](http://zh.wikipedia.org/wiki/SPF)算法）计算得到路由表。OSPF的收敛速度较快。由于其特有的开放性以及良好的扩展性，目前OSPF协议在各种网络中广泛部署。最主要的特征是使用链路状态协议。直接用IP数据报向本自治系统的所有路由器发信息。“”链路状态“说明本路由器和哪些路由器相邻，以及“度量“，可以表示费用、距离、带宽。只有链路状态发生变化时，路由器才向所有路由器用泛洪法发送此消息。所有路由器都能建立一个链路状态数据库。

优点：链路状态数据库较快更新，更新过程收敛快。

OSPF的三张表

①：邻居表

2台路由器的OSPF要协同工作，基本要求就是二者形成全毗邻的邻接关系，而邻居表存储了OSPF路由器邻居的状态和邻居的其他信息。

②：拓扑数据库（LSDB）

OSPF用LSA来描述网络拓扑信息，LSDB中存储着路由器产生或者受到的LSA。

③：OSPF路由表

基于LSDB进行SPF算法运算，计算出的路由存储在此表中，也就是说用于实际数据传送的路由存在此处。

## [http状态码](https://blog.csdn.net/hliq5399/article/details/51351706)

状态码 响应类别 原因短语

1XX 信息性状态码（Informational） 服务器正在处理请求

2XX 成功状态码（Success） 请求已正常处理完毕

3XX 重定向状态码（Redirection） 需要进行额外操作以完成请求

4XX 客户端错误状态码（Client Error）客户端原因导致服务器无法处理请求

5XX 服务器错误状态码（Server Error） 服务器原因导致处理请求出错

200 OK

请求正常处理完毕

204 No Content

请求成功处理，没有实体的主体返回

206 Partial Content

GET范围请求已成功处理

301 Moved Permanently

永久重定向，资源已永久分配新URI

302 Found

临时重定向，资源已临时分配新URI

303 See Other

临时重定向，期望使用GET定向获取

304 Not Modified

发送的附带条件请求未满足

307 Temporary Redirect

临时重定向，POST不会变成GET

400 Bad Request

请求报文语法错误或参数错误

401 Unauthorized

需要通过HTTP认证，或认证失败

403 Forbidden

请求资源被拒绝

404 Not Found

无法找到请求资源（服务器无理由拒绝）

500 Internal Server Error

服务器故障或Web应用故障

503 Service Unavailable

服务器超负载或停机维护

## 网络层分片的原因与具体实现

* **MTU（最大传输单元）**
* MTU前面已经说过了，是链路层中的网络对数据帧的一个限制，依然以以太网为例，MTU为1500个字节。一个IP数据报在以太网中 传输，如果它的长度大于该MTU值，就要进行分片传输，使得每片数据报的长度小于MTU。分片传输的IP数据报不一定按序到达，但IP首部中的信息能让这些数据报片按序组装。IP数据报的分片与重组是在网络层进完成的。
* **MSS（最大分段大小）**
* MSS是TCP里的一个概念（首部的选项字段中）。MSS是TCP数据包每次能够传输的最大数据分段，TCP报文段的长度大于MSS时，要进行分段传输。TCP协议在建立连接的时候通常要协商双方的MSS值，每一方都有用于通告它期望接收的MSS选项（MSS选项只出现在SYN报文段中，即TCP三次握手的前两次）。MSS的值一般为MTU值减去两个首部大小（需要减去IP数据包包头的大小20Bytes和TCP数据段的包头20Bytes）所以如果用链路层以太网，MSS的值往往为1460。而Internet上标准的MTU（最小的MTU，链路层网络为x2.5时）为576，那么如果不设置，则MSS的默认值就为536个字节。很多时候，MSS的值最好取512的倍数。TCP报文段的分段与重组是在运输层完成的。
* 到了这里有一个问题自然就明了了，TCP分段的原因是MSS，IP分片的原因是MTU，由于一直有MSS<=MTU，很明显，分段后的每一段TCP报文段再加上IP首部后的长度不可能超过MTU，因此也就不需要在网络层进行IP分片了。因此TCP报文段很少会发生IP分片的情况。
* 再来看UDP数据报，由于UDP数据报不会自己进行分段，因此当长度超过了MTU时，会在网络层进行IP分片。同样，ICMP（在网络层中）同样会出现IP分片情况。
* 总结：UDP不会分段，就由IP来分。TCP会分段，当然就不用IP来分了！
* 另外，IP数据报分片后，只有第一片带有UDP首部或ICMP首部，其余的分片只有IP头部，到了端点后根据IP头部中的信息在网络层进行重组。而TCP报文段的每个分段中都有TCP首部，到了端点后根据TCP首部的信息在传输层进行重组。IP数据报分片后，只有到达目的地后才进行重组，而不是向其他网络协议，在下一站就要进行重组。
* 最后一点，对IP分片的数据报来说，即使只丢失一片数据也要重新传整个数据报（既然有重传，说明运输层使用的是具有重传功能的协议，如TCP协议）。这是因为IP层本身没有超时重传机制------由更高层（比如TCP）来负责超时和重传。当来自TCP报文段的某一段（在IP数据报的某一片中）丢失后，TCP在超时后会重发整个TCP报文段，该报文段对应于一份IP数据报（可能有多个IP分片），没有办法只重传数据报中的一个数据分片。

## RSA加密算法，MD5原理

一、MD5算法介绍。

（密码散列函数）MD5，即“Message-Digest Algorithm 5（**信息**-摘要算法）”，从名字来看就知道它是从MD3、MD4发展而来的一种加密算法，其主要通过采集文件的信息摘要，以此进行计算并加密。通过MD5算法进行加密，文件就可以获得一个唯一的MD5值，**这个值是独一无二的，就像我们的指纹一样**，因此我们就可以通过文件的MD5值来确定文件是否正确，密码进行加密后也会生成MD5值，论坛就是通过MD5值来验证用户的密码是否正确的。后来有一种SHA代替MD5 叫安全散列算法。

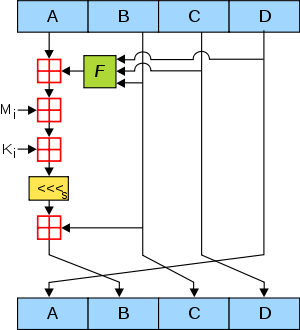
二、MD5算法实现。

MD5是输入不定长度信息，输出固定长度128-bits的算法。经过程序流程，生成四个32位数据，最后联合起来成为一个128-bits散列。基本方式为，求余、取余、调整长度、与链接变量进行循环运算。得出结果。

1、填充编码。

在MD5算法中，首先需要对信息进行填充，使其位长对512求余的结果等于448。因此，信息的位长（Bits Length）将被扩展至N\*512+448，N为一个非负整数，N可以是零。填充的方法如下，在信息的后面填充一个1和无数个0，直到满足上面的条件时才停止用0对信息的填充。然后，在这个结果后面附加一个以64位二进制表示的填充前信息长度。经过这两步的处理，现在的信息的位长=N\*512+448+64=(N+1）\*512，即长度恰好是512的整数倍。这样做的原因是为满足后面处理中对信息长度的要求。

主循环次数 = M / 512   
每个主循环中包含 512 / 32 \* 4 = 64 次 子循环。



1.绿色F   
图中的绿色F，代表非线性函数。官方MD5所用到的函数有四种：

F(X, Y, Z) =(X&Y) | ((~X) & Z)

G(X, Y, Z) =(X&Z) | (Y & (~Z))

H(X, Y, Z) =X^Y^Z

I(X, Y, Z)=Y^(X|(~Z))

在主循环下面64次子循环中，F、G、H、I 交替使用，第一个16次使用F，第二个16次使用G，第三个16次使用H，第四个16次使用I。

2.红色“田”字   
很简单，红色的田字代表相加的意思。

3.Mi   
Mi是第一步处理后的原文。在第一步中，处理后原文的长度是512的整数倍。把原文的每512位再分成16等份，命名为M0~M15，每一等份长度32。在64次子循环中，每16次循环，都会交替用到M1~M16之一。

4.Ki   
一个常量，在64次子循环中，每一次用到的常量都是不同的。

非对称加密

还是这个例子，一个人A想把信息m发给B，B生成了两个有关系的e 和 d ，e称为公钥，d称为私钥，B可以通过公开的方式把e发给A

A通过e里面的某一个算法 例如+e 对m进行加密得到C，A也可以通过公开的方式把C发送给B,不对称的就不对称到这，B收到之后通过不是通过-e的方式，而是-d的方式，得到了信息m

也就是说 通过e加密，然后用d去解密

**RSA加密算法**

其中一个最经典的非对称就是RSA加密算法，那这个RSA是基于什么原理，公钥e和私钥d有一个怎么样的关系呢

首先 B先找出两个质数 p和q

计算   n = p \* q

φ(n) = (p-1)  \* (q-1)      这个函数又叫欧拉函数

公钥e    1 < e < φ(n)   在这个范围取出一个整数，并且 e 和 φ(n) 互质

私钥d    (e \* d ) / φ(n)  = x  ······ 1       (x是几无所谓，余数必须是1，比如 φ(n)是20 e是3 那么d就是7)

 加密  m^e / n =x  ······  c     (这几个x都代表任意数的商，在这里面没有直接意义，可以忽略不看，这三个x也都不相等)

 解密  c^d  / n =x   ······ m   (可以从数学上证明，经过这一系列的计算，得到结果的余数一定是m)

## 应用层协议常用的端口号

FTP(文件传输协议):20 21  
SSH(安全shell协议）:22  
telnet（远程登录协议）:23  
DNS（域名解析系统）:53  
HTTP（超文本传输协议）:80  
SMTP（简单邮件传输协议）:25  
POP3（邮局协议3代）:110

**IMAP 143/tcp**

状态码的类别

1XX Informational（信息性状态码） 接收的请求正在处理

2XX Success（成功状态码） 请求正常处理完毕

3XX Redirection（重定向状态码） 需要进行附加操作以完成请求

4XX Client Error（客户端错误状态码） 服务器无法处理请求

5XX Server Error（服务器错误状态码） 服务器处理请求出错

使用 GET 方法时，对应请求资源的实体会作为响应返回；而使用 HEAD 方法时，对应请求资源的实体首部不随报文主体作为响应返回（即在响应中只返回首部，不会返回实体的主体部

分）。

代理服务器：

每次通过代理服务器转发请求或响应时，会追加写入 Via 首部信息

使用代理服务器的理由有：利用缓存技术（稍后讲解）减少网络带宽

的流量，组织内部针对特定网站的访问控制，以获取访问日志为主要

目的，等等。

缓存代理

代理转发响应时，缓存代理（Caching Proxy）会预先将资源的副本

（缓存）保存在代理服务器上。

透明代理

转发请求或响应时，不对报文做任何加工的代理类型被称为透明代理

（Transparent Proxy）。反之，对报文内容进行加工的代理被称为非

透明代理。

网管：

网关可以由 HTTP 请求转化为其他协议通信

利用网关能提高通信的安全性，因为可以在客户端与网关之间的通信

线路上加密以确保连接的安全。

隧道：

隧道的目的是确保客户端能与服务器进行安全的通信。

缓存是保存的资源副本减少对源服务器的访问节省了通信流量和通信时间

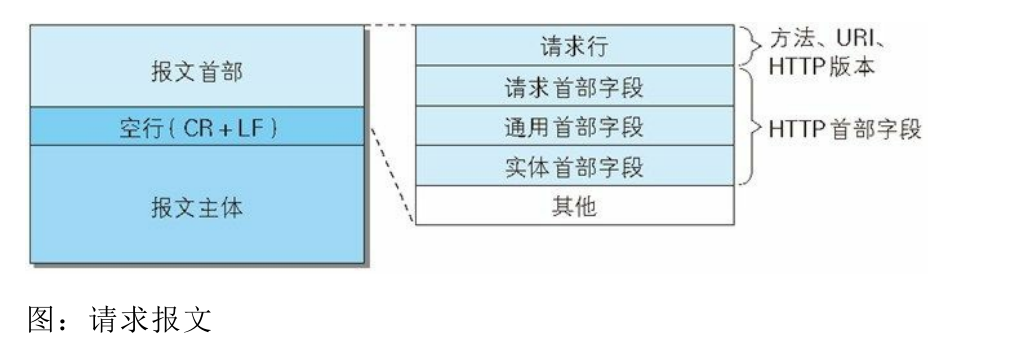
缓存还可以存在客户端浏览器中

http请求报文

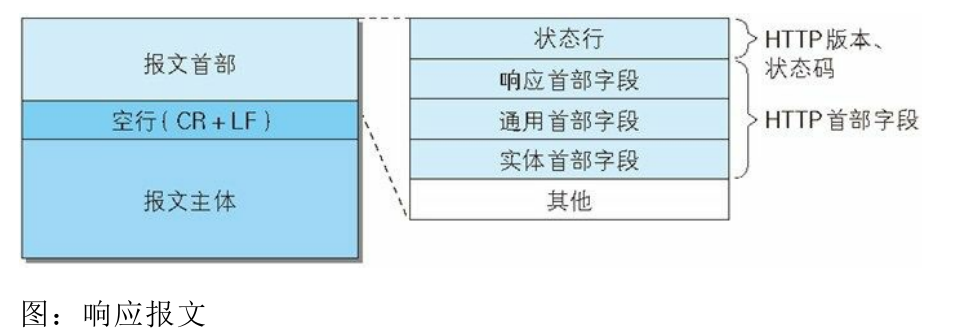
报文首部由 请求行 请求首部字段 其他组成

空行

报文主体



http响应报文

在响应中，HTTP 报文由 HTTP 版本、状态码（数字和原因短语）、HTTP 首部字段 3 部分构成。

首部字段是为了给浏览器和服务器提供报文主体大小、所使用的语言、认证信息等内容。

由首部字段名和字段值构成的，中间用冒号“:” 分隔。

首部字段4 种类型。

通用首部字段 请求报文和响应报文两方都会使用的首部。

请求首部字段 向服务器端发送请求报文时使用的首部。补充了请求的附加

内容、客户端信息、响应内容相关优先级等信息。

响应首部字段 向客户端返回响应报文时使用的首部。补充了响应的附加

内容，也会要求客户端附加额外的内容信息。

实体首部字段 请求报文和响应报文的实体部分使用的首部。补充了资源内容更新时间等与实体有关的信息。

## GET 还是 POST？

GET 比 POST 更简单更快，可用于大多数情况下。

不过，请在以下情况始终使用 POST：

* 缓存文件不是选项（更新服务器上的文件或数据库）
* 向服务器发送大量数据（POST 无大小限制）
* 发送用户输入（可包含未知字符），POST 比 GET 更强大更安全

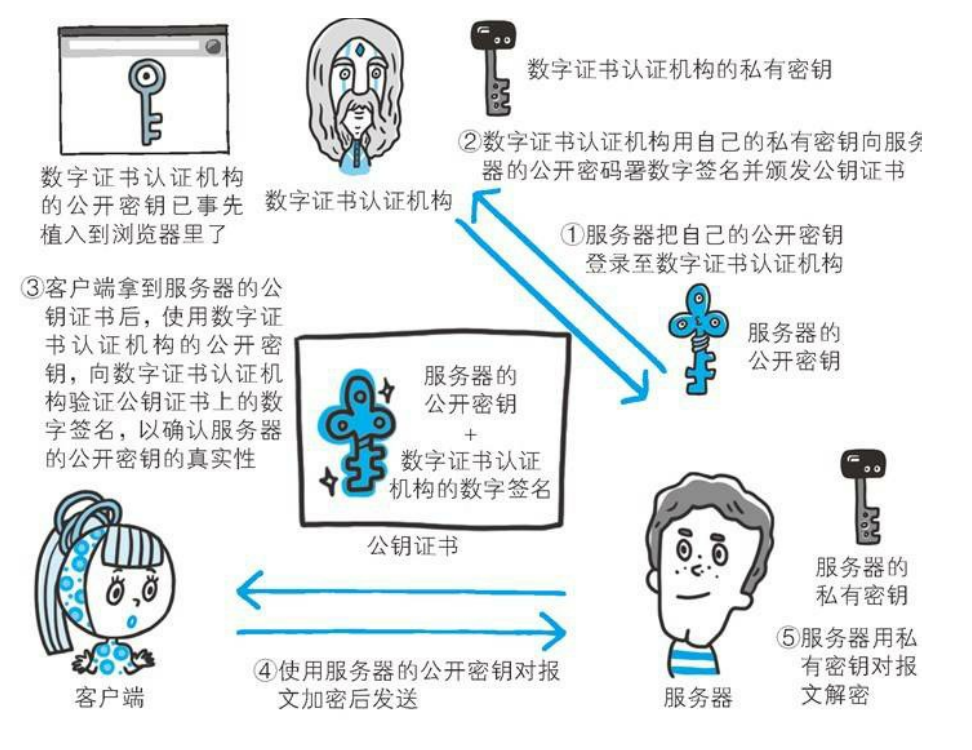
公开密匙 即是 请求方通过公开密匙对明文进行加密，服务器方通过私有密匙解密。

共享密匙是，密匙也通过明文传输，两方公用同一密匙对明文加密和解密，但是密匙本身也可能被监听

如何证明收到的公开密钥就是原本预想的那台服务器发行的公开密钥？

可以使用由数字证书认证机构颁发的公开密钥证书。

数字证书认证机构处于客户端与服务器双方都可信赖的第三方机构的立场上。



HTTPS 中还可以使用客户端证书。以客户端证书进行客户端认证，证明服务器正在通信的对方始终是预料之内的客户端，其作用跟服务器证书如出一辙。

HTTPS 比 HTTP更安全，添加了加密，认证和完整性确认的功能。但是因为SSL本身会带来更多的网络流量开销，以及非对称加密及解密涉及大量运算，因此会导致速度更慢。

非敏感信息则使用 HTTP 通信 。包含个人信息等敏感数据时，才利用 HTTPS 加密通信

在进行加密处理时，并非对所有内容都进行加密处理，而是仅在那些需要信息隐藏时才会加密，以节约资源。

要进行 HTTPS 通信，证书是必不可少的。 威瑞信（VeriSign）

Session 管理及 Cookie 应用

基于表单认证本身是通过服务器端的 Web 应用，将客户端发送过来

的用户 ID 和密码与之前登录过的信息做匹配来进行认证的。

会使用 Cookie 来管理 Session，以弥补 HTTP 协议中不存在的状态管理功能。

服务器会发放用以识别用户的 Session ID

服务器把用户的认证状态与Session ID 绑定后记录在服务器端。

（有点像，我不知道你是谁，你在我这儿存50元，我给你一个信物，下次你来找我就拿着这个东西，我就知道你存了多少钱）

cookie与session的区别有：cookie以文本格式存储在浏览器上，存储量有限；而会话存储在服务端，可以无限量存储多个变量并且比cookie更安全

**session与cookie的区别**

（1）Cookie以文本文件格式存储在浏览器中，而session存储在服务端它存储了限制数据量。它只允许4kb它没有在cookie中保存多个变量。

（2）cookie的存储限制了数据量，只允许4KB，而session是无限量的

（3）我们可以轻松访问cookie值但是我们无法轻松访问会话值，因此它更安全

（4）设置cookie时间可以使cookie过期。但是使用session-destory（），我们将会销毁会话。

总结：如果我们需要经常登录一个站点时，最好用cookie来保存信息，要不然每次登陆都特别麻烦，如果对于需要安全性高的站点以及控制数据的能力时需要用会话效果更佳，当然我们也可以结合两者，使网站按照我们的想法进行运行

拥塞控制和滑动窗口的区别

拥塞控制的四个步骤： 慢开始，拥塞控制，快重传，快恢复。

糊涂窗口综合征 （典型的有telnet应用）

当发送端应用进程产生数据很慢、或接收端应用进程处理接收缓冲区数据很慢，或二者兼而有之；就会使应用进程间传送的报文段很小，特别是有效载荷很小。 极端情况下，有效载荷可能只有1个字节；而传输开销有40字节(20字节的IP头+20字节的TCP头) 这种现象就叫糊涂窗口综合症

解决方法Nagle找到了一个很好的解决方法，发明了Nagle算法。

https://www.cnblogs.com/zhaoyl/archive/2012/09/20/2695799.html

NAGLE算法 ，只是从发送端解决了SWS，客户端可以在接收方要做的是不要通告缓冲空间的很小增长，不通知小窗口，除非缓冲区空间有显著的增长

1. 对于从应用程序中接收到的第一块数据【没有任何等待ACK的包】，立即发送，一个字节的数据也需要发送出去

2. 对于后面的数据，协议栈会进行累计并等待，等待时间是一个RTT,即下个ACK来到时。或者收到一个接收端发出一个ACK，或者累计到一个最大报文段，然后再发送数据

1） Clark解决方法

      Clark解决方法是只要有数据到达就发送确认，但宣布的窗口大小为零，直到或者缓存空间已能放入具有最大长度的报文段，或者缓存空间的一半已经空了。

**延迟ACK，即客户端收到的数据很少的时候，先不马上返回收到，或者把Windows设为0.等收到适量大小再确认**

TCP粘包与拆包 ，怎么防止粘包

由于UDP有消息保护边界（头部中的 标识 字段），粘包拆包问题只发生在TCP协议中

拆包，发送的数据包长达>MSS的时候将发生拆包。

粘包，发送端应用程序写入数据小于套接字缓冲区大小，网卡将应用多次写入的数据发送到网络上，这将会发生粘包。接收端接收方法不及时读取套接字缓冲区数据，这将发生粘包。

如何解决：

1、发送端给每个数据包添加包首部，首部中应该至少包含数据包的长度，这样接收端在接收到数据后，通过读取包首部的长度字段，便知道每一个数据包的实际长度了。

2、发送端将每个数据包封装为固定长度（不够的可以通过补0填充），这样接收端每次从接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。

3、可以在数据包之间设置边界，如添加特殊符号，这样，接收端通过这个边界就可以将不同的数据包拆分开。

**MTU MSS**

MTU: Maxitum Transmission Unit 最大传输单元

MSS: Maxitum Segment Size 最大分段大小

为什么要有MSS呢，因为MSS太大，MSS+TCP头+IP头就大，链路层有MTU即链路层一次最大能接受的报文长度，一般为1500，如果大于这个数，IP数据报会被分片，而分片的数据报在终点会重新装配，如果有丢失，需要全部重传，因此应该尽量不让IP分片，就需要在TCP数据段对大小做限制。

# 拒绝服务类攻击（DoS）

<https://blog.csdn.net/m0_37411189/article/details/78022536>

服务器不会在每次接收到SYN请求的时候立即与客户端建立连接，而是为连接请求分配内存空间，建立会话，并放到一个等待队列之中。如果这个队列已满，就不再接受新的请求，即新的请求将被直接丢弃，这就是拒绝服务。

如果服务器接收到一个RST置位的信息，就判断这是一个有错误的数据段，会根据客户端IP，把对应的连接从缓冲队列中清除掉。这对IP欺骗有影响，也可以作为DoS攻击。

通过了解TCP协议和连接过程，我们要对Server进行DoS攻击，只需要抓住两点：

a.让服务器的缓冲区满，不接受新的请求；

b.使用IP欺骗，使得正常连接被复位，从而影响合法用户。

SYN FLOOD

利用服务器的连接缓冲区，和特殊程序，设置TCP的Header，向服务器不断成倍发送具有只有SYN标志的TCP连接请求。当服务器接收之后，判断为均是没有建立起来的连接请求，于是给这些请求建立会话，排到缓冲区队列中。

直到你发送的SYN请求超过了服务器的缓冲区，服务器便不再接受其他的合法请求了。你可以持续的进行SYN请求，从而致使缓冲区里都是你发送的SYN请求。

IP欺骗DoS攻击

这种攻击利用RST位来实现。假设有一个合法用户（1.1.1.1）与服务器建立了正常的连接，攻击者构造攻击TCP数据，伪装自己的IP也是1.1.1.1，并向服务器发送一个带有RST位的数据段。此时服务器收到该数据段后，便认为从1.1.1.1发送的请求连接有错，便从缓冲区中将该连接清除，从而导致合法用户与服务器断开了连接。如果合法用户要继续传输数据，就得重新发送合法的连接请求。

在进行IP欺骗DoS攻击时，通常伪造大量的合法用户IP进行攻击，向目标发送RST数据，使得服务器拒绝对合法用户服务。

代理服务器可以利用缓存减小网络流量…

[**Cookie/Session机制详解(非原创)**](https://www.cnblogs.com/liuwei0824/p/7699632.html)

<https://www.cnblogs.com/liuwei0824/p/7699632.html>

https://www.cnblogs.com/xxtalhr/p/9053906.html