目录

[**网络分层** 3](#_Toc48205136)

[**TCP和UDP** 5](#_Toc48205137)

[**tcp协议报文头中的字段(这里又是关键点,在讲这个问题的时候,不能光讲头部哪些字段,还要结合字段讲讲作用,然后就顺带把整个TCP的可靠传输原理,以及相关的拥塞控制等全讲了.这就是主动出击的技巧)** 5](#_Toc48205138)

[**tcp三次握手** 5](#_Toc48205139)

[**listen的真正目的** 6](#_Toc48205140)

[**若客户端发起握手请求，服务器无法立刻建立连接怎么办？** 6](#_Toc48205141)

[**第二次握手中的syn的作用** 6](#_Toc48205142)

[**收到服务端的确认后，为什么客户端还需要第三次握手？** 6](#_Toc48205143)

[**netstate可以看到tcp的哪些状态？** 6](#_Toc48205144)

[**tcp四次挥手** 6](#_Toc48205145)

[**为什么要挥四次手？** 6](#_Toc48205146)

[**time\_wait状态产生的原因** 6](#_Toc48205147)

[**如果网络中出现了大量的time\_wait状态，会出现什么问题，如何解决？** 6](#_Toc48205148)

[**关闭连接时，最后一个ACK报文段丢失怎么办？** 7](#_Toc48205149)

[**tcp的流量控制** 7](#_Toc48205150)

[**若滑动窗口过大或过小会产生怎样的影响？** 7](#_Toc48205151)

[**tcp流量控制过程中，传输速率的问题** 7](#_Toc48205152)

[**tcp的拥塞控制** 8](#_Toc48205153)

[**tcp如何保证可靠传输** 8](#_Toc48205154)

[**tcp建立连接后如何保持连接（即如何检测连接是否断开）** 9](#_Toc48205155)

[**tcp黏包是什么，如何解决** 9](#_Toc48205156)

[**tcp三次握手有哪些漏洞** 10](#_Toc48205157)

[**udp协议报文头中的字段** 10](#_Toc48205158)

[**tcp和udp的区别、各自应用场景** 10](#_Toc48205159)

[**如何使用udp实现tcp(如何提高udp的可靠性)?** 11](#_Toc48205160)

[**为什么tcp比udp更安全，但udp还是很多用？** 11](#_Toc48205161)

[**HTTP和HTTPs** 11](#_Toc48205162)

[**输入url到响应页面的过程** 11](#_Toc48205163)

[**http的特点** 12](#_Toc48205164)

[**http为什么是无状态的，如何保持http的状态** 12](#_Toc48205165)

[**http请求报文和响应报文结构** 12](#_Toc48205166)

[**http报文中的字段** 12](#_Toc48205167)

[**http状态码的含义** 13](#_Toc48205168)

[**常用http的方法，get和post的区别** 14](#_Toc48205169)

[**url的长度和编码方式** 14](#_Toc48205170)

[**http1.0v.s.http1.1** 14](#_Toc48205171)

[**http2.0相比http1.x的新特性** 15](#_Toc48205172)

[**http的几个问题** 15](#_Toc48205173)

[**现代浏览器在与服务器建立了一个TCP连接后是否会在一个HTTP请求完成后断开？什么情况下会断开？** 15](#_Toc48205174)

[**一个TCP连接可以对应几个HTTP请求？** 16](#_Toc48205175)

[**一个TCP连接中请求可以一起发送吗？** 16](#_Toc48205176)

[**为什么有的时候刷新页面不需要重新建立SSL连接？** 16](#_Toc48205177)

[**浏览器对同一Host建立TCP连接到数量有没有限制？** 16](#_Toc48205178)

[**收到的HTML如果包含几十个图片标签，这些图片是以什么方式、什么顺序、建立了多少连接、使用什么协议被下载下来的呢？** 16](#_Toc48205179)

[**https的特点和握手过程** 16](#_Toc48205180)

[**如何保证公钥不被篡改？** 17](#_Toc48205181)

[**公钥加密计算量太大，如何减少耗用的时间？** 17](#_Toc48205182)

[**为什么服务端要发送证书给客户端** 17](#_Toc48205183)

[**什么是数字签名和数字证书** 17](#_Toc48205184)

[**httpv.s.https** 17](#_Toc48205185)

[**如何将网站从http切换到https？** 18](#_Toc48205186)

[**零碎的问题** 18](#_Toc48205187)

[**32位ip地址** 18](#_Toc48205188)

[**分类** 18](#_Toc48205189)

[**比较特殊的ip地址（net\_id+host\_id）** 18](#_Toc48205190)

[**ddos攻击的原理，如何防止** 19](#_Toc48205191)

[**攻击网站的方法和原理** 19](#_Toc48205192)

[**路由器和交换机的区别** 19](#_Toc48205193)

[**一些名词** 19](#_Toc48205194)

**网络分层**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TCP/IP模型  （实际） | OSI模型  （理论） | 作用 | 核心协议 | 数据形式 |
| 应用层  （应用程序之间的通信） | 应用层 | 为应用程序提供网络服务。 | ftp  http  自定义 | 报文（message） |
| 表示层 | 可确保一个系统的应用层发送的信息可以被另一个系统的应用层读取。将不同系统的编码转化成一种更为通用的格式。 |
| 会话层 | 实现不同系统间的发起会话会话请求和会话接受请求。 |
| 传输层  （进程之间的通信） | 传输层 | 1）网络中的传输协议和端口号的定义。2）主要是将数据分段和传输，到达目标地址后在进行重组。 | 传输控制协议TCP、用户数据协议UDP  *p.s.*端口:系统默认占用的端口<=5000（e.g.WWW端口号80，https端口号443，http80，数据库3306，DNS53）,用户可使用的端口大小在(5000,2\*\*16-1]区间内。在一系统中，端口可以唯一的标志进程。 | 报文段(segment) |
| 网络层  (主机之间的通信) | 网络层 | 在两个主机系统之间提供连接和路径选择。 | 网络地址(IP)协议：在网络中唯一的标志一台主机。  *p.s.*可以通过IP确定计算机所在的子网络，是由管理员分配的。 | 数据报(datagram) |
| 链路层  （设备之间的通信） | 数据链路层 | 1）定义如何将数据划分成帧，如何控制对物理介质的访问。2）提供错误检测和纠正，以保证数据的可靠传输。 | 地址解析协议ARP：将IP解析为mac地址。  PPP协议  *p.s.*mac地址（物理地址）可以唯一的标志网卡，是绑定在网卡上的。通过mac地址可以将数据包发送到子网络的目标网卡中。 | 帧(frame) |
| 物理层 | 1）定义物理设备的标准，比如网线的接口类型、光纤的接口类型、传输速率等等。2）传输比特流。 | 比特(bit) |

p.s.TCP/IP:TCP/IP协议是一个大家族，不仅仅只有TCP和IP协议，它还包括其它的协议。

数据通信：

封装：应用层——传输层——网络层——链路层。没有经过封装的数据，不能在网络环境中传递。

解封装：链路层——网络层——传输层——应用层。

**TCP和UDP**

**tcp协议报文头中的字段(这里又是关键点,在讲这个问题的时候,不能光讲头部哪些字段,还要结合字段讲讲作用,然后就顺带把整个TCP的可靠传输原理,以及相关的拥塞控制等全讲了.这就是主动出击的技巧)**



**序号**：占4字节，TCP是面向字节流的，在一个TCP连接中传送的字节流的每一个字节都按顺序编号，很简单的例子：比如传送1000个字节的字节流，其中每个字节都是有编号的，比如说是0-999，这里的序号是指发送的时候本报文段的第一个字节的序。

**确认号**：占4字节，由目的端发出，表示确认号之前的数据段都收到了。e.g.确认号为X，则表示前X-1个数据段都收到了,期望收到下一个报文段的SYN为X。

**首部长度：**占4个位，也称为数据偏移，代表本报文的数据起始处距离本报文段的起始处有多远，因为TCP首部中存在可选字段，所以首部长度不固定，所以这个字段是必要的，可以明确指出TCP报文的首部长度。因为其是按4字节为单位的，所以4位二进制数能表示的最大数是15，也就是首部最大长度是60字节。

**保留**：占6位，目前未使用，置0。

**码元比特**：其实是有6个控制位，其中包括：

URG，1时，表明紧急指针字段有效，应尽快传送。

ACK,为1时，确认号才有效，为0,为保证数据的完整性，发送端需重传数据。TCP规定，连接建立后所有传送的报文段ACK=1.

PSH，为1时，尽快交付给应用进程，无需等到整个缓存都填满再向上交付。

RST，为1时，表明TCP连接中出现了严重差错，必须释放连接。

SYN，SYN=1，ACK=0,l连接请求报文；SYN=1,ACK=1,连接响应报文。

FIN，为1时，表明该报文的发送方数据传输完毕，并请求释放连接。

**窗口：**占16个字节，指接收端此时还能接收多少数据，因为接收方接收数据的缓冲区的大小是有限的。此处的窗口值是作为发送方设置其发送窗口大小的依据。

**校验和：**2个字节，用来校验首部和数据部分。紧急指针：占2个字节，只有在URG位=1的时候才有效。

**紧急指针：**16位，本报文断种紧急数据共有多少字节，紧急数据会放在本报文段数据的最前面。

**tcp三次握手**

第一次：客户端发送SYN，进入SYN\_SEND状态。

第二次：服务端收到SYN，并发送SYN和ACK，由LISTEN状态变为SYN\_RECVD状态。

第三次：客户端收到SYN和ACK，发送ACK报文，进入到ESTABLISHED状态。（服务端收到后也进入ESTABLISHED状态）

**listen的真正目的**

listen的函数为侦听端口创建两个队列：未完成队列(SYN\_RCV状态)和已完成队列。如果不调用listen，则客户端过来的SYN请求无法入队接受进一步的处理。因此，listen是服务器的必须过程。

**若客户端发起握手请求，服务器无法立刻建立连接怎么办？**

回复RST报文

**第二次握手中的syn的作用**

丢弃重复的失效的报文。

客户端发送SYN给服务端，若长时间没有收到服务端的回复，会超时重传SYN。此时网络中就会有两个SYN，后到服务器的SYN是无效的，服务器为了判断收到的SYN的有效性，会发送SYN给客户端查询该SYN报文的有效性。

**收到服务端的确认后，为什么客户端还需要第三次握手？**

为了防止失效的请求报文段突然又传到服务端，因而产生错误。

e.g.客户端发送到服务端的SYN报文在网络中长时间滞留，直到连接释放后才到达服务端，若没有第三次握手，则服务端就会建立连接，并一直监听客户端，造成服务端资源的浪费。

**netstate可以看到tcp的哪些状态？**

查看tcp的状态值

**tcp四次挥手**

第一次：客户端传送给服务端的数据传输完成后，发送FIN，进入FIN\_WAIT1状态。

第二次：服务端收到FIN，若有数据还需传送给客户端，则继续传输数据，传输完成后发送ACK，进入CLOSE\_WAIT状态。

第三次：客户端收到ACK，进入FIN\_WAIT2状态。服务端传输FIN，进入LAST\_ACK状态。

第四次：客户端收到FIN，发送ACK。服务端收到ACK，直接进入CLOSED状态。等待2MSL(最大报文生存时间)后进入CLOSED状态。

**为什么要挥四次手？**

保证数据能够完成传输。一方发送FIN，只表示该方没有数据发送给另一方了，但另一方可能仍有数据发送给该方。要等到两方都没有数据发送给对方，才可以断开连接。

**time\_wait状态产生的原因**

1）可靠地实现TCP连接的关闭

我们不能保证服务端一定能够收到最后的ACK报文。若客户端发送ACK后直接进入CLOSED状态，当ACK报文丢失时，服务端长时间没有收到客户端回复，会重传FIN报文，此时会收到一个RST，服务端会认为有错误发生，不能正常的关闭连接，造成服务端资源的浪费。而让客户端处于TIME\_WAIT就不会出现这种情况，在再次收到服务端的FIN后，客户端会重传ACK，并重新计时。

2）保证本连接内的重复报文在网络中消逝，避免影响下一次的连接。

**如果网络中出现了大量的time\_wait状态，会出现什么问题，如何解决？**

如果系统中有很多socket处于TIME\_WAIT状态，当需要创建新的socket连接的时候可能会受到影响，这也会影响到系统的扩展性。之所以TIME\_WAIT能够影响系统的扩展性是因为在一个TCP连接中，一个Socket如果关闭的话，它将保持TIME\_WAIT状态大约1-4分钟。如果很多连接快速的打开和关闭的话，系统中处于TIME\_WAIT状态的socket将会积累很多，由于本地端口数量的限制，同一时间只有有限数量的socket连接可以建立，如果太多的socket处于TIME\_WAIT状态，你会发现，由于用于新建连接的本地端口太缺乏，将会很难再建立新的对外连接。

1)改为长连接2）增加可用端口范围。。。。

**关闭连接时，最后一个ACK报文段丢失怎么办？**

如果最后一个ACK丢失的话，TCP就会认为它的FIN丢失，进行重发FIN。在客户端收到FIN后，就会设置一个2MSL计时器，2MSL计时器可以使客户等待足够长的时间，使得在ACK丢失的情况下，可以等到下一个FIN的到来。如果在TIME\_WAIT状态汇总有一个新的FIN到达了，客户就会发送一个新的ACK，并重新设置2MSL计时器。

**tcp的流量控制**

TCP是全双工的，客户端和服务器均可作为发送方或接收方，我们现在假设一个发送方向接收方发送数据的场景来讲解流量控制。首先我们的接收方有一块接收缓存，当数据来到时会先把数据放到缓存中，上层应用等缓存中有数据时就会到缓存中取数据。假如发送方没有限制地不断地向接收方发送数据，接收方的应用程序又没有及时把接收缓存中的数据读走，就会出现缓存溢出，数据丢失的现象，为了解决这个问题，我们引入流量控制窗口。

假设应用程序最后读走的数据序号是lastByteRead，接收缓存中接收到的最后一个数据序号是lastByteRcv，接收缓存的大小为RcvSize，那么必须要满足lastByteRcv-lastByteRead<=RcvSize才能保证接收缓存不会溢出，所以我们定义流量窗口为接收缓存剩余的空间，也就是Rcv=RcvSize-(lastByteRcv-lastByteRead)。只要接收方在响应ACK的时候把这个窗口的值带给发送方，发送方就能知道接收方的接收缓存还有多大的空间，进而设置滑动窗口的大小。

总结：流量控制就是让发送方的发送速率不要太快，要让接收方来得及接收。利用滑动窗口机制可以很方便地在TCP连接上实现对发送方的流量控制。原理这就是运用TCP报文段中的窗口大小字段来控制，发送方的发送窗口不可以大于接收方发回的窗口大小。

**若滑动窗口过大或过小会产生怎样的影响？**

如果滑动窗口过小，极端的情况就是停止等待协议，发一个报文等一个ACK，会造成通信效率下降。

如果滑动窗口过大，网络容易拥塞，容易造成接收端的缓存不够而溢出，容易产生丢包现象，则需要多次发送重复的数据，耗费了网络带宽。

**tcp流量控制过程中，传输速率的问题**

1.Nagle算法

当有数据要发送时，先不立即发送，而是稍微等一小会，看看在这一小段时间内，还有没有其他需要发送的消息。当等过这一小会以后，再把要发送的数据一次性都发出去。这样就可以有效的减少包头的发送次数，减少使用的网络带宽。

Nagle主要职责是数据的累积，实际上有三个门槛：

1)缓冲区中的字节数达到了一定量(超过阀值MSS,一般是达到窗口大小的一半及以上)；2)等待了一定的时间(一般是等待200ms）；

3）紧急数据发送

2.糊涂窗口综合征

若TCP接收方缓存已满，而交互式的应用进程一次只从接收方缓存中读取一个字节，然后向发送方发送确认，并将窗口设置为1个字节。这样网络的效率极低。

可以让结合搜房等待一段时间，使接收方缓存有足够空间容纳一个最长的报文段或接收方缓存有一半空闲的空间。

这两种方法配合使用，可以更好的保证tcp的传输速率。

**tcp的拥塞控制**

**拥塞**：对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就会变坏。

**拥塞控制：**防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。注意，拥塞控制和流量控制不同，前者是一个全局性的过程，而后者指点对点通信量的控制。

**慢启动**：不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说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：**快重传要求接收方在收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，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：**快重传配合使用的还有快恢复算法，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门限设置成最大窗口的一半，但是接下去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，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。

cwnd<ssthresh,慢开始；cwnd>=ssthresh,拥塞避免。

**tcp如何保证可靠传输**

0、在传递数据之前，会有三次握手来建立连接。

1、分割数据块：应用数据被分割成TCP认为最适合发送的数据块（按字节编号，合理分片）。这和UDP完全不同，应用程序产生的数据报长度将保持不变。(将数据截断为合理的长度)

2、超时重传：当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。

3、确认机制：当TCP收到发自TCP连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒。(对于收到的请求，给出确认响应)(之所以推迟，可能是要对包做完整校验)。

4、校验机制：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。(校验出包有错，丢弃报文段，不给出响应，TCP发送数据端，超时时会重发数据)

5、排序：既然TCP报文段作为IP数据报来传输，而IP数据报的到达可能会失序，因此TCP报文段的到达也可能会失序。如果必要，TCP将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。(对失序数据进行重新排序，然后才交给应用层)

6、丢弃重复数据：既然IP数据报会发生重复，TCP的接收端必须丢弃重复的数据。(对于重复数据，能够丢弃重复数据)

7、流量控制：TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。(TCP可以进行流量控制，防止较快主机致使较慢主机的缓冲区溢出)TCP使用的流量控制协议是可变大小的滑动窗口协议。

8、拥塞控制：当网络拥塞时，减少数据的发送。

**tcp建立连接后如何保持连接（即如何检测连接是否断开）**

有两种技术可以运用。一种是由TCP协议层实现的Keepalive机制，另一种是由应用层自己实现的HeartBeat心跳包。

1.在TCP中有一个Keep-alive的机制可以检测死连接，原理很简单，当连接闲置一定的时间（参数值可以设置，默认是2小时）之后，TCP协议会向对方发一个keepalive探针包（包内没有数据），对方在收到包以后，如果连接一切正常，应该回复一个ACK；如果连接出现错误了（例如对方重启了，连接状态丢失），则应当回复一个RST；如果对方没有回复，那么，服务器每隔一定的时间（参数值可以设置）再发送keepalive探针包，如果连续多个包（参数值可以设置）都被无视了，说明连接被断开了。

2.心跳包之所以叫心跳包是因为：它像心跳一样每隔固定时间发一次，以此来告诉服务器，这个客户端还活着。事实上这是为了保持长连接，至于这个包的内容，是没有什么特别规定的，不过一般都是很小的包，或者只包含包头的一个空包。由应用程序自己发送心跳包来检测连接的健康性。客户端可以在一个Timer中或低级别的线程中定时向服务器发送一个短小精悍的包，并等待服务器的回应。客户端程序在一定时间内没有收到服务器回应即认为连接不可用，同样，服务器在一定时间内没有收到客户端的心跳包则认为客户端已经掉线。

**tcp黏包是什么，如何解决**

**TCP报文粘连**：本来发送的是多个TCP报文，但是在接收端收到的却是一个报文，把多个报文合成了一个报文。

**TCP报文粘连的原因：**"粘包"可发生在发送端，也可发生在接收端。在流传输中出现，UDP不会出现粘包，因为它有消息边界（两段数据间是有界限的）。

1）由Nagle算法造成的发送端的粘包

2）接收端接收不及时造成的接收端粘包

TCP会把接收到的数据存在自己的缓冲区中,然后通知应用层取数据.当应用层由于某些原因不能及时的把TCP的数据取出来,就会造成TCP缓冲区中存放了几段数据，产生报文粘连的现象。

**TCP报文粘连的解决方法：**

1.关闭Nagle算法。在scoket选项中，TCP\_NODELAY表示是否使用Nagle算法。

2.接收端尽可能快速的从缓冲区读数据。

3.可以在发送的数据中，添加一个表示数据的开头和结尾的字符，在收到消息后，通过这些字符来处理报文粘连。

**为什么UDP不会黏包**

tcp是基于字节流的传输，不认为消息是一条一条的，无保护边界，而udp有保护边界，即把数据当做一条独立的消息在网上传输，接收端一次只能接受一条独立的消息。

**tcp三次握手有哪些漏洞**

1.TCP三次握手可能会出现SYNFlood攻击。

2.TCP三次握手可能会出现Land攻击。

3.ConnectionFlood攻击。

**udp协议报文头中的字段**

固定8个字节，源端口，目的端口，校验和，长度。

**tcp和udp的区别、各自应用场景**

1）连接

2）可靠性：TCP提供交付保证，若信息在传输时丢失，将会重发。UDP不提供任何交付保证。

3）有序性：消息到达另一端可能是无序的，TCP为排好序，UDP不会。

4）速度：TCP慢，适合传输大量数据；UDP快，适合传输少量数据。

5）量级：TCP重量级协议，报头包含序列号，ACK号，数据偏移量、保留、控制位、窗口、紧急指针、可选项、填充项、校验位、源端口和目的端口，至少20字节。UDP轻量级，报头只包含长度、源端口号、目的端口、校验和，固定8字节。

6）流量控制和拥塞控制：TCP有，UDP无。

7）TCP面向字节流，UDP面向报文。

8）TCP只能单播，不能发送广播和组播；UDP皆可。

**TCP应用场景：**效率要求相对低，但对准确性要求相对高的场景。因为传输中需要对数据确认、重发、排序等操作，相比之下效率没有UDP高。举几个例子：文件传输、邮件传输、远程登录。

**UDP应用场景**：效率要求相对高，对准确性要求相对低的场景。举几个例子：QQ聊天、QQ视频、网络语音电话（即时通讯，速度要求高，但是出现偶尔断续不是太大问题，并且此处完全不可以使用重发机制）、广播通信（广播、多播）。

**关于QQ**

1)不管UDP还是TCP，最终登陆成功之后，QQ都会有一个TCP连接来保持在线状态。这个TCP连接的远程端口一般是80，采用UDP方式登陆的时候，端口是8000。

2)QQ采用的通信协议以UDP为主，辅以TCP协议。由于QQ的服务器设计容量是海量级的应用，一台服务器要同时容纳十几万的并发连接，因此服务器端只有采用UDP协议与客户端进行通讯才能保证这种超大规模的服务。

3)QQ客户端之间的消息传送也采用了UDP模式，因为国内的网络环境非常复杂，而且很多用户采用的方式是通过代理服务器共享一条线路上网的方式，在这些复杂的情况下，客户端之间能彼此建立起来TCP连接的概率较小，严重影响传送信息的效率。而UDP包能够穿透大部分的代理服务器，因此QQ选择了UDP作为客户之间的主要通信协议。

4)采用UDP协议，通过服务器中转方式。因此，现在的IP侦探在你仅仅跟对方发送聊天消息的时候是无法获取到IP的。大家都知道，UDP协议是不可靠协议，它只管发送，不管对方是否收到的，但它的传输很高效。但是，作为聊天软件，怎么可以采用这样的不可靠方式来传输消息呢？于是，腾讯采用了上层协议来保证可靠传输：如果客户端使用UDP协议发出消息后，服务器收到该包，需要使用UDP协议发回一个应答包。如此来保证消息可以无遗漏传输。之所以会发生在客户端明明看到“消息发送失败”但对方又收到了这个消息的情况，就是因为客户端发出的消息服务器已经收到并转发成功，但客户端由于网络原因没有收到服务器的应答包引起的。

**udp丢包的原因？**

1)发送频率过高导致丢包

原因就是UDP的SendTo不会造成线程阻塞，也就是说，UDP的SentTo不会像TCP中的SendTo那样，直到数据完全发送才会return回调用函数，它不保证当执行下一条语句时数据是否被发送。🡪要保证程序执行后马上开始监听（如果数据包不确定什么时候发过来的话），其次，要在收到一个数据包后最短的时间内重新回到监听状态，其间要尽量避免复杂的操作（比较好的解决办法是使用多线程回调机制）。

2)报文过大丢包超过接收者的缓冲，导致丢包

3)局域网内不丢包，公网上丢包

要么减小流量，要么换tcp协议传输，要么做丢包重传的工作。

**如何使用udp实现tcp(如何提高udp的可靠性)?**

TCP为了保证数据的可靠传输，会切分数据、确认机制、超时重传、校验和、对失序数据重新排序、丢弃重复数据和流量控制。

如果要通过UDP传输数据，但却要保证可靠性的话，要通过应用层来实现的。

可以参照TCP的方式，只是实现不在传输层，而在应用层。

实现确认机制、重传机制、窗口确认机制。

如果你不利用Linux协议栈以及上层socket机制，自己通过抓包和发包的方式去实现可靠性传输，那么必须实现如下功能：

发送：包的分片、包确认、包的重发。

接收：包的序号确认、包的调序。

目前有如下开源程序利用udp实现了可靠的数据传输。分别为RUDP、RTP、UDT。

**为什么tcp比udp更安全，但udp还是很多用？**

1.无需建立连接（减少延迟）；

2.无需维护连接状态；

3.头部开销小，一个TCP数据报的报头大小最少是20字节，UDP数据报的报头固定是8个字节；

4.应用层能更好地控制要发送的数据和发送时间。UDP没有拥塞控制，因此网络中的拥塞不会影响主机的发送频率。某些实时应用要求以稳定的速度发送数据，可以容忍一些数据的丢失，但不允许有较大的延迟，而UDP正好满足这些应用的需求。

**HTTP和HTTPs**

**输入url到响应页面的过程**

一、解析URL：浏览器首先会对输入的URL进行检查，如果不合法，会把输入的文字传给默认的搜索引擎--如google、baidu；通过了URL的验证，那么可以解析得到协议（http以及https）、域名、资源等信息；

二、DNS查询：浏览器会先检查域名信息是否在缓存中；再检查域名是否在本地Hosts文件中；如果还不在，那么浏览器就会向DNS服务器发送一个查询的请求，获得目标服务器的ip地址；

三、TCP封包及传输：浏览器获得了目标服务器的ip、端口，浏览器会调用库函数socket，生成一个TCP流套接字，也就是TCP封包；TCP封包完成之后，就可以传输数据了，浏览器和服务器通过TCP三次握手建立连接，后面就可以请求服务器资源了。

四、服务器接受请求并响应：http有很多请求方法，比如：GET/POST/PUT/DELETE等等，我们浏览器输入URL这种，是GET方法；服务器接受GET请求，服务器根据请求信息，获得相应的资源内容。

五、释放TCP连接：connection字段若为keep-alive模式，则保持连接，可以继续接受请求；若为close模式，则服务端主动释放断开连接，客户端被动关闭连接。

六、浏览器解析并渲染：浏览器从服务器拿到了想要的资源，如html页面，首先对html文档进行解析，生成DOM节点树，然后加载页面的外部资源，如css、js、img等；最后遍历DOM树，并计算每个节点的样式，最终完成渲染，变成我们的页面；

**http的特点**

http是基于TCP/IP关于数据在万维网中如何通信的应用层协议。

1）基于请求-响应的模式（请求必定由客户端发出，服务器端回复响应）

2）无连接(每次连接只处理一个请求）

3）无状态保存(http本身不对请求-响应的信息做保存，即不做持久化处理)

4）简单快速、灵活

5）通信使用明文、请求和响应不会对通信方进行确认、无法保护数据的完整性

**http为什么是无状态的，如何保持http的状态**

结合cookie/session,就可以保存和管理状态。

cookie和session都是用来记录客户端状态的机制。不同的是cookie是保存在客户端，而session是保存在服务端。两者最大的区别在于生存周期，一个是IE启动到IE关闭.(浏览器页面一关,session就消失了)，一个是预先设置的生存周期，或永久的保存于本地的文件(cookie)。

1）cookie是浏览器提供的一种缓存机制，它可以用于维持客户端与服务端之间的会话

2）session指的是维持客户端与服务端会话的一种机制，它可以通过cookie实现，也可以通过别的手段实现。

3）如果用cookie实现会话，那么会话会保存在客户端浏览器中

4）而session机制提供的会话是保存在服务端的

**session使用cookie实现的过程：**

1）首先用户在客户端浏览器发起登陆请求

2）登陆成功后，服务端会把用户信息保存在服务端，并返回一个唯一的session标识给客户端浏览器。

3）客户端浏览器会把这个唯一的session标识保存在起来

4）以后再次访问web应用时，客户端浏览器会把这个唯一的session标识带上，这样服务端就能根据这个唯一标识找到用户信息

**http请求报文和响应报文结构**

**http请求报文：**

1.请求行：请求方法（get/post）、url、版本号(http/1.1)

2.请求头部：首部字段：value（host:hackr.jpconnection:keep-alivecontent-type:application/x-www-form-urlencodedcontent-length:16）

1和2组成了报文首部（报文头）

3.实体主体：存储请求的资源信息（name=ueno&age=37）

**http响应报文：**

1.状态行版本号(http/1.1)、状态码(200)、状态码的原因短语(OK)

2.响应首部

3.实体主体：存放表单数据（如果太大，会把数据分割为多块，分块传送，浏览器逐步显示页面）。

**http报文中的字段**

**通用头部：**

　　Date:报文的发送时间

　　Connection:与连接的控制有关。

　　Transfer\_encoding:WEB服务器表明自己对本响应消息体（不是消息体里面的对象）作了怎样的编码，比如是否分块（chunked）

　　cache\_control:选用的缓存机制e.gno\_cache、public、private

**请求头部**：

　　Host:目标服务器的地址。域名或ip+端口号

　　User\_agent:客户端发起请求所使用的应用程序名

　　Referer:表明客户端是从那个页面发起请求的

　　Accept：可接受的消息类型

　　Accept\_charset、Accept\_encode、Accept\_language

　　if\_Match:如果请求的资源etag未发生改变，就进行请求操作

　　if\_Nonematch

　　if\_modified\_since:如果在规定时间后，资源发生了改变，就进行请求操作

　　if\_unmodified\_sincepost的时候用

**响应头部:**

　　Server:表明服务端使用的程序和版本

　　Accept\_Ranges:表明服务端是否接受请求部分对象。bytesorNone

　　Location:将客户端重定向到的URI

**实体头部：**

　　Allow:资源可支持的http方法（get、post。。。）

　　content\_type:实体主体的类型

　　content\_encodingcontent\_language

　　content\_length:实体主体的字节数

　　content\_range:实体主体的位置范围，一般用于发出部分请求时使用

　　Etag:一个URI的标志值，比如一个html文件，如果被修改了，其Etag也会别修改，所以，ETag的作用跟Last-Modified的作用差不多，主要供WEB服务器判断一个对象是否改变了。比如前一次请求某个html文件时，获得了其ETag，当这次又请求这个文件时，浏览器就会把先前获得ETag值发送给WEB服务器，然后WEB服务器会把这个ETag跟该文件的当前ETag进行对比，然后就知道这个文件有没有改变了。

　　Expires:WEB服务器表明该实体将在什么时候过期，对于过期了的对象，只有在跟WEB服务器验证了其有效性后，才能用来响应客户请求。是HTTP/1.0的头部。例如：Expires：Sat,23May200910:02:12GMT.http1.1做了修改，将时间改为寿命，避免客户端与服务器的时间不同步问题。

Last-Modified:WEB服务器认为对象的最后修改时间，比如文件的最后修改时间，动态页面的最后产生时间等等。例如：Last-Modified：Tue,06May200802:42:43GMT.

在chrome浏览器中，可通过以下操作查看请求头header:按F12进入开发者模式->按F5刷新页面->选择network->点击新出现的内容的任意一行，会出点header，即可点击查看。值得关注的一点是，请求头里面有一个**user-agent字段**，相当于告诉服务端，我是用什么发送的请求，通过判断该字段可以作为反扒机制的一种。

**http状态码的含义**

1XX:通知信息，如请求收到了，或者正在处理。

2XX:表示成功，请求正常处理完毕。e.g.200请求成功

3XX：重定向状态，需要附加操作以完成请求。

4XX：客户端错误，请求无法正常处理，如请求的资源不存在。e.g.404找不到资源

5XX:服务器内部错误，处理请求时出错。

**常用http的方法，get和post的区别**

put,delete,post,get,head. 🡪 增删查改 判断资源是否存在

1.post比get更加安全，get请求的参数直接暴露在url上，而url是对所有人可见的，不能用来传递敏感信息，而post的参数是放在实体主体中。

2.post比get传输的数据量更大且可以使用更多数据类型。因为get受限于浏览器地址栏的字符数量限制，且只能使用ASCII码，而post则不受限制。

3.post比get慢。GET产生一个TCP数据包；POST产生两个TCP数据包。对于GET方式的请求，浏览器会把httpheader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100continue，浏览器再发送data，服务器响应200ok（返回数据），相当于三次握手2个RTT和一个发送数据（实体主体）的RTT,一共三个RTT,而get无实体主体，数据都在url中所以只需三次握手2个RTT,get/post=2/3。

4.GET请求参数会被完整保留在浏览器历史记录或收藏为书签或缓存，而POST中的参数不会。

**url的长度和编码方式**

依据不同的浏览器版本而不同。

字符ascii码的十六进制表示。

**http长连接和短连接的原理、各自的优缺点**

connection字段进行控制。

p.s.

1）TCP的keepalive是检查当前TCP连接是否活着；HTTP的Keep-alive是要让一个TCP连接活久点。

2）HTTP协议的长连接和短连接，实质上是TCP协议的长连接和短连接。

**http长连接的优点：**

1.通过开启和关闭更少的TCP连接，节约CPU时间和内存。

2.通过减少TCP开启和关闭引起的包的数目，降低网络阻塞。

**http长连接的缺点**：服务器维护一个长连接会增加开销。

**http短连接的优点**：服务器不用为每个客户端连接分配内存来记忆大量状态，也不用在客户端失去连接时去清理内存，节省服务器端资源，以更高效地去处理业务。

**http短连接的缺点**：如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽。

**http1.0v.s.http1.1**

HTTP1.0最早在网页中使用是在1996年，那个时候只是使用一些较为简单的网页上和网络请求上，而HTTP1.1则在1999年才开始广泛应用于现在的各大浏览器网络请求中，同时HTTP1.1也是当前使用最为广泛的HTTP协议。主要区别主要体现在：

1)缓存处理

在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，

HTTP1.1则引入了更多的缓存控制策略例如Entitytag，If-Unmodified-Since,If-Match,If-None-Match等更多可供选择的缓存头来控制缓存策略。

2)带宽优化及网络连接的使用

HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能;

HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分，即返回码是206（PartialContent），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

3)错误通知的管理

在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

4)Host头处理

在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homedWebServers），并且它们共享一个IP地址。

HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400BadRequest）。

5)长连接

HTTP1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启Connection：keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。

**断点续传**

要实现断点续传的功能，通常都需要客户端记录下当前的下载进度，并在需要续传的时候通知服务端本次需要下载的内容片段。

HTTP1.1协议（RFC2616）中定义了断点续传相关的HTTP头 Range和Content-Range字段，一个最简单的断点续传实现大概如下：

1.客户端下载一个1024K的文件，已经下载了其中512K 2. 网络中断，客户端请求续传，因此需要在HTTP头中申明本次需要续传的片段： Range:bytes=512000- 这个头通知服务端从文件的512K位置开始传输文件 3. 服务端收到断点续传请求，从文件的512K位置开始传输，并且在HTTP头中增加： Content-Range:bytes 512000-/1024000 并且此时服务端返回的HTTP状态码应该是206，而不是200。

**http2.0相比http1.x的新特性**

1）新的二进制格式（BinaryFormat），HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。

2）多路复用（MultiPlexing），即连接共享，即每一个request都是是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的id将request再归属到各自不同的服务端请求里面。

3）header压缩，对前面提到过HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份headerfields表，既避免了重复header的传输，又减小了需要传输的大小。

4）服务端推送（serverpush），同SPDY一样，HTTP2.0也具有serverpush功能。

如果服务端收到客户端请求，能够‘预测’主请求的依赖资源，在响应主请求的同时，主动并发推送依赖资源至客户端。客户端解析主请求响应后，可以‘无限时’从本地缓存获取依赖资源，减少访问延时，提高访问体验，也加大了链路的并发能力。例如我的网页有一个sytle.css的请求，在客户端收到sytle.css数据的同时，服务端会将sytle.js的文件推送给客户端，当客户端再次尝试获取sytle.js时就可以直接从缓存中获取到，不用再发请求了。

**http的几个问题**

**现代浏览器在与服务器建立了一个TCP连接后是否会在一个HTTP请求完成后断开？什么情况下会断开？**

http1.0会断开，但是代价过大，现在用的比较多的http1.1会默认使用长连接，不会断开。

**一个TCP连接可以对应几个HTTP请求？**

由于不会断开，所以可以对应多个。

**一个TCP连接中请求可以一起发送吗？**

Http1.1单个连接在同一时刻只能处理一个请求，不能。

http2.0使用支持多路复用，共享连接，可以。

**为什么有的时候刷新页面不需要重新建立SSL连接？**

在第一个问题的讨论中已经有答案了，TCP连接有的时候会被浏览器和服务端维持一段时间。TCP不需要重新建立，SSL自然也会用之前的。

**浏览器对同一Host建立TCP连接到数量有没有限制？**

假设我们还处在HTTP/1.1时代，那个时候没有多路传输，当浏览器拿到一个有几十张图片的网页该怎么办呢？肯定不能只开一个TCP连接顺序下载，那样用户肯定等的很难受，但是如果每个图片都开一个TCP连接发HTTP请求，那电脑或者服务器都可能受不了，要是有1000张图片的话总不能开1000个TCP连接吧，你的电脑同意NAT也不一定会同意。

所以答案是：有。Chrome最多允许对同一个Host建立六个TCP连接。不同的浏览器有一些区别。

**收到的HTML如果包含几十个图片标签，这些图片是以什么方式、什么顺序、建立了多少连接、使用什么协议被下载下来的呢？**

如果图片都是HTTPS连接并且在同一个域名下，那么浏览器在SSL握手之后会和服务器商量能不能用HTTP2，如果能的话就使用Multiplexing功能在这个连接上进行多路传输。不过也未必会所有挂在这个域名的资源都会使用一个TCP连接去获取，但是可以确定的是Multiplexing很可能会被用到。如果发现用不了HTTP2呢？或者用不了HTTPS（现实中的HTTP2都是在HTTPS上实现的，所以也就是只能使用HTTP/1.1）。那浏览器就会在一个HOST上建立多个TCP连接，连接数量的最大限制取决于浏览器设置，这些连接会在空闲的时候被浏览器用来发送新的请求，如果所有的连接都正在发送请求呢？那其他的请求就只能等等了。

**https的特点和握手过程**

**https基于http协议，通过ssl或tls提供**

**加密处理数据(采用混合加密技术，中间者无法直接查看铭文内容)、**

**验证对方身份(通过证书认证客户端访问的是自己的服务器)**

**以及数据完整性保护(防止传输的内容被中间人冒充或篡改)。**

**SSL建立连接：**

1、客户端向服务端发送请求url，然后连接到服务端的443端口，发送的信息主要是随机值1和客户端支持的加密算法。

2、服务端接收到信息之后给予客户端响应握手信息，包括随机值2和匹配好的协商加密算法(该加密算法一定是客户端发送给服务端加密算法的子集）。

3、随即服务端给客户端发送第二个响应报文是数字证书。服务端必须要有一套数字证书，可以自己制作，也可以向组织（CA）申请。区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面，这套证书其实就是一对公钥和私钥。传送证书，就是公钥，只是包含了很多信息，如证书的颁发机构，过期时间、服务端的公钥，第三方证书认证机构(CA)的签名，服务端的域名信息等内容。

4、客户端解析证书，这部分工作是由客户端的TLS(传输层的安全协议)来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。如果证书没有问题，那么就生成一个随即值（预主秘钥）。

5、客户端认证证书通过之后，接下来是通过随机值1、随机值2和预主秘钥组装会话秘钥。然后通过证书的公钥加密会话秘钥。

6、传送加密信息，这部分传送的是用证书加密后的会话秘钥，目的就是让服务端使用秘钥解密得到随机值1、随机值2和预主秘钥。

7、服务端解密得到随机值1、随机值2和预主秘钥，然后组装会话秘钥，跟客户端会话秘钥相同。

8、客户端通过会话秘钥加密一条消息发送给服务端，主要验证服务端是否正常接受客户端加密的消息。

9、同样服务端也会通过会话秘钥加密一条消息回传给客户端，如果客户端能够正常接受的话表明SSL连接建立完成了。

使用对称加密方法加密发送的内容，使用非对称加密方法加密回话秘钥，兼顾速度和安全。

**如何保证公钥不被篡改？**

将公钥放在数字证书中，只要证书是可信的，公钥就是可信的。

**公钥加密计算量太大，如何减少耗用的时间？**

每一次对话（session），客户端和服务器端都生成一个"对话密钥"（sessionkey），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。

**为什么服务端要发送证书给客户端**

互联网有太多的服务需要使用证书来验证身份，以至于客户端(操作系统或浏览器等)无法内置所有证书，需要通过服务端将证书发送给客户端。

**什么是数字签名和数字证书**

数字签名：A将一段文本通过哈希（hash）和私钥加密处理后生成数字签名,发送给B。B使用A的公钥解密，得到文本摘要，证明了A的身份。同时对文本本身使用hash函数，得到的结果与文本摘要进行对比，若两者一致，则证明文本未被篡改。

数字签名存在问题：若C将B中存储的A的公钥替换为C的公钥，那么C就可以很容易的冒充A。

解决方法：A找证书颁发机构(CA),为公钥做认证，CA用自己的私钥对A的公钥和一些相关信息一起加密，生成数字证书。A在发送信息的时候，同时附上数字签名和数字证书。B收到后，先用CA的公钥解开数字证书，拿到A的公钥，可以证明A的身份，然后再用A的公钥解开信息。

**https的缺点和使用方法**

http协议加密范围有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面起不到什么作用，且ssl整数的信用链体系并不安全。

考虑到SLL整数需购买申请且等级越高收费越高、https占用服务端资源高很多、握手阶段费时影响响应速度等问题，一般网站(e.g12306)采用分而治之的手段，即主页采用http协议，涉及用户信息则使用https协议。

**httpv.s.https**

1. https更安全

HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，所有传输的内容都经过加密，加密采用对称加密，但对称加密的密钥用服务器方的证书进行了非对称加密。http是超文本传输协议，信息是明文传输，没有加密，通过抓包工具可以分析其信息内容。

1. https需要申请证书
2. https协议需要到CA申请证书，一般免费证书很少，需要交费。而常见的http协议则没有这一项。3）端口不同http使用的是80端口，而https使用的是443端口。
3. 所在层次不同

HTTP协议运行在TCP之上，HTTPS是运行在SSL/TLS之上的HTTP协议，SSL/TLS运行在TCP之上。

**如何将网站从http切换到https？**

这里需要将页面中所有的链接，例如js，css，图片等等链接都由http改为https。例如：http://www.baidu.com改为https://www.baidu.com

这里虽然将http切换为了https，还是建议保留http。所以我们在切换的时候可以做http和https的兼容，具体实现方式是，去掉页面链接中的http头部，这样可以自动匹配http头和https头。例如：将http://www.baidu.com改为//www.baidu.com。然后当用户从http的入口进入访问页面时，页面就是http，如果用户是从https的入口进入访问页面，页面即使https的。

**零碎的问题**

**32位ip地址**

**分类**

A类地址:0+网络号(7位)+主机号(24位)1.0.0.0-127.255.255.255网络数2^7-2主机数2^24-2

B类地址:10+网络号(14位)+主机号(16位)128.0.0.0-191.255.255.255

C类地址:110+网络号(21位)+主机号(8位)192.0.0.0-223.255.255.255

D类地址:1110+组播地址(28位)为多点播送地址，用于多目的地信息的传输和作为备用。

E类地址:11110+保留用于实验和将来使用

**比较特殊的ip地址（net\_id+host\_id）**

1）网络地址：net\_id+全0代表本网段。

2）广播地址：

a.直接广播：net\_id+全1可在本地广播，也可跨网段广播。只可做目的地址。

b.受限广播：255.255.255.255只能用于本地网络广播，不能被路由器转发。只可做目的地址。

3）环回地址：127.X.X.X用于测试本机TCP/IP是否正常。e.g.127.0.0.1用于回环测试。可做源地址和目的地址。

4）所有网络：0.0.0.0代表本机所有IP地址、用于指定默认路由、在主机地址冲突时会获得该地址。只可做源地址。

5）自动私有IP地址：169.254.0.0~169.254.255.255DHCP获取失败会获得该网段地址。代表本地网络目前出现问题，无法联网。

全0+host\_id：本网络上的某个主机。只可做源地址。

224.0.0.1代表组播内的所有主机

224.0.0.2代表组播内的所有路由器

**ddos攻击的原理，如何防止**

1）SYNFlood攻击

攻击方发送大量伪造原ip地址的攻击报文，发送到服务端，服务器端将为了维护一个非常大的半连接队列而消耗非常多的cpu时间和内存，甚至导致服务器拒绝服务。

解决方法：当服务器接受到SYN报文段时，不直接为该TCP分配资源，而只是打开一个半开的套接字。接着会使用SYN报文段的源Id，目的Id，端口号以及只有服务器自己知道的一个秘密函数生成一个cookie，并把cookie作为序列号响应给客户端。

如果客户端是正常建立连接，将会返回一个确认字段为cookie+1的报文段。接下来服务器会根据确认报文的源Id，目的Id，端口号以及秘密函数计算出一个结果，如果结果的值+1等于确认字段的值，则证明是刚刚请求连接的客户端，这时候才为该TCP分配资源。

这样一来就不会为恶意攻击的SYN报文段分配资源空间，避免了攻击。

2）tcp全连接攻击

一般情况下，常规防火墙大多具备过滤Land等DOS攻击的能力，但对于正常的TCP连接是放过的，很多网络服务程序（如：IIS、Apache等Web服务器）能接受的TCP连接数是有限的，一旦有大量的TCP连接，则会导致网站访问非常缓慢甚至无法访问。TCP全连接攻击就是通过许多僵尸主机不断地与受害服务器建立大量的TCP连接，直到服务器的内存等资源被耗尽而被拖跨，从而造成拒绝服务。

优点可绕过一般防火墙的防护而达到攻击目的。

缺点是需要找很多僵尸主机，并且由于僵尸主机的IP是暴露的，因此容易被追踪。

3）Land攻击

land攻击报文的源IP地址和目的IP地址是相同的，都是目标主机的IP地址。这样目标主机接在收到这个SYN报文后，就会向该报文的源地址发送一个ACK报文，并建立一个TCP连接控制结构，而该报文的源地址就是自己。由于目的IP地址和源IP地址是相同的，都是目标主机的IP地址，因此这个ACK报文就发给了目标主机本身。这样如果攻击者发送了足够多的SYN报文，则目标计算机的TCB可能会耗尽，最终不能正常服务。

**攻击网站的方法和原理**

1）ddos攻击

2）XSS攻击(跨站脚本攻击,)

注入攻击的一种,不对服务器端造成任何伤害，而是通过一些正常的站内交互途径，往Web页面里插入恶意html代码，这时服务器端如果没有过滤或转义掉这些脚本，作为内容发布到了页面上，其他用户访问这个页面的时候就会运行这些脚本。比如：有些人在留言板中输入恶意脚本来获取用户的帐号和密码。

3）SQL注入攻击

**路由器和交换机的区别**

1.交换机工作在数据链路层；路由器工作在网络层。

2.交换机转发数据帧；路由器转发IP分组。

3.交换机隔离冲突域，不隔离广播域；路由器隔离冲突域，隔离广播域

**PPP(Point-to-Point Protocol)**

因为IP协议是不可靠的，所以处于链路层的PPP协议也不需要保证可靠。

**简单** 对于链路层的帧，无需纠错，无需序号，无需流量控制，无需保证可靠传输。且不需要支持多点线路，只需支持点对点的连接即可。且只支持全双工链路。

**封装成帧** 在帧头帧尾加上帧定界符。

**透明传输** 不论在数据部分是什么样的比特流，都可以完整的传输过去，即使与到遇到一些与帧定界符一样的比特组合的数据也一样。那如果数据中存在帧定界符应该如何处理？异步线路(逐个字节或字符发送)用字节填充个，同步线路(逐个比特发送)用比特填充。

**多种网络层协议** 封装IP数据报可采用多种协议。

**多种类型链路** 串行/并行,同步/异步。。。

**差错检测** 错就丢弃。无需纠错，只需检错。

**检测连接状态** 链路是否正常工作

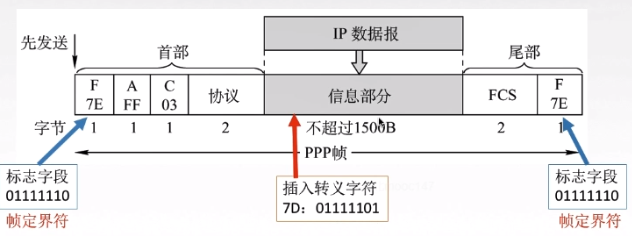
**最大传送单元** 数据部分(即上层交下来的IP数据报)最大长度MTU(默认不超过1500字节)

**网络层地址协商** 知道通信双方的网络层地址。

PPP的三个组成部分：

1. 一个将IP数据报封装到串行链路的方法。e.g.同步串行、异步串行
2. 链路控制协议LCP：建立并维护数据链路连接。身份验证，比如拨号上网需要输入账户密码，如果没有输入正确就不能上网。（更像物理连接）
3. 网络控制协议NCP:封装IP数据报有NCP完成。由于PPP可支持多种网络层协议，每个不同的网络层协议都需要一个相应的NCP来配置，为网络层协议建立和配置连接。（逻辑连接）

PPP帧以字节为单位：



信息部分如果出现转义字符或帧定界符，就在该字符前面加入转移字符。

协议字段标志信息是什么：IP数据报、LCP的数据、网络层控制数据。。。

FCS 用于差错检测的冗余循环校验码。

A 地址字段，用于指定哪个站正在处理，但是PPP只关心一个站，所以设置了oxFF（所有站）。

C 控制字段，用于帧序列和重传行为，在PPP没有用。

**一些名词**

**MSL最大报文生存时间**，一般是30s,1min或2min。

**TTL（time-to-live生存时间)**，并不是具体的时间，而是设置了数据报可以经过的最大路由器的数目，每经过一个路由器，该值减一。当TTL为0,该数据报被丢弃。

**RTT往返时间**，TCP的超时与重传需要对一个给定连接的往返时间RTT的测量。由于路由器和网络流量均会变化，因此RTT也经常变化。