计算机网络

三次握手(为什么不是两次)

TCP报文段

一个TCP报文段分成tcp首部和tcp数据报两个部分，其中首部20字节固定，后面有4N字节是可选的增加选项。TCP报文段既可以用来运载数据，也可以用来建立连接、释放连接和应答。

图 5-6 TCP 报文段的首部

- 源端口和目的端口：各占2字节，是传输层与应用层的服务接口，复用和分用都通过端口实现
- 序号字段：seq，4字节。TCP是面向字节流的，所以tco连接中传送的数据流每一个字节上都有一个序号，序号字段的值表示本报文段所发送的数据的第一个字节的序号。

  例如，一报文段的序号字段值是 301，而携带的数据共有 100 字节，这就表明本报文段的数据的最后一个字节的序号是 400，故下一个报文段的数据序号应从 401 开始。
- **确认号字段ack**：4字节。是期望收到对方的下一个报文段的数据的第一个字节的序号，若为N，则表示序号N-1为止的数据都正确收到
- 数据偏移（首部长度）：4位，表示的是首部长度，指出TCP报文段的数据起始处距离TCP报文段的起始处有多远
- 保留字段：6位，保留为今后使用，目前为0
- 紧急位URG：URG=1时，表示紧急指针字段有效，应尽快传递
- **确认为ACK**：当ACK=1时，确认字段有效，TCP规定在建立连接后所有传送报文段都必须把ACK=1
- 推送为PSH：接收到PSH=1的报文段，就尽快交付接收应用程序，而不等缓存存完再去
- 复位位RST: RST=1表明TCP连接出现烟冲差错，必须释放连接，然后重新建立运输连接
- **同步位SYN**：SYN=1表示这是一个连接请求或连接接受报文。（SYN=1,ACK=0表明这是一个连接请求报文。如果对方同意建立连接，则相应报文SYN=1,ACK=1） +** 终止位FIN**: 用来释放一个连接，FIN=1表明此报文段的发送方数据已经发送完毕，并要求释放传输连接

- 窗口字段：2字节，指出允许对方发送的数据量，接收方的数据缓存空间有限
- 检验和：2字节。检验和字段的检验范围包括首部和数据两部分，且同样要加上伪首部
- 紧急指针字段：16位，指出本报文段中紧急数据共有多少字节（紧急数据放在报文段数据的最前面
- 选项字段：长度可变，表示最大报文段长度MSS
- 填充字段：为了使整个首部长度是4字节的整数倍

TCP连接的建立-三次握手

- 第一步：客户机TCP向服务器TCP发送连接请求报文段，这个报文段不包含应用层数据，首部SYN=1，另外，客户机会随机选择一个起始序号seq=x（连接强求报文不携带数据，但要消耗序号）
- 第二步：服务器TCP接收到请求报文后，如果同意建立连接，则发回确认，并为该TCP连接分配TCP缓存和变量。再确认报文段中，SYN=1,ACK=1，确认号字段值=x+1，服务器随机产生起始序号seq=y。确认报文段同样不包括应用层数据
- 第三步：客户机收到确认报文段后，还要向服务器给出确认，并为该连接分配缓存和变量。ACK=1,序号字段为x+1，确认号字段为ack=y+1。该报文段可以携带数据，如果不携带数据则不消耗序号。
    - 由于三次握手客户端资源和服务器端资源分配的时间不一样，使得服务器易受到SYN洪泛攻击

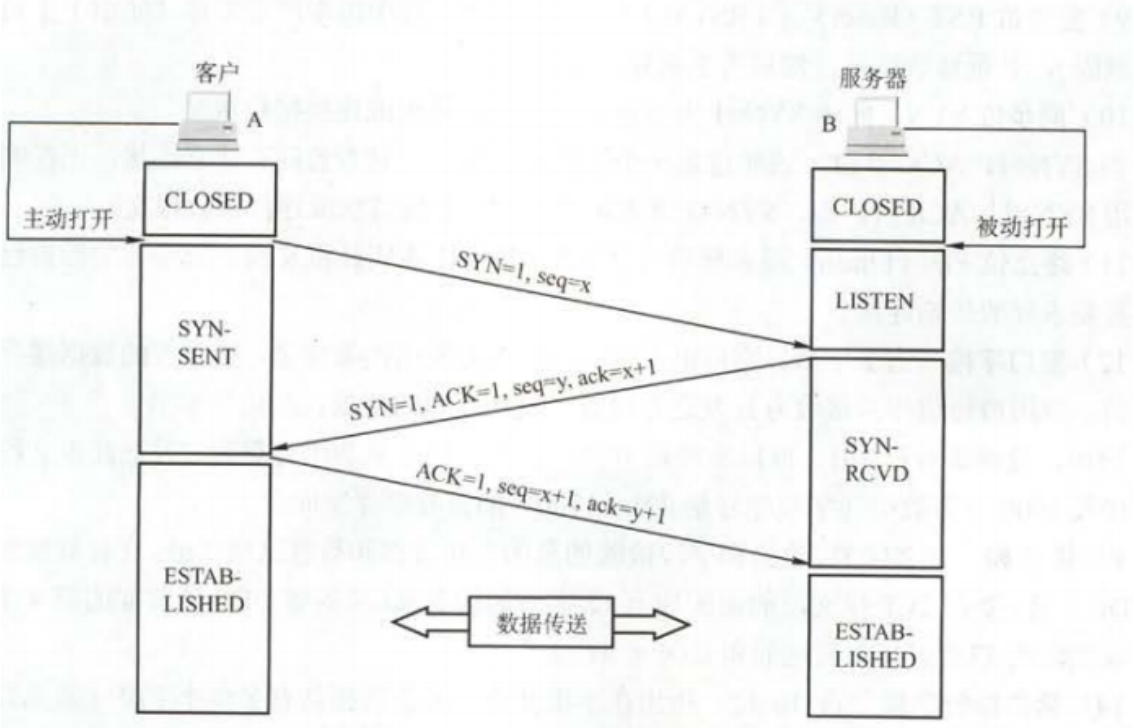


图 5-7　用“三次握手”建立 TCP 连接

连接建立编码

- SYN =1, seq=x
- SYN=1,ACK=1，seq=y， ack=x+1
- ACK =1, swq=x+1，ack=y+1

为什么是3次握手而不是2次或者4次

在谢希仁著《计算机网络》第四版中讲“三次握手”的目的是“为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误”

TCP可靠传输的精髓在于：让双方都明确对方发送的序号字段。之后对发送的数据进行编号，这样能保证接收端在接受数据的有序和可靠性。所以可以总结说三次握手来约定双方的ISN(Initial Sequence Number)也就是初始的seq。

四次握手效率低，会造成资源的浪费。而两次握手造成的结果就是两者就服务端的seq没有达成一致，这个时候服务端会一直重传自己的SYN信号，直到收到客户端的ACK确认为止。

一些特殊情况

A=客户端， B=服务端。 **第一个包，即A发送给B的SYN中途被丢，没有达到B**
A会周期性超时重传，直到收到B的确认

第二个包，即B给A发送的SYN+ACK被丢，没有达到A
B会周期性的超时重传，直到收到A的ACK确认。

第三个包，即A给B发送的ACK被弄丢了，没有到B
A发完ACK 是established的状态，而B还是TCP-Active的状态

- 假如此时双方都没有数据进行传输，B会周期性的进行重传，直到收到A的确认回复，这时B的状态会变成established，双方可以正常传输数据。
- 假如此时A想传输数据，B收到A的Data+ACK，B会自然把自己的状态切换成established，并接受A的data
- 假定B此时想传输数据，数据发送不了，会一直周期性的重传SYN+ACK.直到收到A的确认才可以发送数据。

四次挥手

四次挥手步骤

- 客户端打算关闭连接，向TCP发送连接释放报文段，并停止再发送数据，主动关闭TCP连接,FIN=1，seq=u，它等于前面一传送过的数据的最后一个字节的序号加1。FIN报文段不携带数据，
- 服务器接收到连接释放报文段，确认号ack=u+1，而这个报文段自己序号是v，等于它前面已经传过的数据的最后一个字节+1.此时，客户机到服务器方向的连接就被释放，TCP处于半关闭状态。但服务器还可以向客户机发数据。
- 若服务器没有要向客户机发送的数据，那发送FIN=1的连接释放报文段。
- 客户端收到后要发送确认，在去确认报文段中，ACK=1，确认号ack=w+1，此时TCP连接还未释放，

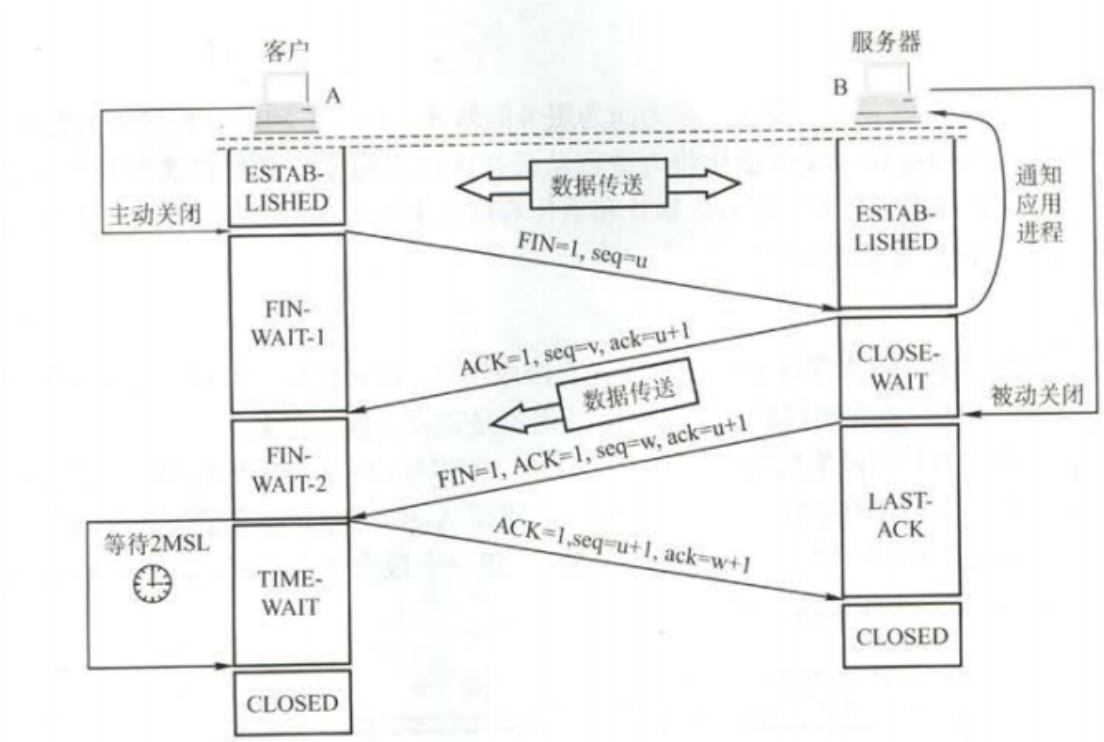


图 5-8　用“四次握手”释放 TCP 连接

释放连接信号码

- FIN = 1 ，seq = u
- ACK =1, seq = v， ack=u+1
- FIN = 1, ACK =1, seq=w，ack=u+1
- ACK=1，seq=u+1，ack=w+1

对关闭TCP连接的理解

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。
TCP协议的连接是全双工连接，一个TCP连接存在双向的读写通道。
简单说来是 “先关读，后关写” ，一共需要四个阶段。以客户机发起关闭连接为例：

- 1.服务器读通道关闭
- 2.客户机写通道关闭
- 3.客户机读通道关闭
- 4.服务器写通道关闭

关闭行为是在发起方数据发送完毕之后，给对方发出一个FIN（finish）数据段。直到接收到对方发送的FIN，且对方收到了接收确认ACK之后，双方的数据通信完全结束，过程中每次接收都需要返回确认数据段ACK。

详细过程：
第一阶段 客户机发送完数据之后，向服务器发送一个FIN数据段，序列号为i；

- 1.服务器收到FIN(i)后，返回确认段ACK，序列号为i+1，关闭服务器读通道；
- 2.客户机收到ACK(i+1)后，关闭客户机写通道；（此时，客户机仍能通过读通道读取服务器的数据，服务器仍能通过写通道写数据）

第二阶段 服务器发送完数据之后，向客户机发送一个FIN数据段，序列号为j；

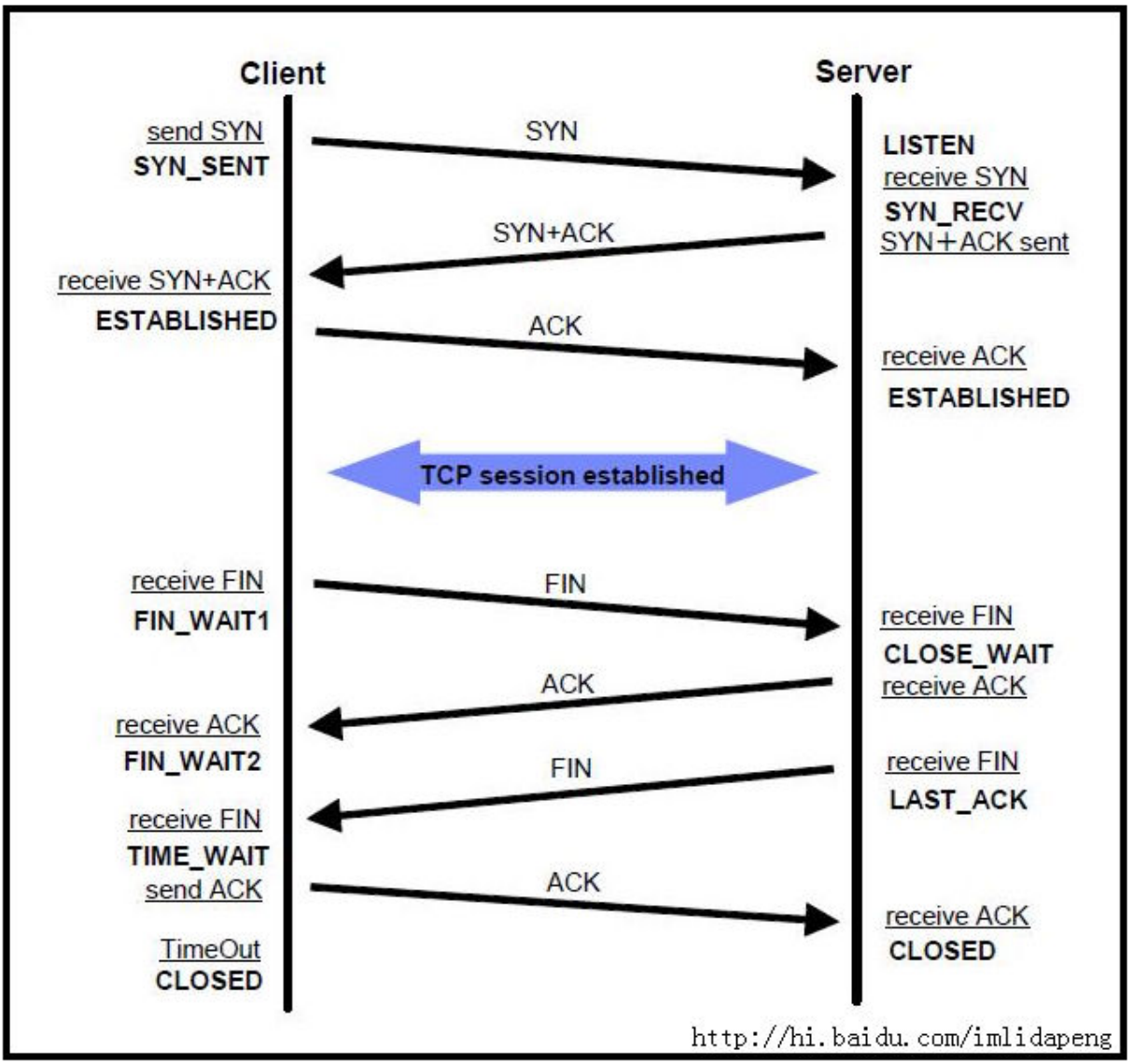
- 3.客户机收到FIN(j)后，返回确认段ACK，序列号为j+1，关闭客户机读通道；
- 4.服务器收到ACK(j+1)后，关闭服务器写通道。 这是标准的TCP关闭两个阶段，服务器和客户机都可以发起关闭，完全对称。

TCP连接及释放的客户端和服务器信息

- LISTEN - 侦听来自远方TCP端口的连接请求；
- SYN-SENT -在发送连接请求后等待匹配的连接请求；
- SYN-RECEIVED - 在收到和发送一个连接请求后等待对连接请求的确认；
- ESTABLISHED- 代表一个打开的连接，数据可以传送给用户；
- FIN-WAIT-1 - 等待远程TCP的连接中断请求，或先前的连接中断请求的确认；
- FIN-WAIT-2 - 从远程TCP等待连接中断请求；
- CLOSE-WAIT - 等待从本地用户发来的连接中断请求；
- CLOSING -等待远程TCP对连接中断的确认；
- LAST-ACK - 等待原来发向远程TCP的连接中断请求的确认；
- TIME-WAIT -等待足够的时间以确保远程TCP接收到连接中断请求的确认；
- CLOSED - 没有任何连接状态；

客户端TCP状态迁移：
CLOSED->SYN_SENT->ESTABLISHED->FIN_WAIT_1->FIN_WAIT_2->TIME_WAIT->CLOSED

服务器TCP状态迁移：
CLOSED->LISTEN->SYN收到->ESTABLISHED->CLOSE_WAIT->LAST_ACK->CLOSED



四次挥手的原因

那四次分手又是为何呢？TCP协议是一种面向连接的、可靠的、基于字节流的运输层通信协议。TCP是全双工模式，这就意味着，当主机1发出FIN报文段时，只是表示主机1已经没有数据要发送了，主机1告诉主机2，它的数据已经全部发送完毕了；但是，这个时候主机1还是可以接受来自主机2的数据；当主机2返回ACK报文段时，表示它已经知道主机1没有数据发送了，但是主机2还是可以发送数据到主机1的；当主机2也发送了FIN报文段时，这个时候就表示主机2也没有数据要发送了，就会告诉主机1，我也没有数据要发送了，之后彼此就会愉快的中断这次TCP连接。如果要正确的理解四次分手的原理，就需要了解四次分手过程中的状态变化。

FIN_WAIT_1: 这个状态要好好解释一下，其实FIN_WAIT_1和FIN_WAIT_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别是：FIN_WAIT_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入到FIN_WAIT_1状态。而当对方回应ACK报文后，则进入到FIN_WAIT_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN_WAIT_1状态一般是比较难见到的，而FIN_WAIT_2状态还有时常常可以用netstat看到。（主动方）

FIN_WAIT_2：上面已经详细解释了这种状态，实际上FIN_WAIT_2状态下的SOCKET，表示半连接，也即有一方要求close连接，但另外还告诉对方，我暂时还有点数据需要传送给你(ACK信息)，稍后再关闭连接。（主动方）

CLOSE_WAIT：这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方close一个SOCKET后发送FIN报文给自己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE_WAIT状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以 close这个SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE_WAIT状态下，需要完成的事情是等待你去关闭连接。（被动方）

LAST_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也即可以进入到CLOSED可用状态了。（被动方）

TIME_WAIT: 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。如果FINWAIT1状态下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME_WAIT状态，而无须经过FIN_WAIT_2状态。（主动方） CLOSED: 表示连接中断。

Time-wait的意义所在

TIME_WAIT是TCP协议用以保证被重新分配的socket不会受到之前残留的延迟重发报文影响的机制，是必要的逻辑保证。

从上面可以看到，主动发起关闭连接的操作的一方将达到TIME_WAIT状态，而且这个状态要保持Maximum Segment Lifetime的两倍时间。

原因有二： 一、保证TCP协议的全双工连接能够可靠关闭 二、保证这次连接的重复数据段从网络中消失

先说第一点，如果Client直接CLOSED了，那么由于IP协议的不可靠性或者是其它网络原因，导致Server没有收到Client最后回复的ACK。那么Server就会在超时之后继续发送FIN，此时由于Client已经CLOSED了，就找不到与重发的FIN对应的连接，最后Server就会收到RST而不是ACK，Server就会以为是连接错误把问题报告给高层。这样的情况虽然不会造成数据丢失，但是却导致TCP协议不符合可靠连接的要求。所以，Client不是直接进入CLOSED，而是要保持TIME_WAIT，当再次收到FIN的时候，能够保证对方收到ACK，最后正确的关闭连接。

再说第二点，如果Client直接CLOSED，然后又再向Server发起一个新连接，我们不能保证这个新连接与刚关闭的连接的端口号是不同的。也就是说有可能新连接和老连接的端口号是相同的。一般来说不会发生什么问题，但是还是有特殊情况出现：假设新连接和已经关闭的老连接端口号是一样的，如果前一次连接的某些数据仍然滞留在网络中，这些延迟数据在建立新连接之后才到达Server，由于新连接和老连接的端口号是一样的，又因为TCP协议判断不同连接的依据是socket pair，于是，TCP协议就认为那个延迟的数据是属于新连接的，这样就和真正的新连接的数据包发生混淆了。所以TCP连接还要在TIME_WAIT状态等待2倍MSL，这样可以保证本次连接的所有数据都从网络中消失。

weburl访问的全过程

1.DHCP配置主机信息（找本机IP）

- 主机生成一个DHCP请求报文，并将这个报文放入目的端口67和源端口68的UDP报文段中

- 该报文段放在一个广播IP地址（255.255.255.255）和源IP地址（0.0.0.0）的IP数据报中
- 该数据报被放在MAC帧中，目的地址FF:FF:FF:FF:FF:FF，广播到交换机连接的所有设备
- 交换机的DHCP服务器收到广播帧后，不断向上解析得到IP、UDP、DHCP报文，之后生成DHCP的ACK报文，该报文包括 IP地址、DNS服务器IP地址、默认网关路由器的IP地址和子网掩码，再经过层层封装到MAC帧中
- 该帧的目的地址是主机的mac地址，主机收到该帧后分解得DHCP报文，之后配置IP地址，子网掩码、DNS服务器IP地址，安装默认网关

2.ARP解析网关MAC地址（找网关MAC地址）

- 主机通过浏览器生成一个TCP套接字，为了发送HTTP请求，需要知道网站对应的IP地址
- 生成一个DNS查询报文，端口53（DNS服务器）
- DNS查询报文放入目的地址为DNS服务器IP地址的IP数据报中
- IP数据报放入一个以太网帧中，发送至网关路由器
- DHCP过程只知道网关的IP地址，为了获取网关的MAC地址，需要用ARP协议
- 主机生成一个目的地址为网关路由器IP的ARP查询报文，放入一个广播帧中，并发送这个以太网帧，交换机将其发送给所有的连接设备
- 网关接收到该帧后，分解得到ARP报文，发现IP地址与自己想匹配，发送一个ACK报文回应自己的MAC地址

3.DNS解析域名（找服务器IP）

- 知道了DNS的MAC地址后就可以继续DNS解析过程
- 网关接收到DNS查询报文后，抽出IP数据报，并根据该表选择该转发的路由器
- 路由器根据内部网关协议（RIP、OSPF）和外部网关协议（BGP）配置路由器到DNS的路由表项
- 之前的DNS报文到DNS服务器后，照常依次抽出报文，在DNS库中查找解析域名
- 找到DNS记录后发送DNS回答报文，然后将其放入UDP报文段、IP数据报，通过路由器反转发回网关路由器，经过交换机到主机

4. HTTP请求页面

- 有了HTTP服务器的IP地址后，主机便可以生成TCP套接字，向web服务器发送HTTP get报文
- 建立HTTP连接前需要进行TCP连接，进行三次握手，过程略
- 建立连接后发送HTTP的GET报文，交付给HTTP服务器
- HTTP服务器从TCP中读出报文，生成HTTP相应报文，将web页面放入HTTP报文主体中发挥主机
- 浏览器收到HTTP相应报文后抽取WEB页面内容进行渲染，显示web页面

TCP和UDP的区别，TCP是如何保证其可靠性的

UDP和TCP 的基本区别

TCP和UDP协议特性的区别，主要从连接性、可靠性、有序性、拥塞控制、传输速度、头部大小来讲

- TCP面向连接，UDP无连接。TCP3次握手建立连接，UDP发送前不需要建立连接
- TCP可靠，UDP不可靠，TCP丢包有确认重传机制，UDP不会
- TCP有序，会对报文进行重排；而UDP无序，后发送信息可能先到达
- TCP必须进行数据验校，UDP的校验可选
- TCP有流量控制（滑动窗口）和拥塞控制，UDP没有
- TCP传输慢，UDP传输快，因为TCP要建立连接、保证可靠有序，还有流量、拥塞控制
- TCP包头较大（20字节）UDP较小（8字节）
- 基于TCP协议：HTTP/HTTPS、Telnet、FTP、SMTP 基于UDP的协议 DHCP、DNS、SNMP、TFTP、BOOTP

TCP实现可靠性

TCP提供的可靠数据传输服务就是要保证接收方进程从缓存区读出的字节流与发送方的字节流是完全一样的。使用了校验、序号、确认和重传机制来达到这个目的。

- **序号**：TCP首部的序号字段用来保证数据能够有序提交给应用层，序号是建立在传送的字节流之上，值指的是本报文段所发送的数据的第一个字节的序号
- **确认**：TCP首部的确认号是期望收到的对方的下一个报文段的数据的第一个字节的序号，TCP默认使用的是累计确认，在确认阶段，可以不按顺序。
- **重传**：超时和冗余会导致TCP对报文段重传
  - 超时：TCP每发送一个报文段，就对报文段设置计时器，如果计时器设置的重传时间到期还没收到确认，就重传这一段报文。重传时间的计算加权平均往返时间+4*RRT偏差的加权平均值
  - 冗余ACK:TCP规定每当比期望序号大的时序报文段到达时，发送一个冗余ACK，表明它期待的下一个字节的序号，若发送端收到同一个报文段的n次冗余ACK，则可以认为该报文段丢失，重传。
- **流量控制**：TCP 连接的每一方都有固定大小的缓冲空间，TCP的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。（利用滑动窗口实现）
- **拥塞控制**：当网络拥塞时，减少数据的发送。

TCP流量控制

TCP 中采用滑动窗口来进行传输控制，滑动窗口的大小意味着接收方还有多大的缓冲区可以用于接收数据。发送方可以通过滑动窗口的大小来确定应该发送多少字节的数据。当滑动窗口为 0 时，发送方一般不能再发送数据报，但有两种情况除外，一种情况是可以发送紧急数据，例如，允许用户终止在远端机上的运行进程。另一种情况是发送方可以发送一个 1 字节的数据报来通知接收方重新声明它希望接收的下一字节及发送方的滑动窗口大小。

- TCP 利用滑动窗口实现流量控制。
- 流量控制是为了控制发送方发送速率，保证接收方来得及接收。
- 接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为 0，则发送方不能发送数据。

TCP拥塞控制

拥塞控制是让网络能够承受现在的网络符合，是一个全局性的过程，涉及所有的主机及路由器等；而流量控制是指点对点的通信量控制。所以发送方在确定发送报文的速率时，要维护两个窗口，接受窗口rwnd（反应接收方容量）和拥塞窗口cwnd（反映当前网络容量）

- 慢开始和拥塞避免
    - 慢开始：在TCP连接好的阶段，令拥塞窗口=1，最大报文段长度MSS，每收到一个确认，cwnd+1，增大一个MSS，这样每经过一个传输轮次，拥塞窗口cwnd就会加倍，直到到达规定的阈值(ssthresh)，然后才去拥塞避免算法。
    - 拥塞避免算法：当cwnd每经过一个往返时延就增加一个MSS大小，而不是加倍，样拥塞窗口cwnd按线性规律缓慢的增长，比慢开始算法的拥塞窗口增长速率缓慢的多。而当出现异常超时（网络拥塞情况时），则令阈值(ssthresh)等于当前cwnd一半，即慢算法+拥塞避免算法

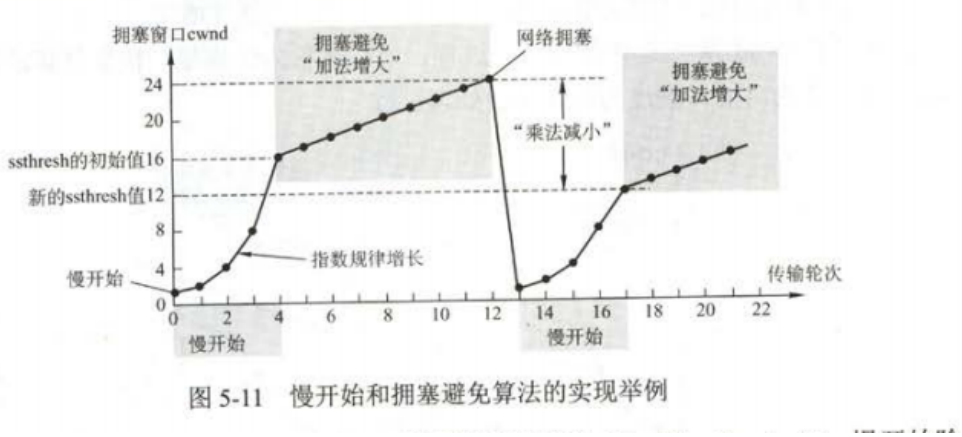
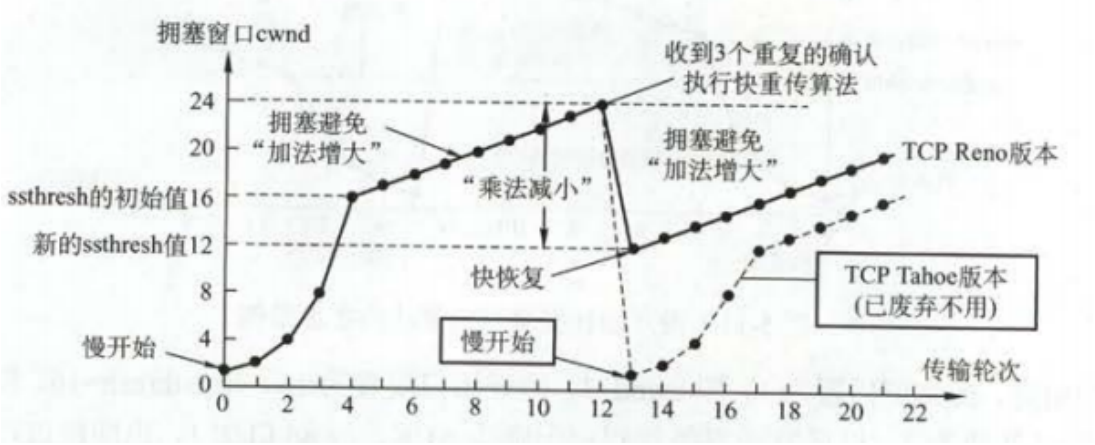


图 5-11　慢开始和拥塞避免算法的实现举例

- 快重传
    - 当发送方连续收到三个重复的ACK报文时，直接重传，不必等待重传计时器超时。
- 快恢复
    - 当发送端连续收到三个冗余ACK确认时，执行乘法减小，但cwnd的值是阈值减半后的值，而不是从1开始



http请求信息(状态码，请求头等)

HTTP是面向文本的，因此在报文中的每个字段都是ASII码串，并且每个字段长度不确定。有两类报文，请求报文和响应报文。

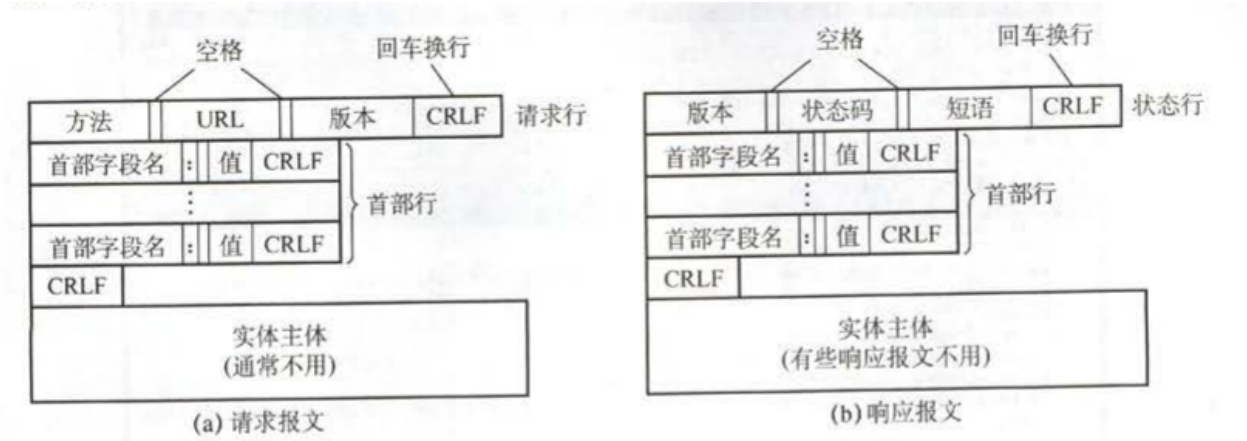


图 6-14　HTTP 的报文结构

HTTP request常见的9种请求方法：

- OPTIONS 返回服务器针对特定资源的HTTP请求方法
- HEAD 向服务器索要与GET请求一致的相应，而响应体不会被返回
- GET 向特定资源发出请求（GET可能会被爬虫随意访问）
- POST 向指定资源提交数据处理的请求（提交表单、上传文件）数据被包含在请求体中。POST可能会导致新建资源或已有资源的修改
- PUT 向指定资源上传最新内容
- DELETE 请求服务器删除 REQUEST_URL 所标识的资源
- TRACE 回显服务器收到的请求
- CONNECT 预留给连接方式改为管道的代理服务器
- PATCH 局部修改某一资源

HTTP request常见的首部请求头（可以在www.baidu.com上查看对应值）

- Host：（发送请求，请求头是必须的）主要用于执行被请求资源的Internet主机和端口号，它从HTTP的URL中提取出来，
- Connection: 是否需要持久连接，keep-alive 和close。 HTTP 是一个请求响应模式的典型范例，即客户端向服务器发送一个请求信息，服务器来响应这个信息。在之前的HTTP版本中，每个请求被创建一个新的到服务器端的连接，在这个连接上发送请求，然后接受请求。keep-alive被用来解决效率了低的问题。keep-alive 使客户端到服务端的连接持续有效，当出现了后续请求时， keep-alive避免了建立或者重新建立。对于市面上大部分服务器keep-alive都被支持，但是对于负担较重的网站，保留的客户端会影响性能。
- Accept：浏览器可以接受的MIME类型。 Accept：text/html, 如果不能接受将返回406错误
- Cache-control: 指定请求和响应遵循缓存机制。缓存指令是单向的，且独立。

- Accept-Encoding：浏览器生命可接受的编码方法，通常说明是否支持压缩。
- Accept-Language：浏览器声明接受语言
- Accept-Charset：浏览器可以接受的字符集，默认任何字符集都可以接收
- User-agent：用于告诉HTTP服务器，客户端使用操作系统浏览器的名称和版本。

http response的状态码

- 100-199信息状态码
  - 100 continue：收到了请求的初始部分，请客户端继续
- 200-299成功状态码
  - 200 OK：请求被正常处理
  - 204 No Content：请求被接受，但是响应报文只有首部和状态行，没有实体部分
  - 206 Partial Content：客户端只强求了一部分的资源，服务器只针对请求的资源进行返回。
- 300-399重定向状态码
  - 301 Moved Permanently: 永久重定向
  - 302 Found: 临时重定向，资源被临时移动了
  - 303 See Other: 表示用户请求的资源代表着另一个URI，应该使用GET去获取资源
  - 304 Not Modified: 当客户端发送的请求附带条件时，服务器找到资源但未符合请求
  - 307 Temporary Redirect: 同302，临时重定向
- 400-499客户端错误状态码
  - 400 Bad Request: 告知客户端它发送了一个错误的请求
  - 401 Unauthorized: 请求需进行认证
  - 403 Forbidden: 请求被服务器拒绝了
  - 404 Not Found：服务器无法找到对应资源
  - 405 Method Not Allowed：请求中带有不支持的方法
- 500-599服务器错误状态码
  - 500 Internet Serverr Error:服务器内部错误
  - 502 Bad GateWay:代理或网关服务器从下一条链路收到了伪响应
  - 503 Server Unavaialble:服务器正忙
  - 504 GateWay Timeout:一个代理网关等待另一个服务器时超时了

HTTP response首部请求头

- Cacht-Control:告诉所有的缓存机制是否可以缓存以及缓存那种类型
- Connection：是否保持持久连接
- Content-Encoding：返回内容压缩编码类型
- Content-type：返回内容的MIME类型
- Date：原始服务器消息发出时间
- Expires：响应过期的时间
- Server：Web服务器软件名称
- Set-Cookie：设置浏览器缓存
- Transfer-Encoding：文件传输编码

Cookie & Session & Application

Cookie

Cookie机制
在程序中，会话跟踪是很重要的事情。理论上，一个用户的所有请求操作都应该属于同一个会话，而另一个用户的所有请求操作则应该属于另一个会话，二者不能混淆。例如，用户A在超市购买的任何商品都应该放在A的购物车内，不论是用户A什么时间购买的，这都是属于同一个会话的，不能放入用户B或用户C的购物车内，这不属于同一个会话。

而Web应用程序是使用HTTP协议传输数据的。HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话。即用户A购买了一件商品放入购物车内，当再次购买商品时服务器已经无法判断该购买行为是属于用户A的会话还是用户B的会话了。要跟踪该会话，必须引入一种机制。

Cookie就是这样的一种机制。它可以弥补HTTP协议无状态的不足。在Session出现之前，基本上所有的网站都采用Cookie来跟踪会话。

单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。
什么是Cookie
Cookie意为“甜饼”，是由W3C组织提出，最早由Netscape社区发展的一种机制。目前Cookie已经成为标准，所有的主流浏览器如IE、Netscape、Firefox、Opera等都支持Cookie。

由于HTTP是一种无状态的协议，服务器单从网络连接上无从知道客户身份。怎么办呢？就给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是Cookie的工作原理。

Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。Google也只能操作Google的Cookie，而不能操作Baidu的Cookie。Cookie具有不可跨域名性

Cookie基本属性
HttpCookie类

```
    // The value of the cookie itself.
    private final String name;  // NAME= ... "$Name" style is reserved
    private String value;       // value of NAME

    // Attributes encoded in the header's cookie fields.
    private String comment;     // Comment=VALUE ... describes cookie's use
    private String commentURL;  // CommentURL="http URL" ... describes cookie's use
    private boolean toDiscard;  // Discard ... discard cookie unconditionally
    private String domain;      // Domain=VALUE ... domain that sees cookie
    private long maxAge = MAX_AGE_UNSPECIFIED;  // Max-Age=VALUE ... cookies auto-expire
    private String path;        // Path=VALUE ... URLs that see the cookie
```

```
    private String portlist;    // Port[="portlist"] ... the port cookie may be returned
to
    private boolean secure;     // Secure ... e.g. use SSL
    private boolean httpOnly;   // HttpOnly ... i.e. not accessible to scripts
    private int version = 1;    // Version=1 ... RFC 2965 style
```

Session

session机制 除了使用Cookie，Web应用程序中还经常使用Session来记录客户端状态。Session是服务器端使用的一种记录客户端状态的机制，使用上比Cookie简单一些，相应的也增加了服务器的存储压力。

什么是session Session是另一种记录客户状态的机制，不同的是Cookie保存在客户端浏览器中，而Session保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录(HashTable)在服务器上。这就是Session。客户端浏览器再次访问时只需要从该Session中查找该客户的状态就可以了。

如果说Cookie机制是通过检查客户身上的“通行证”来确定客户身份的话，那么Session机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。Session对象是在客户端第一次请求服务器的时候创建的

Application

网络编程(Socket编程)

socket通信

客户端通过使用Socket的构造器来连接到指定的服务器

- Socket(InetAddress/String remoteAddress, in port):创建连接到指定的远程主机、远程端口的Socket，本地ip和本地端口采用系统规默认。
- Socket(InetAddress/String remoteAddress,in port, InetAddress/String localAddress, in localPort):创建连接到指定远程主机，端口的Socket，并指定本地主机和端口。

服务端通过ServerSockete创建TCP通信服务端。

- ServerSocket(int port)：用指定的端口port来创建一个ServerSocket
- Socket accept():如果接收到一个客户端的请求，该方法返回一个与客户端Socket对应的Socket。

基本的服务端和客户端socket创建和通信

```
public class Server {
    public static void main(String[] args) throws IOException {
       ServerSocket serverSocket = new ServerSocket(30000);
```

```
        while(true){
            Socket s = serverSocket.accept();
            PrintStream ps = new PrintStream(s.getOutputStream());
            ps.println("get message from server");
            ps.close();
            s.close();
        }

    }
}


public class Client {
    public static void main(String[] args) throws IOException {
        /*
        也可以创建无连接的socket，然后进行连接。
         Socket socket = new Socket();
         socket.connect(new InetAddress(host,port),1000)
         */
        Socket socket = new Socket("127.0.0.1", 30000);
        socket.setSoTimeout(1000);

        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader((socket.getInputStream())));
        String s = bufferedReader.readLine();
        System.out.println(s);
        bufferedReader.close();
        socket.close();

    }
}
```

多线程的socket编程

http和https的区别

> 在**http**（超文本传输协议）下加入ssl协议(secure socket layer)，依靠证书来验证服务器的身份，并未客户端和服务器间额通信加密

SSL协议位于TCP/IP协议与各种应用层协议之间，为数据通讯提供安全支持。SSL协议可分为两层：

- SSL记录协议（SSL Record Protocol）：它建立在可靠的传输协议（如TCP）之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。
- SSL握手协议（SSL Handshake Protocol）：它建立在SSL记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

工作步骤

- 客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

- Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。
- 客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。
- 客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。
- Web服务器利用自己的私钥解密出会话密钥。
- Web服务器利用会话密钥加密与客户端之间的通信。

加密方法

- DES加密算法（对称加密）：该算法是一个利用56+8奇偶校验位（第8,16,24,32,40,48,56,64）=64位的密钥对以64位为单位的块数据进行加解密。先按8*7的规则表进行位交换，奇偶校验，然后进行左移的操作，生成16个子钥。用得到的子密钥再对它进行加密。生成明文
- 3DES加密（对称加密）：3DES加加密过程为：C=Ek3(Dk2(Ek1(M)))，其中EK就代表了DES算法的加密过程，也就是进行3次DES加密，简单增加DES的密钥长度来避免攻击。
- AES加密（对称加密）：以字节作为输入，对应密钥长度和加密轮数，将明文转换为状态矩阵，然后状态矩阵的内容不断变化（逆行移位，逆字节代换-查表变换，伦密加钥-逐位异或操作，轮列混合-矩阵相乘）最终生成明文
- RSA（非对称加密） 通过选取很大的素数，进行互质等操作，分别得到秘钥和公钥，并对原始的素数丢弃。加密方式是将加密数据分成等长数据块，利用秘钥计算加密。

适用场景

- 由于非对称加密算法的运行速度比对称加密算法的速度慢很多，当我们需要加密大量的数据时，建议采用对称加密算法，提高加解密速度。
- 对称加密算法不能实现签名，因此签名只能非对称算法。
- 由于对称加密算法的密钥管理是一个复杂的过程，密钥的管理直接决定着他的安全性，因此当数据量很小时，我们可以考虑采用非对称加密算法。 在实际的操作过程中，我们通常采用的方式是：采用非对称加密算法管理对称算法的密钥，然后用对称加密算法加密数据，这样我们就集成了两类加密算法的优点，既实现了加密速度快的优点，又实现了安全方便管理密钥的优点。
- 如果在选定了加密算法后，那采用多少位的密钥呢？一般来说，密钥越长，运行的速度就越慢，应该根据的我们实际需要的安全级别来选择，一般来说，RSA建议采用1024位的数字，ECC建议采用160位，AES采用128为即可