# 1、基本概念

为什么会有TCP/IP协议

在世界上各地，各种各样的电脑运行着各自不同的[**操作系统**](http://lib.csdn.net/base/operatingsystem)为大家服务，这些电脑在表达同一种信息的时候所使用的方法是千差万别。就好像圣经中上帝打乱了各地人的口音，让他们无法合作一样。计算机使用者意识到，计算机只是单兵作战并不会发挥太大的作用。只有把它们联合起来，电脑才会发挥出它最大的潜力。于是人们就想方设法的用电线把电脑连接到了一起。

但是简单的连到一起是远远不够的，就好像语言不同的两个人互相见了面，完全不能交流信息。因而他们需要定义一些共通的东西来进行交流，TCP/IP就是为此而生。TCP/IP不是一个协议，而是一个协议族的统称。里面包括了IP协议，IMCP协议，TCP协议，以及我们更加熟悉的http、ftp、pop3协议等等。电脑有了这些，就好像学会了外语一样，就可以和其他的计算机终端做自由的交流了。

TCP/IP协议分层

提到协议分层，我们很容易联想到ISO-OSI的七层协议经典[**架构**](http://lib.csdn.net/base/architecture)，但是TCP/IP协议族的结构则稍有不同。如图所示



TCP/IP协议族按照层次由上到下，层层包装。最上面的就是应用层了，这里面有http，ftp,等等我们熟悉的协议。而第二层则是传输层，著名的TCP和UDP协议就在这个层次（不要告诉我你没用过udp玩星际）。第三层是网络层，IP协议就在这里，它负责对数据加上IP地址和其他的数据（后面会讲到）以确定传输的目标。第四层是叫数据链路层，这个层次为待传送的数据加入一个以太网协议头，并进行CRC编码，为最后的数据传输做准备。再往下则是硬件层次了，负责网络的传输，这个层次的定义包括网线的制式，网卡的定义等等（这些我们就不用关心了，我们也不做网卡），所以有些书并不把这个层次放在tcp/ip协议族里面，因为它几乎和tcp/ip协议的编写者没有任何的关系。发送协议的主机从上自下将数据按照协议封装，而接收数据的主机则按照协议从得到的数据包解开，最后拿到需要的数据。这种结构非常有栈的味道，所以某些文章也把tcp/ip协议族称为tcp/ip协议栈。

一些基本的常识

在学习协议之前，我们应该具备一些基本知识。

            互联网地址(ip地址)

网络上每一个节点都必须有一个独立的Internet地址（也叫做IP地址）。现在，通常使用的IP地址是一个32bit的数字，也就是我们常说的IPv4标准，这32bit的数字分成四组，也就是常见的255.255.255.255的样式。IPv4标准上，地址被分为五类，我们常用的是B类地址。具体的分类请参考其他文档。需要注意的是IP地址是网络号+主机号的组合，这非常重要。

            域名系统

域名系统是一个分布的[**数据库**](http://lib.csdn.net/base/mysql)，它提供将主机名（就是网址啦）转换成IP地址的服务。

            RFC

RFC是什么？RFC就是tcp/ip协议的标准文档，在[这里](http://www.ietf.org/iesg/1rfc_index.txt)我们可以看到RFC那长长的定义列表，现在它一共有4000多个协议的定义，当然，我们所要学习的，也就是那么十几个协议而已。

            端口号(port)

注意，这个号码是用在TCP，UDP上的一个逻辑号码，并不是一个硬件端口，我们平时说把某某端口封掉了，也只是在IP层次把带有这个号码的IP包给过滤掉了而已。

            应用编程接口

现在常用的编程接口有socket和TLI。而前面的有时候也叫做“Berkeley socket”，可见Berkeley对于网络的发展有多大的贡献。

# [2、数据链路层](http://blog.csdn.net/goodboy1881/archive/2006/04/16/665061.aspx)

 数据链路层有三个目的：

            为IP模块发送和 接收IP数据报。

            为ARP模块发送ARP请求和接收ARP应答。

            为RARP发送RARP请 求和接收RARP应答

ip大家都听说过。至于ARP和RARP，ARP叫做地址解析协议，是用IP地址换MAC地址的一种协议，而RARP则叫做逆地址解析协议，在tcp/ip协议的后面章节会介绍它们（在局域网里面用ARP协议可以很容易的搞瘫痪网络哦）

数据链路层的协议还是很多的，有我们最常用的以太网（就是平时我们用的网卡）协议，也有不太常见的令牌环，还有FDDI，当然，还有国内现在相当普及的PPP协议（就是adsl宽带），以及一个loopback协议。

联系[**Linux**](http://lib.csdn.net/base/linux)里面的ifconfig -a命令，这个命令通常会得到如下的结果

eth0 Link encap:Ethernet HWaddr 00:01:4A:03:5B:ED  
inet addr:192.168.11.2 Bcast:192.168.11.255 Mask:255.255.255.0  
inet6 addr: fe80::201:4aff:fe03:5bed/64 Scope:Link  
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
RX packets:2819 errors:0 dropped:0 overruns:0 frame:0  
TX packets:76 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:1000  
RX bytes:241609 (235.9 KiB) TX bytes:9596 (9.3 KiB)

lo Link encap:Local Loopback  
inet addr:127.0.0.1 Mask:255.0.0.0  
inet6 addr: ::1/128 Scope:Host  
UP LOOPBACK RUNNING MTU:16436 Metric:1  
RX packets:2713 errors:0 dropped:0 overruns:0 frame:0  
TX packets:2713 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:0  
RX bytes:3516032 (3.3 MiB) TX bytes:3516032 (3.3 MiB)

其中，eth0就是以太网接口，而lo则是loopback接口。这也说明这个主机在网络链路层上至少支持loopback协议和以太网协议。

以太网（Ether-net）的定是指数字设备公司（ Digital Equipment Corp.）、英特尔公司（Intel Corp.）和Xerox公司在1982年联合公布的一个标准，这个标准里面使用了一种称作CSMA/CD的接入方法。而IEEE802提供的标准集802.3(还有一部分定义到了802.2中)也提供了一个CSMA/CD的标准。这两个标准稍有不同，TCP/IP协议对这种情况的处理方式如下:

            以太网的IP数据报封装在RFC894中定义，而IEEE802网络的IP数据报封装在RFC1042中定义。

            一台主机一定要能发送和接收RFC894定义的数据报。

            一台主机可以接收RFC894和RFC1042的封装格式的混合数据报。

            一台主机也许能够发送RFC1042数据报。。如果主机能同时发送两种类型的分组数 据，那么发送的分组必须是可以设置的，而且默认条件下必须是RFC 894分组。

可见，RFC1042在TCP/IP里面处于一个配角的地位。这两种不同的数据报格式请参考教材。

ppp(点对点协议)是从SLIP的替代品。他们都提供了一种低速接入的解决方案。而每一种数据链路层协议，都有一个MTU（最大传输单元）定义，在这个定义下面，如果IP数据报过大，则要进行分片(fragmentation)，使得每片都小于MTU，注意PPP的MTU并不是一个物理定义，而是指一个逻辑定义（个人认为就是用程序控制）。可以用netstat来打印出MTU的结果，比如键入netstat -in

Kernel Interface table  
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg  
eth0       1500   0     1774      0      0      0      587      0      0      0 BMRU  
lo        16436   0     2667      0      0      0     2667      0      0      0 LRU

就可以观察到eth0的MTU是1500。而lo（环回接口）的MTU则是16436。

最后说说那个环回接口（loopback）。平时我们用127.0.0.1来尝试自己的机器服务器好使不好使。走的就是这个loopback接口。对于环回接口，有如下三点值得注意:

            传给环回地址（一般是127.0.0.1）的任何数据均作为I P输入。

            传给广播地址或多播地址的数据报复制一份传给环回接口，然后送到以太网上。这是 因为广播传送和多播传送的定义包含主机本身。

            任何传给该主机IP地址的数据均送到环回接口。

# 3、TCP/IP详解之IP协议ARP协议和RARP协议

 　　把这三个协议放到一起学习是因为这三个协议处于同一层，ARP协议用来找到目标主机的Ethernet网卡Mac地址，IP则承载要发送的消息。数据链路层可以从ARP得到数据的传送信息，而从IP得到要传输的数据信息。

　　1.IP协议

　　IP协议是TCP/IP协议的核心，所有的TCP，UDP，IMCP，IGCP的数据都以IP数据格式传输。要注意的是，IP不是可靠的协议，这是说，IP协议没有提供一种数据未传达以后的处理机制--这被认为是上层协议--TCP或UDP要做的事情。所以这也就出现了TCP是一个可靠的协议，而UDP就没有那么可靠的区别。这是后话，暂且不提

　　1.1.IP协议头

　　如图所示



　TTL字段，规定该数据包在穿过多少个路由之后才会被抛弃(这里就体现出来IP协议包的不可靠性，它不保证数据被送达)，某个ip数据包每穿过一个路由器，该数据包的TTL数值就会减少1，当该数据包的TTL成为零，它就会被自动抛弃。这个字段的最大值也就是255，也就是说一个协议包也就在路由器里面穿行255次就会被抛弃了，根据系统的不同，这个数字也不一样，一般是32或者是64，Tracerouter这个工具就是用这个原理工作的，tranceroute的-m选项要求最大值是255，也就是因为这个TTL在IP协议里面只有8bit。

　　现在的ip版本号是4，所以也称作IPv4。现在还有IPv6，而且运用也越来越广泛了。

　　1.2.IP路由选择

　　当一个IP数据包准备好了的时候，IP数据包(或者说是路由器)是如何将数据包送到目的地的呢?它是怎么选择一个合适的路径来"送货"的呢?

　　最特殊的情况是目的主机和主机直连，那么主机根本不用寻找路由，直接把数据传递过去就可以了。至于是怎么直接传递的，这就要靠ARP协议了，后面会讲到。

　　稍微一般一点的情况是，主机通过若干个路由器(router)和目的主机连接。那么路由器就要通过ip包的信息来为ip包寻找到一个合适的目标来进行传递，比如合适的主机，或者合适的路由。路由器或者主机将会用如下的方式来处理某一个IP数据包

　　如果IP数据包的TTL(生命周期)以到，则该IP数据包就被抛弃。

　　搜索路由表，优先搜索匹配主机，如果能找到和IP地址完全一致的目标主机，则将该包发向目标主机

　　搜索路由表，如果匹配主机失败，则匹配同子网的路由器，这需要“子网掩码(1.3.)”的协助。如果找到路由器，则将该包发向路由器。

　　搜索路由表，如果匹配同子网路由器失败，则匹配同网号(第一章有讲解)路由器，如果找到路由器，则将该包发向路由器。

　　搜索陆游表，如果以上都失败了，就搜索默认路由，如果默认路由存在，则发包

　　如果都失败了，就丢掉这个包。

　　这再一次证明了，ip包是不可靠的。因为它不保证送达。

　　1.3.子网寻址

　　IP地址的定义是网络号+主机号。但是现在所有的主机都要求子网编址，也就是说，把主机号在细分成子网号+主机号。最终一个IP地址就成为 网络号码+子网号+主机号。例如一个B类地址：210.30.109.134。一般情况下，这个IP地址的红色部分就是网络号，而蓝色部分就是子网号，绿色部分就是主机号。至于有多少位代表子网号这个问题上，这没有一个硬性的规定，取而代之的则是子网掩码，校园网相信大多数人都用过，在校园网的设定里面有一个255.255.255.0的东西，这就是子网掩码。子网掩码是由32bit的二进制数字序列,形式为是一连串的1和一连串的0，例如：255.255.255.0(二进制就是11111111.11111111.11111111.00000000)对于刚才的那个B类地址，因为210.30是网络号，那么后面的109.134就是子网号和主机号的组合，又因为子网掩码只有后八bit为0，所以主机号就是IP地址的后八个bit，就是134，而剩下的就是子网号码--109。

　　2. ARP协议

　　还记得数据链路层的以太网的协议中，每一个数据包都有一个MAC地址头么?我们知道每一块以太网卡都有一个MAC地址，这个地址是唯一的，那么IP包是如何知道这个MAC地址的?这就是ARP协议的工作。

　　ARP(地址解析)协议是一种解析协议，本来主机是完全不知道这个IP对应的是哪个主机的哪个接口，当主机要发送一个IP包的时候，会首先查一下自己的ARP高速缓存(就是一个IP-MAC地址对应表缓存)，如果查询的IP-MAC值对不存在，那么主机就向网络发送一个ARP协议广播包，这个广播包里面就有待查询的IP地址，而直接收到这份广播的包的所有主机都会查询自己的IP地址，如果收到广播包的某一个主机发现自己符合条件，那么就准备好一个包含自己的MAC地址的ARP包传送给发送ARP广播的主机，而广播主机拿到ARP包后会更新自己的ARP缓存(就是存放IP-MAC对应表的地方)。发送广播的主机就会用新的ARP缓存数据准备好数据链路层的的数据包发送工作。

　　一个典型的arp缓存信息如下，在任意一个系统里面用“arp -a”命令:

　　Interface: 192.168.11.3 --- 0x2

　　Internet Address Physical Address Type

　　192.168.11.1 00-0d-0b-43-a0-2e dynamic

　　192.168.11.2 00-01-4a-03-5b-ed dynamic

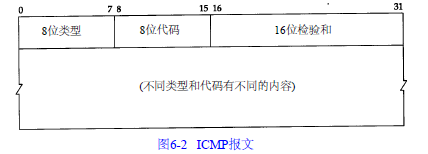
　　都会得到这样的结果。

　　这样的高速缓存是有时限的，一般是20分钟(伯克利系统的衍生系统)。

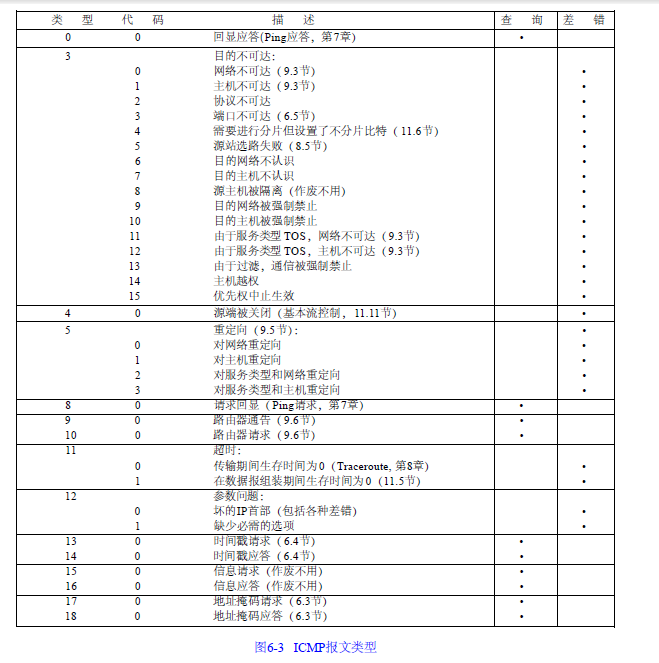
# [4、ICMP协议，ping和Traceroute](http://blog.csdn.net/goodboy1881/archive/2006/04/20/670761.aspx)

 1.IMCP协议介绍

前面讲到了，IP协议并不是一个可靠的协议，它不保证数据被送达，那么，自然的，保证数据送达的工作应该由其他的模块来完成。其中一个重要的模块就是ICMP(网络控制报文)协议。



当传送IP数据包发生错误－－比如主机不可达，路由不可达等等，ICMP协议将会把错误信息封包，然后传送回给主机。给主机一个处理错误的机会，这 也就是为什么说建立在IP层以上的协议是可能做到安全的原因。ICMP数据包由8bit的错误类型和8bit的代码和16bit的校验和组成。而前 16bit就组成了ICMP所要传递的信息。书上的图6－3清楚的给出了错误类型和代码的组合代表的意思。



尽管在大多数情况下，错误的包传送应该给出ICMP报文，但是在特殊情况下，是不产生ICMP错误报文的。如下

1.         ICMP差错报文不会产生ICMP差错报文（出IMCP查询报文）（防止IMCP的无限产生和传送）

2.         目的地址是广播地址或多播地址的IP数据报。

3.         作为链路层广播的数据报。

4.         不是IP分片的第一片。

5.         源地址不是单个主机的数据报。这就是说，源地址不能为零地址、环回地址、广播地 址或多播地址。

虽然里面的一些规定现在还不是很明白，但是所有的这一切规定，都是为了防止产生ICMP报文的无限传播而定义的。

ICMP协议大致分为两类，一种是查询报文，一种是差错报文。其中查询报文有以下几种用途:

1.         ping查询（不要告诉我你不知道ping程序）

2.         子网掩码查询（用于无盘工作站在初始化自身的时候初始化子网掩码）

3.         时间戳查询（可以用来同步时间）

2.ICMP的应用--ping

ping可以说是ICMP的最著名的应用，当我们某一个网站上不去的时候。通常会ping一下这个网站。ping会回显出一些有用的信息。一般的信息如下:

Reply from 10.4.24.1: bytes=32 time<1ms TTL=255  
Reply from 10.4.24.1: bytes=32 time<1ms TTL=255  
Reply from 10.4.24.1: bytes=32 time<1ms TTL=255  
Reply from 10.4.24.1: bytes=32 time<1ms TTL=255  
  
Ping statistics for 10.4.24.1:  
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

ping这个单词源自声纳定位，而这个程序的作用也确实如此，它利用ICMP协议包来侦测另一个主机是否可达。原理是用类型码为0的ICMP发请 求，受到请求的主机则用类型码为8的ICMP回应。ping程序来计算间隔时间，并计算有多少个包被送达。用户就可以判断网络大致的情况。我们可以看到， ping给出来了传送的时间和TTL的数据。我给的例子不太好，因为走的路由少，有兴趣地可以ping一下国外的网站比如sf[**.NET**](http://lib.csdn.net/base/dotnet)，就可以观察到一些 丢包的现象，而程序运行的时间也会更加的长。  
ping还给我们一个看主机到目的主机的路由的机会。这是因为，ICMP的ping请求数据报在每经过一个路由器的时候，路由器都会把自己的ip放到该数 据报中。而目的主机则会把这个ip列表复制到回应icmp数据包中发回给主机。但是，无论如何，ip头所能纪录的路由列表是非常的有限。如果要观察路由， 我们还是需要使用更好的工具，就是要讲到的Traceroute(windows下面的名字叫做tracert)。

3.ICMP的应用--Traceroute

Traceroute是用来侦测主机到目的主机之间所经路由情况的重要工具，也是最便利的工具。前面说到，尽管ping工具也可以进行侦测，但是，因为ip头的限制，ping不能完全的记录下所经过的路由器。所以Traceroute正好就填补了这个缺憾。

Traceroute的原理是非常非常的有意思，它受到目的主机的IP后，首先给目的主机发送一个TTL=1（还记得TTL是什么吗？）的UDP(后面就 知道UDP是什么了)数据包，而经过的第一个路由器收到这个数据包以后，就自动把TTL减1，而TTL变为0以后，路由器就把这个包给抛弃了，并同时产生 一个主机不可达的ICMP数据报给主机。主机收到这个数据报以后再发一个TTL=2的UDP数据报给目的主机，然后刺激第二个路由器给主机发ICMP数据 报。如此往复直到到达目的主机。这样，traceroute就拿到了所有的路由器ip。从而避开了ip头只能记录有限路由IP的问题。

有人要问，我怎么知道UDP到没到达目的主机呢？这就涉及一个技巧的问题，TCP和UDP协议有一个端口号定义，而普通的网络程序只监控少数的几个号码较 小的端口，比如说80,比如说23,等等。而traceroute发送的是端口号>30000(真变态)的UDP报，所以到达目的主机的时候，目的 主机只能发送一个端口不可达的ICMP数据报给主机。主机接到这个报告以后就知道，主机到了，所以，说Traceroute是一个骗子一点也不为过:)

# 5、IP选路，动态选路，和一些细节

1.静态IP选路

1.1.一个简单的路由表

选路是IP层最重要的一个功能之一。前面的部分已经简单的讲过路由器是通过何种规则来根据IP数据包的IP地址来选择路由。这里就不重复了。首先来看看一个简单的系统路由表。

Destination     Gateway         Genmask         Flags Metric Ref    Use Iface  
192.168.11.0    \*               255.255.255.0   U     0      0        0 eth0  
169.254.0.0     \*               255.255.0.0     U     0      0        0 eth0  
default         192.168.11.1    0.0.0.0         UG    0      0        0 eth0

对于一个给定的路由器，可以打印出五种不同的flag。

1.         U表明该路由可用。

2.         G表明该路由是到一个网关。如果没有这个标志，说明和Destination是直连的，而相应的Gateway应该直接给出Destination的地址。

3.         H表明该路由是到一个主机，如果没有该标志，说明Destination是一个网络，换句话说Destination就应该写成一个网络号和子网号的组合，而不包括主机号(主机号码处为0)，例如 192.168.11.0

4.         D表明该路由是为重定向报文创建的

5.         M该路由已经被重定向报文修改

U没啥可说的，G说明这是一个网关，如果你要发数据给Destination，IP头应该写Destination的IP地址，而数据链路层的MAC地址就应该是GateWay的Mac地址了；反之，如果没有G标志，那么数据链路层和IP层的地址应该是对应的。H说明了Destination的性质，如果是H的，则说明该地址是一个完整的地址，既有网络号又有主机号，那么再匹配的时候就既要匹配网络号，又要匹配主机号；反之，Destination就代表一个网络，在匹配的时候只要匹配一下网络号就可以了。

这样，IP选路的方式就可以更加具体化了。如下

1.         首先用IP地址来匹配那些带H标志的DestinationIP地址。

2.         如果1失败就匹配那些网络地址。

3.         如果2失败就发送到Default网关

顺便提一下那个GenMask（还记得子网掩码么），它指定了目的地址的子网号，例如第一条的子网就是11。

1.2.其他有关路由表的知识

一般，我们在配置好一个网络接口的时候，一个路由就被直接创建好了。当然我们也可以手动添加路由。用route add命令就可以了。

而当一个IP包在某一个路由器的时候发现没有路由可走，那么该路由器就会给源主机发送“主机不可达”或者“网络不可达”的ICMP包来报错。

注意，一般的[**操作系统**](http://lib.csdn.net/base/operatingsystem)默认是没有路由功能的，这需要自己配置。

1.3.ICMP的IP重定向报文和路由发现报文

当IP包在某一个地方转向的时候，都回给发送IP报的源主机一个ICMP重定向报文，而源主机就可以利用这个信息来更新自己的路由表，这样，随着网络通信的逐渐增多，路由表也就越来越完备，数据转发的速度也会越来越快。我们需要注意的是：

1.         重定向报文只能由路由器发出。

2.         重定向报文为主机所用，而不是为路由器所用。

在主机引导的时候，一般会发送在网内广播一个路由请求的ICMP报文，而多个路由器则会回应一个路由通告报文。而且，路由其本身不定期的在网络内发布路由通告报文，这样，根据这些报文，每一个主机都会有机会建立自己的路由表而实现网络通信。路由器在一份通告报文中可以通告多个地址，并且给出每一个地址的优先等级，这个优先等级是该IP作为默认路由的等级，至于怎么算的就不深究了。

路由器一般会在450-600秒的时间间隔内发布一次通告，而一个给定的通告报文的寿命是30分钟。而主机在引导的时候会每三秒发送一次请求报文，一旦接受到一个有效的通告报文，就停止发送请求报文。

在TCP/IP详解编写的时候，只有Solaris2.x支持这两种报文，大多数系统还不支持这两种报文。

动态选路协议

前面的选路方法叫做静态选路，简要地说就是在配置接口的时候，以默认的方式生成路由表项。并通过route来增加表项，或者通过ICMP报文来更新表项（通常在默认方式出错的情况下）。 而如果上诉三种方法都不能满足，那么我们就使用动态选路。

动态选路协议是用于动态选路的重要组成部分，但是他们只是使用在路由器之间，相邻路由器之间互相通信。系统（路有选择程序）选择比较合适的路有放到核心路由表中，然后系统就可以根据这个核心路有表找到最合适的网路。也就是说，动态选路是在系统核心网络外部进行的，它只是用一些选路的策略影响路由表，而不会影响到最后通过路由表选择路由的那一部分。选路协议有一大类常用的叫做内部网关协议(IGP)，而在IGP中，RIP就是其中最重要的协议。一种新的IGP协议叫做开放最短路经优先(OSPF)协议，其意在取代RIP。另一种最早用在网路骨干网上的IGP协议--HELLO，现在已经不用了。

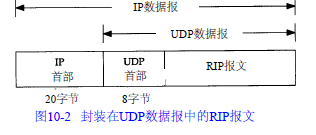
如今，任何支持动态选路的路由器都必须同时支持OSPF和RIP，还可以选择性的支持其他的IGP协议。

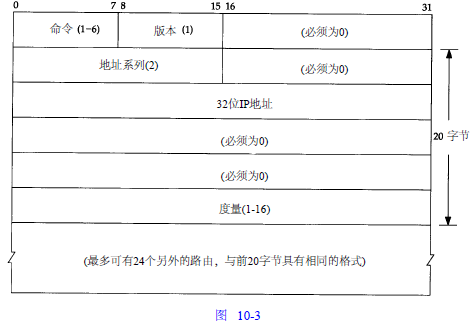
2.1.Unix选路程序

Unix系统上面通常都有路由守护程序－－routed。还有一个叫做gate。gate所支持的协议要比routed多，routed只是支持RIPv1版本。而gate则支持RIPv1、v2，BGPv1 等等。

2.1.RIP：选路信息协议

它的定义可以在RFC1058内找到，这种协议使用UDP作为载体（也就是UDP的上层协议）。我们最关心的就是RIP其中的一个段，叫做度量的段，这是一个以hop作为计数器（就是以走过多少路由为计数器）的段（IP协议里面也有一个TTL不是么）。这个度量段将最终影响到路由表的建立。参考图:





一般说来routed要承担如下的工作：

1.         给每一个已知的路由器发送rip请求报文，要求其他路由器给出完整的路由表。这种报文的命令字段为1，地址字段为0，度量地段为16（相当于无穷大）。

2.         接受请求，如果接收到刚才的那个请求，就把自己的完整的路由表交给请求者。如果没有，就处理IP请求表项，把表项中自己有的部分添上跳数，没有的部分添上16。然后发给请求者。

3.         接受回应。更新自己的路由表。使用hop数小的规则。

4.         定期更新路由表，一般是30s(真频繁)给相邻的路有启发一次自己的路由表。这种形式可以使广播形式的。

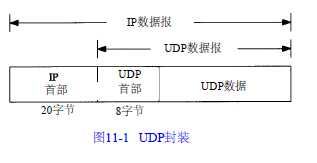
这个协议看起来会工作的很好，但是，这里面其实有很多隐藏的忧患，比如说RIP没有子网的概念，比如说环路的危险。而且hop数的上限也限制了网络的大小。

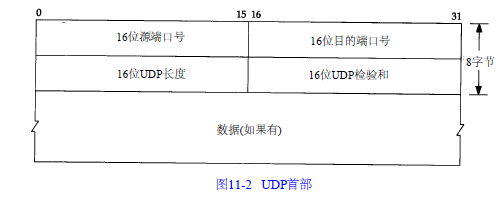
因此，出现了很多RIPv1的替代品，比如说RIPv2,比如说OSPF。他们都是通过某种策略来影响路由表，所以就不说了。

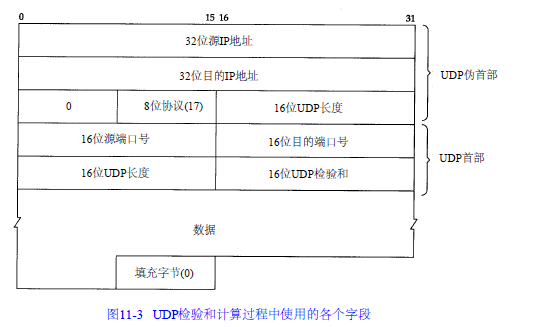
# 6、UDP协议

 1.UDP简要介绍

UDP是传输层协议，和TCP协议处于一个分层中，但是与TCP协议不同，UDP协议并不提供超时重传，出错重传等功能，也就是说其是不可靠的协议。







2.UDP协议头

2.1.UDP端口号

由于很多软件需要用到UDP协议，所以UDP协议必须通过某个标志用以区分不同的程序所需要的数据包。端口号的功能就在于此，例如某一个UDP程序A在系统中注册了3000端口，那么，以后从外面传进来的目的端口号为3000的UDP包都会交给该程序。端口号理论上可以有2^16这么多。因为它的长度是16个bit

2.2.UDP检验和

这是一个可选的选项，并不是所有的系统都对UDP数据包加以检验和数据(相对TCP协议的必须来说)，但是RFC中标准要求，发送端应该计算检验和。

UDP检验和覆盖UDP协议头和数据，这和IP的检验和是不同的，IP协议的检验和只是覆盖IP数据头，并不覆盖所有的数据。UDP和TCP都包含一个伪首部，这是为了计算检验和而摄制的。伪首部甚至还包含IP地址这样的IP协议里面都有的信息，目的是让UDP两次检查数据是否已经正确到达目的地。如果发送端没有打开检验和选项，而接收端计算检验和有差错，那么UDP数据将会被悄悄的丢掉（不保证送达），而不产生任何差错报文。

2.3.UDP长度

UDP可以很长很长，可以有65535字节那么长。但是一般网络在传送的时候，一次一般传送不了那么长的协议（涉及到MTU的问题），就只好对数据分片，当然，这些是对UDP等上级协议透明的，UDP不需要关心IP协议层对数据如何分片。

3.IP分片

IP在从上层接到数据以后，要根据IP地址来判断从那个接口发送数据（通过选路），并进行MTU的查询，如果数据大小超过MTU就进行数据分片。数据的分片是对上层和下层透明，而数据也只是到达目的地还会被重新组装，不过不用担心，IP层提供了足够的信息进行数据的再组装。

在IP头里面，16bit识别号唯一记录了一个IP包的ID,具有同一个ID的IP片将会被重新组装；而13位片偏移则记录了某IP片相对整个包的位置；而这两个表示中间的3bit标志则标示着该分片后面是否还有新的分片。这三个标示就组成了IP分片的所有信息，接受方就可以利用这些信息对IP数据进行重新组织（就算是后面的分片比前面的分片先到，这些信息也是足够了）。

因为分片技术在网络上被经常的使用，所以伪造IP分片包进行流氓攻击的软件和人也就层出不穷。

可以用Trancdroute程序来进行简单的MTU侦测。请参看教材。

3.UDP和ARP之间的交互式用

这是不常被人注意到的一个细节，这是针对一些系统地实现来说的。当ARP缓存还是空的时候。UDP在被发送之前一定要发送一个ARP请求来获得目的主机的MAC地址，如果这个UDP的数据包足够大，大到IP层一定要对其进行分片的时候，想象中，该UDP数据包的第一个分片会发出一个ARP查询请求，所有的分片都辉等到这个查询完成以后再发送。事实上是这样吗？

结果是，某些系统会让每一个分片都发送一个ARP查询，所有的分片都在等待，但是接受到第一个回应的时候，主机却只发送了最后一个数据片而抛弃了其他，这实在是让人匪夷所思。这样，因为分片的数据不能被及时组装，接受主机将会在一段时间内将永远无法组装的IP数据包抛弃，并且发送组装超时的ICMP报文（其实很多系统不产生这个差错），以保证接受主机自己的接收端缓存不被那些永远得不到组装的分片充满。

4.ICMP源站抑制差错

当目标主机的处理速度赶不上数据接收的速度，因为接受主机的IP层缓存会被占满，所以主机就会发出一个“我受不了”的一个ICMP报文。

5.UDP服务器设计

UDP协议的某些特性将会影响我们的服务器程序设计，大致总结如下：

1.         关于客户IP和地址：服务器必须有根据客户IP地址和端口号判断数据包是否合法的能力（这似乎要求每一个服务器都要具备）

2.         关于目的地址：服务器必须要有过滤广播地址的能力。

3.         关于数据输入：通常服务器系统的每一个端口号都会和一块输入缓冲区对应，进来的输入根据先来后到的原则等待服务器的处理，所以难免会出现缓冲区溢出的问题，这种情况下，UDP数据包可能会被丢弃，而应用服务器程序本身并不知道这个问题。

4.         服务器应该限制本地IP地址，就是说它应该可以把自己绑定到某一个网络接口的某一个端口上。

# [7、广播和多播，IGMP协议](http://blog.csdn.net/goodboy1881/archive/2006/05/13/726860.aspx)

 1.单播，多播，广播的介绍

1.1.单播(unicast)

单播是说，对特定的主机进行数据传送。例如给某一个主机发送IP数据包。这时候，数据链路层给出的数据头里面是非常具体的目的地址，对于以太网来 说，就是网卡的MAC地址（不是FF-FF-FF-FF-FF-FF这样的地址）。现在的具有路由功能的主机应该可以将单播数据定向转发，而目的主机的网 络接口则可以过滤掉和自己MAC地址不一致的数据。

1.2.广播(unicast)

广播是主机针对某一个网络上的所有主机发送数据包。这个网络可能是网络，可能是子网，还可能是所有的子网。如果是网络，例如A类网址的广播就是 netid.255.255.255，如果是子网，则是netid.netid.subnetid.255；如果是所有的子网（B类IP）则是则是 netid.netid.255.255。广播所用的MAC地址FF-FF-FF-FF-FF-FF。网络内所有的主机都会收到这个广播数据，网卡只要把 MAC地址为FF-FF-FF-FF-FF-FF的数据交给内核就可以了。一般说来ARP，或者路由协议RIP应该是以广播的形式播发的。

1.3.多播(multicast)

可以说广播是多播的特例，多播就是给一组特定的主机（多播组）发送数据，这样，数据的播发范围会小一些(实际上播发的范围一点也没有变小)，多播的MAC地址是最高字节的低位为一，例 如01-00-00-00-00-00。多播组的地址是D类IP，规定是224.0.0.0-239.255.255.255。

虽然多播比较特殊，但是究其原理，多播的数据还是要通过数据链路层进行MAC地址绑定然后进行发送。所以一个以太网卡在绑定了一个多播IP地址之后，必 定还要绑定一个多播的MAC地址，才能使得其可以像单播那样工作。这个多播的IP和多播MAC地址有一个对应的[**算法**](http://lib.csdn.net/base/datastructure)，在书的p133到p134之间。可以看到 这个对应不是一一对应的，主机还是要对多播数据进行过滤。

个人的看法：广播和多播的性质是一样的，路由器会把数据放到局域网里面，然后网卡对这些数据进行过滤，只拿到自己打算要的数据，比如自己感兴趣的多 播数据，自己感兴趣的组播数据。当一个主机运行了一个处理某一个多播IP的进程的时候，这个进程会给网卡绑定一个虚拟的多播mac地址，并做出来一个多播 ip。这样，网卡就会让带有这个多播mac地址的数据进来，从而实现通信，而那些没有监听这些数据的主机就会把这些数据过滤掉，换句话说，多播，是让主机 的内核轻松了，而网卡，对不起，您就累点吧。

一些文章也印证了这种想法，最明显的就是[局域网监听的原理、实现与防范](http://news.softhouse.com.cn/news/show/440.html)

2.一些验证性实验

这些实验并不是很复杂，我们只是要ping一下一般的ip和一个广播地址。首先我ping一下自己所在的子网的某一台主机:

Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time=1ms TTL=255

可以看到，机器返回的是一台主机的回应结果，进而推测，如果我ping一个广播地址呢？结果如下

Reply from 192.168.11.9: bytes=32 time=1ms TTL=255  
Reply from 192.168.11.174: bytes=32 time<1ms TTL=64  
Reply from 192.168.11.174: bytes=32 time<1ms TTL=64  
Reply from 192.168.11.174: bytes=32 time<1ms TTL=64  
Reply from 192.168.11.218: bytes=32 time<1ms TTL=64  
Reply from 192.168.11.174: bytes=32 time<1ms TTL=64

可以看到，ping返回了一些随机的ip的结果，这些ip都是与主机在同一子网内的ip。我们可以看到，广播实际上是给处于子网内的所有ip发信。

再来一个多播的例子，但是要实现这个多播并不容易，因为我不知道网络内有多少个多播组，就只好利用几个特殊的多播地址来验证了。

对于多播地址，有几个特殊的多播地址被占用，他们是

1.         224.0.0.1--该子网内所有的系统组。

2.         224.0.0.2--该子网内所有的路由器。

3.         224.0.1.1--网络实现协议NTP专用IP。

4.         224.0.0.9--RIPv2专用IP

所以只要ping这几个IP，就应该能得到一些结果，比如说我ping 224.0.0.2。

Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255  
Reply from 192.168.11.1: bytes=32 time<1ms TTL=255

我们可以看到，这回ping只返回了一个ip的回应。而这个就是我的网关的地址，这也验证了224.0.0.2是所有路由器的多播(组播)地址

3.IGMP协议

IGMP的作用在于，让其他所有需要知道自己处于哪个多播组的主机和路由器知道自己的状态。一般多播路由器根本不需要知道某一个多播组里面有多少个主机，而只要知道自己的子网内还有没有处于某个多播组的主机就可以了。只要某一个多播组还有一台主机，多播路由器就会把数据传输出去，这样，接受方就会通过网卡过滤功能来得到自己想要的数据。为了知道多播组的信息，多播路由器需要定时的发送IGMP查询，IGMP的格式可以看书，各个多播组里面的主机要根据查询来回复自己的状态。路由器来决定有几个多播组，自己要对某一个多播组发送什么样的数据。

这种查询回应数据报的TTL一般是1，而且就算是出错也不产生ICMP差错（没必要）。

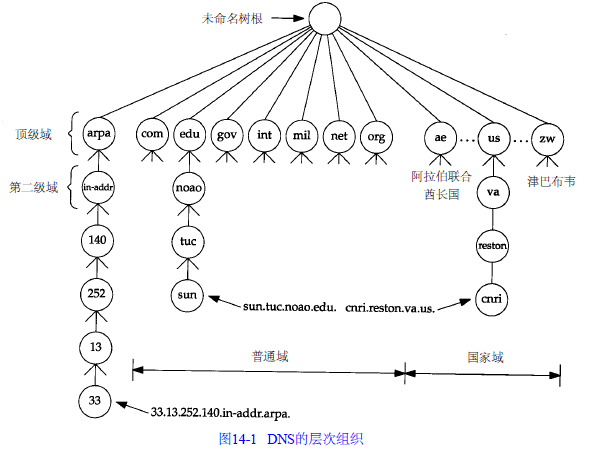
# [8、DNS域名系统](http://blog.csdn.net/goodboy1881/archive/2006/05/18/743816.aspx)

前面已经提到了访问一台机器要靠IP地址和MAC地址，其中，MAC地址可以通过ARP协议得到，所以这对用户是透明的，但是IP地址就不行，无论如何用户都需要用一个指定的IP来访问一台计算机，而IP地址又非常不好记，于是就出现了DNS系统

1.DNS系统介绍

DNS的全称是Domain Name System。它负责把FQDN(就是以"."分隔结尾的名字)翻译成一个IP。最初的DNS系统使用的是一个巨大的hosts.txt文件（很吃惊，用 这个就好使了？），可是一段时间以后，开发这就不得不用[**数据库**](http://lib.csdn.net/base/mysql)来代替hosts.txt文件，最终发展到了现在的分布式数据库。

DNS系统是一个巨大的树，最上方有一个无名树根，下一层是arpa,com,edu,gov,int,mil，us, cn。等等，其中arpa，是域名反解析树的顶端；而com，edu，等域名本来只用在美国（这就是技术特权啊），但是现在几乎全世界通用；而us， cn，等叫做国家域。这个树里面的域名并不是统一管理的，网络信息中心（NIS）负责分配顶级域合委派其他制定地区域的授权机构。



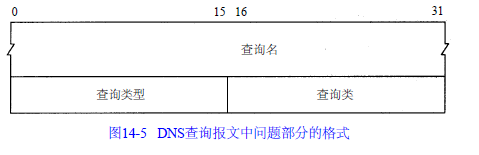
一个独立管理的DNS子树叫做zone，最常见的区域就是二级域名，比如说.com.cn。我们还可以把这个二级域名给划分成更小的区域，比如说sina.com.cn。

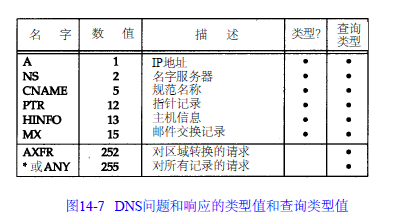
DNS系统是一个分布式的数据库，当一个数据库发现自己并没有某查询所需要的数据的时候，它将把查询转发出去，而转发的目的地通常是根服务器，根服 务器从上至下层层转发查询，直到找到目标为止。DNS还有一个特点就是使用高速缓存，DNS把查询过的数据缓存在某处，以便于下次查询时使用。

2.DNS协议

DNS报文定义了一个既可以查询也可以响应的报文格式。







对各个字段简单解释如下

1.         最前面的16个bit唯一的标示了问题号码，用于查询端区别自己的查询。

2.         紧接着的16个bit又可以做进一步的细分，标示了报文的性质和一些细节，比如说是查询报文还是响应报文，需要递归查询与否（一般服务器都支持递归查询，而且不需要任何设置，BIND就是这样）

3.         查询问题后面有查询类型，包括A，NS，CNAME，PTR，HINFO，MX，如果熟悉BIND的话，就知道在zong的配置文件里面，每一条记录都记载了各自的类型，比如A就是IP地址，NS就是名字服务器。

4.         响应报文可以回复多个IP，也就是说，域名可以和多个IP地址对应，并且有很多CNAME。

3.反向查询

正向查询指的是通过域名得到IP的查询，而反向查询就是通过IP得到域名。例如用host命令，host ip就可以得到服务器的域名，host domainName 就得到IP。

稍微知道一点[**数据结构**](http://lib.csdn.net/base/datastructure)的人都能意识到，在正向查询的域里面做反向查询，其做法只有遍历整个数据集合----对于DNS来说，那就是遍历整个数据库， 这将带来巨大的负担，所以DNS采取了另一种方法，使用另一棵子树来维护IP-〉域名的对应表。这个子树的根节点是in-addr.arpa,而一个IP 例如192.168.11.2)所具有的DNS地址就是 2.11.168.192.in-addr.arpa（ip倒置）。在DNS系统里面，一个反向地址对应一个PTR纪录（对应A纪录），所以反向查询又叫 做指针(PTR)查询。

4.其他问题的讨论

4.1.DNS服务器高速缓存

BIND9默认是作为一个高速缓存服务器，其将所有的查询都转交到根服务器去，然后得到结果并放在本地的缓冲区，以加快查询速度。如果有兴趣可以安装一个BIND9来尝试一下。而自己定义的zone则可以规定其在缓存中的时间，一般是1天（就是配置文件中的1D）。

4.2.用UDP还是TCP

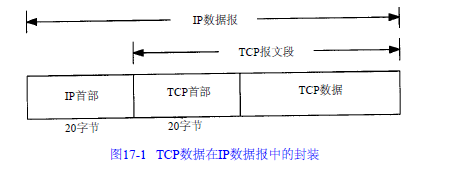
DNS服务器支持TCP和UDP两种协议的查询方式，而且端口都是53。而大多数的查询都是UDP查询的，一般需要TCP查询的有两种情况：

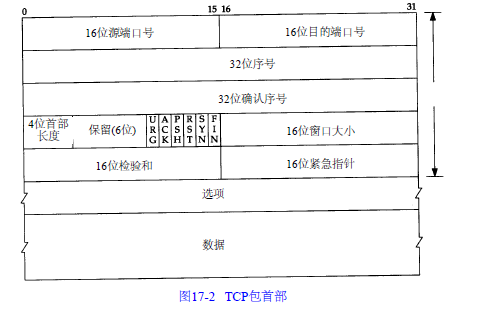
1.         当查询数据多大以至于产生了数据截断(TC标志为1)，这时，需要利用TCP的分片能力来进行数据传输（看TCP的相关章节）。

2.         当主（master）服务器和辅（slave）服务器之间通信，辅服务器要拿到主服务器的zone信息的时候。

# 9、TCP协议概述

TCP和UDP处在同一层---运输层，但是TCP和UDP最不同的地方是，TCP提供了一种可靠的数据传输服务，TCP是面向连接的，也就是说，利用TCP通信的两台主机首先要经历一个“拨打电话”的过程，等到通信准备结束才开始传输数据，最后结束通话。所以TCP要比UDP可靠的多，UDP是把数据直接发出去，而不管对方是不是在收信，就算是UDP无法送达，也不会产生ICMP差错报文。





把TCP保证可靠性的简单工作原理摘抄如下

            应用数据被分割成TCP认为最适合发送的数据块。这和UDP完全不同，应用程序产生的 数据报长度将保持不变。由TCP传递给IP的信息单位称为报文段或段。

            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。在第21章我们将了解TCP协议中自适应的超时 及重传策略。

            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。

            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， T P将丢弃这个报文段和不确认收到此报文段（希望发端超时并重发）。

            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。

            TCP还能提供流量控制。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。

从这段话中可以看到，TCP中保持可靠性的方式就是超时重发，这是有道理的，虽然TCP也可以用各种各样的ICMP报文来处理这些，但是这也不是可靠的，最可靠的方式就是只要不得到确认，就重新发送数据报，直到得到对方的确认为止。

TCP的首部和UDP首部一样，都有发送端口号和接收端口号。但是显然，TCP的首部信息要比UDP的多，可以看到，TCP协议提供了发送和确认所需要的所有必要的信息。TCP数据发送的过程。

            双方建立连接

            发送方给接受方TCP数据报，然后等待对方的确认TCP数据报，如果没有，就重新发，如果有，就发送下一个数据报。

            接受方等待发送方的数据报，如果得到数据报并检验无误，就发送ACK(确认)数据报，并等待下一个TCP数据报的到来。直到接收到FIN(发送完成数据报)

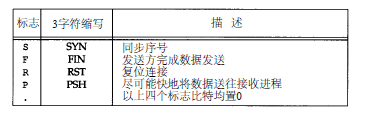
            中止连接

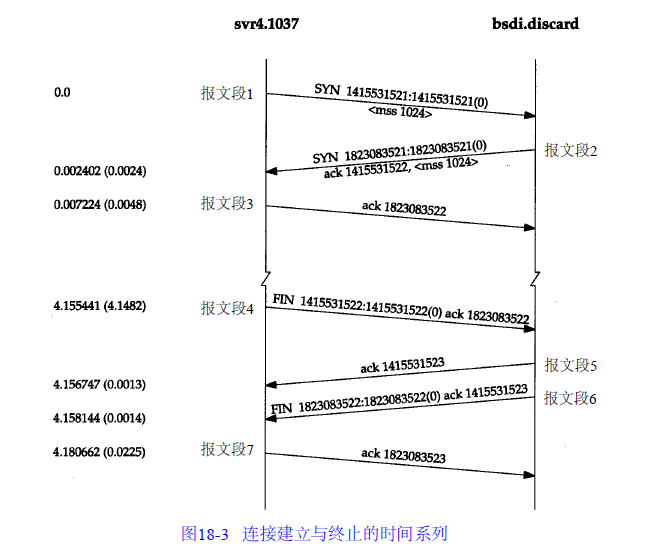
可以想见，为了建立一个TCP连接，系统可能会建立一个新的进程（最差也是一个线程），来进行数据的传送。

# [10、TCP连接的建立与中止](http://blog.csdn.net/goodboy1881/archive/2006/05/18/744534.aspx)

TCP是一个面向连接的协议，所以在连接双方发送数据之前，都需要首先建立一条连接。这和前面讲到的协议完全不同。前面讲的所有协议都只是发送数据而已，大多数都不关心发送的数据是不是送到，UDP尤其明显，从编程的角度来说，UDP编程也要简单的多----UDP都不用考虑数据分片。

书中用telnet登陆退出来解释TCP协议连接的建立和中止的过程，可以看到，TCP连接的建立可以简单的称为三次握手，而连接的中止则可以叫做四次握手。





1.连接的建立

在建立连接的时候，客户端首先向服务器申请打开某一个端口(用SYN段等于1的TCP报文)，然后服务器端发回一个ACK报文通知客户端请求报文收到，客户端收到确认报文以后再次发出确认报文确认刚才服务器端发出的确认报文（绕口么），至此，连接的建立完成。这就叫做三次握手。如果打算让双方都做好准备的话，一定要发送三次报文，而且只需要三次报文就可以了。

可以想见，如果再加上TCP的超时重传机制，那么TCP就完全可以保证一个数据包被送到目的地。

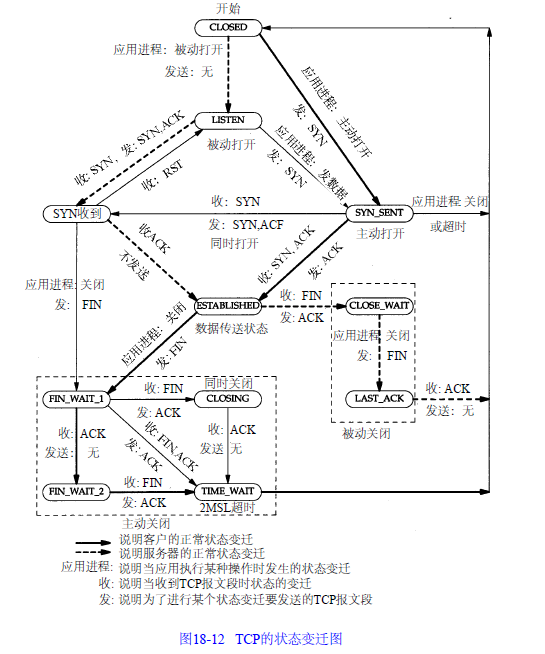
2.结束连接

TCP有一个特别的概念叫做half-close，这个概念是说，TCP的连接是全双工（可以同时发送和接收）连接，因此在关闭连接的时候，必须关闭传和送两个方向上的连接。客户机给服务器一个FIN为1的TCP报文，然后服务器返回给客户端一个确认ACK报文，并且发送一个FIN报文，当客户机回复ACK报文后（四次握手），连接就结束了。

3.最大报文长度

在建立连接的时候，通信的双方要互相确认对方的最大报文长度(MSS)，以便通信。一般这个SYN长度是MTU减去固定IP首部和TCP首部长度。对于一个以太网，一般可以达到1460字节。当然如果对于非本地的IP，这个MSS可能就只有536字节，而且，如果中间的传输网络的MSS更佳的小的话，这个值还会变得更小。

4.TCP的状态迁移图



这是一个看起来比较复杂的状态迁移图，因为它包含了两个部分---服务器的状态迁移和客户端的状态迁移，如果从某一个角度出发来看这个图，就会清晰许多，这里面的服务器和客户端都不是绝对的，发送数据的就是客户端，接受数据的就是服务器。

4.1.客户端应用程序的状态迁移图

客户端的状态可以用如下的流程来表示：

CLOSED->SYN\_SENT->ESTABLISHED->FIN\_WAIT\_1->FIN\_WAIT\_2->TIME\_WAIT->CLOSED

以上流程是在程序正常的情况下应该有的流程，从书中的图中可以看到，在建立连接时，当客户端收到SYN报文的ACK以后，客户端就打开了数据交互地连接。而结束连接则通常是客户端主动结束的，客户端结束应用程序以后，需要经历FIN\_WAIT\_1，FIN\_WAIT\_2等状态，这些状态的迁移就是前面提到的结束连接的四次握手。

4.2.服务器的状态迁移图

服务器的状态可以用如下的流程来表示：

CLOSED->LISTEN->SYN收到->ESTABLISHED->CLOSE\_WAIT->LAST\_ACK->CLOSED

在建立连接的时候，服务器端是在第三次握手之后才进入数据交互状态，而关闭连接则是在关闭连接的第二次握手以后（注意不是第四次）。而关闭以后还要等待客户端给出最后的ACK包才能进入初始的状态。

4.3.其他状态迁移

书中的图还有一些其他的状态迁移，这些状态迁移针对服务器和客户端两方面的总结如下

1.         LISTEN->SYN\_SENT，对于这个解释就很简单了，服务器有时候也要打开连接的嘛。

2.         SYN\_SENT->SYN收到，服务器和客户端在SYN\_SENT状态下如果收到SYN数据报，则都需要发送SYN的ACK数据报并把自己的状态调整到SYN收到状态，准备进入ESTABLISHED

3.         SYN\_SENT->CLOSED，在发送超时的情况下，会返回到CLOSED状态。

4.         SYN\_收到->LISTEN，如果受到RST包，会返回到LISTEN状态。

5.         SYN\_收到->FIN\_WAIT\_1，这个迁移是说，可以不用到ESTABLISHED状态，而可以直接跳转到FIN\_WAIT\_1状态并等待关闭。

4.4.2MSL等待状态

书中给的图里面，有一个TIME\_WAIT等待状态，这个状态又叫做2MSL状态，说的是在TIME\_WAIT2发送了最后一个ACK数据报以后，要进入TIME\_WAIT状态，这个状态是防止最后一次握手的数据报没有传送到对方那里而准备的（注意这不是四次握手，这是第四次握手的保险状态）。这个状态在很大程度上保证了双方都可以正常结束，但是，问题也来了。

由于插口的2MSL状态（插口是IP和端口对的意思，socket），使得应用程序在2MSL时间内是无法再次使用同一个插口的，对于客户程序还好一些，但是对于服务程序，例如httpd，它总是要使用同一个端口来进行服务，而在2MSL时间内，启动httpd就会出现错误（插口被使用）。为了避免这个错误，服务器给出了一个平静时间的概念，这是说在2MSL时间内，虽然可以重新启动服务器，但是这个服务器还是要平静的等待2MSL时间的过去才能进行下一次连接。

4.5.FIN\_WAIT\_2状态

这就是著名的半关闭的状态了，这是在关闭连接时，客户端和服务器两次握手之后的状态。在这个状态下，应用程序还有接受数据的能力，但是已经无法发送数据，但是也有一种可能是，客户端一直处于FIN\_WAIT\_2状态，而服务器则一直处于WAIT\_CLOSE状态，而直到应用层来决定关闭这个状态。

5.RST，同时打开和同时关闭

RST是另一种关闭连接的方式，应用程序应该可以判断RST包的真实性，即是否为异常中止。而同时打开和同时关闭则是两种特殊的TCP状态，发生的概率很小。

6.TCP服务器设计

前面曾经讲述过UDP的服务器设计，可以发现UDP的服务器完全不需要所谓的并发机制，它只要建立一个数据输入队列就可以。但是TCP不同，TCP服务器对于每一个连接都需要建立一个独立的进程（或者是轻量级的，线程），来保证对话的独立性。所以TCP服务器是并发的。而且TCP还需要配备一个呼入连接请求队列（UDP服务器也同样不需要），来为每一个连接请求建立对话进程，这也就是为什么各种TCP服务器都有一个最大连接数的原因。而根据源主机的IP和端口号码，服务器可以很轻松的区别出不同的会话，来进行数据的分发。

# 11、TCP交互数据流，成块数据流

目前建立在TCP协议上的网络协议特别多，有telnet，ssh，有ftp，有http等等。这些协议又可以根据数据吞吐量来大致分成两大类：(1)交互数据类型，例如telnet，ssh，这种类型的协议在大多数情况下只是做小流量的数据交换，比如说按一下键盘，回显一些文字等等。(2)数据成块类型，例如ftp，这种类型的协议要求TCP能尽量的运载数据，把数据的吞吐量做到最大，并尽可能的提高效率。针对这两种情况，TCP给出了两种不同的策略来进行数据传输。

1.TCP的交互数据流

对于交互性要求比较高的应用，TCP给出两个策略来提高发送效率和减低网络负担：（1）捎带ACK。(2)Nagle[**算法**](http://lib.csdn.net/base/datastructure)（一次尽量多的发数据）。通常，在网络速度很快的情况下，比如用lo接口进行telnet通信，当按下字母键并要求回显的时候，客户端和服务器将经历 发送按键数据->服务器发送按键数据的ack -> 服务器端发送回显数据->客户端发送回显数据的ACK的过程，而其中的数据流量将是40bit + 41bit+41bit+40bit = 162bit，如果在广域网里面，这种小分组的TCP流量将会造成很大的网络负担。

1.1.捎带ACK的发送方式

这个策略是说，当主机收到远程主机的TCP数据报之后，通常不马上发送ACK数据报，而是等上一个短暂的时间，如果这段时间里面主机还有发送到远程主机的TCP数据报，那么就把这个ACK数据报“捎带”着发送出去，把本来两个TCP数据报整合成一个发送。一般的，这个时间是200ms。可以明显地看到这个策略可以把TCP数据报的利用率提高很多。

1.2.Nagle算法

上过bbs的人应该都会有感受，就是在网络慢的时候发贴，有时键入一串字符串以后，经过一段时间，客户端“发疯”一样突然回显出很多内容，就好像数据一下子传过来了一样，这就是Nagle算法的作用。

Nagle算法是说，当主机A给主机B发送了一个TCP数据报并进入等待主机B的ACK数据报的状态时，TCP的输出缓冲区里面只能有一个TCP数据报，并且，这个数据报不断地收集后来的数据，整合成一个大的数据报，等到B主机的ACK包一到，就把这些数据“一股脑”的发送出去。虽然这样的描述有些不准确，但还算形象和易于理解，我们同样可以体会到这个策略对于低减网络负担的好处。

在编写插口程序的时候，可以通过TCP\_NODELAY来关闭这个算法。

2.TCP的成块数据流

对于FTP这样对于数据吞吐量有较高要求的要求，将总是希望每次尽量多的发送数据到对方主机，就算是有点“延迟”也无所谓。TCP也提供了一整套的策略来支持这样的需求。TCP协议中有16个bit表示“窗口”的大小，这是这些策略的核心。

2.1.传输数据时ACK的问题

在解释滑动窗口前，需要看看ACK的应答策略，一般来说，发送端发送一个TCP数据报，那么接收端就应该发送一个ACK数据报。但是事实上却不是这样，发送端将会连续发送数据尽量填满接受方的缓冲区，而接受方对这些数据只要发送一个ACK报文来回应就可以了，这就是ACK的累积特性，这个特性大大减少了发送端和接收端的负担。

2.2.滑动窗口

滑动窗口本质上是描述接受方的TCP数据报缓冲区大小的数据，发送方根据这个数据来计算自己最多能发送多长的数据。如果发送方收到接受方的窗口大小为0的TCP数据报，那么发送方将停止发送数据，等到接受方发送窗口大小不为0的数据报的到来。

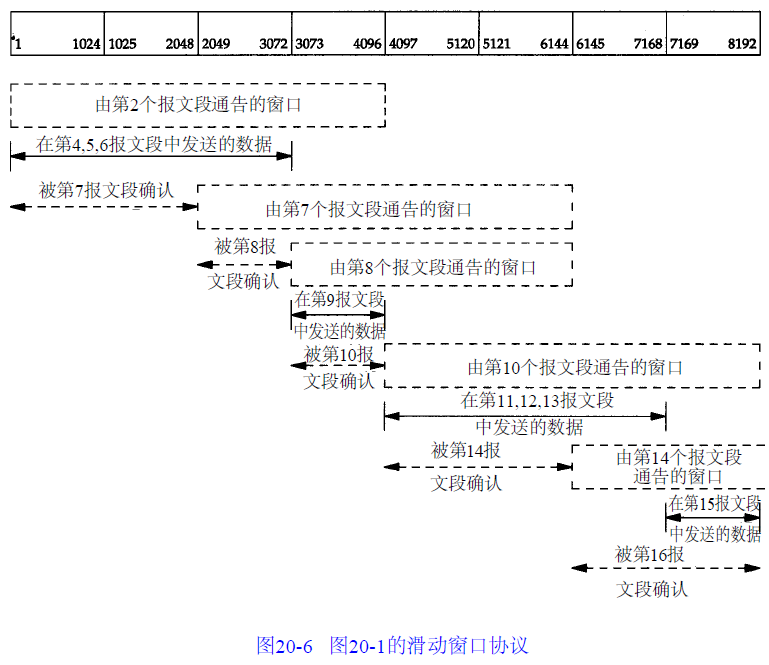
关于滑动窗口协议，书上还介绍了三个术语，分别是：

1.         窗口合拢：当窗口从左边向右边靠近的时候，这种现象发生在数据被发送和确认的时候。

2.         窗口张开：当窗口的右边沿向右边移动的时候，这种现象发生在接受端处理了数据以后。

3.         窗口收缩：当窗口的右边沿向左边移动的时候，这种现象不常发生。

TCP就是用这个窗口，慢慢的从数据的左边移动到右边，把处于窗口范围内的数据发送出去（但不用发送所有，只是处于窗口内的数据可以发送。）。这就是窗口的意义。图20-6解释了这一点。窗口的大小是可以通过socket来制定的，4096并不是最理想的窗口大小，而16384则可以使吞吐量大大的增加。



2.3.数据拥塞

上面的策略用于局域网内传输还可以，但是用在广域网中就可能会出现问题，最大的问题就是当传输时出现了瓶颈（比如说一定要经过一个slip低速链路）所产生的大量数据堵塞问题（拥塞），为了解决这个问题，TCP发送方需要确认连接双方的线路的数据最大吞吐量是多少。这，就是所谓的拥塞窗口。

拥塞窗口的原理很简单，TCP发送方首先发送一个数据报，然后等待对方的回应，得到回应后就把这个窗口的大小加倍，然后连续发送两个数据报，等到对方回应以后，再把这个窗口加倍（先是2的指数倍，到一定程度后就变成现行增长，这就是所谓的慢启动），发送更多的数据报，直到出现超时错误，这样，发送端就了解到了通信双方的线路承载能力，也就确定了拥塞窗口的大小，发送方就用这个拥塞窗口的大小发送数据。要观察这个现象是非常容易的，我们一般在下载数据的时候，速度都是慢慢“冲起来的”

以上就是TCP数据传输的大致流程，虽然并不细致，但是足以描述TCP的工作原理，重点是TCP的流量控制原理，滑动窗口，拥塞窗口，ACK累计确认等知识点。

# 12、TCP的超时与重传

超时重传是TCP协议保证数据可靠性的另一个重要机制，其原理是在发送某一个数据以后就开启一个计时器，在一定时间内如果没有得到发送的数据报的ACK报文，那么就重新发送数据，直到发送成功为止。

1.超时

超时时间的计算是超时的核心部分，TCP要求这个算法能大致估计出当前的网络状况，虽然这确实很困难。要求精确的原因有两个：(1)定时长久会造成网络利用率不高。(2)定时太短会造成多次重传，使得网络阻塞。所以，书中给出了一套经验公式，和其他的保证计时器准确的措施。

1.1.递推公式概说

最早的TCP曾经用了一个非常简单的公式来估计当前网络的状况，如下

R<-aR+(1-a)M

RTP=Rb

其中a是一个经验系数为0.1，b通常为2。注意，这是经验，没有推导过程，这个数值是可以被修改的。这个公式是说用旧的RTT(R)和新的RTT(M)综合到一起来考虑新的RTT(R)的大小。但是，我们又看到，这种估计在网络变化很大的情况下完全不能做出“灵敏的反应”（Jacoboson说的），于是就有下面的修正公式：

Err=M-A

A<-A+gErr

D<-D+h(|Err|-D)

RTO=A+4D

这个递推公式甚至把方差这种统计概念也使用了进来，使得偏差更加的小。而且，必须要指出的是，这两组公式更新，都是在数据成功传输的情况下才进行，在发生数据重新传输的情况下，并不使用上面的公式进行网络估计，理由很简单，因为程序已经不在正常状态下了，估计出来的数据也是没有意义的。

1.2.RTO的初始化

RTO的初始化是由公式决定的，例如最初的公式，初始的值应该是1。而修正公式，初始RTO应该是A+4D。

1.3.RTO的更新

当数据正常传输的情况下，我们就会用上面的公式来更新各个数据，并重开定时器，来保证下一个数据被顺利传输。要注意的是：重传的情况下，RTO不用上面的公式计算，而采用一种叫做“指数退避”的方式。例如：当RTO为1S的情况下，发生了数据重传，我们就用RTO=2S的定时器来重新传输数据，下一次用4S。一直增加到64S为止。

1.4.估计器的初始化

在这里，SYN用的估计器初始化似乎和传输用的估计器不一样，在修正公式中，SYN的情况下，A初始化为0,D初始化为3S。

而在得到传输第一个数据的ACK的时候，应该按照下面的公式进行初始化：

A=M+0.5

D=A/2

1.5.估计器的更新

和上面的讨论差不多，就是在正常情况下，用上面的公式计算，在重传的情况下，不更新估计器的各种参数。原因还是因为估计不准确。

1.6.Karn算法

这不算是一个算法，这应该是一个策略，说的就是更新RTO和估计器的值的时机选择问题，1.3.和1.5.所说得更新时机就是Karn算法。

1.7.计时器的使用

1.         一个连接中，有且仅有一个测量定时器被使用。也就是说，如果TCP连续发出3组数据，只有一组数据会被测量。

2.         ACK数据报不会被测量，原因很简单，没有ACK的ACK回应可以供结束定时器测量。

2.重传

有了超时就要有重传，但是就算是重传也是有策略的，而不是将数据简单的发送。

2.1.重传时发送数据的大小

前面曾经提到过，数据在传输的时候不能只使用一个窗口协议，我们还需要有一个拥塞窗口来控制数据的流量，使得数据不会一下子都跑到网路中引起“拥塞”。也曾经提到过，拥塞窗口最初使用指数增长的速度来增加自身的窗口，直到发生超时重传，再进行一次微调。但是没有提到，如何进行微调，拥塞避免算法和慢启动门限就是为此而生。

所谓的慢启动门限就是说，当拥塞窗口超过这个门限的时候，就使用拥塞避免算法，而在门限以内就采用慢启动算法。所以这个标准才叫做门限，通常，拥塞窗口记做cwnd，慢启动门限记做ssthresh。下面我们来看看拥塞避免和慢启动是怎么一起工作的



算法概要

1.         对一个给定的连接，初始化cwnd为1个报文段，ssthresh为65535个字节。

2.         TCP输出例程的输出不能超过cwnd和接收方通告窗口的大小。拥塞避免是发送方使用 的流量控制，而通告窗口则是接收方进行的流量控制。前者是发送方感受到的网络拥塞的估 计，而后者则与接收方在该连接上的可用缓存大小有关。

3.         当拥塞发生时（超时或收到重复确认），ssthresh被设置为当前窗口大小的一半（cwnd 和接收方通告窗口大小的最小值，但最少为2个报文段）。此外，如果是超时引起了拥塞，则 cwnd被设置为1个报文段（这就是慢启动）。

4.         当新的数据被对方确认时，就增加cwnd，但增加的方法依赖于我们是否正在进行慢启 动或拥塞避免。如果cwnd小于或等于ssthresh，则正在进行慢启动，否则正在进行拥塞避免。 慢启动一直持续到我们回到当拥塞发生时所处位置的半时候才停止（因为我们记录了在步骤2 中给我们制造麻烦的窗口大小的一半），然后转为执行拥塞避免。

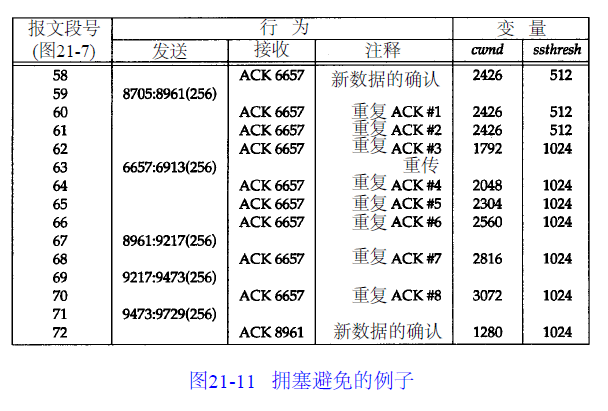
2.2.快速重传和快速恢复算法

这是数据丢包的情况下给出的一种修补机制。一般来说，重传发生在超时之后，但是如果发送端接受到3个以上的重复ACK的情况下，就应该意识到，数据丢了，需要重新传递。这个机制是不需要等到重传定时器溢出的，所以叫做快速重传，而重新传递以后，因为走的不是慢启动而是拥塞避免算法，所以这又叫做快速恢复算法。流程如下：

1.         当收到第3个重复的ACK时，将ssthresh设置为当前拥塞窗口cwnd的一半。重传丢失的 报文段。设置cwnd为ssthresh加上3倍的报文段大小。

2.         每次收到另一个重复的ACK时， cwnd增加1个报文段大小并发送1个分组（如果新的 cwnd允许发送）。

3.         当下一个确认新数据的ACK到达时，设置cwnd为ssthresh（在第1步中设置的值）。这个 ACK应该是在进行重传后的一个往返时间内对步骤1中重传的确认。另外，这个ACK也应该 是对丢失的分组和收到的第1个重复的ACK之间的所有中间报文段的确认。这一步采用的是拥 塞避免，因为当分组丢失时我们将当前的速率减半。



2.3.ICMP会引起重新传递么？

答案是：不会，TCP会坚持用自己的定时器，但是TCP会保留下ICMP的错误并且通知用户。

2.4.重新分组

TCP为了提高自己的效率，允许再重新传输的时候，只要传输包含重传数据报文的报文就可以，而不用只重传需要传输的报文。

# 13、TCP坚持定时器，TCP保活定时器

TCP一共有四个主要的定时器，前面已经讲到了一个－－超时定时器－－是TCP里面最复杂的一个，另外的三个是：

1.         坚持定时器

2.         保活定时器

3.         2MSL定时器

其中坚持定时器用于防止通告窗口为0以后双方互相等待死锁的情况；而保活定时器则用于处理半开放连接

1.坚持定时器

坚持定时器的原理是简单的，当TCP服务器收到了客户端的0滑动窗口报文的时候，就启动一个定时器来计时，并在定时器溢出的时候向向客户端查询窗口是否已经增大，如果得到非零的窗口就重新开始发送数据，如果得到0窗口就再开一个新的定时器准备下一次查询。通过观察可以得知，TCP的坚持定时器使用1，2，4，8，16……64秒这样的普通指数退避序列来作为每一次的溢出时间。

糊涂窗口综合症

TCP的窗口协议，会引起一种通常叫做糊涂窗口综合症的问题，具体表现为，当客户端通告一个小的非零窗口时，服务器立刻发送小数据给客户端并充满其缓冲区，一来二去就会让网络中充满小TCP数据报，从而影响网络利用率。对于发送方和接收端的这种糊涂行为。TCP给出了一些建议（或者是规定）。

1.         接收方不通告小窗口。通常的算法是接收方不通告一个比当前窗口大的窗口（可以为0），  
除非窗口可以增加一个报文段大小（也就是将要接收的MSS）或者可以增加接收方缓存空间  
的一半，不论实际有多少。

2.         发送方避免出现糊涂窗口综合症的措施是只有以下条件之一满足时才发送数据： ( a )可  
以发送一个满长度的报文段； ( b )可以发送至少是接收方通告窗口大小一半的报文段； ( c )可以  
发送任何数据并且不希望接收ACK（也就是说，我们没有还未被确认的数据）或者该连接上  
不能使用Nagle算法。

ok，现在我们回忆一下，可以发现TCP的很多规定都是为了在一次传送中发送尽量多的数据，例如捎带ACK数据报文的策略，Nagle算法，重传时发送包含原数据报文的策略，等等。

2.保活定时器

保活定时器更加的简单，还记得FTP或者Http服务器都有Sesstion Time机制么，因为TCP是面向连接的，所以就会出现只连接不传送数据的“半开放连接”，服务器当然要检测到这种连接并且在某些情况下释放这种连接，这就是保活定时器的作用。其时限根据服务器的实现不同而不通。另外要提到的是，当其中一端如果崩溃并重新启动的情况下，如果收到该端“前生”的保活探察，则要发送一个RST数据报文帮助另一端结束连接。