# [使用wireshark分析TCP/IP协议中TCP包头的格式](http://blog.csdn.net/ningxuezhu/article/details/39892091)

1. [摘要](http://blog.csdn.net/ningxuezhu/article/details/39892091/#t0)
2. [一概述](http://blog.csdn.net/ningxuezhu/article/details/39892091/#t1)
3. [二TCP报文格式](http://blog.csdn.net/ningxuezhu/article/details/39892091/#t2)
4. [三实例解析](http://blog.csdn.net/ningxuezhu/article/details/39892091/#t3)

**摘要：**

本文简单介绍了TCP面向连接理论知识，详细讲述了TCP报文各个字段含义，并从Wireshark俘获分组中选取TCP连接建立相关报文段进行分析。

**一、概述**

TCP是面向连接的可靠传输协议，两个进程互发数据之前需要建立连接，这里的连接只不过是端系统中分配的一些缓存和状态变量，中间的分组交换机不维护任何连接状态信息。连接建立整个过程如下(即三次握手协议)：

首先，客户机发送一个特殊的TCP报文段；

其次，服务器用另一个特殊的TCP报文段来响应；

最后，客户机再用第三个特殊报文段作为响应。

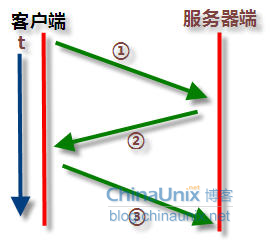


图1 三次握手协议示意图[1]

**二、TCP报文格式**

**2.1 概述**

为了提供可靠的数据传输，TCP报文首部字段有较多的字段，TCP报文格式如下图：

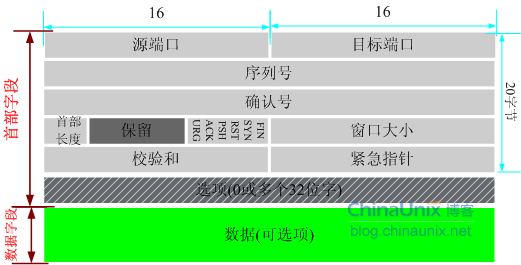


图2 TCP报文格式

**源和目标端口**

用于多路复用/多路分解来自或送至上层应用的数据，可以这样理解，端口用来标识同一台计算机的不同进程。

**序列号和确认号**

这两个字段是TCP可靠传输服务的关键部分，序列号是该报文段首字节的字节流编号(**TCP把数据看成是有序的字节流**，TCP隐式地对数据流的每个字节进行编号)。这样理解可能更直观，当报文被分解成多个报文段时，序列号就是报文段首字节在整个报文的偏移量。**确定号指定下一个期待的字节**。TCP是全双工的，假设从主机A接收到主机B的数据，则主机A填充进报文段的确认号是主机A期望从主机B收到的下一个字节序号。还没理清这两者的关系？见下图(三次握手)：

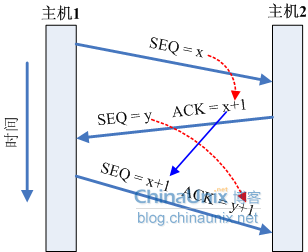


图3 正常情况下TCP连接建立过程

**首部长度(4位)**

因为选项是不定长的，这就需要标识整个首部字段的长度(单位是32位字)，即5+选项个数。4位，单位是32位字(双字节)，所以首部最长是15\*4=60字节，即选项最长是40字节(10个选项)。

**标志**

**URG**

指示报文段里存在着被发送方的上层实体标记为”紧急”数据，当URG=1时，其后的紧急指针指示紧急数据在当前数据段中的位置(相对于当前序列号的字节偏移量)，TCP接收方必须通知上层实体。

**ACK**

当ACK=0时，表示该数据段不包含确认信息，当ACK=1时，表示该报文段包括一个对已被成功接收报文段的确认。

**PSH**

当PSH=1时，接收方在收到数据后立即将数据交给上层，而不是直到整个缓冲区满。

**RST**

用于重置一个已经混乱的连接(如主崩溃)，也可用于拒绝一个无效的数据段或者拒绝一个连接请求。一般而言，如果你得到的数据段被设置了RST位，那说明你这一端有问题了。

**SYN**

用于建立连接过程，在连接请求中，**SYN=1和ACK=0**表示该数据段没有使用**捎带**的确认域，而连接应答捎带一个确认，即**SYN=1和ACK=1**。

注：捎带是指对客户机到服务器数据的**确认**被装载在一个承载服务器到客户机的数据报文段中。

**FIN**

用于释放一个连接，表示发送方已经没有数据要传输了。此时，接收方可能继续接收数据，好在SYN和FIN数据段都有序列号，从而保证了这两种数据段以正确顺序被处理。

**窗口大小**

用于流控制(确保连接的任何一方都不会过快地发送过量的分组而淹没另一方)，窗口大小指定了从**被确认的字节**算起可以发送多少个字节。

**校验和**

提供了额外可靠性，在计算检验和的时候，TCP的Checksum域设为0，如果数据域的字节数为奇数，则数据域填补一个额外的0字节。校验和[**算法**](http://lib.csdn.net/base/datastructure)：将所有的16位字按1的补码形式累加起来，取累加结果的补码。因此，当接收方执行同样计算时(包括Checksum域)，结果应该是0。

**紧急指针**

参考标志字段的URG位。

**选项**

选项部分是为了适合复杂网络环境和更好地服务于应用层设计的。TCP选项最长是40字节。详情见2.2。

**数据**

无任何数据的TCP段也是合法的，通常用于确认和控制信息。

**2.2 选项字段[2]**

TCP选项部分很好出现在已经建立连接的会话中，只要出现在TCP连接建立阶段，即三次握手。TCP选项部分实际运用有以下几种：

**(1)最大报文传输段(MMS, Maximum Segment Size)**

用于发送发与接收方协商**最大报文段长度**(仅仅是净荷数据，不包括TCP首部字段)。TCP在三次握手中，每一方都会通告期望收到的MSS(MSS只出现在SYN数据包中)，如果一方不接受另一方的MSS值，则使用默认的**536**字节净荷数据，即主机能够接受20+536字节的TCP报文段。

**(2)窗口扩大选项(Window scaling)**

TCP报文的窗口大小字段占16位，即最大值是65535，**但随着时延和带宽比较大的通信产生(如卫星通信)，需要更大的窗口满足性能和吞吐率**，这就是窗口扩大选项存在的意义。例子见参考资料[2]。

Windows scaling占3个字节，最后一个字节是**移位值(Shift count)**，即首部的窗口位数16向左移动，如移位值为14，则新的窗口最大值增大到65535\*(2^14)。

窗口扩大选项是在TCP建立之初进行协商，如果已实现了窗口扩大，当不再需要扩大窗口时，发送**移位值=0**就可以恢复到原窗口大小，即65535。

**(3)选择确认选项(SACK, Selective Acknowledgements)**

考虑这样情况，主机A发送报文段12345，主机B收到135且报文无差错，**SACK用来确保只重传缺少的报文段**，而不是重传所有报文段。

SACK选项需要2个功能字节，一个用来指明使用SACK选项(SACK Permission)，另一指明这个选项占多少字节。

那怎么形容丢失的报文段2，说明2的左右边界分别是1、3。TCP的数据报文是有字块边界的，而这种边界是由序列号表示的。

最多能指明多少个字节块的边界信息呢？答案是4个。这是因为选项字段最大是40字节，去除2个功能字节，序列号是32位即4字节，并且需要左右边界，所以(40-2)/8 = 4。

**(4)时间戳选项(timestamps)**

时间戳选项用来计算往返时间RTT，发送方在发送报文段时把当前时钟的时间值放入时间戳字段，接收方将该时间戳字段的值复制到确认报文中，当接收方收到确认报文，对比确认报文的时间戳(等于发送方发送报文段的时间戳)和现在的时钟，即可算出RTT。

时间戳选项还可用于防止回绕序号PAWS。序列号只有32位，每2^32个序列号就会回绕(想想环形队列)，采用时间戳选项很容易区分相同序列号的报文段。

**(5)NOP(NO-Operation)**

TCP的头部必须是4字节的倍数，而大多数选项不是4字节倍数，不足的用NOP填充。除此之外，NOP也用于分割不同的选项数据，如窗口扩大选项和SACK之间使用NOP隔离(下面的实例将看到这一点)。

**三、实例解析**

**3.1 概述**

还是以访问百度首页为例，首先用DNS协议将URL解析成IP地址，接着在客户机和服务器间建立TCP连接，用Wireshark俘获的分组如下图：

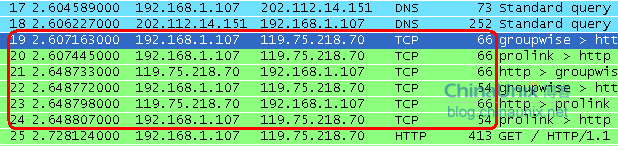


图4 Wireshark俘获建立TCP连接分组

你一看会觉得有些奇怪，理论上应该是3个分组的，怎么有6个分组？先不急，先把这6个报文收发示意图作出来(结合时间和报文含义)，如下：

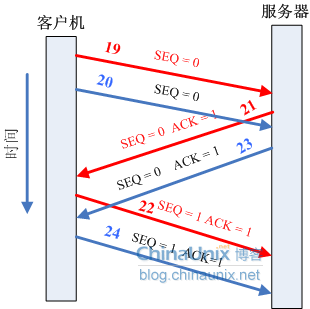


图5 TCP连接建立实例

从图可知，连接建立伊始，客户机发了两个报文段，这也许是为了更快建立连接(假设有个请求报文段丢失，也不至于要等一段时间，重发报文)。接下来，以19、21、22(上图红色线条所示)分析TCP连接建立过程。

**3.1 第一次握手19**

Wireshark俘获TCP连接第一次握手的报文段如下：

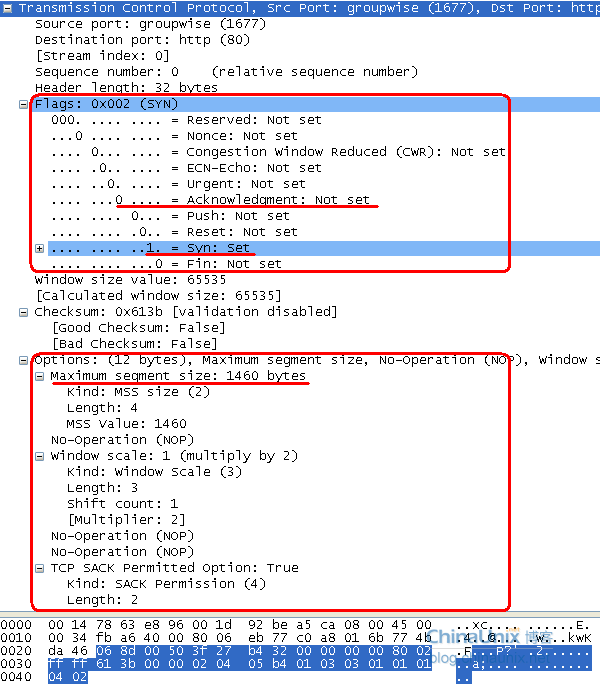


图6 TCP连接第一次握手实例

这里主要挑几个字段分析：

标志字段，SYN=1、ACK=0表示该数据段没有使用**捎带**的确认域。

最大报文段长度(MMS)1460是怎么来的，链路层的以太网物理特性决定数据帧长度为1500(即MTU，最大传输单元)，1460=1500-20(IP首部长度)-20(TCP首部长度)。不要被该报文首部长度32字节所迷惑，这只是建立连接过程。MSS与MTU关系见下图[2]：

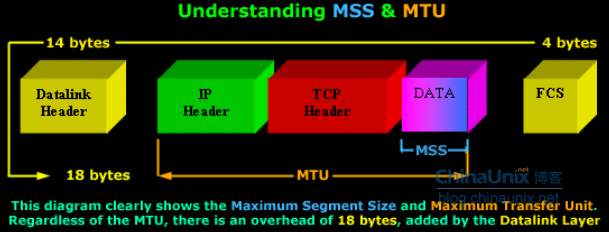


图7 MSS与MTU关系

NOP字段，可以作为不足4倍数字节填充，也可作为选项间分隔，该报文段出现了3个NOP，具体功能见下图：

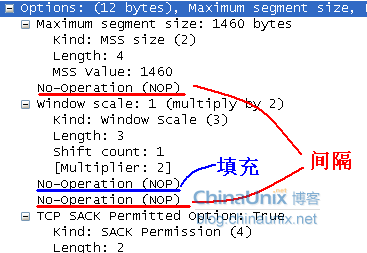


图8 TCP报文NOP字段

**3.3 第二次握手21**

服务器响应客户端TCP报文段，此时确认号为1了，SYN=1、ACK=1表明连接应答捎带一个确认，Wireshark俘获分组如下：

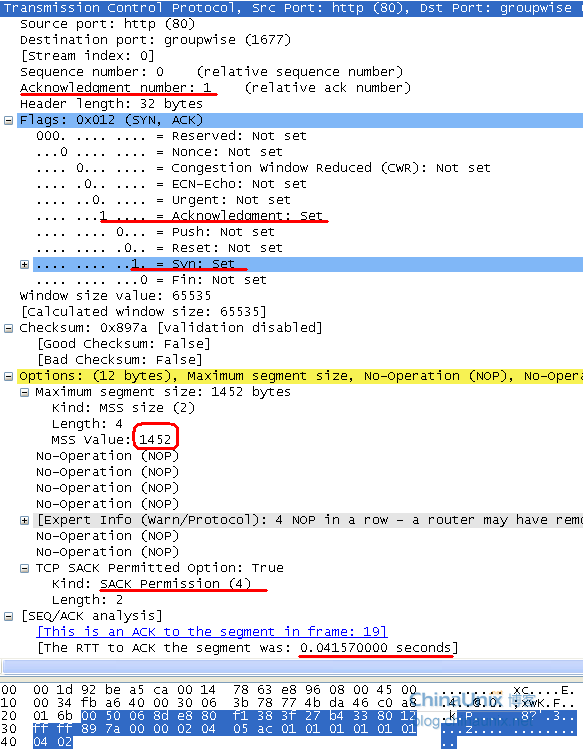


图9 TCP连接第二次握手实例

为什么MSS是1452而不是1460?这是因为使用PPPoE(Point-to-Point over Ethernet，可以使以太网的主机通过一个简单的桥接设备连到一个无端的接入集中器上[3])拨号上网，PPoP首部是8个字节，所以PPPoE的MTU是1492，MSS也就为1492-40=1452。

那么，TCP连接建立后数据传输的MSS是多少呢，1460 or 1452 or 536 ？我的理解是默认值536，这样理解对吗？求指点！

**3.4 第三次握手22**

客户机再次服务器的报文段，此时序列号和确认号都为1，没有选项字段，Wireshark俘获的分组信息如下：

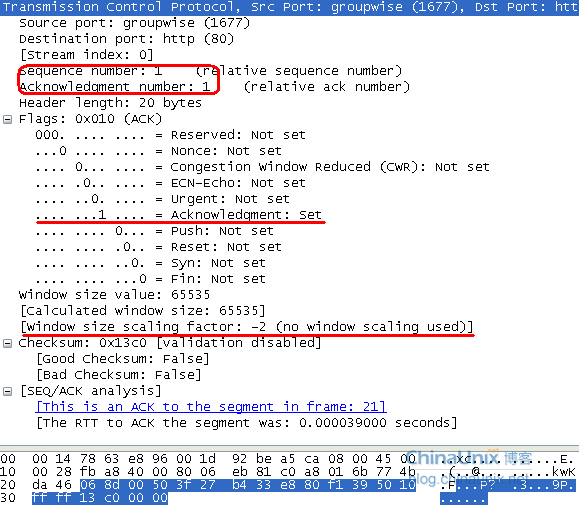


图10 TCP连接第三次握手实例

值得注意的，因为窗口扩展大小协商未果，所以就不扩大窗口了，即窗口大小最大为65535。

如此，TCP连接建立:-)

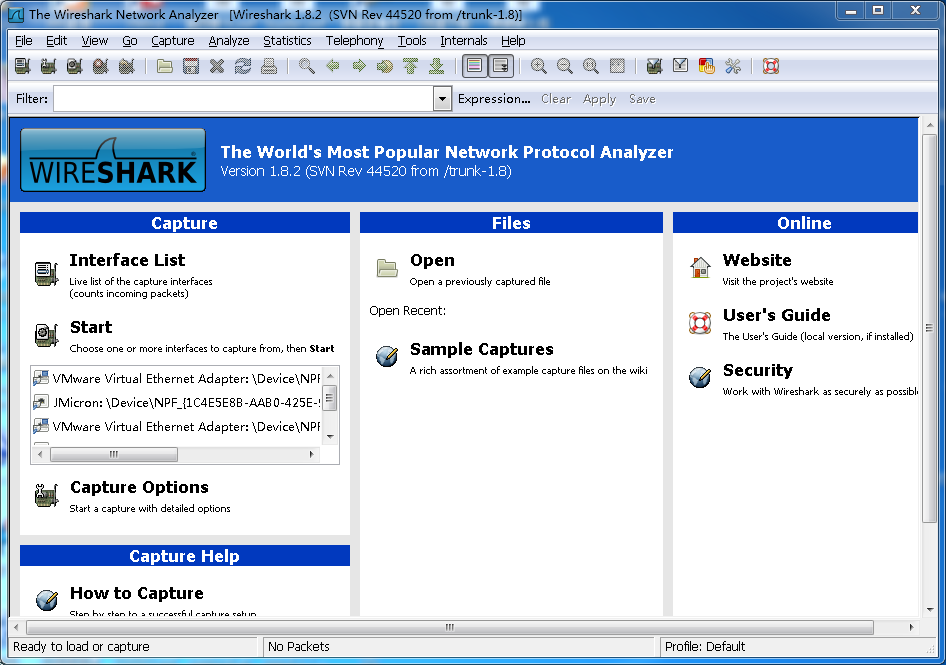
# wireshark抓包详细图文教程

wireshark是非常流行的网络封包分析软件，功能十分强大。可以截取各种网络封包，显示网络封包的详细信息。使用wireshark的人必须了解网络协议，否则就看不懂wireshark了。  
为了安全考虑，wireshark只能查看封包，而不能修改封包的内容，或者发送封包。

wireshark能获取HTTP，也能获取HTTPS，但是不能解密HTTPS，所以wireshark看不懂HTTPS中的内容，总结，如果是处理HTTP,HTTPS 还是用Fiddler,**其他协议比如TCP,UDP 就用wireshark.**

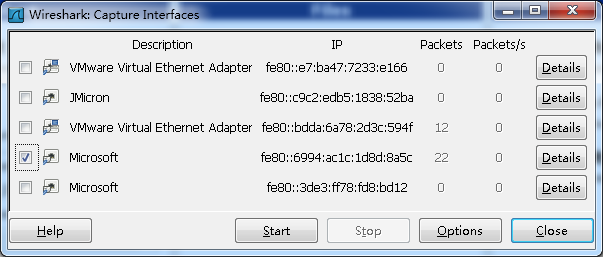
**wireshark 开始抓包**

开始界面

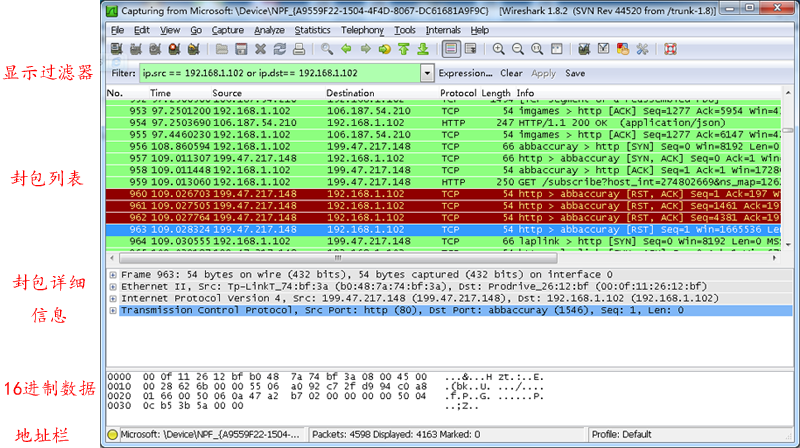


**wireshark是捕获机器上的某一块网卡的网络包**，当你的机器上有多块网卡的时候，你需要选择一个网卡。

点击Caputre->Interfaces.. 出现下面对话框，选择正确的网卡。然后点击"Start"按钮, 开始抓包



Wireshark 窗口介绍



**WireShark 主要分为这几个界面**

1. Display Filter(显示过滤器)，  用于过滤

2. Packet List Pane(封包列表)， 显示捕获到的封包， 有源地址和目标地址，端口号。 颜色不同，代表

3. Packet Details Pane(封包详细信息), 显示封包中的字段

4. Dissector Pane(16进制数据)

5. Miscellanous(地址栏，杂项)

**第 2 页 Wireshark 显示过滤**

**http://www.9upk.com/article/UploadPic/2013-5/20135220328859.png**

使用过滤是非常重要的， 初学者使用wireshark时，将会得到大量的冗余信息，在几千甚至几万条记录中，以至于很难找到自己需要的部分。搞得晕头转向。

过滤器会帮助我们在大量的数据中迅速找到我们需要的信息。

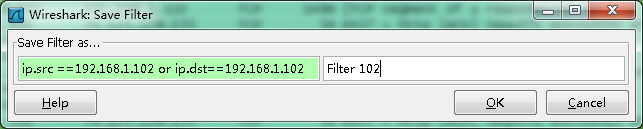
过滤器有两种，

一种是显示过滤器，就是主界面上那个，用来在捕获的记录中找到所需要的记录

一种是捕获过滤器，用来过滤捕获的封包，以免捕获太多的记录。 在Capture -> Capture Filters 中设置

保存过滤

在Filter栏上，填好Filter的表达式后，点击Save按钮， 取个名字。比如"Filter 102",



Filter栏上就多了个"Filter 102" 的按钮。

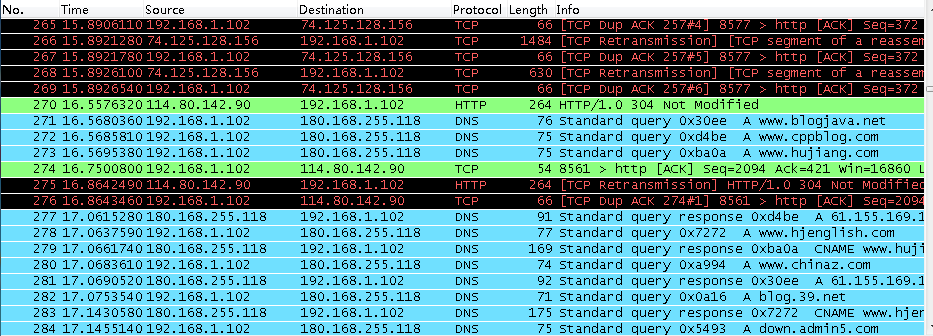
http://www.9upk.com/article/UploadPic/2013-5/20135220328630.png

|  |  |
| --- | --- |
| 过滤表达式 | 用途 |
| http | 只查看HTTP协议的记录 |
| ip.src ==192.168.1.102 or ip.dst==192.168.1.102 | 源地址或者目标地址是192.168.1.102 |
|  |  |
|  |  |

封包列表(Packet List Pane)

封包列表的面板中显示，编号，时间戳，源地址，目标地址，协议，长度，以及封包信息。 你可以看到不同的协议用了不同的颜色显示。

你也可以修改这些显示颜色的规则，  View ->Coloring Rules.



封包详细信息 (Packet Details Pane)

这个面板是我们最重要的，用来查看协议中的每一个字段。

各行信息分别为

Frame:   物理层的数据帧概况

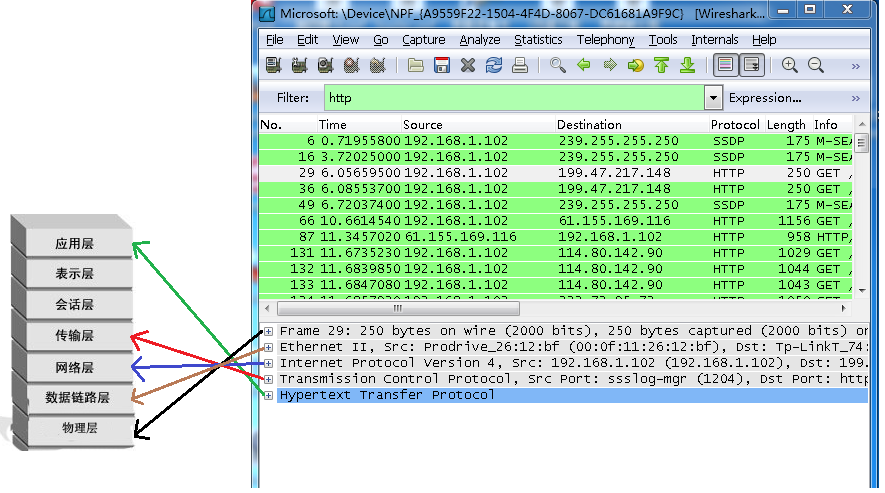
Ethernet II: 数据链路层以太网帧头部信息

Internet Protocol Version 4: 互联网层IP包头部信息

Transmission Control Protocol:  传输层T的数据段头部信息，此处是TCP

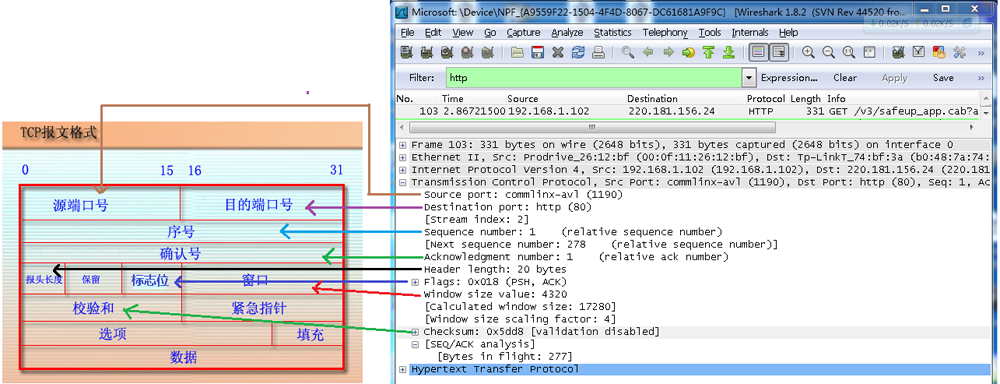
Hypertext Transfer Protocol:  应用层的信息，此处是HTTP协议

**第 3 页 wireshark与对应的OSI七层模型**

****

**TCP包的具体内容**

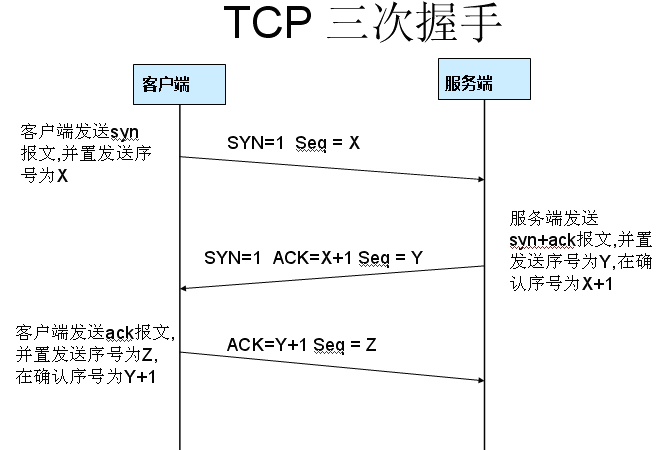
 从下图可以看到wireshark捕获到的TCP包中的每个字段。



**第 4 页 实例分析TCP三次握手过程**

看到这， 基本上对wireshak有了初步了解， 现在我们看一个TCP三次握手的实例

 三次握手过程为

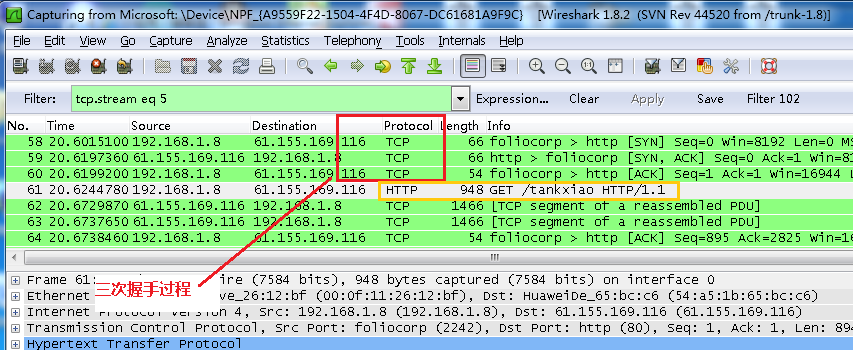


这图我都看过很多遍了， 这次我们用wireshark实际分析下三次握手的过程。

打开wireshark, 打开浏览器输入 [http://www.9upk.com](http://www.9upk.com/)

在wireshark中输入http过滤， 然后选中GET /tankxiao HTTP/1.1的那条记录，右键然后点击"Follow TCP Stream",

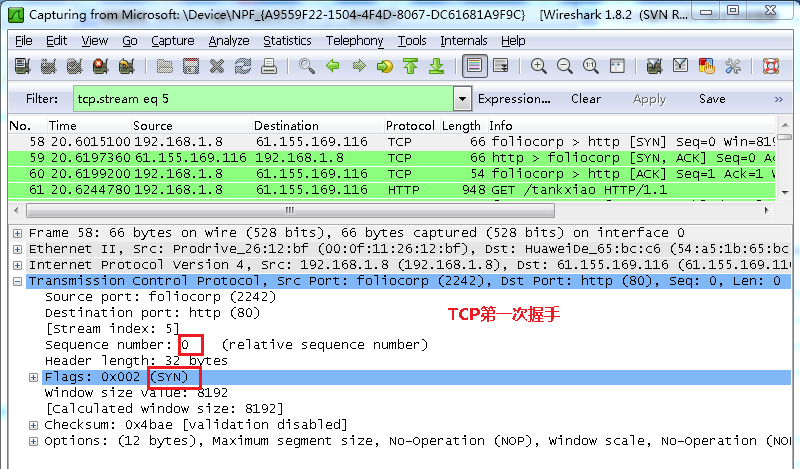
这样做的目的是为了得到与浏览器打开网站相关的数据包，将得到如下图



图中可以看到wireshark截获到了三次握手的三个数据包。第四个包才是HTTP的， 这说明HTTP的确是使用TCP建立连接的。

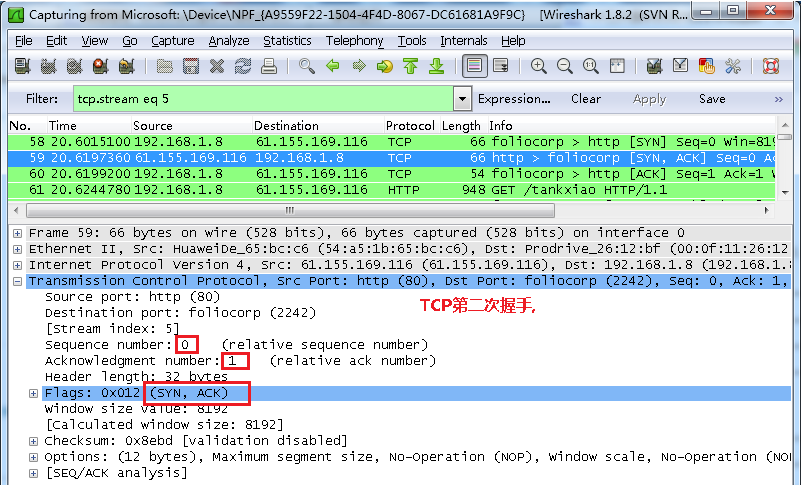
**第一次握手数据包**

客户端发送一个TCP，标志位为SYN，序列号为0， 代表客户端请求建立连接。 如下图



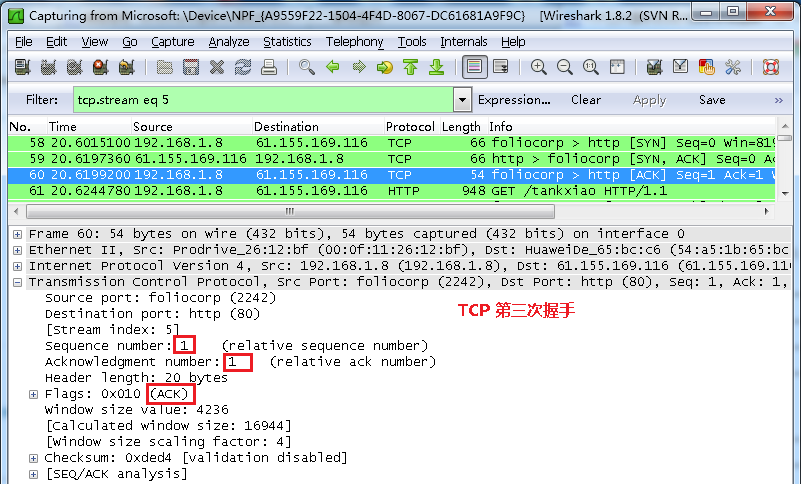
**第二次握手的数据包**

服务器发回确认包, 标志位为 SYN,ACK. 将确认序号(Acknowledgement Number)设置为客户的I S N加1以.即0+1=1, 如下图



**第三次握手的数据包**

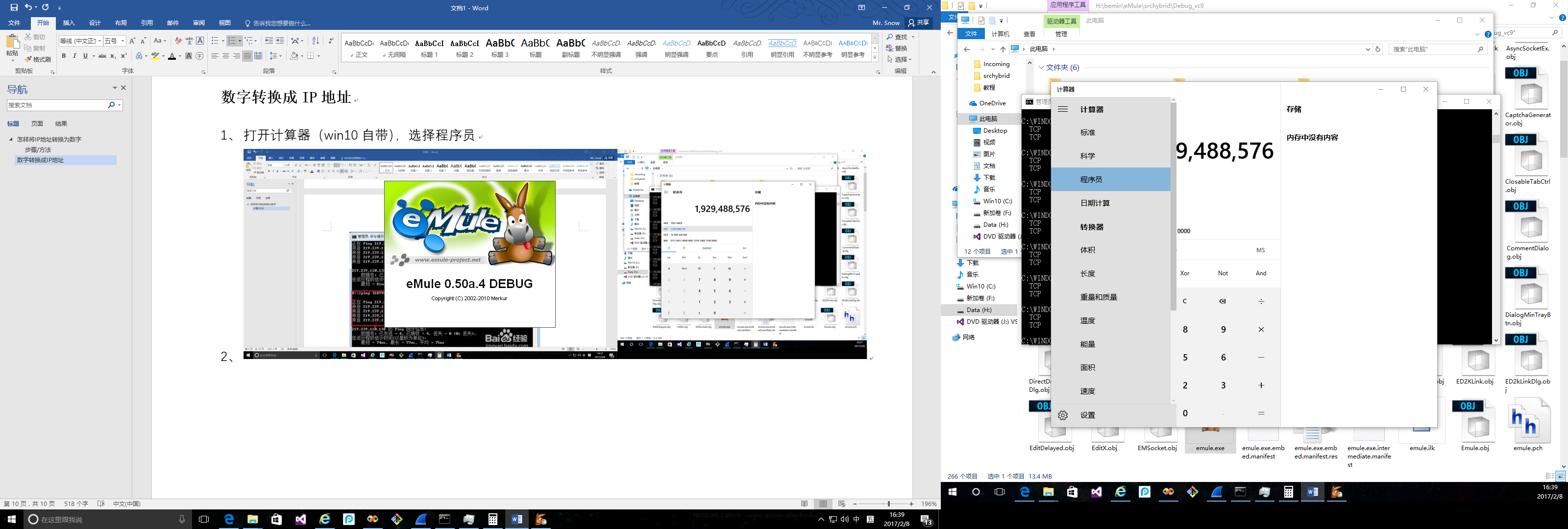
客户端再次发送确认包(ACK) SYN标志位为0,ACK标志位为1.并且把服务器发来ACK的序号字段+1,放在确定字段中发送给对方.并且在数据段放写ISN的+1, 如下图:



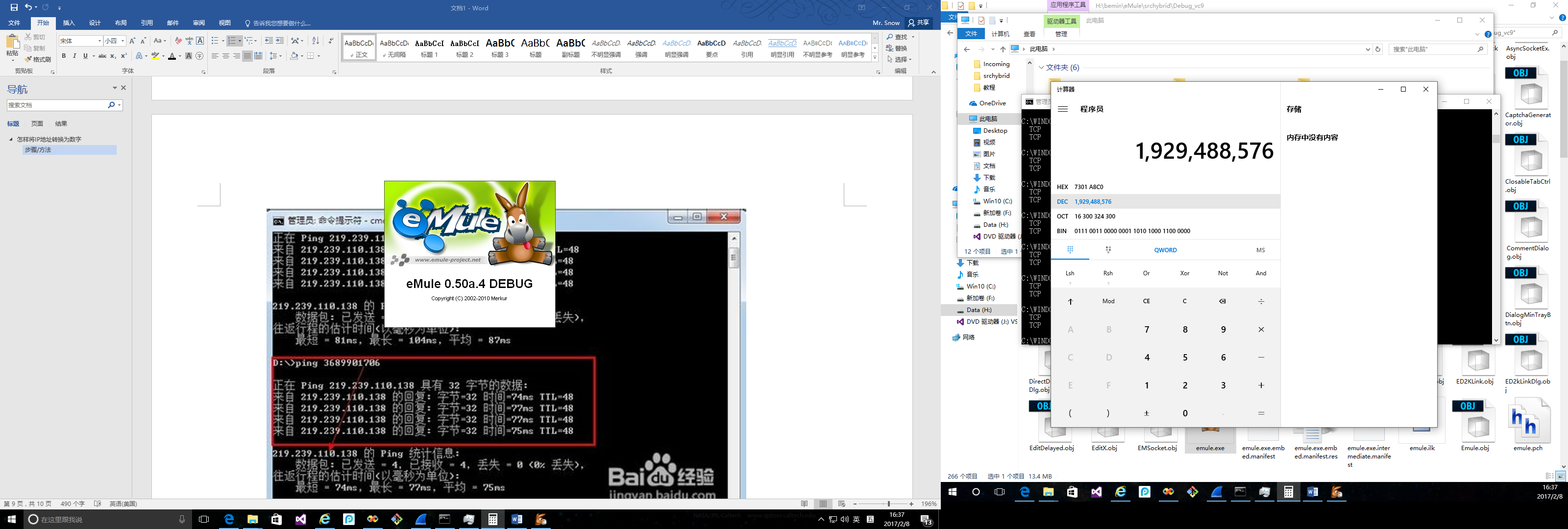
 就这样通过了TCP三次握手，建立了连接

# 数字转换成IP地址

1. 打开计算器（win10自带），选择程序员



2、点击DEC,输入整数1929488576



3、得到对应的十六进制数7301A8C0

4、区分是顺字节序还是逆字节序，本例中的数字是逆字节序，所以应从右往左输入数字

5、点击HEX，输入C0，得到DEC 192，依次输入A8，01，73，分别得到十进制数168，1，115。

6、IP地址就是192.168.1.115

7、如果是顺字节序，则得到的整数应该是**3232235891，HEX值就是C0A80173**

# ICMP

ICMP是（Internet Control Message Protocol）Internet控制[报文](http://baike.baidu.com/view/175122.htm)协议。它是[TCP/IP协议族](http://baike.baidu.com/view/2221037.htm)的一个子协议，用于在IP[主机](http://baike.baidu.com/view/23880.htm)、[路由](http://baike.baidu.com/view/18655.htm)器之间传递控制消息。控制消息是指[网络通](http://baike.baidu.com/view/8079702.htm)不通、[主机](http://baike.baidu.com/view/23880.htm)是否可达、[路由](http://baike.baidu.com/view/18655.htm)是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。

中文名

ICMP

外文名

Internet Control Message Protocol

类    型

控制[报文](http://baike.baidu.com/view/175122.htm)协议

协议族

[TCP/IP协议族](http://baike.baidu.com/view/2221037.htm)

归    属

网络层协议

作    用

在主机与路由器之间传递控制信息

## 目录

1. 1 [定义](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#1)
2. 2 [协议内容](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#2)
3. 3 [重要性](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#3)
4. 4 [校验算法](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#4)
5. 5 [抵御攻击](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#5)
6. 6 [防御方法](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#6)
7. 7 [ICMP类型](http://baike.baidu.com/link?url=IwJMNAKRYLDhGzLochvs-AMwtiXoFXqp9E4OoVxVrdqntoVZJOiVvHg_eQ4mGGgLkzqvz2z2hGhWKwsxNo6SC8yU20Fc34XaD4PGP1d0LZXBE6ZTnjURNHiaeGBSQyhf7BQe35TmdMSfxtR-7lLauK#7)

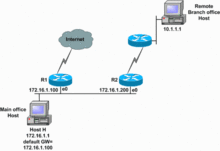
## ICMP定义

ICMP协议是一种面向无连接的协议，用于传输出错报告控制信息。它是一个非常重要的协议，它对于[网络安全](http://baike.baidu.com/view/17495.htm)具有极其重要的意义。[1]

它是[TCP/IP协议](http://baike.baidu.com/view/7649.htm)族的一个子协议，属于网络层协议，主要用于在主机与路由器之间传递控制信息，包括报告错误、交换受限控制和状态信息等。当遇到IP数据无法访问目标、IP[路由器](http://baike.baidu.com/view/1360.htm)无法按当前的传输速率转发[数据包](http://baike.baidu.com/view/25880.htm)等情况时，会自动发送ICMP消息。ICMP报文在IP帧结构的首部协议类型字段（Protocol 8bit)的值=1.

如下图所示，ICMP包有一个8字节长的包头，其中前4个字节是固定的格式，包含8位类型字段，8位代码字段和16位的校验和；后4个字节根据ICMP包的类型而取不同的值。

[](http://baike.baidu.com/pic/ICMP/572452/0/ae51f3deb48f8c54ca50c28039292df5e1fe7f99?fr=lemma&ct=single)ICMP报文格式

[](http://baike.baidu.com/pic/ICMP/572452/0/dbf554ed9529cbccb31cb1d0?fr=lemma&ct=single)ICMP原理

ICMP提供一致易懂的出错报告信息。发送的出错[报文](http://baike.baidu.com/view/175122.htm)返回到发送原数据的设备，因为只有发送设备才是出错报文的逻辑接受者。发送设备随后可根据ICMP报文确定发生错误的类型，并确定如何才能更好地重发失败的数据包。但是ICMP唯一的功能是报告问题而不是纠正错误，纠正错误的任务由发送方完成。

我们在网络中经常会使用到ICMP协议，比如我们经常使用的用于检查网络通不通的[Ping](http://baike.baidu.com/view/709.htm)命令（Linux和Windows中均有），这个“Ping”的过程实际上就是ICMP协议工作的过程。还有其他的[网络命令](http://baike.baidu.com/view/41338.htm)如[跟踪路由](http://baike.baidu.com/view/493712.htm)的Tracert命令也是基于ICMP协议的。

## ICMP协议内容

ICMP的全称是 Internet Control Message Protocol 。从技术角度来

[](http://baike.baidu.com/pic/ICMP/572452/0/f392492cc28fe9f48a1399ce?fr=lemma&ct=single)ICMP常用类型

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﹐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。

· 重导资料传送路径（[ICMP重定向](http://baike.baidu.com/view/5622994.htm)）。

· 资料[流量控制](http://baike.baidu.com/view/190232.htm)。ICMP在沟通之中，主要是透过不同的[类别](http://baike.baidu.com/view/738139.htm)(Type)与[代码](http://baike.baidu.com/view/41.htm)(Code) 让机器来识别不同的连线状况。常用的[类别](http://baike.baidu.com/view/738139.htm)如下表所列﹕

ICMP 是个非常有用的协议﹐尤其是当我们要对网路连接状况进行判断的时候。

## ICMP重要性

ICMP协议对于[网络安全](http://baike.baidu.com/view/17495.htm)具有极其重要的意义。ICMP协议本身的特

[](http://baike.baidu.com/pic/ICMP/572452/0/aa251d4fd9a2b72eafc3abc8?fr=lemma&ct=single)ICMP常用类型

点决定了它非常容易被用于攻击网络上的[路由器](http://baike.baidu.com/view/1360.htm)和[主机](http://baike.baidu.com/view/23880.htm)。例如，在1999年8月[海信集团](http://baike.baidu.com/view/61422.htm)“悬赏”50万元人民币测试防火墙的过程中，其防火墙遭受到的ICMP攻击达334050次之多，占整个攻击总数的90%以上！可见，ICMP的重要性绝不可以忽视！

比如，可以利用[操作系统](http://baike.baidu.com/view/880.htm)规定的ICMP数据包最大尺寸不超过64[KB](http://baike.baidu.com/view/170925.htm)这一规定，向[主机](http://baike.baidu.com/view/23880.htm)发起“Ping of Death”（[死亡之Ping](http://baike.baidu.com/view/113328.htm)）攻击。“Ping of Death” 攻击的原理是：如果ICMP数据包的尺寸超过64KB上限时，[主机](http://baike.baidu.com/view/23880.htm)就会出现[内存分配](http://baike.baidu.com/view/549628.htm)错误，导致TCP/IP[堆栈](http://baike.baidu.com/view/93201.htm)崩溃，致使主机死机。（[操作系统](http://baike.baidu.com/view/880.htm)已经取消了发送ICMP数据包的大小的限制，解决了这个漏洞）

此外，向目标[主机](http://baike.baidu.com/view/23880.htm)长时间、连续、大量地发送ICMP数据包，也会最终使系统瘫痪。大量的ICMP数据包会形成“ICMP风暴”，使得目标[主机](http://baike.baidu.com/view/23880.htm)耗费大量的CPU资源处理，疲于奔命。

## ICMP校验算法

以下[代码](http://baike.baidu.com/view/41.htm)在Visual Studio 2008 + Windows 7下调试通过。

lpsz指定要计算的数据包首地址，\_dwSize指定该数据包的长度。

int CalcCheckSum(char\* lpsz,DWORD \_dwSize)

{

int dwSize;

\_\_asm // 嵌入汇编

{

mov ecx,\_dwSize

shr ecx,1

xor ebx,ebx

mov esi,lpsz

read: //所有word相加，保存至EBX寄存器

lodsw

movzx eax,ax

add ebx,eax

loop read

test \_dwSize,1 //校验数据是否是奇数位的

jz calc

lodsb

movzx eax,al

add ebx,eax

calc:

mov eax,ebx //高低位相加

and eax,0ffffh

shr ebx,16

add eax,ebx

not ax

mov dwSize,eax

}

return dwSize;

}

## ICMP抵御攻击

虽然ICMP协议给[黑客](http://baike.baidu.com/view/1960.htm)以可乘之机，但是ICMP攻击也并非无药可医。只要在日常[网络管理](http://baike.baidu.com/view/325702.htm)中未雨绸缪，提前做好准备，就可以有效地避免ICMP攻击造成的损失。

对于“Ping of Death”攻击，可以采取两种方法进行防范：第一种方法是在[路由器](http://baike.baidu.com/view/1360.htm)上对ICMP[数据包](http://baike.baidu.com/view/25880.htm)进行[带宽](http://baike.baidu.com/view/10821.htm)限制，将ICMP占用的带宽控制在一定的范围内，这样即使有ICMP攻击，它所占用的带宽也是非常有限的，对整个网络的影响非常少；第二种方法就是在[主机](http://baike.baidu.com/view/23880.htm)上设置ICMP数据包的处理规则，最好是设定拒绝所有的ICMP数据包。

设置ICMP数据包处理规则的方法也有两种，一种是在[操作系统](http://baike.baidu.com/view/880.htm)上设置[包过滤](http://baike.baidu.com/view/150853.htm)，另一种是在[主机](http://baike.baidu.com/view/23880.htm)上安装[防火墙](http://baike.baidu.com/view/3067.htm)。具体设置如下：

**1．在Windows 2000 Server中设置ICMP过滤**

[Windows 2000 Server](http://baike.baidu.com/view/765690.htm)提供了“[路由](http://baike.baidu.com/view/18655.htm)与[远程访问](http://baike.baidu.com/view/183974.htm)”服务，但是默认情况下是没有启动的，因此首先要启动它：点击“[管理工具](http://baike.baidu.com/view/1251417.htm)”中的“路由与远程访问”，启动设置向导。在其中选择“手动配置服务器”项，点击[下一步]按钮。稍等片刻后，系统会提示“路由和[远程访问服务](http://baike.baidu.com/view/253162.htm)现在已被安装。要开始服务吗？”，点击[是]按钮启动服务。

[](http://baike.baidu.com/pic/ICMP/572452/0/a28d62d9a83a41f139012fe7?fr=lemma&ct=single)图1

服务启动后，在[计算机](http://baike.baidu.com/view/3314.htm)名称的分支下会出现一个“IP[路由选择](http://baike.baidu.com/view/632880.htm)”，点击它展开分支，再点击“常规”，会在右边出现服务器中的网络连接（即[网卡](http://baike.baidu.com/view/4230.htm)）。用[鼠标](http://baike.baidu.com/view/2199.htm)右键点击你要配置的网络连接，在弹出的菜单中点击“属性”，会弹出一个网络连接属性的窗口，如图1所示。

图1中有两个按钮，一个是“输入筛选器”（指对此服务器接受的[数据包](http://baike.baidu.com/view/25880.htm)进行筛选），另一个是“输出筛选器”（指对此服务器发送的数据包进行筛选），这里应该点击[输入筛选器] 按钮，会弹出一个“添加筛选器”窗口，再点击[添加]按钮，表示要增加一个筛选条件。

在“协议”右边的下拉列表中选择“ICMP”，在随后出现的“ICMP类型”和“ICMP编码”中均输入“255”，代表所有

[](http://baike.baidu.com/pic/ICMP/572452/0/35e940dfc4818b0b495403ed?fr=lemma&ct=single)图2

的ICMP类型及其编码。ICMP有许多不同的类型（Ping就是一种类型），每种类型也有许多不同的状态，用不同的“编码”来表示。因为其类型和编码很复杂，这里不再叙述。

点击[确定]按钮返回“输入筛选器”窗口，此时会发现“筛选器”列表中多了一项内容（如图2所示）。点击[确定]按钮返回“[本地连接](http://baike.baidu.com/view/179248.htm)”窗口，再点击[确定]按钮，此时筛选器就生效了，从其他[计算机](http://baike.baidu.com/view/3314.htm)上Ping这台[主机](http://baike.baidu.com/view/23880.htm)就不会成功了。

[](http://baike.baidu.com/pic/ICMP/572452/0/b94f65ec7fd176c92f2e21eb?fr=lemma&ct=single)图3

**2． 用防火墙设置ICMP过滤**

许多[防火墙](http://baike.baidu.com/view/3067.htm)在默认情况下都启用了ICMP过滤的功能。如果没有启用，只要选中“防御ICMP攻击”、“防止别人用[ping命令](http://baike.baidu.com/view/36175.htm)[探测](http://baike.baidu.com/view/716064.htm)”就可以了，如图3所示。

## ICMP防御方法

**选择合适的防火墙**

有效防止ICMP攻击，[防火墙](http://baike.baidu.com/view/3067.htm)应该具有状态检测、细致的[数据包](http://baike.baidu.com/view/25880.htm)完整性检查和很好的过滤规则控制功能。

[状态检测防火墙](http://baike.baidu.com/view/1638561.htm)通过跟踪它的连接状态，动态允许外出数据包的响应信息进入防火墙所保护的网络。例如，状态检测防火墙可以记录一个出去的 PING（ICMP Echo Request），在接下来的一个确定的时间段内，允许目标主机响应的ICMP Echo Reply直接发送给前面发出了PING命令的IP，除此之外的其他ICMP Echo Reply消息都会被防火墙阻止。与此形成对比的是，包过滤类型的防火墙允许所有的ICMP Echo Reply消息进入防火墙所保护的网络了。许多路由器和基于[Linux内核](http://baike.baidu.com/view/573460.htm)2.2或以前版本的防火墙系统，都属于包过滤型，用户应该避免选择这些系统。

新的攻击不断出现，防火墙仅仅能够防止已知攻击是远远不够的。通过对所有数据包进行细致分析，删除非法的数据包，防火墙可以防止已知和未知的 DoS攻击。这就要求防火墙能够进行数据包一致性检查。[安全策略](http://baike.baidu.com/view/160028.htm)需要针对ICMP进行细致的控制。因此防火墙应该允许对ICMP类型、代码和包大小进行过滤，并且能够控制连接时间和ICMP包的生成速率。

**配置防火墙以预防攻击**

一旦选择了合适的防火墙，用户应该配置一个合理的安全策略。以下是被普遍认可的防火墙安全配置惯例，可供管理员在系统安全性和易用性之间作出权衡。

防火墙应该强制执行一个缺省的拒绝策略。除了出站的ICMP Echo Request、出站的ICMP Source Quench、进站的TTL Exceeded和进站的ICMP Destination Unreachable之外，所有的ICMP消息类型都应该被阻止。

## ICMPICMP类型

这是一个完整的ICMP类型的列表：**Table C-1. ICMP类型**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TYPE** | **CODE** | **Description** | **Query** | **Error** |
| 0 | 0 | Echo Reply——回显应答（Ping应答） | x |  |
| 3 | 0 | Network Unreachable——网络不可达 |  | x |
| 3 | 1 | Host Unreachable——主机不可达 |  | x |
| 3 | 2 | Protocol Unreachable——协议不可达 |  | x |
| 3 | 3 | Port Unreachable——端口不可达 |  | x |
| 3 | 4 | Fragmentation needed but no frag. bit set——需要进行分片但设置不分片比特 |  | x |
| 3 | 5 | Source routing failed——源站选路失败 |  | x |
| 3 | 6 | Destination network unknown——目的网络未知 |  | x |
| 3 | 7 | Destination host unknown——目的主机未知 |  | x |
| 3 | 8 | Source host isolated (obsolete)——源主机被隔离（作废不用） |  | x |
| 3 | 9 | Destination network administratively prohibited——目的网络被强制禁止 |  | x |
| 3 | 10 | Destination host administratively prohibited——目的主机被强制禁止 |  | x |
| 3 | 11 | Network unreachable for TOS——由于服务类型TOS，网络不可达 |  | x |
| 3 | 12 | Host unreachable for TOS——由于服务类型TOS，主机不可达 |  | x |
| 3 | 13 | Communication administratively prohibited by filtering——由于过滤，通信被强制禁止 |  | x |
| 3 | 14 | Host precedence violation——主机越权 |  | x |
| 3 | 15 | Precedence cutoff in effect——优先中止生效 |  | x |
| 4 | 0 | Source quench——源端被关闭（基本流控制） |  |  |
| 5 | 0 | Redirect for network——对网络重定向 |  |  |
| 5 | 1 | Redirect for host——对主机重定向 |  |  |
| 5 | 2 | Redirect for TOS and network——对服务类型和网络重定向 |  |  |
| 5 | 3 | Redirect for TOS and host——对服务类型和主机重定向 |  |  |
| 8 | 0 | Echo request——回显请求（Ping请求） | x |  |
| 9 | 0 | Router advertisement——路由器通告 |  |  |
| 10 | 0 | Route solicitation——路由器请求 |  |  |
| 11 | 0 | TTL equals 0 during transit——传输期间生存时间为0 |  | x |
| 11 | 1 | TTL equals 0 during reassembly——在数据报组装期间生存时间为0 |  | x |
| 12 | 0 | IP header bad (catchall error)——坏的IP首部（包括各种差错） |  | x |
| 12 | 1 | Required options missing——缺少必需的选项 |  | x |
| 13 | 0 | Timestamp request (obsolete)——时间戳请求（作废不用） | x |  |
| 14 |  | Timestamp reply (obsolete)——时间戳应答（作废不用） | x |  |
| 15 | 0 | Information request (obsolete)——信息请求（作废不用） | x |  |
| 16 | 0 | Information reply (obsolete)——信息应答（作废不用） | x |  |
| 17 | 0 | Address mask request——地址掩码请求 | x |  |
| 18 | 0 | Address mask reply——地址掩码应答 |  |  |

# [ping 原理与ICMP协议](http://www.cnblogs.com/Akagi201/archive/2012/03/26/2418475.html)

## **ping** 的**原理**

**ping** 程序是用来探测**主机到主机之间是否可通信**，如果不能**ping**到某台主机，表明不能和这台主机建立连接。**ping** 使用的是**ICMP协议**，**它发送icmp回送请求消息给目的主机**。**ICMP协议规定：目的主机必须返回ICMP回送应答消息给源主机。**如果源主机在一定时间内收到应答，则认为主机可达。

**ICMP协议通过IP协议发送的，IP协议是一种无连接的，不可靠的数据包协议。**在Unix/Linux，序列号从0开始计数，依次递增。而Windows　**ping**程序的ICMP序列号是没有规律。

**ICMP协议在实际传输中数据包：20字节IP首部 + 8字节ICMP首部+ 1472字节<数据大小>38字节**

**ICMP报文格式:IP首部(20字节)+8位类型+8位代码+16位校验和+(不同的类型和代码，格式也有所不同)**

**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

## ICMP——

### 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。

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
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）

而差错报文则产生在数据传送发生错误的时候。就不赘述了。

### 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，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。

# TTL –百度百科

TTL是 Time To Live的缩写，该字段指定IP包被路由器丢弃之前允许通过的最大网段数量。TTL是IPv4包头的一个8 bit字段。

## TTL功能特点

在IPv4包头中TTL是一个8 bit字段，它位于IPv4包的第9个字节。如下图所示，每

一行表示 32 bit（4字节），位从0开始编号，即0~31。

[](http://baike.baidu.com/pic/TTL/130248/0/8326cffc1e178a8249c94535f503738da877e8aa?fr=lemma&ct=single)ipv4报文头部的TTL字段

TTL的作用是限制IP数据包在计算机网络中的存在的时间。TTL的最大值是255，TTL的一个推荐值是64。

虽然TTL从字面上翻译，是可以存活的时间，但实际上TTL是IP数据包在计算机网络中可以转发的最大跳数。TTL字段由IP数据包的发送者设置，在IP数据包从源到目的的整个转发路径上，每经过一个路由器，路由器都会修改这个TTL字段值，具体的做法是把该TTL的值减1，然后再将IP包转发出去。如果在IP包到达目的IP之前，TTL减少为0，路由器将会丢弃收到的TTL=0的IP包并向IP包的发送者发送 ICMP time exceeded消息。

TTL的主要作用是避免IP包在网络中的无限循环和收发，节省了网络资源，并能使IP包的发送者能收到告警消息。

TTL 是由发送[主机](http://baike.baidu.com/view/23880.htm)设置的，以防止数据包不断在[IP](http://baike.baidu.com/view/8370.htm)互联网络上永不终止地循环。转发[IP](http://baike.baidu.com/view/8370.htm)数据包时，要求路由器至少将 TTL 减小 1。

TTL值的[注册表](http://baike.baidu.com/view/979.htm)位置[HKEY\_LOCAL\_MACHINE](http://baike.baidu.com/view/1387918.htm)\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters 其中有个DefaultTTL的DWORD值，其数据就是默认的TTL值了，我们可以修改，但不能大于十进制的255。Windows系统设置后重启才生效。

生存时间，就是一条[域名解析](http://baike.baidu.com/view/30676.htm)记录在[DNS](http://baike.baidu.com/view/22276.htm)服务器中的存留时间。当各地的DNS服务器接受到解析请求时，就会向域名指定的NS服务器(权威域名服务器）发出解析请求从而获得解析记录；在获得这个记录之后，记录会在DNS服务器(各地的缓存服务器，也叫递归域名服务器）中保存一段时间，这段时间内如果再接到这个域名的解析请求，DNS服务器将不再向NS服务器发出请求，而是直接返回刚才获得的记录；而这个记录在DNS服务器上保留的时间，就是TTL值。[1]

## TTL示例

[生存时间](http://baike.baidu.com/view/7854346.htm)，简单的说它表示DNS记录在DNS服务器上缓存时间。要理解它的值，请先看下面的一个例子：[2]

假设，有这样一个[域名](http://baike.baidu.com/view/43.htm)myhost.baiwan-han.c o m（其实，这就是一条DNS记录，通常表示在baiwan-han.c o m域中有一台名为myhost的主机）对应IP地址为1.1.1.1，它的TTL为10分钟。这个域名或称这条记录存储在一台名为-ns.baiwan-han.c o m的DNS服务器上。

此时如果有一个用户在浏览器中键入一下地址（又

[](http://baike.baidu.com/pic/TTL/130248/0/246cca2ad6177488033bf61c?fr=lemma&ct=single)TTL

称URL）：myhost.baiwan-han.c o m 那么会发生些什么呢？

该访问者指定的DNS服务器（或是他的ISP，互联网服务商，动态分配给他的)8.8.8.8就会试图为他解释myhost.baiwan-han.c o m，当然8.8.8.8这台DNS服务器由于没有包含myhost.baiwan-han.c o m这条信息，因此无法立即解析，但是通过全球DNS的递归查询后，最终定位到-ns.baiwan-han.c o m这台DNS服务器，-ns.baiwan-han.c o m这台DNS服务器将myhost.baiwan-han.c o m对应的IP地址1.1.1.1告诉8.8.8.8这台DNS服务器，然有再由8.8.8.8告诉用户结果。8.8.8.8为了以后加快对myhost.baiwan-han.c o m这条记录的解析，就将刚才的1.1.1.1结果保留一段时间，这就是TTL时间，在这段时间内如果用户又有对myhost.baiwan-han.c o m这条记录的解析请求，它就直接告诉用户1.1.1.1，当TTL到期则又会重复上面的过程。

## 典型的例子

C:\>ping 192.168.0.1  
  
Pinging 192.168.0.1 with 32 bytes of data:  
  
Reply from 192.168.0.1: bytes=32 time<10ms TTL=128  
Reply from 192.168.0.1: bytes=32 time<10ms TTL=128  
Reply from 192.168.0.1: bytes=32 time<10ms TTL=128  
Reply from 192.168.0.1: bytes=32 time<10ms TTL=128  
  
Ping statistics for 192.168.0.1:  
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
Minimum = 0ms, Maximum = 0ms, Average = 0ms  
  
TTL：生存时间  
  
 指定数据报被路由器丢弃之前允许通过的网段数量。  
  
 TTL 是由发送主机设置的，以防止数据包不断在 IP 互联网络上永不终止地循环。转发 IP 数据包时，要求路由器至少将 TTL 减小 1。   
  
 使用PING时涉及到的 ICMP 报文类型  
  
一个为ICMP请求回显（ICMP Echo Request）  
  
一个为ICMP回显应答（ICMP Echo Reply）  
  
TTL 字段值可以帮助我们识别[操作系统](http://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf)类型。  
  
 UNIX 及类 UNIX [操作系统](http://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf) ICMP 回显应答的 TTL 字段值为 255   
 Compaq Tru64 5.0 ICMP 回显应答的 TTL 字段值为 64   
 微软 [Windows](http://www.baidu.com/s?wd=Windows&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf) NT/[2K](http://www.baidu.com/s?wd=2K&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf)[操作系统](http://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf) ICMP 回显应答的 TTL 字段值为 128   
 微软 [Windows](http://www.baidu.com/s?wd=Windows&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf) [95](http://www.baidu.com/s?wd=95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf) 操作系统 ICMP 回显应答的 TTL 字段值为 32  
 当然，返回的[TTL值](http://www.baidu.com/s?wd=TTL%E5%80%BC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf)是相同的  
  
 但有些情况下有所特殊  
  
LINUX Kernel 2.2.x & 2.4.x ICMP 回显应答的 TTL 字段值为 64   
FreeBSD 4.1, 4.0, 3.4;   
Sun Solaris 2.5.1, 2.6, 2.7, 2.8;   
OpenBSD 2.6, 2.7,   
NetBSD  
HP UX 10.20  
ICMP 回显应答的 TTL 字段值为 255   
[Windows](http://www.baidu.com/s?wd=Windows&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf) [95](http://www.baidu.com/s?wd=95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf)/98/98SE  
Windows ME  
ICMP 回显应答的 TTL 字段值为 32   
Windows NT4 WRKS  
Windows NT4 Server  
Windows 2000  
ICMP 回显应答的 TTL 字段值为 128  
 这样，我们就可以通过这种方法来辨别操作系统  
  
 TTL  
  
LINUX 64  
WIN[2K](http://www.baidu.com/s?wd=2K&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYuW7brHIWuWFbPynYPjbv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWc1rjmdrf)/NT 128  
WINDOWS 系列 32  
UNIX 系列 255

# TTL expired in transit （TTL 传输中过期）

**ping一个ip却返回另一个ip**

简单来说，TTL全程Time to Live，意思就是生存周期以下是参考资料ping是一个很常用的小工具，它主要用于确定网络的连通性问题 使用ping命令后，常见的出错信息通常分为3种：  
1、Unknown host：不知名主机这种出错信息的意思是，该远程主机的名字不能被域名服务器 DNS 转换成IP地址  
故障原因可能是域名服务器有故障，或者其名字不正确，或者网络管理员的系统与远程主机之间的通信线路有故障 飞  
2、Noanswer：无响应这种故障说明本地系统有一条通向中心主机的路由，但却接收不到它发给该中心主机的任何信?  
息 故障原因可能是下列之一：中心主机没有工作；本地或中心主机网络配置不正确：本地或中心的路由器没有；1：：作：  
通信线路有故障；中心主机存在路由选择问题  
3、Request timbd out：超时工作站与中心主机的连接超时，数据包全部丢失of原因：可能是到路由器的连接出现  
问题，或路由器不能通过，也可能是中心主机已经关机或死机 如何用ping命令查找无法上网的原因  
1．Ping命令的语法格式：  
有必要先给不了解Ping命令的人介绍一卜Ping命令的具体语法格式：ping目的地址[参数1J[参数2]……  
其中目的地址是指被测试计算机的IP地址或域名 主要参数有：  
a：解析主机地址  
n：数据：发出的测试包的个数，缺省值为4  
l：数值：所发送缓冲区的大小  
t：继续执行Ping命令，直到用户按Ctrl^C终上  
有关hng的其他参数，可通过在MS-DOS提示符—卜运行Ping或Ping— 命令来查看  
2．hng命令的应用技巧：  
用Ping：：[：具检查网络服务器和任意一台客户端上TCP／IP协议的：]二作情况时，只要在网络中其他任何一台计算机上Ping  
该计算机的IP地址即可 例如要检查网络文件服务器192．192．225．225HPQW上的TCP／IP协议二[：作是否正常，只要在  
开始菜单下的“运行”子项中键入Ping 192．192．225．225就可以了 如果HPQW的TCP／IP协议：[：作正常，即会以DOS  
屏幕方式显示如下所示的信息：  
Pinging 192．192．225．225 with 32 byteS of dara：  
Reply from 192．192．225，225：bytes=32 time=lms TTL=128  
Reply from 192．192，225．225：bytes=32 time<1mS TTL=128

Reply from 192．192．225．225：byteS’32 time

Reply from 192．192．225．225：byteS‘32 timePing StatiStiCe for 192．192．225．225：

PacketS：Sent二4，ReceiVed二4，LOSt二0 0％lOSS  
Approximate round trip timeS in milli-secondS：  
Minimum=Oms，Maximum=1mS，Average=OmS  
以上返回了4个测试数据包，其中bytes=32表示测试中发送的数据包大小是32个字节，“me<10ms表示与对方主机  
往返一次所用的时间小于10毫秒，TTL=128表示当前测试使用的TTL Time to Live 值为128 系统默认值  
如果网络有问题，则返回如下所示的响应失败信息：  
Pinging 192．192，225．225 with 32 bytes of data  
RequeSt timed out．  
RequeSt timed out．  
RequeSt timed Out．  
RequeSt timed out．  
Ping StatiStiCe for 192．192．225，225：  
PacketS：Sent=4，ReceiVed=0，LOSt＼二4 100％lOSS  
Minimum‘0ms，Maximum=OmS，Average’0mS  
网络故障：出现第二种情况时，建议从以上几个方面来着手排查：一是看被测试计算机是否已安装了TCP／IP协议：  
二是检查一下被测试计算机的网卡安装是否正确且是否已经连通：三是看被测试计算机的TCP／IP协议是否与网F  
有效的绑定 具体方法是通过选择“开始一设置一控制面板一网络”来查看 ：如果通过以上几个步骤的检查还没有  
发现问题的症结，建议重新安装并设置一，‘厂TCP／”协议，如果是TCP／IP协议的问题，这时绝对可以彻底解决  
按照上述方法，我们还可以用Ping命令来检查任意一台客户湍计算机上TCP／IP的工作情况 例如我们要检查网络任  
  
一客户端“机房0厂上的TCP／IP协议的配置和工作情况，可直接在该台机器上Ping本机的IP地址，若返回成功的信  
息，说明IP地虹LB己置无误，若失败则应检查IP地址的配置? 赏&uuml; 韵虏街杞 校菏紫认燃觳橐弧げ氛 鐾 纾 氐?  
看一下该IP地址是否正在被其他用户使用，然后再看一下该工作站是否已正确连入网络 很多情况下用户没有登陆网  
络也会出现此种情况，这可是低级错误啊 最后检查网—E的I／0地址lIRQ值和DMA值，这些值是否与其他设备发生  
了冲突 其中最后一项的检查非常重要，也常被许多用户所忽视，即使是Ping成功后也要进行此项的检查 因为当Ping  
本机的IP地址成功后，仅表明本机的IP地址配置没有问题，但并不能说明网卡的配置完全正确 这时虽然在本机的  
“网上邻居”中能够看到本机的计算机名，可就是无法与其他的用户连通，不知问题出在何处，其实问题往往就出在  
网卡上  
简单来说，TTL全程Time to Live，意思就是生存周期  
首先要说明ping命令是使用的网络层协议ICMP，所以TTL指的是一个网络层的网络数据包 package 的生存周期，这句话不懂的先回去复习OSI7层协议去 第一个问题，为什么要有生存周期这个概念 很显然，一个package从一台机器到另一台机器中间需要经过很长的路径，显然这个路径不是单一的，是很复杂的，并且很可能存在环路如果一个数据包在传输过程中进入了环路，如果不终止它的话，它会一直循环下去，如果很多个数据包都这样循环的话，那对于网络来说这就是灾难了所以需要在包中设置这样一个值，包在每经过一个节点，将这个值减1，反复这样操作，最终可能造成2个结果：包在这个值还为正数的时候到达了目的地，或者是在经过一定数量的节点后，这个值减为了0 前者代表完成了一次正常的传输，后者代表包可能选择了一条非常长的路径甚至是进入了环路，这显然不是我们期望的，所以在这个值为0的时候，网络设备将不会再传递这个包而是直接将他抛弃，并发送一个通知给包的源地址，说这个包已死  
其实TTL值这个东西本身并代表不了什么，对于使用者来说，关心的问题应该是包是否到达了目的地而不是经过了几个节点后到达 但是TTL值还是可以得到有意思的信息的 每个操作系统对TTL值得定义都不同，这个值甚至可以通过修改某些系统的网络参数来修改，例如Win2000默认为128，通过注册表也可以修改而Linux大多定义为64 不过一般来说，很少有人会去修改自己机器的这个值的，这就给了我们机会可以通过ping的回显TTL来大体判断一台机器是什么操作系统 以我公司2台机器为例  
看如下命令

D:Documents and Settingshx>ping 61 152 93 131

Pinging 61 152 93 131 with 32 bytes of data:Reply from 61 152 93 131: bytes=32 time=21ms TTL=118

Reply from 61 152 93 131: bytes=32 time=19ms TTL=118  
Reply from 61 152 93 131: bytes=32 time=18ms TTL=118  
Reply from 61 152 93 131: bytes=32 time=22ms TTL=118Ping statistics for 61 152 93 131:  
Packets: Sent = 4， Received = 4， Lost = 0 0% loss  
Approximate round trip times in milli-seconds:  
Minimum = 18ms， Maximum = 22ms， Average = 20msD:Documents and Settingshx>ping 61 152 104 40Pinging 61 152 104 40 with 32 bytes of data:Reply from 61 152 104 40: bytes=32 time=28ms TTL=54  
Reply from 61 152 104 40: bytes=32 time=18ms TTL=54  
Reply from 61 152 104 40: bytes=32 time=18ms TTL=54  
Reply from 61 152 104 40: bytes=32 time=13ms TTL=54Ping statistics for 61 152 104 40:  
Packets: Sent = 4， Received = 4， Lost = 0 0% loss  
Approximate round trip times in milli-seconds:  
  
Minimum = 13ms， Maximum = 28ms， Average = 19ms  
第一台TTL为118，则基本可以判断这是一台Windows机器，从我的机器到这台机器经过了10个节点，因为128-118=10 而第二台应该是台Linux，理由一样64-54=10  
了解了上面的东西，可能有人会有一些疑问，例如以下：1，不是说包可能走很多路径吗，为什么我看到的4个包TTL都是一样的，没有出现不同 这是由于包经过的路径是经过了一些最优选择算法来定下来的，在网络拓扑稳定一段时间后，包的路由路径也会相对稳定在一个最短路径上 具体怎么算出来的要去研究路由算法了，不在讨论之列 2，对于上面例子第二台机器，为什么不认为它是经过了74个节点的Windows机器 因为128-74=54 对于这个问题，我们要引入另外一个很好的ICMP协议工具 不过首先要声明的是，一个包经过74个节点这个有些恐怖，这样的路径还是不用为好 要介绍的这个工具是tracert \*nix下为traceroute ，让我们来看对上面的第二台机器用这个命令的结果

D:Documents and Settingshx>tracert 61 152 104 40

Tracing route to 61 152 104 40 over a maximum of 30 hops1 13 ms 16 ms 9 ms 10 120 32 1

2 9 ms 9 ms 11 ms 219 233 244 105  
3 12 ms 10 ms 10 ms 219 233 238 173  
4 15 ms 15 ms 17 ms 219 233 238 13  
5 14 ms 19 ms 19 ms 202 96 222 73  
6 14 ms 17 ms 13 ms 202 96 222 121  
7 14 ms 15 ms 14 ms 61 152 81 86  
8 15 ms 14 ms 13 ms 61 152 87 162  
9 16 ms 16 ms 28 ms 61 152 99 26  
10 12 ms 13 ms 18 ms 61 152 99 94

11 14 ms 18 ms 16 ms 61 152 104 40

Trace complete 从这个命令的结果能够看到从我的机器到服务器所走的路由，确实是11个节点 上面说10个好像是我犯了忘了算0的错误了，应该是64-54+1，嘿嘿 ，而不是128的TTL经过了70多个节点

既然已经说到这里了，不妨顺便说说关于这两个ICMP命令的高级一点的东西  
首先是ping命令，其实ping有这样一个参数，可以无视操作系统默认TTL值而使用自己定义的值来发送ICMP Request包  
例如还是用那台Linux机器，用以下命令：

D:Documents and Settingshx>ping 61 152 104 40 -i 11

Pinging 61 152 104 40 with 32 bytes of data:Reply from 61 152 104 40: bytes=32 time=10ms TTL=54

Reply from 61 152 104 40: bytes=32 time=13ms TTL=54  
Reply from 61 152 104 40: bytes=32 time=10ms TTL=54  
Reply from 61 152 104 40: bytes=32 time=13ms TTL=54Ping statistics for 61 152 104 40:  
Packets: Sent = 4， Received = 4， Lost = 0 0% loss ，  
Approximate round trip times in milli-seconds:  
Minimum = 10ms， Maximum = 13ms， Average = 11msD:Documents and Settingshx>  
这个命令我们定义了发包的TTL为11，而前面我们知道，我到这台服务器是要经过11个节点的，所以这个输出和以前没什么不同 现在再用这个试试看：

D:Documents and Settingshx>ping 61 152 104 40 -i 10

Pinging 61 152 104 40 with 32 bytes of data:Reply from 61 152 99 94: TTL expired in transit

Reply from 61 152 99 94: TTL expired in transit  
Reply from 61 152 99 94: TTL expired in transit  
Reply from 61 152 99 94: TTL expired in transit Ping statistics for 61 152 104 40:  
Packets: Sent = 4， Received = 4， Lost = 0 0% loss ，  
Approximate round trip times in milli-seconds:  
Minimum = 0ms， Maximum = 0ms， Average = 0msD:Documents and Settingshx>可以看到，结果不一样了，我定义了TTL为10来发包，结果是TTL expired in transit 就是说在到达服务器之前这个包的生命周期就结束了注意看这句话前面的ip，这个ip恰好是我们前面tracert结果到服务器之前的最后1个ip，包的TTL就是在这里减少到0了，根据我们前面的讨论，当TTL减为0时设备会丢弃包并发送一个TTL过期的ICMP反馈给源地址，这里的结果就是最好的证明  
通过这里再次又证明了从我机器到服务器是经过了11个节点而不是70多个，呵呵  
最后再巩固一下知识，有人可能觉得tracer这个命令很神奇，可以发现一个包所经过的路由路径 其实这个命令的原理就在我们上面的讨论中 想象一下，如果我给目的服务器发送一个TTL为1的包，结果会怎样  
根据前面的讨论，在包港出发的第一个节点，TTL就会减少为0，这时这个节点就会回应TTL失效的反? 飧龌赜Π 松璞副旧淼?p地址，这样我们就得到了路由路径的第一个节点的地址  
因此，我们继续发送TTL=2的包，也就受到第二个节点的TTL失效回应依次类推，我们一个一个的发现，当最终返回的结果不是TTL失效而是ICMP Response的时候，我们的tracert也就结束了，就是这么简单

顺便补一句ping命令还有个-n的参数指定要发包的数量，指定了这个数字就会按照你的要求来发包了而不是默认的4个包 如果使用-t参数的话，命令会一直发包直到你强行中止它

<http://sandyuk8.spaces.live.com/blog/cns!e212824a1a70151f!468.entry>

<http://www.sainao.com/blog/u/10/archives/2007/177.html>

**首先我们要知道什么是Ping命令，怎么用Ping命令**

点击开始菜单，点运行，输入“cmd”点确定，（不用输入引号，大小写均可。）

在打开的黑色cmd窗口中，输入“ping xxxxxx”（ 不用输入引号，大小写均可），按回车即可看到ping结果，其中xxxx可以是网址也可以是IP。

比如：

Ping bbs.lgole.com

或

Ping 127.0.0.1

其中，ping网址的时候不需要输入“http://”这个东西。

**“Ping”的幕后过程**

我们以下面一个网络为例：有A、B、C、D四台机子，一台路由RA，子网掩码均为255.255.255.0，默认路由为192.168.0.1

**1.在同一网段内**

　　在主机A上运行“Ping 192.168.0.5”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“192.168.0.5”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“192.168.0.5”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，并想办法得到192.168.0.5的MAC地址（物理地址，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），以便交给数据链路层构建一个数据帧。关键就在这里，IP层协议通过机器B的IP地址和自己的子网掩码，发现它跟自己属同一网络，就直接在本网络内查找这台机器的MAC，如果以前两机有过通信，在A机的ARP缓存表应该有B机IP与其MAC的映射关系，如果没有，就发一个ARP请求广播，得到B机的MAC， 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。

　　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。

**2.不在同一网段内**

　　在主机A上运行“Ping 192.168.1.4”后，开始跟上面一样，到了怎样得到MAC地址时，IP协议通过计算发现D机与自己不在同一网段内，就直接将交由路由处理，也就是将路由的MAC取过来，至于怎样得到路由的MAC，跟上面一样，先在ARP缓存表找，找不到就广播吧。路由得到这个数据帧后，再跟主机D进行联系，如果找不到，就向主机A返回一个超时的信息。

**对Ping后返回信息的分析**

　1.Request timed out

　　这是大家经常碰到的提示信息，很多文章中说这是对方机器置了过滤ICMP数据包，从上面工作过程来看，这是不完全正确的，至少有下几种情况。

　　（1） 对方已关机，或者网络上根本没有这个地址：比如在上图中主机A中PING 192.168.0.7 ，或者主机B关机了，在主机A中PING 192.168.0.5 都会得到超时的信息。

　　（2）对方与自己不在同一网段内，通过路由也无法找到对方，但有时对方确实是存在的，当然不存在也是返回超时的信息。

　　（3）对方确实存在，但设置了ICMP数据包过滤（比如防火墙设置）。

　　怎样知道对方是存在，还是不存在呢，可以用带参数 -a 的Ping命令探测对方，如果能得到对方的NETBIOS名称，则说明对方是存在的，是有防火墙设置，如果得不到，多半是对方不存在或关机，或不在同一网段内。

　　（4）错误设置IP地址

　　正常情况下，一台主机应该有一个网卡，一个IP地址，或多个网卡，多个IP地址（这些地址一定要处于不同的IP子网）。但如果一台电脑的“拨号网络适配器”（相当于一块软网卡）的TCP/IP设置中，设置了一个与网卡IP地址处于同一子网的IP地址，这样，在IP层协议看来，这台主机就有两个不同的接口处于同一网段内。当从这台主机Ping其他的机器时，会存在这样的问题：

　　A.主机不知道将数据包发到哪个网络接口，因为有两个网络接口都连接在同一网段。

　　B.主机不知道用哪个地址作为数据包的源地址。因此，从这台主机去Ping其他机器，IP层协议会无法处理，超时后，Ping 就会给出一个“超时无应答”的错误信息提示。但从其他主机Ping这台主机时，请求包从特定的网卡来，ICMP只须简单地将目的、源地址互换，并更改一些标志即可，ICMP应答包能顺利发出，其他主机也就能成功Ping通这台机器了。

　2.Destination host Unreachable

　　（1） 对方与自己不在同一网段内，而自己又未设置默认的路由，比如上例中A机中不设定默认的路由，运行Ping 192.168.0.1.4就会出现“Destination host Unreachable”。

　　（2）网线出了故障

　　这里要说明一下“destination host unreachable”和 “time out”的区别，如果所经过的路由器的路由表中具有到达目标的路由，而目标因为原因不可到达，这时候会出现“time out”，如果路由表中连到达目标的路由都没有，那就会出现“destination host unreachable”。

　3.Bad IP address

　　这个信息表示您可能没有连接到DNS服务器，所以无法解析这个IP地址，也可能是IP地址不存在。

　4.Source quench received

　　这个信息比较特殊，它出现的机率很少。它表示对方或中途的服务器繁忙无法回应。

　5.Unknown host——不知名主机

　　这种出错信息的意思是，该远程主机的名字不能被域名服务器（DNS）转换成IP地址。故障原因可能是域名服务器有故障，或者其名字不正确，或者网络管理员的系统与远程主机之间的通信线路有故障。

　6.No answer——无响应

这种故障说明本地系统有一条通向中心主机的路由，但却接收不到它发给该中心主机的任何信息。故障原因可能是下列之一：中心主机没有工作；本地或中心主机网络配置不正确；本地或中心的路由器没有工作；通信线路有故障；中心主机存在路由选择问题。

7.Ping 127.0.0.1：127.0.0.1是本地循环地址 ，如果本地址无法Ping通，则表明本地机TCP/IP协议不能正常工作。

　8.no rout to host：网卡工作不正常。

　9.transmit failed，error code：10043网卡驱动不正常。

　10.unknown host name：DNS配置不正确。

# [IOCP之accept、AcceptEx、WSAAccept的区别 .](http://blog.csdn.net/zhongguoren666/article/details/7858588)

1. accept、WSAAccept是同步操作，AcceptEx是异步操作

2. WSAAccept函数早accept函数基础上添加了条件函数判断是否接受客户端连接

3. AcceptEx是异步的，可以同时发出多个AcceptEx请求，支持重叠IO操作.

# WSAAccept()

### 目录

* 1[**简述：**](javascript:void(0))
* 2[**返回值：**](javascript:void(0))
* 3[**错误代码：**](javascript:void(0))
* 1[**简述：**](javascript:void(0))
* 2[**返回值：**](javascript:void(0))
* 3[**错误代码：**](javascript:void(0))

## 简述：/WSAAccept()

根据条件[函数](http://www.baike.com/sowiki/%E5%87%BD%E6%95%B0?prd=content_doc_search)的[返回值](http://www.baike.com/sowiki/%E8%BF%94%E5%9B%9E%E5%80%BC?prd=content_doc_search)有条件地接受连接，同时（可选地）[创建](http://www.baike.com/sowiki/%E5%88%9B%E5%BB%BA?prd=content_doc_search)和/或加入一个[套接口组](http://www.baike.com/sowiki/%E5%A5%97%E6%8E%A5%E5%8F%A3%E7%BB%84?prd=content_doc_search)。  
  
          SOCKET WSAAPI WSAAccept ( SOCKET s, struct  
          sockaddr FAR \* addr, int FAR \* addrlen,  
          LPCONDITIONPROC lpfnCondition, DWORD  
          dwCallbackData );  
  
s：标识一个套接口的描述字，该套接口在listen()后[监听连接](javascript:linkredwin('监听连接');)。  
addr：（可选）指针，指向存放通讯层所知的连接实体地址的缓冲区。addr参数的具体格式由套接口创建时产生的地址族决定。  
addrlen：（可选）指针，指向存放addr地址长度的整形数。  
lpfnCondition：（可选的）用户提供的条件函数的进程实例地址。该函数根据参数传入的[调用者](javascript:linkredwin('调用者');)信息作出接受或拒绝的决定，并通过给结果参数赋予特定的值来（可选地）创建和/或加入一个[套接口组](http://www.baike.com/sowiki/%E5%A5%97%E6%8E%A5%E5%8F%A3%E7%BB%84?prd=content_doc_search)。  
dwCallbackData：作为条件函数参数返回给应用程序的回调数据。WinSock不分析该参数。

## 返回值：/WSAAccept()

  若无错误发生，WSAAccept()函数返回所接受套接口的描述字。否则的话，将返回INVALID\_SOCKET错误，应用程序可通过WSAGetLastError()来获取相应的错误代码。  
  addrlen参数引用的整形数初始时包含了addr参数所指向的空间数，在调用返回时包含了返回地址的实际长度。

## 错误代码：/WSAAccept()

WSANOTINITIALISED                在调用本API之前应成功调用WSAStartup()。  
WSAECONNREFUSED                根据条件函数的返回值（CF\_REJECT）强制拒绝连接请求。  
WSAENETDOWN                        网络子系统失效。  
WSAEFAULT                         addrlen参数太小（小于sockaddr结构的大小），或者lpfnCondition并不是用户空间的一部分。  
WSAEINTR                                通过[WSACancelBlockingCall()](http://www.baike.com/sowiki/WSACancelBlockingCall%28%29?prd=content_doc_search)函数取消（阻塞）调用。  
WSAEINPROGRESS                一个阻塞WinSock调用正在进行。  
WSAEINVAL                                WSAAccept()调用前未执行listen()调用；条件函数中的g[参数非法](http://www.baike.com/sowiki/%E5%8F%82%E6%95%B0%E9%9D%9E%E6%B3%95?prd=content_doc_search)；条件函数的返回值非法；套接口处于非法状态。  
WSAEMFILE                                WSAAccept()调用时排队队列非空，且无可用套接口描述字。  
WSAENOBUFS                        无可用缓冲区空间。  
WSAENOTSOCK                        描述字不是一个套接口。  
WSAEOPNOTSUPP                所引用的套接口不是支持面向连接服务类型的。  
WSATRY\_AGAIN                        根据条件函数的返回值(CF\_DEFER) ，连接请求被推迟。  
WSAEWOULDBLOCK                套接口标志为非阻塞，无连接请求供接受。  
WSAEACCES                                被推迟的连接请求超时或撤销。

# [FD\_READ FD\_WRITE触发时机](http://blog.csdn.net/whatday/article/details/46692619)

2015-06-30 10:51 849人阅读 [评论](http://blog.csdn.net/whatday/article/details/46692619#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/whatday/article/details/46692619#report)

本文章已收录于：

FD\_WRITE触发条件：  
1.client 通过connect（WSAConnect）首次和server建立连接时，在client端会触发FD\_WRITE事件  
2.server通过accept（WSAAccept）接受client连接请求时，在server端会触发FD\_WRITE事件  
3.send（WSASend）/sendto（WSASendTo）发送失败返回WSAEWOULDBLOCK，并且当缓冲区有可用空间时，则会触发FD\_WRITE事件  
第1.2条其实是同一种情况，在第一次建立连接时，C/S端都会触发一个FD\_WRITE事件。  
主要是3这种情况：send出去的数据其实都先存在winsock的发送缓冲区中，然后才发送出去，如果缓冲区满了，那么再调用send（WSASend,sendto,WSASendTo）的话，就会返回一个 WSAEWOULDBLOCK的错误码，接下来随着发送缓冲区中的数据被发送出去，缓冲区中出现可用空间时，一个 FD\_WRITE 事件才会被触发，这里比较容易混淆的是 FD\_WRITE 触发的前提是 缓冲区要先被充满然后随着数据的发送又出现可用空间，而不是缓冲区中有可用空间，  
  
  
  
FD\_READ事件触发条件：   
1.在数据到达socket后，并且从来没有触发过FD\_READ(也就是最开始的阶段)   
2.在数据到达socket后，并且前一个recv()调用后   
3.调用recv()后，缓冲区还有未读完的数据  
  
FD\_READ过程如下：   
1.100 bytes 数据到达,winsock2发出FD\_READ   
2.程序用recv()只读入50 bytes,还剩下50 bytes   
3.winsock2继续发出FD\_READ消息  
  
recv()返回WSAEWOULDBLOCK的情况：   
1.有数据到达，FD\_READ触发，该消息加入程序的消息队列   
2.在还没处理该消息前，程序就把数据recv()了   
3.等到处理该FD\_READ消息时，程序调用recv()就会返回WSAEWOULDBLOCK(因为数据在这之前就recv()了)  
  
FD\_READ注意：   
1.winsock2发出一个FD\_READ后，如果程序没有用recv()，即使还有数据没接收，FD\_READ也不会再触发另一个FD\_READ，要等到recv()调用后FD\_READ才会发出。   
2.对一个FD\_READ多次recv()的情形：如果程序对一个FD\_READ多次recv()将会造成触发多个空的FD\_READ，所以程序在第2次recv()前要关掉FD\_READ(可以使用WSAAsynSelect关掉FD\_READ)，然后再多次recv()。

3.recv()返回WSAECONNABORTED,WSAECONNRESET...等消息，可以不做任何处理，可以等到FD\_CLOSE事件触发时再处理

# [send和WSASend区别](http://blog.csdn.net/whatday/article/details/46691893)

2015-06-30 09:58 1400人阅读 [评论](http://blog.csdn.net/whatday/article/details/46691893#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/whatday/article/details/46691893#report)

本文章已收录于：

socket本身有一个send函数，这个函数的只能一次发送一个缓冲区，这对于在发送大量数据的时候或者数据包很多的时候就可能导致可能导致系统的低性能，主要原因在于调用太多次的send函数，导致从用户态到核心态的不断切换，而耗费了当前的CPU时钟周期。

那么解决办法就是减少调用send的次数，一种办法就是使用一个大一点的BUFFER，在发送数据的时候将多个数据包的内容COPY到这个BUFFER中，然后一次发送，这样在一定程度上减少了send的调用次数，但需要一定的编码工作。

Windows平台上有一个WSASend函数，可以支持一次发送多个BUFFER的请求，每个被发送的数据被填充到WSABUF结构中，然后传递给WSASend函数同时提供BUF的数量，这样WSASend就能上面的工作而减少send的调用次数，来提高了性能。

实际检验证明，使用WSASend可以提高50%的性能甚至更多

# [NAT穿越技术之UDP打洞](http://blog.csdn.net/markman101/article/details/5853703)

根据STUN协议(RFC3489),NAT大致分为下面四类：  
1) Full Cone

这种NAT内部的机器A连接过外网机器C后,NAT会打开一个端口.然后外网的任何发到这个打开的端口的UDP数据报都可以到达A.不管是不是C发过来的.  
例如 A:192.168.8.100 NAT:202.100.100.100 C:292.88.88.88  
A(192.168.8.100:5000) -> NAT(202.100.100.100:8000) -> C(292.88.88.88:2000)  
任何发送到 NAT(202.100.100.100:8000)的数据都可以到达A(192.168.8.100:5000)

2) Restricted Cone

这种NAT内部的机器A连接过外网的机器C后,NAT打开一个端口.然后C可以用任何端口和A通信.其他的外网机器不行.  
例如 A:192.168.8.100 NAT:202.100.100.100 C:292.88.88.88  
A(192.168.8.100:5000) -> NAT(202.100.100.100 : 8000) -> C(292.88.88.88:2000)  
任何从C发送到 NAT(202.100.100.100:8000)的数据都可以到达A(192.168.8.100:5000)

3) Port Restricted Cone

这种NAT内部的机器A连接过外网的机器C后,NAT打开一个端口.然后C可以用原来的端口和A通信.其他的外网机器不行.  
例如 A:192.168.8.100 NAT:202.100.100.100 C:292.88.88.88  
A(192.168.8.100:5000) -> NAT(202.100.100.100 : 8000) -> C(292.88.88.88:2000)  
C(202.88.88.88:2000)发送到 NAT(202.100.100.100:8000)的数据都可以到达A(192.168.8.100:5000)

以上三种NAT通称Cone NAT(圆锥形NAT).我们只能用这种NAT进行UDP打洞.

4) Symmetric（对称形）

对于这种NAT.连接不同的外部目标.原来NAT打开的端口会变化.而Cone NAT则不会.虽然可以用端口猜测.但是成功的概率很小.因此放弃这种NAT的UDP打洞.

二. UDP hole punching

首先需要明白的是，如果双方的NAT都是Full Cone NAT的话，则不需要打洞就可以直接进行通信。  
而对于双方都是Port Restricted Cone NAT的时候，则需要利用UDP打洞原理进行“先打洞，然后才能直接通信”。  
对于Cone NAT.要采用UDP打洞.需要一个公网机器server C来充当”介绍人”.处于NAT之后的内网的A,B先分别和C通信,打开各自的NAT端口.C这个时候知道A,B的公网IP: Port. 现在A和B想直接连接.比如A给B直接发包，除非B是Full Cone，否则不能通信.反之亦然.  
为什么啊？因为对于处于NAT之后的A,B。如果想A要与外界的D通信，则首先必须要A发包到D，然后A经过NAT设备NA，NA把A的内网地址和端口转换为NA的外网地址和端口。和D通信之后，D才能经过NA和A通信。也就是说，只能A和外界主动通信，外界不能主动和处于NA之后的A通信。这种包会被NA直接丢弃的。这也就是上面所说的Port Restricted Cone 的情形啊！ A(192.168.8.100:5000) -> NA(202.100.100.100:8000) -> D(292.88.88.88:2000)但是我们可以这样.  
A --- NA --- Server C --- NB --- B  
A,B 为主机；  
NA, NB 为NAT设备；  
Server C为外网的机器；  
1，如果A想与B通信；  
2，A首先连接 C, C得到A的外网NA的地址和端口；  
3，B也要连接C，C得到B的外网NB的地址和端口；  
4，A告诉C说我要和B通讯；  
5，C通过NB发信息给B，告诉B A的外网NA的地址和端口；  
6，B向NA发数据包（肯定会被NA丢弃，因为NA上并没有 A->NB 的合法session），  
但是NB上就建立了有B->NA的合法session了；  
7，B发数据包给C，让 C 通知 A，我已经把洞打好了；  
8，A接受到通知后向 B 的外网发NB数据包，这样就不会被丢弃掉了。因为对于NB来说，它看到的是A的外网NA的地址，  
而通过第6步，B已经让NA成为NB的合法通信对象了。所以当NA发数据包给NB时，NB就会接收并转发给B；  
注意： 路由器和防火墙的UDP打洞的端口有个时间限制的，在一定时间内如果没有数据通讯会自动关闭

三. 同一个NAT后的情况

如果A,B在同一个NAT后面.如果用上面的技术来进行互连.那么如果NAT支持loopback(就是本地到本地的转换),A,B可以连接,但是比较浪费带宽和NAT.  
有一种办法是,A,B和介绍人通信的时候,同时把自己的local IP也告诉服务器.A,B通信的时候,同时发local ip和公网IP.谁先到就用哪个IP.但是local ip就有可能不知道发到什么地方去了.比如A,B在不同的NAT后面但是他们各自的local ip段一样.A给B的local IP发的UDP就可能发给自己内部网里面的xxxx了.

//----------------------------------------------------------------

UDP打洞技术，其实很简单。过程很清晰，一目了然!

以上是本人从网上找的，将的比较清晰，很适合初学者。

以下是本人自己写的一个实例，感觉过程比较清晰，对初学者应该有点帮助!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//作者：紫色溟渊

//完成时间：2010年1月1日

//功能实现：UDP打洞技术

//说明：仅供技术交流，欢迎转载，转载请注明出处!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

以下给出源码，实现UDP打洞!

//client a.......source code

#include <Winsock2.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include "HostToIp.h"

void main()

{

 WORD wVersionRequested;  
 WSADATA wsaData;  
 int err;  
   
 wVersionRequested = MAKEWORD( 2, 2 );  
   
 err = WSAStartup( wVersionRequested, &wsaData );  
 if ( err != 0 ) {  
  return;  
 }  
   
 if ( LOBYTE( wsaData.wVersion ) != 2 ||  
   HIBYTE( wsaData.wVersion ) != 2 ) {  
  WSACleanup( );  
  return;   
 }  
 char c\_server\_name[]="psni.3322.org";  
 char c\_server\_ip[16];  
 HostToIP(c\_server\_name,c\_server\_ip);  
 SOCKET sockClient=socket(AF\_INET,SOCK\_DGRAM,0);  
 sockaddr\_in addrServer;  
 addrServer.sin\_addr.S\_un.S\_addr=inet\_addr(c\_server\_ip);  
 addrServer.sin\_family=AF\_INET;  
 addrServer.sin\_port=htons(6000);

// bind(sockServer,(sockaddr\*)&sockServer,sizeof(sockServer));

 char cGetInfo[100],cSendInfo[100];

 sockaddr\_in addrClient;  
 int len=sizeof(sockaddr);  
 //.........A 或者 B  
 //如果client==A，那么接收到的就是B的信息  
 memset(cSendInfo,0,100);  
 printf("local client a......请输入数据！/n");  
 gets(cSendInfo);  
 sendto(sockClient,cSendInfo,strlen(cSendInfo)+1,0,(sockaddr\*)&addrServer,len);

 //得到B的信息  
 memset(cGetInfo,0,100);  
 recvfrom(sockClient,cGetInfo,100,0,(sockaddr \*)&addrServer,&len);  
 char c\_ip[16];  
 int port;  
 char \*token ;  
 token=strtok(cGetInfo,"#");  
 strcpy(c\_ip,token);  
 token=strtok(NULL,"#");  
 port=atoi(token);

 //如果第一次失败  
 memset(cGetInfo,0,100);  
 recvfrom(sockClient,cGetInfo,100,0,(sockaddr \*)&addrServer,&len);  
 //如果说一次尝试打洞成功  
 if(strcmp(cGetInfo,"hello")==0)  
 {  
  //等待server告诉 A  ,B的port和ip  
  printf("B第一次打洞尝试打动成功/n!");  
  memset(cGetInfo,0,100);  
  recvfrom(sockClient,cGetInfo,100,0,(sockaddr \*)&addrServer,&len);  
 }  
 else if(strcmp(cGetInfo,"ok,you can send to b")==0)  
 {  
  printf("wait ok,send to b now...../n");  
 }

 //向b发送消息  
 sockaddr\_in addr\_b;  
 addr\_b.sin\_addr.S\_un.S\_addr=inet\_addr(c\_ip);  
 addr\_b.sin\_family=AF\_INET;  
 addr\_b.sin\_port=port;

 sendto(sockClient,"hello....i'm a, let's talk....",strlen("hello....i'm a, let's talk....")+1,0,(sockaddr\*)&addr\_b,len);  
 //A自己说的话  
 printf("a:--hello....i'm a, let's talk..../n");

 //等待B发送信息过来，然后A在发送信息过去  
 while(TRUE)  
 {  
  memset(cGetInfo,0,100);  
  recvfrom(sockClient,cGetInfo,100,0,(sockaddr \*)&addr\_b,&len);  
  printf("b:--%s/n",cGetInfo);  
  printf("请输入您要发送的信息 a:");  
  memset(cSendInfo,0,100);  
  gets(cSendInfo);  
  sendto(sockClient,cSendInfo,100,0,(sockaddr \*)&addr\_b,len);  
 }

 printf("udp holing......ok/n");

 closesocket(sockClient);  
 WSACleanup();  
 system("PAUSE");

}

//client b......source code

#include <Winsock2.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include "HostToIp.h"

void main()

{

 WORD wVersionRequested;  
 WSADATA wsaData;  
 int err;  
   
 wVersionRequested = MAKEWORD( 2, 2 );  
   
 err = WSAStartup( wVersionRequested, &wsaData );  
 if ( err != 0 ) {  
  return;  
 }  
   
 if ( LOBYTE( wsaData.wVersion ) != 2 ||  
   HIBYTE( wsaData.wVersion ) != 2 ) {  
  WSACleanup( );  
  return;   
 }  
 char c\_server\_name[]="psni.3322.org";  
 char c\_server\_ip[16];  
 HostToIP(c\_server\_name,c\_server\_ip);  
 SOCKET sockClient=socket(AF\_INET,SOCK\_DGRAM,0);  
 sockaddr\_in addrServer;  
 addrServer.sin\_addr.S\_un.S\_addr=inet\_addr(c\_server\_ip);  
 addrServer.sin\_family=AF\_INET;  
 addrServer.sin\_port=htons(6001);

// bind(sockServer,(sockaddr\*)&sockServer,sizeof(sockServer));

 char cGetInfo[100],cSendInfo[100];  
 char choleinfo[]="hello";  
 char choleok[]="ok";  
 sockaddr\_in addrClient;  
 int len=sizeof(sockaddr);

 //.........A 或者 B  
 //如果client==B ，那么接受到的就是A的信息

 memset(cSendInfo,0,100);  
 printf("local client b....请输入数据！/n");  
 gets(cSendInfo);  
 sendto(sockClient,cSendInfo,strlen(cSendInfo)+1,0,(sockaddr\*)&addrServer,len);

 memset(cGetInfo,0,100);  
 recvfrom(sockClient,cGetInfo,100,0,(sockaddr \*)&addrServer,&len);  
 char c\_ip[16];  
 int port;  
 char \*token ;  
 token=strtok(cGetInfo,"#");  
 strcpy(c\_ip,token);  
 token=strtok(NULL,"#");  
 port=atoi(token);  
 sockaddr\_in addr\_a;  
 addr\_a.sin\_addr.S\_un.S\_addr=inet\_addr(c\_ip);  
 addr\_a.sin\_family=AF\_INET;  
 addr\_a.sin\_port=port ;

 //向A打洞，  
 sendto(sockClient,choleinfo,strlen(choleinfo)+1,0,(sockaddr \*)&addr\_a,len);

 //告诉server打洞完成  
 sendto(sockClient,choleok,3,0,(sockaddr \*)&addrServer,len);

 //等待A发送信息过来  
 memset(cGetInfo,0,100);  
 recvfrom(sockClient,cGetInfo,100,0,(sockaddr \* )& addr\_a,&len);  
 printf("a:--%s/n",cGetInfo);

 //自己发送信息给A  
 sendto(sockClient,"hello, i get it....",strlen("hello, i get it....")+1,0,(sockaddr\*)&addr\_a,len);  
 printf("b:--hello, i get it..../n");  
 //udp hole ok，可以和A 无限发送信息  
 while(TRUE)  
 {  
  memset(cGetInfo,0,100);  
  recvfrom(sockClient,cGetInfo,100,0,(sockaddr\*)&addr\_a,&len);  
  printf("a:--%s/n",cGetInfo);  
  memset(cSendInfo,0,100);  
  printf("请输入您要发送的信息 b:");  
  gets(cSendInfo);  
  sendto(sockClient,cSendInfo,100,0,(sockaddr \*)&addr\_a,len);  
 }

 printf("udp holing .....ok....../n");

 closesocket(sockClient);  
 WSACleanup();  
 system("PAUSE");  
}

//server...................source code

#include <Winsock2.h>  
#include <stdio.h>  
#include <string.h>

void main()

{

 WORD wVersionRequested;  
 WSADATA wsaData;  
 int err;  
   
 wVersionRequested = MAKEWORD( 2, 2 );  
   
 err = WSAStartup( wVersionRequested, &wsaData );  
 if ( err != 0 ) {  
  return;  
 }  
   
 if ( LOBYTE( wsaData.wVersion ) != 2 ||  
   HIBYTE( wsaData.wVersion ) != 2 ) {  
  WSACleanup( );  
  return;   
 }

 //------------------------sock\_server\_a  
 SOCKET sockServer=socket(AF\_INET,SOCK\_DGRAM,0);  
 sockaddr\_in addrServer;  
 addrServer.sin\_addr.S\_un.S\_addr=htonl(INADDR\_ANY);  
 addrServer.sin\_family=AF\_INET;  
 addrServer.sin\_port=htons(6000);

 bind(sockServer,(sockaddr\*)&addrServer,sizeof(sockaddr));

 printf("sock\_servser\_a ...bind ....ok!/n");  
 //----------------------sock\_server\_b  
 SOCKET sockServer1=socket(AF\_INET,SOCK\_DGRAM,0);  
 sockaddr\_in addrServer1;  
 addrServer1.sin\_addr.S\_un.S\_addr=htonl(INADDR\_ANY);  
 addrServer1.sin\_family=AF\_INET;  
 addrServer1.sin\_port=htons(6001);

 bind(sockServer1,(sockaddr\*)&addrServer1,sizeof(sockaddr));  
 printf("sock\_server\_b....bind...ok!/n");

 char cGetInfo[100],cSendInfo[100]="welcome";  
 char cWriteInfo[100] ;

 sockaddr\_in addrClienta;//保存A的地址信息  
 sockaddr\_in addrClientb;//保存B的地址信息

 int len=sizeof(sockaddr);

 memset(cWriteInfo,0,100);

 //等待A来连接  
 memset(cGetInfo,0,100);  
 if(recvfrom(sockServer,cGetInfo,100,0,(sockaddr \*)&addrClienta,&len)==SOCKET\_ERROR)  
 {  
  printf("recv...error/n");  
  return ;  
 }  
 //sendto(sockServer,cWriteInfo,strlen(cWriteInfo)+1,0,(sockaddr \*)&addrClienta,len);  
 printf("clienta.......connected ...ok/n");  
   
 //等待B来连接  
 memset(cGetInfo,0,100);  
 if(recvfrom(sockServer1,cGetInfo,100,0,(sockaddr \*)&addrClientb,&len)==SOCKET\_ERROR)  
 {  
  printf("recv...error/n");  
  return ;  
 }  
 //sendto(sockServer,cWriteInfo,strlen(cWriteInfo)+1,0,(sockaddr \*)&addrClientb,len);  
 printf("clientb.......connected ...ok/n");

 //获得A的信息，然后发送给B  
 //char  str\_ip[16],str\_info[30];  
 char \*str\_ip=new char[16];  
 char \*str\_info=new char[30];   
 char \*c\_port=new char[6];

 memset(str\_ip,0,16);  
 memset(str\_info,0,30);  
 memset(c\_port,0,6);  
 str\_ip=inet\_ntoa(addrClienta.sin\_addr) ;  
 itoa(addrClienta.sin\_port,c\_port,10);  
 //str\_info=str\_ip+"#"+c\_port ;  
 strcpy(str\_info,str\_ip);  
 strcat(str\_info,"#");  
 strcat(str\_info,c\_port);  
 sendto(sockServer1,str\_info,strlen(str\_info)+1,0,(sockaddr \*)&addrClientb,len);

 memset(str\_ip,0,16);  
 memset(str\_info,0,30);  
 memset(c\_port,0,6);

 //获得B的信息，然后发送给A  
 str\_ip=inet\_ntoa(addrClientb.sin\_addr) ;  
 itoa(addrClientb.sin\_port,c\_port,10);

 strcpy(str\_info,str\_ip);  
 strcat(str\_info,"#");  
 strcat(str\_info,c\_port);

 sendto(sockServer,str\_info,strlen(str\_info)+1,0,(sockaddr \*)&addrClienta,len);

 //等待B的打洞ok消息  
 memset(cGetInfo,0,100);  
 recvfrom(sockServer1,cGetInfo,100,0,(sockaddr \*)&addrClientb,&len);

 //接受到B得打洞ok消息后，向A发送信息  
 sendto(sockServer,"ok,you can send to b",strlen("ok,you can send to b")+1,0,(sockaddr \*)&addrClienta,len);

 //服务端server的使命就完成了  
 closesocket(sockServer);  
 closesocket(sockServer1);  
 WSACleanup();  
 system("PAUSE");

}

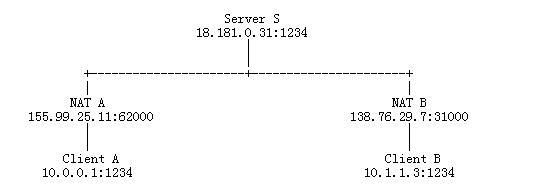
//代码写的不好，但是大致的流程已经写出来了。大家可以将其修改，让其适合自己的实际需要!

# [UDP打洞技术2](http://blog.csdn.net/markman101/article/details/5853730)

这篇文章主要要研究的，就是非常有名的"UDP打洞技术"，UDP打洞技术依赖于由公共防火墙和cone NAT，允许适当的有计划的端对端应用程序通过NAT"打洞"，即使当双方的主机都处于NAT之后。这种技术在 [RFC3027的5.1节[NAT PROT]](http://rfc.net/rfc3027.html) 中进行了重点介绍，并且在Internet[KEGEL]中进行了非正式的描叙，还应用到了最新的一些协议，例如[TEREDO,ICE]协议中。不过，我们要注意的是，"术"如其名，UDP打洞技术的可靠性全都要依赖于UDP。

这里将考虑两种典型场景，来介绍连接的双方应用程序如何按照计划的进行通信的，第一种场景，我们假设两个客户端都处于不同的NAT之后；第二种场景，我们假设两个客户端都处于同一个NAT之后，但是它们彼此都不知道(他们在同一个NAT中)。

处于不同NAT之后的客户端通信

我们假设 Client A 和 Client B 都拥有自己的私有IP地址，并且都处在不同的NAT之后，端对端的程序运行于 CLIENT A,CLIENT B,S之间，并且它们都开放了UDP端口1234。 CLIENT A和CLIENT B首先分别与S建立通信会话，这时NAT A把它自己的UDP端口62000分配给CLIENT A与S的会话，NAT B也把自己的UDP端口31000分配给CLIENT B与S的会话。如下图所示：  


假如这个时候 CLIENT A 想与 CLIENT B建立一条UDP通信直连，如果 CLIENT A只是简单的发送一个UDP信息到CLIENT B的公网地址138.76.29.7:31000的话，NAT B会不加考虑的将这个信息丢弃（除非NAT B是一个 full cone NAT），因为 这个UDP信息中所包含的地址信息，与CLIENT B和服务器S建立连接时存储在NAT B中的服务器S的地址信息不符。同样的，CLIENT B如果做同样的事情，发送的UDP信息也会被 NAT A 丢弃。

假如 CLIENT A 开始发送一个 UDP 信息到 CLIENT B 的公网地址上，与此同时，他又通过S中转发送了一个邀请信息给CLIENT B，请求CLIENT B也给CLIENT A发送一个UDP信息到 CLIENT A的公网地址上。这时CLIENT A向CLIENT B的公网IP(138.76.29.7:31000)发送的信息导致 NAT A 打开一个处于 CLIENT A的私有地址和CLIENT B的公网地址之间的新的通信会话，与此同时，NAT B 也打开了一个处于CLIENT B的私有地址和CLIENT A的公网地址(155.99.25.11:62000)之间的新的通信会话。一旦这个新的UDP会话各自向对方打开了，CLIENT A和CLIENT B之间就可以直接通信，而无需S来牵线搭桥了。(这就是所谓的打洞技术)！

# getsockname函数与getpeername函数的使用

getsockname和getpeername函数

getsockname函数用于获取与某个套接字关联的本地协议地址   
getpeername函数用于获取与某个套接字关联的外地协议地址

定义如下：

#include<sys/socket.h>

int getsockname(int sockfd, struct sockaddr \*localaddr, socklen\_t \*addrlen);

int getpeername(int sockfd, struct sockaddr \*peeraddr, socklen\_t \*addrlen);

对于这两个函数，如果函数调用成功，则返回0，如果调用出错，则返回-1。

使用这两个函数，我们可以通过套接字描述符来获取自己的IP地址和连接对端的IP地址，如在未调用bind函数的TCP客户端程序上，可以通过调用getsockname()函数获取由内核赋予该连接的本地IP地址和本地端口号，还可以在TCP的服务器端accept成功后，通过getpeername()函数来获取当前连接的客户端的IP地址和端口号。

如下面的客户端-服务器程序：

服务器端代码

/\*服务器端\*/

#define MAXLINE 4096

#define PORT 6563

#define LISTENQ 1024

#include<stdio.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

#include<arpa/inet.h>

int main() {

int listenfd, connfd;

struct sockaddr\_in servaddr;//服务器绑定的地址

struct sockaddr\_in listendAddr, connectedAddr, peerAddr;//分别表示监听的地址，连接的本地地址，连接的对端地址

int listendAddrLen, connectedAddrLen, peerLen;

char ipAddr[INET\_ADDRSTRLEN];//保存点分十进制的地址

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

servaddr.sin\_port = htons(PORT);

bind(listenfd, (struct sockaddr\*)&servaddr, sizeof(servaddr));//服务器端绑定地址

listen(listenfd, LISTENQ);

listendAddrLen = sizeof(listendAddr);

getsockname(listenfd, (struct sockaddr \*)&listendAddr, &listendAddrLen);//获取监听的地址和端口

printf("listen address = %s:%d\n", inet\_ntoa(listendAddr.sin\_addr), ntohs(listendAddr.sin\_port));

while(1) {

connfd = accept(listenfd, (struct sockaddr \*)NULL, NULL);

connectedAddrLen = sizeof(connectedAddr);

getsockname(connfd, (struct sockaddr \*)&connectedAddr, &connectedAddrLen);//获取connfd表示的连接上的本地地址

printf("connected server address = %s:%d\n", inet\_ntoa(connectedAddr.sin\_addr), ntohs(connectedAddr.sin\_port));

getpeername(connfd, (struct sockaddr \*)&peerAddr, &peerLen); //获取connfd表示的连接上的对端地址

printf("connected peer address = %s:%d\n", inet\_ntop(AF\_INET, &peerAddr.sin\_addr, ipAddr, sizeof(ipAddr)), ntohs(peerAddr.sin\_port));

}

return 0;

}

上面的代码中，在调用listen函数之后就获取监听套接字描述符对应的本地地址，在accept()函数后，由于accept返回了一个套接字描述符connfd用于表示该连接，所以可以对这个connfd调用getsockname函数和getpeername函数，分别获取内核赋予该连接的本地IP地址和连接的对端地址。

客户端代码

/\*客户端\*/

#define PORT 6563

#include<stdio.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<unistd.h>

#include<string.h>

#include<arpa/inet.h>

int main(int argc, char \*\*argv) {

struct sockaddr\_in servaddr;//服务器端地址

struct sockaddr\_in clientAddr;//客户端地址

int sockfd;

int clientAddrLen = sizeof(clientAddr);

char ipAddress[INET\_ADDRSTRLEN];//保存点分十进制的ip地址

if(argc < 2) {

printf("parameter error");

}

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

if(inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr) <= 0) {

printf("server address error\n");//地址参数不合法

}

connect(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr));//向服务器端发起连接请求

getsockname(sockfd, (struct sockaddr\*)&clientAddr, &clientAddrLen);//获取sockfd表示的连接上的本地地址

printf("client:client ddress = %s:%d\n", inet\_ntop(AF\_INET, &clientAddr.sin\_addr, ipAddress, sizeof(ipAddress)), ntohs(clientAddr.sin\_port));

return 0;

}

在客户端的代码中，调用connect函数后，即可调用getsockname来获连接上的本地地址。

代码的运行结果如下：

http://img1.tuicool.com/bm2yem.png!web

http://img0.tuicool.com/nEZ3umY.png!web

从上面的代码中可以看到，服务器端listenfd套接字描述符对应的地址即为绑定的通配IP地址和指定的端口，而connfd套接字描述符对应的连接的服务器端的地址为内核赋予的地址和用户指定的端口。

上面的客户端与服务器端的代码中使用了函数inet\_ntoa，inet\_pton对32位的地址进行转换，其中inet\_ntoa是较老的函数，与它一起的还有函数inet\_addr和inet\_ntoa，这三个函数的定义如下：

#include<arpa/inet.h>

int inet\_aton(const char \*strptr, struct in\_addr \*addrptr);

in\_addr\_t inet\_addr(const char \*strptr);

char \*inet\_ntoa(struct in\_addr inaddr);

inet\_aton与inet\_addr函数的功能类似，都是将点分十进制的字符串表示的IP地址转换成32位的网络字节序的IPv4地址。

inet\_ntoa函数将32位的网络字节序的IPv4地址转换成点分十进制的字符串表示的IP地址，inet\_addr函数已被废弃，并且这三个函数只针对IPv4地址有效，在点分十进制数串和32位的网络字节序二进制值间进行转换，如果要对于IPv4和IPv6都适用，那么使用下面两个函数：

#include<arpa/inet.h>

int inet\_pton(int family, const char \*strptr, void \*addrptr);

const char \*inet\_ntop(int family, const void \*addrptr, char \*strptr, size\_t len);

函数中的p和n分别代表 **表达式(presentation)** 和 **数值(numeric)**

所以inet\_pton函数将strptr指针所指的字符串转换为网络地址(IPv4和IPv6)，再将地址保存到addrptr指向的结构体中，inet\_ntop将网络地址转为字符串表示的地址，结果存放在strptr中，len参数是strptr的大小。   
这两个函数支持IPv4和IPv6，所以需要通过参数family来指定，当前要转换的是IPv4地址还是IPv6地址。

# 字节顺序

[编辑](javascript:;) [锁定](https://baike.baidu.com/view/10812319.htm)

本词条缺少**名片图**，补充相关内容使词条更完整，还能快速升级，赶紧来编辑吧！

字节顺序是指占内存多于一个字节类型的数据在内存中的存放顺序，通常有小端、[大端](https://baike.baidu.com/item/%E5%A4%A7%E7%AB%AF)两种字节顺序。小端[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F)指低字节数据存放在内存低地址处，高字节数据存放在内存高地址处；大端字节序是高字节数据存放在低地址处，低字节数据存放在高地址处。

中文名

字节顺序

概    念

占内存多于一个字节类型存放顺序

通常有

小端、[大端](https://baike.baidu.com/item/%E5%A4%A7%E7%AB%AF)

基    于

X86平台的PC机是小端[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F)

## 目录

1. 1 [简介](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F#1)
2. 2 [跨平台](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F#2)
3. 3 [概念](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F#3)
4. 4 [验证](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F#4)
5. ▪ [案例1](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F#4_1)
6. ▪ [案例2](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F#4_2)

## 字节顺序简介

[编辑](javascript:;)

基于X86平台的PC机是小端[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F)的，而有的嵌入式平台则是[大端](https://baike.baidu.com/item/%E5%A4%A7%E7%AB%AF)字节序的。因而对int、uint16、uint32等多于1字节类型的数据，在这些嵌入式平台上应该变换其存储顺序。通常我们认为，在空中传输的字节的顺序即网络字节序为标准顺序，考虑到与协议的一致以及与同类其它平台产品的互通，在程序中发数据包时，将[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)字节序转换为[网络字节序](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%AD%97%E8%8A%82%E5%BA%8F/12610557)，收数据包处将网络字节序转换为[主机字节序](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA%E5%AD%97%E8%8A%82%E5%BA%8F/12664928)。

## 字节顺序跨平台

[编辑](javascript:;)

在本LINUX的书里介绍到INTEL的CPU使用的小端[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F)其他比MOTOROLA

68000系列CPU使用的是[大端](https://baike.baidu.com/item/%E5%A4%A7%E7%AB%AF)字节序如果不转换 将数据通过网络发出时 比如MOTOROLA发一个16位数据：0X1234 传送到INTEL时

就被INTEL解释为0X3412 也就是4660成了13330 所以有时候需要一些函数来进行大小端字节序的转换

## 字节顺序概念

[编辑](javascript:;)

关于这大小[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F/1457160)的概念不是很想的明白[1]  数据在

内存里是具体怎么存放的形式？为什么会有CPU解释的不同？数据不是按12345678……这样的顺序一直排列的么？

如: 一个多字节值 0xFECDBA98,内存从地址100开始存放

降序: FE 对应地址100 | CD 对应地址 101 | BA 对应地址 102 | 98 对应地址 103

升序: 98 | BA | CD | FE ---->;same above

注意,我们的书写字数据表示法是从高字节位--->低字节位（从左到右）

[内存地址](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E5%9C%B0%E5%9D%80)生长方向为: 从左到右 由低到高(这是不变的)

数据为: 0x98BADCFE

降序(Big-endian)[大端](https://baike.baidu.com/item/%E5%A4%A7%E7%AB%AF)[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F)存储时 由左到右

升序([Little-endian](https://baike.baidu.com/item/Little-endian/4118225))小端字节序存储时 由右向左

## 字节顺序验证

[编辑](javascript:;)

可以自己编一个小程序验证一下(用C的[数组](https://baike.baidu.com/item/%E6%95%B0%E7%BB%84))

更简单的调用VC里的checkEndian()

[Intel处理器](https://baike.baidu.com/item/Intel%E5%A4%84%E7%90%86%E5%99%A8/2157617)的字节顺序是和DEC VAX处理器的字节顺序一致的。因此它与68000型处理器以及Internet的顺序是不同的，所以用户在使用时要特别小心以保证正确的顺序。

任何从Windows Sockets函数对[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80/150859)和[端口号](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3%E5%8F%B7)的引用和传送给Windows Sockets函数的IP地址和端口号均是按照网络顺序组织的，这也包括了[sockaddr\_in](https://baike.baidu.com/item/sockaddr_in/3917215)结构这一数据类型中的IP地址域和端口域（但不包括sin\_family域）。

### 字节顺序案例1

**如何手算主机字节顺序转换为网络字节顺序？**

**假设某16位的整数，主机字节顺序的值是21，那么它的网络字节顺序是多少？  
　　解决的步骤是：**

**1、将21化成**[**二进制**](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6/361457)**，二进制，如果不足16位就在其前面补0，补满16位。**

**21转换成二进制是：10101，在它前面补0，补满16位后就得到：**

**00000000#00010101**

**2、将这个16位二进制字符平分成两段，每段8位**

**0000000 000010101 == > 00000000#00010101**

**3、颠倒这两段的顺序，然后去掉第一个字符“1”前面的0，化成**[**十进制**](https://baike.baidu.com/item/%E5%8D%81%E8%BF%9B%E5%88%B6/6521392)**就得到了网络字节顺序的值了。**

**00000000#00010101颠倒后：00010101#00000000**

**即00 01 01 01 00 000000**

**去掉第一个“1”前面的0得到：10 10 10 00 00 00 0**

**化成十进制得到：5376**

**通过这个程序可以进行验证：**

[?](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include<stdio.h>  #include<string.h>  #include<netdb.h>  intmain(  void)  {  structservent\*server=NULL;  charsername[16]="\0";  charproname[4]="\0";    printf("请输入服务名称：");    gets(sername);  printf("请输入使用的协议：");  gets(proname);  //如果能够通过getserbyname函数找到结果，就输出结果，否则报错  if((server=getservbyname(sername,proname))!=NULL)  {    printf("  网络服务端口：\n");  printf("网络字节顺序值：%d\n",server->s\_port);  printf("主机字节顺序值：%d\n",ntohs(server->s\_port));  }  else  {  printf("该服务不存在\n");  }  return0;  } |

**编译后，按提示输入ftp和tcp，就得到网络字节顺序值是5376，主机字节顺序值是21。**

**16位的**[**转换函数**](https://baike.baidu.com/item/%E8%BD%AC%E6%8D%A2%E5%87%BD%E6%95%B0/5271305)**：ntohs和htons  
　　ntohs(network to host short)是将网络字节顺序转换为主机字节顺序，返回值是一个16位的整数，即2个字节长度的整数（1字节=8位）short int，也可以写作uint16\_t。  
　　htons(host to network short)是将主机字节顺序转换为网络字节顺序，返回值也是一个16位的整数short int。**

**32位转换函数ntohl和htonl。  
　　ntohl(network to host long)是将网络字节顺序转换为主机字节顺序，返回值是一个16位的整数，即2个字节长度的整数（1字节=8位）long int，也可以写作uint32\_t。  
　　htonl(host to network long)是将主机字节顺序转换为网络字节顺序，返回值也是一个16位的整数long int。**  
　　[**IP地址**](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80/150859)**使用的是32位的**[**无符号整数**](https://baike.baidu.com/item/%E6%97%A0%E7%AC%A6%E5%8F%B7%E6%95%B4%E6%95%B0/9203544)**，所以，在对IP地址进行处理的时候，就需要用到32位的转换了。  
　　一个**[**点分十进制**](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6/2186100)**的IP地址是192.168.0.1，还原成原来的**[**二进制**](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6/361457)[**原码**](https://baike.baidu.com/item/%E5%8E%9F%E7%A0%81/1097586)**是：**

**11000000 10101000 0000000 00000 0001**

**这是主机字符顺序存储的值。  
　　将其按照每个字节分隔开来，即每8位分隔开来，得到：  
　　11000000 10101000 00000000 00000001**  
　　 **将这4段**[**二进制编码**](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6%E7%BC%96%E7%A0%81/1758517)**进行完全颠倒，就得到了网络字节顺序：  
　　00000001000000001010100011000000**  
　　 **去掉第一个字符“1”前面无效的0，得到：  
　　10 00 00 00 01 01 01 00 01 10 00 00 0**

**化成**[**十进制**](https://baike.baidu.com/item/%E5%8D%81%E8%BF%9B%E5%88%B6/6521392)**得到：16820416  
　　inet\_addr函数可以将一个**[**点分十进制**](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6/2186100)**的**[**IP地址**](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80/150859)**转换成网络字节顺序的长整形十进制值。同样的，inet\_ntoa函数将网络字节顺序的长整形十进制数值的IP地址转换成点分十进制。**

### 字节顺序案例2

**将一个**[**点分十进制**](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6/2186100)**的**[**IP地址**](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80/150859)**转换成它的**[**二进制**](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6/361457)[**原码**](https://baike.baidu.com/item/%E5%8E%9F%E7%A0%81/1097586) **分析：用inet\_addr函数，转换点分十进制的IP为网络字节顺序的**[**长整型**](https://baike.baidu.com/item/%E9%95%BF%E6%95%B4%E5%9E%8B/325017)[**十进制**](https://baike.baidu.com/item/%E5%8D%81%E8%BF%9B%E5%88%B6/6521392)**数值后，再用ntohl函数将其转换成主机字节顺序的长整型整数，然后使用一个自定义函数将这个主机字节顺序的长整形整数转换成二进制。**

[?](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E9%A1%BA%E5%BA%8F)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include<stdio.h>  #include<sys/socket.h>  #include<netdb.h>  #include<arpa/inet.h>  #include<netinet/in.h>  #include<string.h>  //  十进制转二进制函数，参数为无符号超长整形（防止溢出）  voiddec(constunsignedlonglongintx)  {  if(x/2>0)  {  dec(x/2);    printf("%d",x%2);  }  else  {  printf("%d",x);  }  }  intmain(  void)  {  charipaddr[15]="\0";  unsignedlonglonginthostaddr=0;//主机字节IP的值  printf("请输入  IP地址：");    gets(ipaddr);  hostaddr=ntohl(  inet\_addr(ipaddr));    printf("IP地址的网络字节顺序的值：%ld\n",inet\_addr(ipaddr));  //调用dec函数，为了防止溢出将hostaddr  强制转换成无符号超长整形后再转换成  二进制  printf("二进制形式是：\n");  dec((unsignedlonglongint)hostaddr);  printf("\n");  return0;  } |