本文由mofeellassie贡献
doc1。
第一篇
目录： 目录：
一． 关于 ARP 协议的基础知识 1. 2. ARP 的工作原理 ARP 包的格式
手把手教你玩转 ARP 包
作者： 作者：
CSDN VC/MFC 网络编程 PiggyXP ^_^
一.
关于 ARP 协议的基础知识
1. ARP 的工作原理 本来我不想在此重复那些遍地都是的关于 ARP 的基本常识，但是为了保持文章的完整性以及照顾初学 者，我就再啰嗦一些文字吧，资深读者可以直接跳过此节。
我们都知道以太网设备比如网卡都有自己全球唯一的 MAC 地址，它们是以 MAC 地址来传输以太网数据 包的，但是它们却识别不了我们 IP 包中的 IP 地址，所以我们在以太网中进行 IP 通信的时候就需要一个协 议来建立 IP 地址与 MAC 地址的对应关系，以使 IP 数据包能发到一个确定的地方去。这就是 ARP(Address Resolution Protocol，地址解析协议)。
讲到此处，我们可以在命令行窗口中，输入 arp －a 来看一下效果，类似于这样的条目 210.118.45.100 00-0b-5f-e6-c5-d7 dynamic
就是我们电脑里存储的关于 IP 地址与 MAC 地址的对应关系，dynamic 表示是临时存储在 ARP 缓存中的 条目，过一段时间就会超时被删除(xp/2003 系统是 2 分钟)。
这样一来，比如我们的电脑要和一台机器比如 210.118.45.1 通信的时候，它会首先去检查 arp 缓存， 查找是否有对应的 arp 条目，如果没有，它就会给这个以太网络发 ARP 请求包 请求包广播询问 210.118.45.1 的对 应 MAC 地址，当然，网络中每台电脑都会收到这个请求包，但是它们发现 210.118.45.1 并非自己，就不会 做出相应，而 210.118.45.1 就会给我们的电脑回复一个 ARP 应答包 应答包，告诉我们它的 MAC 地址是 xx-xx-xxxx-xx-xx,于是我们电脑的 ARP 缓存就会相应刷新,多了这么一条： 210.118.45.1 xx-xx-xx-xx-xx-xx dynamic
为什么要有这么一个 ARP 缓存呢， 试想一下如果没有缓存， 我们每发一个 IP 包都要发个广播查询地址， 岂不是又浪费带宽又浪费资源？ 而且我们的网络设备是无法识别 ARP 包的真伪的，如果我们按照 ARP 的格式来发送数据包，只 要信息有效计算机就会根据包中的内容做相应的反应.
试想一下,如果我们按照 ARP 响应包的相应的内容来刷新自己的 ARP 缓存中的列表，嘿嘿，那我们岂不 是可以根据这点在没有安全防范的网络中玩些 ARP 包的小把戏了？在后面的文章里我就手把手来教你们如 何填充发送 ARP 包，不过先别急，我们再继续学点基础知识^_^
2. ARP 包的格式 ARP
既然我们要来做一个我们自己的 ARP 包，当然首先要学习一下 ARP 包的格式。

从网络底层看来，一个 ARP 包是分为两个部分的，前面一个是物理帧头 物理帧头，后面一个才是 ARP 帧。 物理帧头 首先，物理帧头，它将存在于任何一个协议数据包的前面，我们称之为 DLC Header，因为这个 帧头是在数据链路层构造的，并且其主要内容为收发双方的物理地址，以便硬件设备识别。 DLC Header 字段 接收方 MAC 发送方 MAC Ethertype 长 度(Byte) 6 6 2 0x0806 图 1 物理帧头格式 0x0806 是 ARP 帧的类型值 默认值 备注 广播时,为 ff-ff-ff-ff-ff-ff
图 1 是需要我们填充的物理帧头的格式,我们可以看到需要我们填充的仅仅是发送端和接收端 的物理地址罢了,是不是很简单呢? 接下来我们看一下 ARP 帧的格式. ARP Frame 字段 硬件类型 上层协议 类型 MAC 地址长 度 IP 地址长 度 操作码 发送方 MAC 发送方 IP 接收方 MAC 接收方 IP 填充数据 长 度(Byte) 2 2 默认值 0x1 0x0800 备注 以太网类型值 上层协议为 IP 协议

1
0x6
以太网 MAC 地址长度为 6
1 2 6 4 6 4 18
0x4
IP 地址长度为 4 0x1 表示 ARP 请求包,0x2 表示应答包
因为物理帧最小长度为 64 字节,前面的 42 字节 再加上 4 个 CRC 校验字节,还差 18 个字节 图 2 ARP 帧格式
我们可以看到需要我们填充的同样也只是 MAC,IP,再加上一个 1 或 2 的操作码而已。
……
3.ARP 包的填充 1) 请求包的填充： 比如我们的电脑 MAC 地址为 aa-aa-aa-aa-aa-aa, IP 为 192.168.0.1 我们想要查询 192.168.0.99 的 MAC 地址, 应该怎么来做呢？
首先填充 DLC Header, 通过前面的学习我们知道, 想要知道某个计算机对应的 MAC 地址是 要给全网发送广播的, 所以接收方 MAC 肯定是 ffffffffffff, 发送方 MAC 当然是自己啦, 于是 接收方 发送方 我们的 DLC Header 就填充完成了， 如图, 加粗的是我们要手动输入的值(当然我编的程序比较智 能, 会根据你选择的 ARP 包类型帮你自动填入一些字段, 你一用便知^_^)。
DLC Header 长度 字段 (Byte) 接收方 MAC 发送方 MAC Ethertype 6 6 2 ffffffffffff aaaaaaaaaaaa 0x0806 填充值
图 3 ARP 请求包中 DLC Header 内容
接下来是 ARP 帧, 请求包的操作码 操作码当然是 1,发送方的 MAC 以及 IP 当然填入我们自己的， 发 操作码 然后要注意一下, 这里的接收方 IP 填入我们要查询的那个 IP 地址, 就是 192.168.0.99 了, 而 接收方 接收方 MAC 填入任意值就行,不起作用, 于是, 如图,
ARP Frame 长度 字段 (Byte) 硬件类型 上层协议类型 MAC 地址长度 2 2 1 1 0800 6 填充值
IP 地址长度 操作码 发送方 MAC 发送方 IP 接收方 MAC 接收方 IP 填充数据

1 2 6 4 6 4 18 图 4 ARP 请求包中 ARP 帧的内容
4 1 aaaaaaaaaaaa 192.168.0.1 任意值 xxxxxxxxxxxx 192.168.0.99 0
如果我们构造一个这样的包发送出去, 如果 192.168.0.99 存在且是活动的, 我们马上就会 收到一个 192.168.0.99 发来的一个响应包, 我们可以查看一下我们的 ARP 缓存列表, 是不是多 了一项类似这样的条目： 192.168.0.99 是不是很神奇呢？ 我们再来看一下 ARP 响应包的构造 bb-bb-bb-bb-bb-bb
2) 响应包的填充 有了前面详细的解说， 你肯定就能自己说出响应包的填充方法来了吧， 所以我就不细说了， 列两个表就好了
比如说给 192.168.0.99 (MAC 为 bb-bb-bb-bb-bb-bb) 发一个 ARP 响应包, 告诉它我们的 MAC 地址为 aa-aa-aa-aa-aa-aa, 就是如此来填充各个字段
DLC Header 长度 字段 (Byte) 接收方 MAC 发送方 MAC 6 6 bbbbbbbbbbbb aaaaaaaaaaaa 填充值
Ethertype
2 图 5 ARP 响应包中 DLC Header 内容
0x0806
ARP Frame 长度 字段 (Byte) 硬件类型 上层协议类型 MAC 地址长度 IP 地址长度 操作码 发送方 MAC 发送方 IP 接收方 MAC 接收方 IP 填充数据 2 2 1 1 2 6 4 6 4 18 1 0800 6 4 2 aaaaaaaaaaaa 192.168.0.1 bbbbbbbbbbbb 192.168.0.99 0 填充值
图 6 ARP 响应包中 ARP 帧的内容
这样 192.168.0.99 的 ARP 缓存中就会多了一条关于我们 192.168.0.1 的地址映射。 好了, 终于到了编程实现它的时候了^_^

二. 发送 ARP 包的编程实现 1. 填充数据包 上面的那些关于 ARP 包各个字段的表格，对应在程序里就是结构体，对应于上面的表格，于是我们需 要三个下面这样的结构体 // DLC Header typedef struct tagDLCHeader { unsigned char unsigned char unsigned short DesMAC[6]; SrcMAC[6]; Ethertype; /* destination HW addrress */ /* source HW addresss */ /* ethernet type */

} DLCHEADER, *PDLCHEADER;

// ARP Frame typedef struct tagARPFrame { unsigned short unsigned short unsigned char unsigned char unsigned short HW_Type; Prot_Type; HW_Addr_Len; Prot_Addr_Len; Opcode; /* hardware address */ /* protocol address */ /* length of hardware address */ /* length of protocol address */ /* ARP/RARP */

unsigned char unsigned long unsigned char unsigned long unsigned char } ARPFRAME, *PARPFRAME;

Send_HW_Addr[6]; Send_Prot_Addr; Targ_HW_Addr[6]; Targ_Prot_Addr; padding[18];

/* sender hardware address */ /* sender protocol address */ /* target hardware address */ /* target protocol address */

// ARP Packet = DLC header + ARP Frame typedef struct tagARPPacket { DLCHEADER ARPFRAME dlcHeader; arpFrame;

} ARPPACKET, *PARPPACKET;

这些结构体一定能看懂吧，在程序中就是对号入座就好了

……

1.

填充数据包

下面我举个填充包头的例子，我首先定义个了一个转换字符的函数，如下

/** ********* * * * * * Name & Params:: formatStrToMAC (const LPSTR lpHWAddrStr : 用户输入的 MAC 地址字符串 unsigned char *HWAddr : 返回的 MAC 地址字符串(赋给数据包结构体)

* * *

) Purpose: 将用户输入的 MAC 地址字符转成数据包结构体需要的格式

** ********/ void formatStrToMAC(const LPSTR lpHWAddrStr, unsigned char *HWAddr) { unsigned int i, index = 0, value, temp; unsigned char c;

_strlwr(lpHWAddrStr);

// 转换成小写

for (i = 0; i < strlen(lpHWAddrStr); i++) { c = *(lpHWAddrStr + i); if ((c>='0' && c<='9') || (c>='a' && c<='f')) { if (c>='0' && c<='9') temp = c - '0'; if (c>='a' && c<='f') temp = c - 'a' + 0xa; if ((index % 2) == 1) { value = value*0x10 + temp; HWAddr[index/2] = value; } else value = temp; index++; } if (index == 12) break; } } // 数字 // 字母

// 开始填充各个字段 ARPPACKET ARPPacket; 构体变量 // 定义 ARPPACKET 结

memset(&ARPPacket, 0, sizeof(ARPPACKET));

// 数据包初始化

formatStrToMAC(“DLC 源 MAC 字符串”,ARPPacket.dlcHeader.SrcMAC); LC 帧头 formatStrToMAC(“DLC 目的 MAC 字符串”,ARPPacket.dlcHeader.DesMAC);

// D

formatStrToMAC(“ARP 源 MAC 字符串”,ARPPacket.arpFrame.Send_HW_Addr); // 源 MAC ARPPacket.arpFrame.Send_Prot_Addr = inet_addr(srcIP); // 源 IP

formatStrToMAC(“ARP 目的 MAC 字符串”,ARPPacket.arpFrame.Targ_HW_Add

r); // 目的 MAC ARPPacket.arpFrame.Targ_Prot_Addr = inet_addr(desIP); // 目的 IP
ARPPacket.arpFrame.Opcode = htons((unsigned short)arpType); 包类型
// arp
// 自动填充的常量 ARPPacket.dlcHeader.Ethertype = htons((unsigned short)0x0806); // DLC H eader 的以太网类型 ARPPacket.arpFrame.HW_Type = htons((unsigned short)1); 型 ARPPacket.arpFrame.Prot_Type = htons((unsigned short)0x0800); 协议类型 ARPPacket.arpFrame.HW_Addr_Len = (unsigned char)6; 地址长度 // MAC // 上层 // 硬件类
ARPPacket.arpFrame.Prot_Addr_Len = (unsigned char)4; 长度
// IP 地址
That's all ! ^_^ 填充完毕之后，我们需要做的就是把我们的 ARPPACKET 结构体发送出去
2.发送 ARP 数据包：
我们发送 ARP 包就要用到 winpcap 的 api 了，具体步骤及函数是这样的，为了简单易懂，我 把错误处理的地方都去掉了，详见代码 /** ****** * * * * * * Name & Params:: SendARPPacket() Purpose: 发送 ARP 数据包 Remarks: 用的是 winpcap 的 api 函数
** ******/ void SendARPPacket() { char *AdapterDeviceName =GetCurAdapterName(); 字 // 首先获得获得网卡名
lpAdapter = PacketOpenAdapter(AdapterDeviceName); 网卡
// 根据网卡名字打开
lpPacket = PacketAllocatePacket();
// 给 PACKET 结构指针分配内存
PacketInitPacket(lpPacket, &ARPPacket, sizeof(ARPPacket)); //初始化 PACKET 结构指针 // 其中的 ARPPacket 就是我们先前填充的 ARP 包
PacketSetNumWrites(lpAdapter, 1);
// 每次只发送一个包
PacketSendPacket(lpAdapter, lpPacket, true)
// Send !!!!! ^_^
PacketFreePacket(lpPacket); PacketCloseAdapter(lpAdapter); }
// 释放资源
呵呵，至此，关于 ARP 包最关键的部分就讲完了，你现在就可以来随心所欲的发送自己的 ARP 包了
既然作为一篇“科普文章”，接下来我再讲一讲与整个项目有关的附加步骤以及说明
三. 附加步骤以及说明 1. 如何在 VC 中使用 winpcap 驱动 虽然 winpcap 开发包使用起来非常简便， 但是前期准备工作还是要费一番功夫的， 缺一不可。 ^_^ 首先就是要安装它的驱动程序了，可以到它的主页下载，更新很快的 http://winpcap.polito.it/install/default.htm 下载 WinPcap auto-installer (driver +DLLs)，直接安装就好了，或者我提供的代码包 里面也有。 希望以后用 winpcap 作开发的朋友，还需要下载 Developer's pack，解压即可。
然后， 需要设置我们工程的附加包含目录为我们下载 Developer's pack 开发包的 Inclul de 目录，连接器的附加依赖库设置为 Developer's pack 的 lib 目录。 当然，因为我们的工作比较简单，就是借用 winpcap 发送数据包而已，所以只用从
winpcap 开发包的 include 文件夹中，拷贝 Packet32.h，到我们的工程来，并且包含它就可 以，但是要注意，Packet32.h 本身还要包含一个 Devioctl.h，也要一并拷贝进来，当然还有运 行库 Packet.lib，一共就是需要拷贝 3 个文件了，如果加入库不用我多说了吧，在工程里面设 置，或者是在需要它的地方加入 #pragma comm

ent(lib, "Packet.lib")了。
整个项目其实可以分为四个部分，填充数据包、发送数据包、枚举系统网卡列表和 填充数据包、发送数据包、枚举系统网卡列表 填充数据包 相关信息以及枚举系统 ARP 缓存列表 枚举系统 缓存列表，下面我再讲一下如何获得系统的网卡以及 ARP 列 表， 这两个部分都要用到 IP Helper 的 api， 所以要包含以及库文件 Iphlpapi.lib， 其实都是很简单的，只用寥寥几行就 OK 了 2. 枚举系统网卡以及信息
最好是先定义关于网卡信息的一个结构体，这样显得结构比较清晰 // 网卡信息 typedef struct tagAdapterInfo { char szDeviceName[128]; char szIPAddrStr[16]; char szHWAddrStr[18]; DWORD dwIndex; }INFO_ADAPTER, *PINFO_ADAPTER; // 名字 // IP // MAC // 编号
/** **********) Purpose: 获得系统的网卡信息，并将其添加到 list 控件中 Remarks: Name & Params:: AddAdapInfoToList (CListCtrl& list : CARPPlayerDlg 传入的 list 句柄
*
获得网卡 IP 及 MAC 用到了 IpHelper api GetAdaptersInfo
** */
void AddAdapInfoToList(CListCtrl& list) { char tempChar; ULONG uListSize=1; PIP_ADAPTER_INFO pAdapter; 信息 int nAdapterIndex = 0; // 定义 PIP_ADAPTER_INFO 结构存储网卡
DWORD dwRet = GetAdaptersInfo((PIP_ADAPTER_INFO)&tempChar, &uListS ize);//关键函数
if (dwRet == ERROR_BUFFER_OVERFLOW) { PIP_ADAPTER_INFO pAdapterListBuffer = (PIP_ADAPTER_INFO)new(char[uList Size]); dwRet = GetAdaptersInfo(pAdapterListBuffer, &uListSize); if (dwRet == ERROR_SUCCESS) { pAdapter = pAdapterListBuffer; while (pAdapter) 到 List 中 { // 网卡名字 CString strTemp = pAdapter->AdapterName; strTemp = "\\Device\\NPF_" + strTemp; list.InsertItem(nAdapterIndex,strTemp); strcpy(AdapterList[nAdapterIndex].szDeviceName,strTemp); // 加上前缀 // 枚举网卡然后将相关条目添加
// IP strcpy(AdapterList[nAdapterIndex].szIPAddrStr, pAdapter->IpAddressList.IpAddress.String); list.SetItemText(nAdapterIndex,1,AdapterList[nAdapterIndex].szIPAddrSt r); // MAC formatMACToStr(AdapterList[nAdapterIndex].szHWAddrStr, pAdapter->A ddress); list.SetItemText(nAdapterIndex,2,AdapterLis[nAdapterIndex].szHWAddrSt r); // 网卡编号 AdapterList[nAdapterIndex].dwIndex = pAdapter->Index;
pAdapter = pAdapter->Next; nAdapterIndex ++; } delete pAdapterListBuffer; } } }
2)获取 ARP 条目列表 // ARP 条目信息 typedef struct tagARPInfo { char szIPAddrStr[16]; char szHWAddrStr[18]; DWORD dwType; }INFO_ARP, *PINFO_ARP; // IP // MAC // 类型
/** **********) Purpose: 读入系统的 ARP 缓存列表,.并添加到对话框中 Remarks: 用到了 IpHelper api GetIpNetTable 而且用到了 WinSock 的 api, 所以要包含<WinSock2.h> Name & Params:: AddARPInfoToList (CListCtrl& list : CARPPlayerDlg 传入的 list 句柄 用户选中的网卡编号
const short nAdapterIndex :
** / void AddARPInfoToList(CListCtrl& list,const short nAdapterIndex) { char tempChar; DWORD dwListSize = 1; DWORD dwRet; in_addr inaddr; list.DeleteAllItems();
dwRet = GetIpNetTable((PMIB_IPNETTABLE)&tempChar, &dwListSize, TRU E); // 关键函数 if (dwRet == ERROR_INSUFFICIENT_BUFFER) { PMIB_IPNETTABLE pI

```
pNetTable = (PMIB_IPNETTABLE)new(char[dwListSiz e]);
    dwRet = GetIpNetTable(pIpNetTable, &dwListSize, TRUE); if (dwRet == E
RROR_SUCCESS) { for (int i=0; i<(int)pIpNetTable->dwNumEntries; i++) { //
IP inaddr.S_un.S_addr = pIpNetTable->table[i].dwAddr; strcpy( ARPList[i].s
zIPAddrStr, inet_ntoa(inaddr) ); // MAC formatMACToStr( ARPList[i].szHWAdd
rStr, pIpNetTable->table[i].bP hysAddr ); // Type ARPList[i].dwType = pIpN
etTable->table[i].dwType;
    if (AdapterList[nAdapterIndex].dwIndex != pIpNetTable->table[i].d wIn
dex) continue;
    list.InsertItem(i,ARPList[i].szIPAddrStr); list.SetItemText(i,1,ARPLi
st[i].szHWAddrStr); switch(ARPList[i].dwType) { case 3: list.SetItemText(i
,2,"Dynamic"); break; case 4: list.SetItemText(i,2,"Static"); break; case
1: list.SetItemText(i,2,"Invalid"); default: list.SetItemText(i,2,"Other")
; // 根据 type 的值来转换成字符显示
    } } } delete pIpNetTable; } } 这样一来，我们基本上大功告成了，其他还有
一些东西在这里就不讲了，大家可以下载我 的代码看看就好了。 下面我们来用 ARP
 包玩一些小把戏 ^_^。
    既然我们可以自己来填充数据包，那么来玩些 ARP 的“ 小游戏 ”欺骗就是易如
反掌了，当然， 是在没有安全防护的网络里 ，比如只有 hub 或者交换机把你们相连
，而没有路由分段……^_^ 下面我就由浅入深的讲一些介绍一些关于 ARP 的小伎俩。

    1. 小伎俩 1) 你可以试着发一个请求包广播，其中的 ARP 帧里关于你的信息填
成这样： (为了节省篇幅，我只写需要特别指出的填充字段) 发送方 MAC 发送方 IP
 6 4 随便乱填一个错误的 填上你的 IP
    出现什么结果？是不是弹出一个 IP 地址冲突的提示？呵呵，同样的道理，如果
发送方 IP 填成别人的，然后每隔 1 秒发一次……..-_-b
    2)
    比如你们都靠一个网关 192.168.0.1 上网 ，如果你想让 192.168.0.77 上不了
网，就可以伪 装成网关给 192.168.0.77 发一个错误的 ARP 响应包，like this
    发送方 MAC 发送方 IP
    6 4
    随便乱填一个错误的 网关 IP 192.168.0.1
    接收方就填 192.168.0.77 的相关信息，发送之后，它还能上网不？ 这样能折
腾他好一阵子了，只要它的系统得不到正确的到网关的 ARP 映射表它就一直上不了网
 了 ^_^ 呵呵类似的伎俩还有很多，不过只停留在这点东西上也没什么意思，还是看
看稍微高深 一点的吧^_^
    2.
    ARP 欺骗
    因为在以太网里，网络设备就是靠 MAC 信息来识别的计算机的，比如 A 电脑知
道 MAC 地址为 22 -22-22-22-22-22 的电脑是 B，而如果我给 A 发送一个 ARP 响应
包，告诉它我的 MAC 是 22-22-22-22 -22-22 的话，A 同样会认为我的计算机是 B
了，那么好，我们设想有这么一个环境，
    A 的防火墙只对 IP 为 192.168.0.2 MAC 为 22-22-22-22-22-22 的 B 有信任
关系， 而且 A 打开了 21 端口提供 FTP 服务，正常情况下因为防火墙的缘故我们的
计算机是连不到 A 的， 于是我们想办法让 B down 掉，或者在它关机的时候，我们
把我们的 IP 改成 B 的 192.168.0.2， 然后给 A 发送一个 ARP 回应包，告诉 A 更
新一下 ARP 缓存列表，192.168.0.2 的 IP 映射到我们的 MAC 地址上来，于是，奇
迹出现了，我们可以连到 A 的 FTP 上了，防火墙失效了^_^ 不过这个办法只能在同
网段内生效，如果我们和 A 不在一个网段内，那就要复杂的多了，还要 配合 ICMP
的重定向来控制报文的路由，这个我准备在以后阐述 ICMP 包的时候详细讲解，就不
再此 多说了。 3. 基于 ARP 欺骗的监听原理 监听的技术有很多了，不过我们常用的
 sniffer 工具只能在基于 hub 的网络中起作用，碰到哪怕 是交换机都无能为力了，
```

这个时候我们的 ARP 欺骗技术就派上用场了。 还是假设有三台主机 A,B,还有我们的主机，位于同一个交换式局域网中 A 与 B 正在通信，如果我们想要刺探 A——>B 通信的内容，于是我们就可以给 A 发送一个伪造的 ARP 回应包，告诉 A，B 的 IP 对应的 MAC 条目为我们的 MAC 地址，于是，A 也就会相应的刷新自己的 A RP 缓存，将发给 B 的数据，源源不断的发送到我们的主机上来，这样我就可以对接收到的数据包进行 分析就好了，达到了监听的目的。当然，因为动态 ARP 缓存是动态的，有超时时间的，所以我们必须 每隔一段时间就给 A 发送一个 ARP 回应包 虽然我们这样达到了目的，但是 A 到 B 的通信却被停止了，为了不让 B 发现，我们还要对每次接 收到的数据包进行转发，全部都转发给 B，这样就天衣无缝了^_^ 同样的，如果我们还想监听 B A 的数据包，一样给 B 发一个 ARP 回应包，告诉 B，A 的 IP 对

应的 MAC 是我们的主机 MAC，于是 B 到 A 的数据包也源源不断的发到我们的主机上来了，当然我们也 是一样要对这些数据包进行转发，如图： A <> 我们的主机 <> B

一切都无误的话，A 和 B 的通信内容就这样不知不觉的被我们监听到了^_^ 具体的代码实现由于篇幅的关系我就不放在这里讲了， 如果需要我就专门另写篇文章附上完整代 码吧 至此，我们的 ARP 基础知识就讲完了，但愿您能从中有所收获

后记： 因为本人开发都是使用 VC++.net 2003，所以没有安装.net 的朋友是打不开工程的，可以试一下 vckbase 上的工程转换工具，本人没有试过，不保证有效 http://www.vckbase.com/tools/assist/prjconverter.rar 而且本文的代码使用了 winpcap 开发包，是要另外安装 ainpcap 驱动。 读者可以安装我代码包里的驱动，不过它更新很快，可以到它主页上去下载最新版本 http://winpcap.polito.it/install/default.htm 不做开发的读者，只用下载并安装这个就可以了 WinPcap auto-installer (driver +DLLs) 我的原文及源码下载地址稍后贴出，请关注本帖^_^