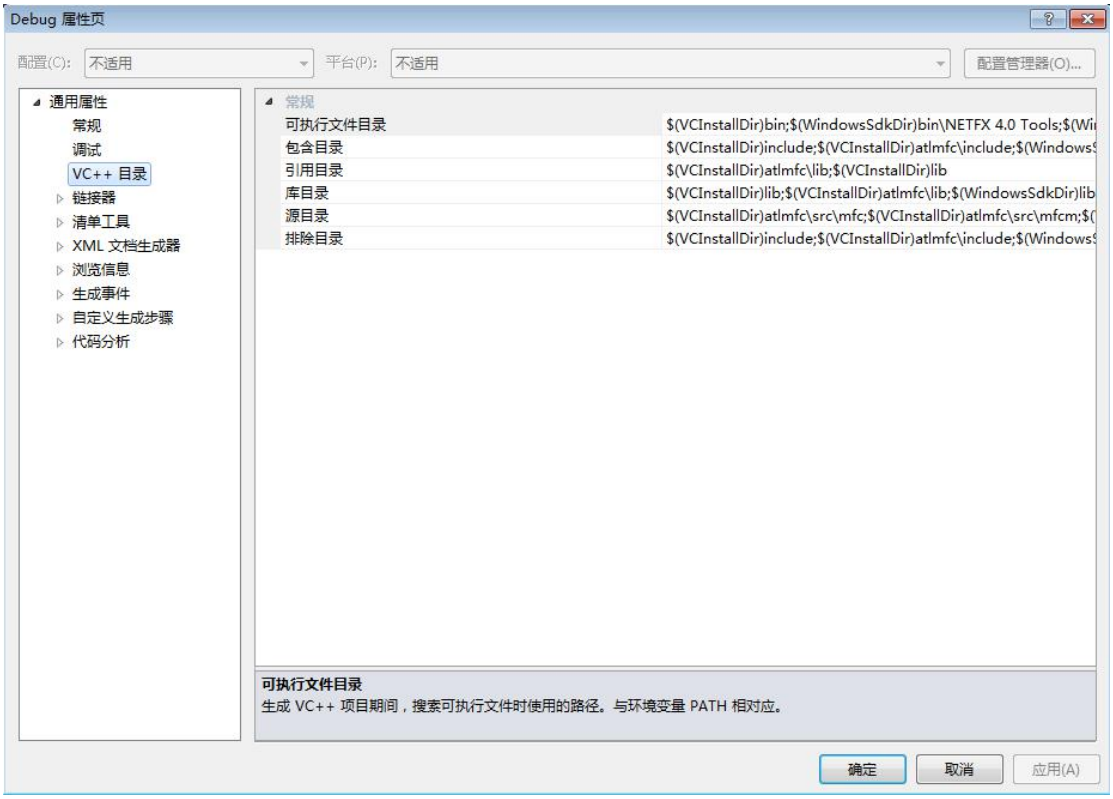
一. Winpcap 环境搭建

本实验采用 visual studio 10.0
（仅供参考）

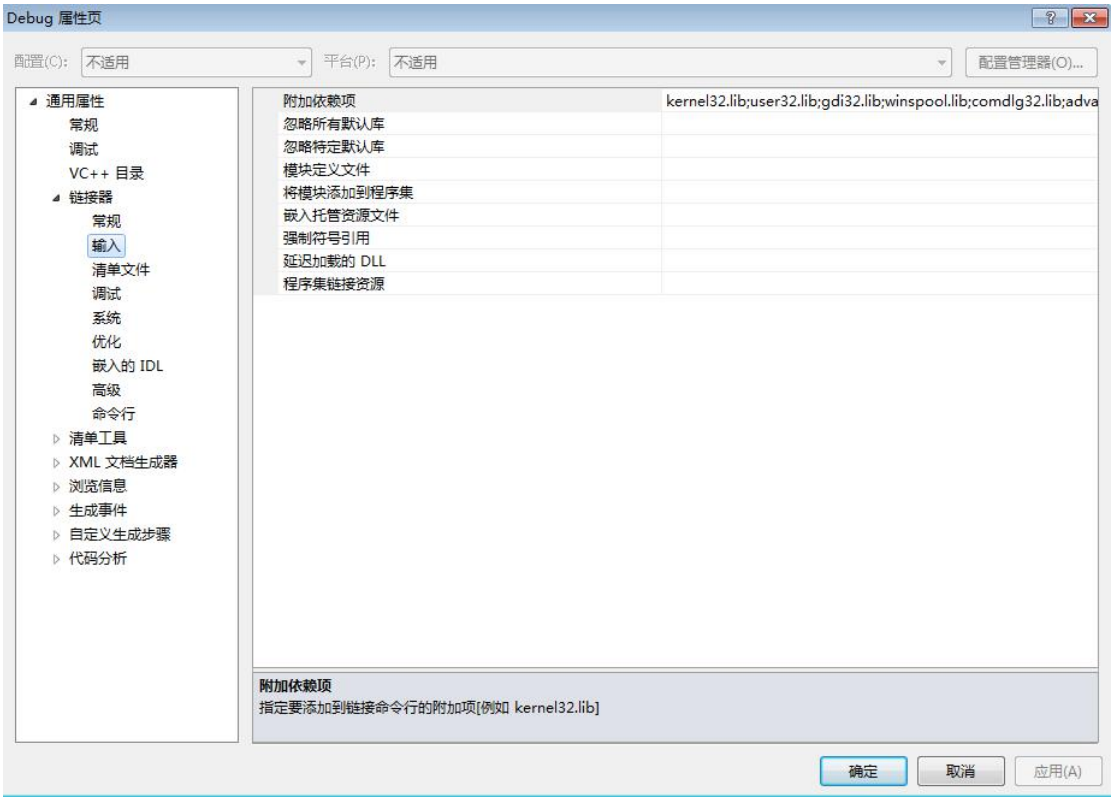
1.View->Property Manager
Debug|Win32 -> Mircrosoft.Cpp.Win32.user（右键）->Properties
2.设置环境目录：VC++ Directiories -> Include Directories 和 Library Directories 中添加路径。
假如将 wpdpack 放到 c 盘。则：
Include Directories:c:\wpdpack\Include;
Library Directories:c:\wpdpack\Lib;
3.Linker（连接器）下的 Command Line（命令行）
Additional Options （附加项）中输入：
wpcap.lib ws2_32.lib（注：用空格分隔。）

上述方法可以只设置一次

（1） 首先建立一个工程 networkstack;
文件- 新建 - 项目 - ， 选中 “WIN 32 控制台应用程序”，输入项目名称(networkstack)和指定存储目录（D:\）-在应用程序设置对话框中
应用程序类型选择：WIN 32 控制台应用程序
附加选项：空项目；
添加公共头文件以用于： 不选；
（2）在解决方案标签，增加或产生新文件；
（3） 对环境进行配置
1） 安装 WINPCAP.EXE;
2） 将 WpdPack_4_1_2.zip 解压，拷贝到 d:\根目录；
3）在属性管理器标签，点击刚创建的工程文件名，双击“debug|win32”,在弹出的“debug”属性对话框中，双击“VC++目录”，在包含目录，加入 WpdPack 拷贝的 INCLUDE 目录路径；在库目录中，加入 WpdPack 拷贝的 LIBARY 目录路径；



在“debug”属性对话框中，双击“链接器”-“输入”属性，



在上图的“附件依赖项”中，加入“wpcap.lib” 和“ws2_32.lib”两个库文件即可。

VC++6.0 环境配置

（1）下载 Winpcap 安装包；

地址 http://www.winpcap.org/install/default.htm。

（2）http://www.winpcap.org/devel.htm 下 载 WinPcap developer's pack 包解压，里面有配置好的例子和 include library，以及 docs，include，lib，example 等文件夹；

（3）在 VC 中设置 include 和 library 目录，具体方法是：

打开 VC，点击“TOOLS-option-directories”，在 include files 中 添加…\wpdpack\include 目录（步骤 2 中得到的目录），在 library fillies 中添加 …\wpdpack\lib 目录；

（4）点击“ project-settings-link ”，在 object\library modules 中添加 wpcap.lib 和 *Packet.lib* ws2_32.lib

二．数据帧发送

1.设计思路：
现有一个字节数组 2048 字节长，因为数据帧最小 64，最大 1518，作为数据帧的载体，
然后依次加上以太网数据头，要发送的数据，计算循环冗余码（采用查表法），这样符合数据帧
最后是发送数据帧，打开适配器，选择混杂模式，发送封包。

2.代码实现：

```
#include<stdio.h>
#include<stdlib.h>

#define HAVE_REMOTE
#include<pcap.h>

#pragma warning(disable:4996)
#define ETHERNET_IP 0x0800
#define MAX_SIZE 2048

int size_of_packet = 0;
u_int32_t crc32_table[256];

//ethernet header
struct ethernet_header
{
    u_int8_t dest_mac[6];
    u_int8_t src_mac[6];
    u_int16_t ethernet_type;
};

//generate table
void generate_crc32_table()
{
    int i, j;
    u_int32_t crc;
    for (i = 0; i < 256; i++)
    {
```

```
        crc = i;
        for (j = 0; j < 8; j++)
        {
            if (crc & 1)
                crc = (crc >> 1) ^ 0xEDB88320;
            else
                crc >>= 1;
        }
        crc32_table[i] = crc;
    }
}

u_int32_t calculate_crc(u_int8_t *buffer, int len)
{
    int i, j;
    u_int32_t crc;
    crc = 0xffffffff;
    for (i = 0; i < len; i++)
    {
        crc = (crc >> 8) ^ crc32_table[(crc & 0xFF) ^ buffer[i]];
    }
    crc ^= 0xffffffff;
    return crc;
}


void load_ethernet_header(u_int8_t *buffer)
{
    struct ethernet_header *hdr = (struct ethernet_header*)buffer;
    int i = 0;
    for (i = 0; i < 6; i++)
    {
        hdr->dest_mac[i] = 0x11;//this is where you can define the mac address
    }
    for (i = 0; i < 6; i++)
    {
        hdr->src_mac[i] = 0x22;//source mac address
    }
    hdr->ethernet_type = ETHERNET_IP;
    size_of_packet += sizeof(ethernet_header);
}


int load_ethernet_data(u_int8_t *buffer, FILE *fp)
```

```
{
    int size_of_data = 0;
    char tmp[MAX_SIZE], ch;
    while ((ch = fgetc(fp)) != EOF)
    {
        tmp[size_of_data] = ch;
        size_of_data++;
    }
    if (size_of_data < 46 || size_of_data>1500)
    {
        printf("Size of data is not satisfied with condition!!!\n");
        return -1;
    }

    u_int32_t crc = calculate_crc((u_int8_t*)tmp, size_of_data);
    //printf("%d\n", crc);
    int i;
    for (i = 0; i < size_of_data; i++)
    {
        *(buffer + i) = tmp[i];
    }
    *(u_int32_t*)(buffer + i) = crc;
    size_of_packet += size_of_data + 4;
    return 1;
}



int main()
{
    u_int8_t buffer[MAX_SIZE];//as a carrier of packet
    generate_crc32_table();

    //generate a packet
    size_of_packet = 0;
    FILE *fp = fopen("data.txt", "r");
    if (load_ethernet_data(buffer + sizeof(ethernet_header), fp) == -1)
    {
        return -1;
    }
    load_ethernet_header(buffer);

    //send the packet
    pcap_t *handle;
```

```
    char *device;
    char error_buffer[PCAP_ERRBUF_SIZE];
    device = pcap_lookupdev(error_buffer);
    if (device == NULL)
    {
        printf("%s\n", error_buffer);
        return -1;
    }
    handle = pcap_open_live(device, size_of_packet, PCAP_OPENFLAG_PROMISCUOUS, 1,
error_buffer);
    if (handle == NULL)
    {
        printf("Open adapter is failed..\n");
        return -1;
    }
    pcap_sendpacket(handle, (const u_char*)buffer, size_of_packet);;
    //printf("%d", *(int*)(buffer + size_of_packet - 4));
    pcap_close(handle);
    return 0;

}
```

三．数据接收

```
#include<stdio.h>
#include<stdlib.h>


#define HAVE_REMOTE
#include<pcap.h>
#include<WinSock2.h>

#pragma warning(disable:4996)

void ethernet_protocol_packet_callback(u_char *argument, const struct pcap_pkthdr
*packet_header, const u_char *packet_content);

//ethernet protocol header format
struct ethernet_header
{
    u_int8_t ether_dhost[6];//destination mac
    u_int8_t ether_shost[6];//src mac
    u_int16_t ether_type;
```

```
};

u_int8_t accept_dest_mac[2][6] = { { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 }, { 0x33,
0x33, 0x33, 0x33, 0x33, 0x33 } };
u_int32_t crc32_table[256];
//generate table
void generate_crc32_table()
{
    int i, j;
    u_int32_t crc;
    for (i = 0; i < 256; i++)
    {
        crc = i;
        for (j = 0; j < 8; j++)
        {
            if (crc & 1)
                crc = (crc >> 1) ^ 0xEDB88320;
            else
                crc >>= 1;
        }
        crc32_table[i] = crc;
    }
}

u_int32_t calculate_crc(u_int8_t *buffer, int len)
{
    int i, j;
    u_int32_t crc;
    crc = 0xffffffff;
    for (i = 0; i < len; i++)
    {
        crc = (crc >> 8) ^ crc32_table[(crc & 0xFF) ^ buffer[i]];
    }
    crc ^= 0xffffffff;
    return crc;
}


//ethernet protocol analysis
void ethernet_protocol_packet_callback(u_char *argument, const struct pcap_pkthdr
*packet_header, const u_char *packet_content)
{
    u_short ethernet_type;
    struct ethernet_header *ethernet_protocol;
```

```
    u_char *mac_string;
    static int packet_number = 1;
    ethernet_protocol = (struct ethernet_header*)packet_content;
    int len = packet_header->len;
    int i, j;

    ////check the mac address
    ////if the packet is sended to my pc or broadcast
    int flag = 2;
    for (i = 0; i < 2; i++)
    {
        flag = 2;
        for (j = 0; j < 6; j++)
        {
            if (ethernet_protocol->ether_dhost[j] == accept_dest_mac[i][j])
                continue;
            else
            {
                flag = i;
                break;
            }
        }
        if (flag != 2)continue;
        else
            break;
    }
    if (flag != 2)
    {
        return;
    }
    if (i == 0)
    {
        printf("It's broadcasted.\n");
    }
    // if the source is acceptable
    u_int8_t accept_source_mac[2][6] = { { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 },
{ 0x22, 0x22, 0x22, 0x22, 0x22, 0x22 } };
    for (i = 0; i < 2; i++)
    {
        flag = 1;
        for (j = 0; j < 6; j++)
        {
            if (ethernet_protocol->ether_shost[j] == accept_source_mac[i][j])
                continue;
```

```
            else
            {
                flag = 0;
                break;
            }
        }
        if (flag)
            break;
    }
    if (flag == 0)return;

    //see if the data is changed or not
    u_int32_t crc = calculate_crc((u_int8_t*)(packet_content +
sizeof(ethernet_header)), len - 4 - sizeof(ethernet_header));
    if (crc != *((u_int32_t*)(packet_content + len - 4)))
    {
        printf("The data has been changed.\n");
        return;
    }

    printf("----------------------------\n");
    printf("capture %d packet\n", packet_number);
    printf("capture time: %d\n", packet_header->ts.tv_sec);
    printf("packet length: %d\n", packet_header->len);
    printf("-----Ethernet protocol-------\n");

    ethernet_type = ethernet_protocol->ether_type;
    printf("Ethernet type: %04x\n", ethernet_type);
    switch (ethernet_type)
    {
    case 0x0800:printf("Upper layer protocol: IPV4\n"); break;
    case 0x0806:printf("Upper layer protocol: ARP\n"); break;
    case 0x8035:printf("Upper layer protocol: RARP\n"); break;
    case 0x814c:printf("Upper layer protocol: SNMP\n"); break;
    case 0x8137:printf("Upper layer protocol: IPX\n"); break;
    case 0x86dd:printf("Upper layer protocol: IPV6\n"); break;
    case 0x880b:printf("Upper layer protocol: PPP\n"); break;
    default:
        break;
    }

    mac_string = ethernet_protocol->ether_shost;
    printf("MAC source address: %02x:%02x:%02x:%02x:%02x:%02x\n", *mac_string,
*(mac_string + 1), *(mac_string + 2), *(mac_string + 3),
```

```
        *(mac_string + 4), *(mac_string + 5));
    mac_string = ethernet_protocol->ether_dhost;
    printf("MAC destination address: %02x:%02x:%02x:%02x:%02x:%02x\n",
*mac_string, *(mac_string + 1), *(mac_string + 2),
        *(mac_string + 3), *(mac_string + 4), *(mac_string + 5));

    /*if (ethernet_type == 0x0800)
    {
        ip_protocol_packet_callback(argument, packet_header, packet_content +
sizeof(ethernet_header));
    }
*/
    //show the data;
    for (u_int8_t *p = (u_int8_t*)(packet_content + sizeof(ethernet_header)); p !=
(u_int8_t*)(packet_content + packet_header->len - 4); p++)
    {
        printf("%c", *p);
    }
    printf("\n");

    printf("----------------------\n");
    packet_number++;
}

int main()
{
    generate_crc32_table();

    pcap_if_t *all_adapters;
    pcap_if_t *adapter;
    pcap_t *adapter_handle;
    char error_buffer[PCAP_ERRBUF_SIZE];

    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &all_adapters, error_buffer)
== -1)
    {
        fprintf(stderr, "Error in findalldevs_ex function: %s\n", error_buffer);
        return -1;
    }
    if (all_adapters == NULL)
    {
        printf("\nNo adapters found! Make sure WinPcap is installed!!!\n");
        return 0;
    }
```

```
    int id = 1;
    for (adapter = all_adapters; adapter != NULL; adapter = adapter->next)
    {
        printf("\n%d.%s\n", id++, adapter->name);
        printf("--- %s\n", adapter->description);
    }
    printf("\n");

    int adapter_id;
    printf("Enter the adapter id between 1 and %d: ", id - 1);
    scanf("%d", &adapter_id);
    if (adapter_id<1 || adapter_id>id - 1)
    {
        printf("\n Adapter id out of range.\n");
        pcap_freealldevs(all_adapters);
        return -1;
    }

    adapter = all_adapters;
    for (id = 1; id < adapter_id; id++)
    {
        adapter = adapter->next;
    }
    adapter_handle = pcap_open(adapter->name, 65535, PCAP_OPENFLAG_PROMISCUOUS, 5,
NULL, error_buffer);
    if (adapter_handle == NULL)
    {
        fprintf(stderr, "\n Unable to open adapter: %s\n", adapter->name);
        pcap_freealldevs(all_adapters);
        return -1;
    }

    pcap_loop(adapter_handle, NULL, ethernet_protocol_packet_callback, NULL);
    pcap_close(adapter_handle);
    pcap_freealldevs(all_adapters);
    return 0;
}
```

这个没有考虑大小端的问题。