**数据通信作业-5**

姓名： 费扬 学号： 519021910917 日期： 2022.04.27

1. 实验名称及内容

Lab2：

Build a 2-hop Point-to-Point network as illustrated below,

192.168.10.0 192.168.50.0

n0 ------------------------ n1-------------------------------- n2

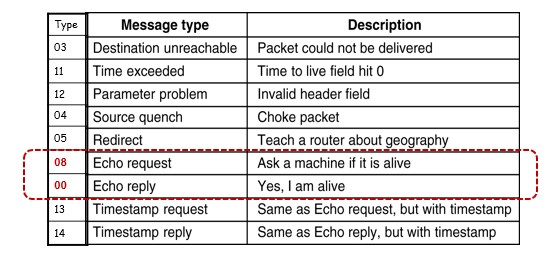
point-to-point point-to-point

5Mbps, 2ms 1Mbps, 2ms

* n0 sends a total number of 2000 bytes to n2.
* Use onoff-application (TCP) on n0 with packet size 512, data rate 50kb/s, set the OnTime random variable to 1 and OffTime random variable to 0.
* Use PacketSink (TCP) application on n2 to receive the packets.
* Enable NS\_LOG on both onoff-application and PacketSink, turn on pacp tracing on all nodes.
* Use filename: lab2.cc

Hints: For onoff-application and packet-sink application, the source codes are in /home/workspace/ns-allinone-3.28/ns-3.28/src/applications/model. Or go to <https://www.nsnam.org/doxygen/index.html>, click Modules→Applications→OnOffApplication to see a detailed description of the application.

1. 实验过程和结果



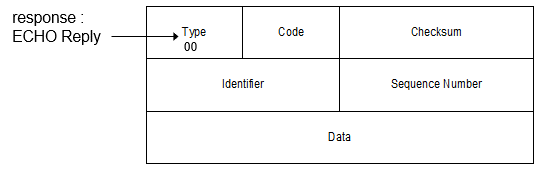
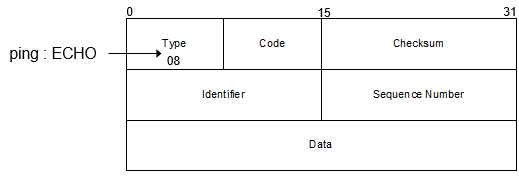
ICMP报文的类型

这里使用了08和00两种，即Echo request和Echo reply，确认连接的机器是处于活跃状态。



ICMP被认为是IP的一部分，但也是IP用户，ICMP的报文被封装在IP报文内部。

因此，本次通信任务的报文处理如下：



**基本操作步骤：**

Myping.cpp：

* 1. Create a raw socket：socktype=SOCK\_RAW, protocol=IPPROTO\_ICMP;
  2. Construct an ICMP message;
  3. Use “sendto” to send the ICMP message to the remote machine;
  4. Use “recvfrom” to receive any response.
  5. Wireshark

代码见打包文件中myping.cpp，这里只介绍思路：

**首先定义icmp\_hdr的结构体，包括变量如下：**

typedef struct icmp\_hdr {

unsigned char icmp\_type;

unsigned char icmp\_code;

unsigned short icmp\_checksum;

unsigned short icmp\_id;

unsigned short icmp\_sequence;

} ICMP\_HDR;

**然后在Main函数中：**

**WSAStartup并创建socket；**

s = socket(remote->ai\_family, SOCK\_RAW, IPPROTO\_ICMP);

if (s == INVALID\_SOCKET) {

cout << "socket() failed with " << WSAGetLastError() << endl;

freeaddrinfo(remote);

freeaddrinfo(local);

WSACleanup();

return -1;

}

**进行setsockopt、分配空间的初始化，并对参数如下定义；**

icmp\_hdr = (ICMP\_HDR\*)icmpbuf;

icmp\_hdr->icmp\_type = 8;

icmp\_hdr->icmp\_code = 0;

icmp\_hdr->icmp\_id = (unsigned short)GetCurrentProcessId();

icmp\_hdr->icmp\_sequence = 0;

icmp\_hdr->icmp\_checksum = 0;

datapart = icmpbuf + sizeof(ICMP\_HDR);

memset(datapart, 'Q', DEFAULT\_SIZE);

**进行Bind；**

iResult = bind(s, local->ai\_addr, (int)local->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

cout << "bind failed with " << WSAGetLastError() << endl;

freeaddrinfo(remote);

freeaddrinfo(local);

closesocket(s);

free(icmpbuf);

WSACleanup();

return -1;

}

**Receive并打印输出；**

RecvFrom(s, recvbuf, recvbuflen, (SOCKADDR\*)&from, &fromlen, &recvol);

cout << "Pinging: " << destHost;

PrintAddress(remote->ai\_addr, remote->ai\_addrlen);

cout << " with " << DEFAULT\_SIZE << " bytes of data." << endl;

**(☆)循环四次传输，使用sendto和WaitForSingleObject进行收发；**

* **首先调用函数确定ICMP格式，并进行校验和计算：**

SetIcmpSequence(icmpbuf);

ComputeIcmpchecksum(icmpbuf, packetlen);

* **Sendto和SingleObject捕获：**

iResult = sendto(s, icmpbuf, packetlen, 0, remote->ai\_addr, (int)remote->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

cout << "sendto failed with %" << WSAGetLastError() << endl;

freeaddrinfo(remote);

freeaddrinfo(local);

closesocket(s);

free(icmpbuf);

WSACloseEvent(recvol.hEvent);

WSACleanup();

return -1;

}

iResult = WaitForSingleObject((HANDLE)recvol.hEvent, DEFAULT\_RECV\_TIMEOUT);

if (iResult == WAIT\_FAILED) {

cout << "WaitForSigleObject failed with " << WSAGetLastError() << endl;

freeaddrinfo(remote);

freeaddrinfo(local);

closesocket(s);

free(icmpbuf);

WSACloseEvent(recvol.hEvent);

WSACleanup();

return -1;

}

else if (iResult == WAIT\_TIMEOUT) {

cout << "Request Time Out." << endl;

}

else {

time = (ULONG)GetTickCount64() - time;

WSAResetEvent(recvol.hEvent);

RecvPack += 1;

cout << "Reply From";

PrintAddress((SOCKADDR\*)&from, fromlen);

if (time == 0) {

printf(": bytes = %d time < 1 ms TTL = %d\n", DEFAULT\_SIZE, TTL);

}

else {

printf(": bytes = %d time = %d ms TTL = %d\n", DEFAULT\_SIZE, time, TTL);

}

if (i < 3) {

fromlen = sizeof(SOCKADDR\_STORAGE);

RecvFrom(s, recvbuf, recvbuflen, (SOCKADDR\*)&from, &fromlen, &recvol);

}

}

Sleep(1000);

* **操作输出**
* **Checksum函数定义如下：**

USHORT checksum(USHORT\* buffer, int size) {

unsigned long cksum = 0;

while (size > 1) {

cksum += \*buffer++;

size -= sizeof(USHORT);

}

if (size) {

cksum += \*(UCHAR\*)buffer;

}

cksum = (cksum >> 16) + (cksum & 0xffff);

cksum += (cksum >> 16);

return (USHORT)(~cksum);

}

* **ComputeIcmpchecksum函数定义如下：**

void ComputeIcmpchecksum(char\* buf, int packetlen) {

ICMP\_HDR\* icmpv4 = NULL;

icmpv4 = (ICMP\_HDR\*)buf;

icmpv4->icmp\_checksum = 0;

icmpv4->icmp\_checksum = checksum((USHORT\*)buf, packetlen);

}

* **SetIcmpSequence函数定义如下：**

void SetIcmpSequence(char\* buf) {

ULONG sequence = 0;

sequence = (ULONG)GetTickCount64();

ICMP\_HDR\* icmpv4 = NULL;

icmpv4 = (ICMP\_HDR\*)buf;

icmpv4->icmp\_sequence = (USHORT)sequence;

}

* **RecvFrom函数定义如下：**

int RecvFrom(SOCKET s, char\* buf, int buflen, SOCKADDR\* from, int\* fromlen, WSAOVERLAPPED\* ol) {

WSABUF wbuf;

DWORD flags, bytes;

int iResult;

wbuf.buf = buf;

wbuf.len = buflen;

flags = NULL;

iResult = WSARecvFrom(s, &wbuf, 1, &bytes, &flags, from, fromlen, ol, NULL);

if (iResult == SOCKET\_ERROR) {

if (WSAGetLastError() != WSA\_IO\_PENDING) {

printf("WSARecvfrom failed: %d\n", WSAGetLastError());

return SOCKET\_ERROR;

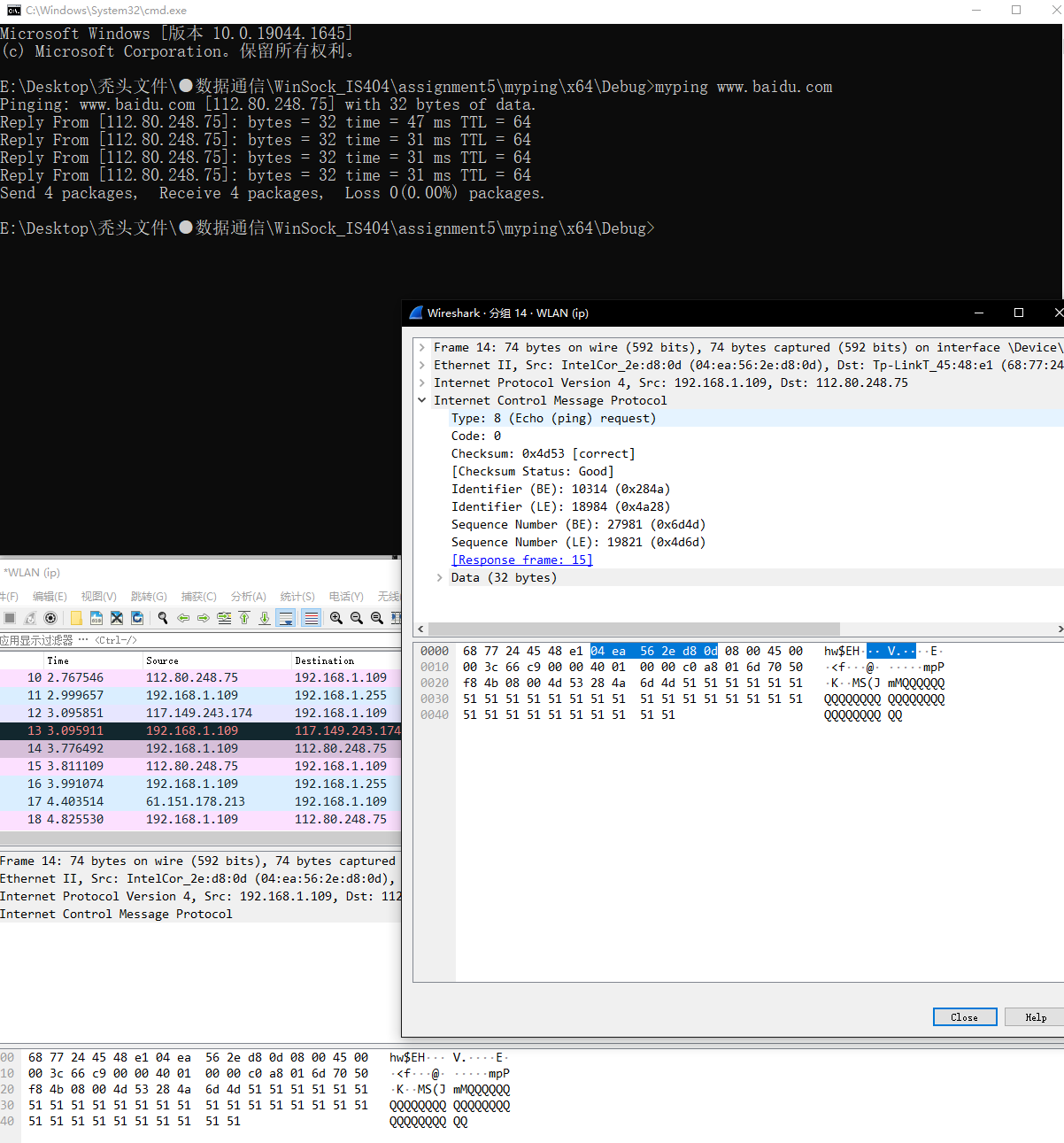
}

}

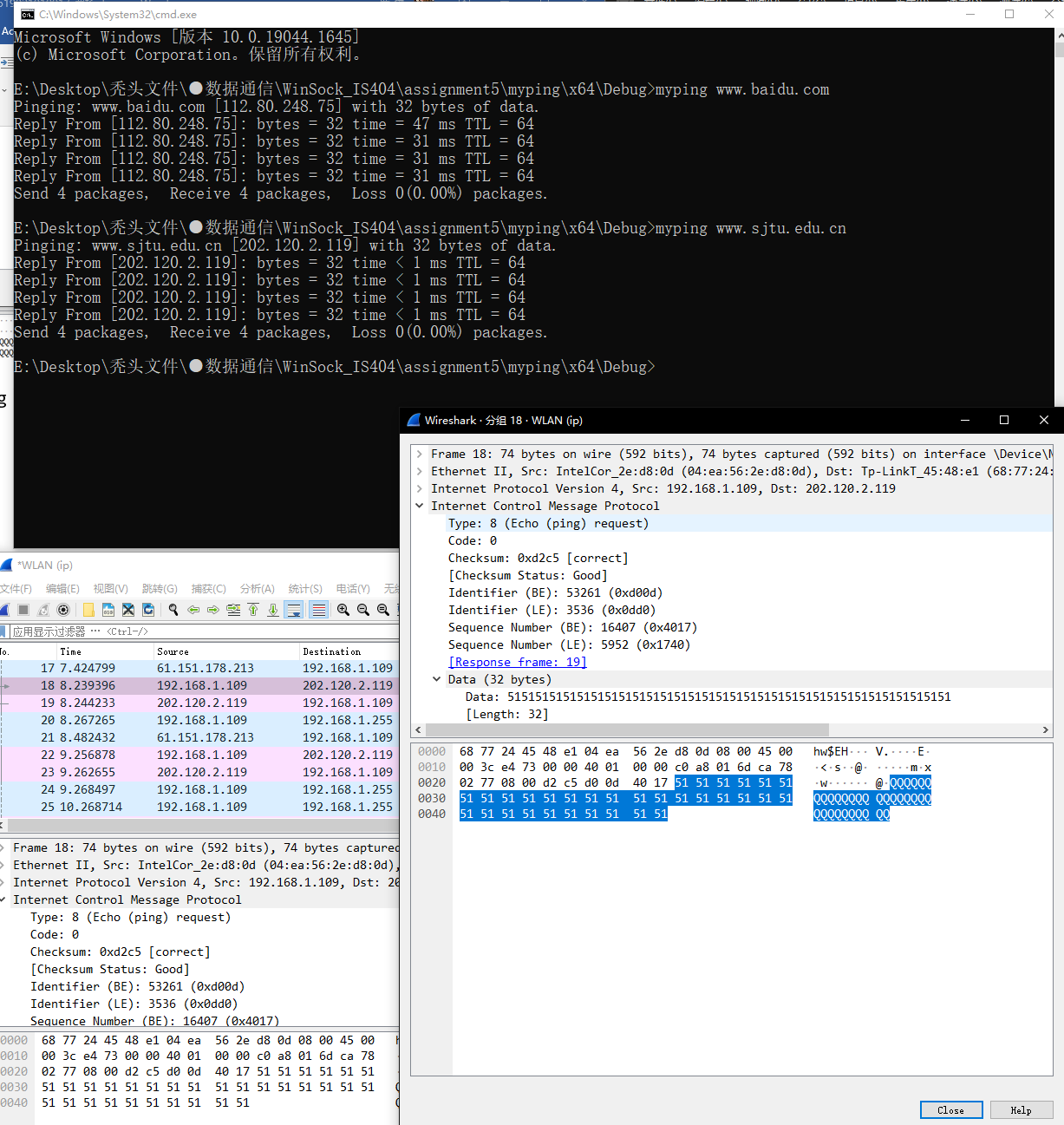
return NO\_ERROR;

}

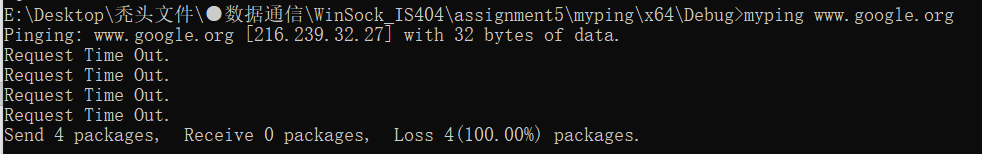
**之后在命令行操作输出，同时打开wireshark进行监测：**



在cmd中实现ping百度，wireshark抓包icmp



在cmd中实现ping交大官网，wireshark抓包icmp



Ping谷歌会被墙，丢包

三、问题与思考

1. 对比IP和ICMP传输：

ICMP 的全称是 Internet Control Message Protocol(互联网控制协议)，它是一种互联网套件，它用于IP 协议中发送控制消息。也就是说，ICMP 是依靠 IP 协议来完成信息发送的，它是 IP 的主要部分，但是从体系结构上来讲，它位于 IP 之上，因为 ICMP 报文是承载在 IP 分组中的，就和 TCP 与 UDP 报文段作为 IP 有效载荷被承载那样。这也就是说，当主机收到一个指明上层协议为 ICMP 的 IP 数据报时，它会分解出该数据报的内容给 ICMP，就像分解数据报的内容给 TCP 和 UDP 一样。

ICMP 协议和 TCP、UDP 等协议不同，它不用于传输数据，只是用来发送消息。因为 IP 协议现在有两类版本：IPv4 和 IPv6 ，所以 ICMP 也有两个版本：ICMPv4 和 ICMPv6。

2. 几种错误：

11004：网路连接错误，可能是网址不对；

Timeout：比如访问外网时ping不通，需要挂VPN；

Time<1ms：访问非常快，TTL正常则无误，有可能是访问了局域网，比如访问127.0.0.1就是time<1ms；

**Ref：**

Getting started with Winsock

<https://msdn.microsoft.com/en-us/library/ms738545(v=vs.85).aspx>

Winsock reference

<https://msdn.microsoft.com/en-us/library/ms741416(v=vs.85).aspx>