/\*\*\*\*\* Assignment File of wireshark \*\*\*\*\*/

Part A-Ethernet

#include <iostream>

#include <pcap.h>

#include <net/ethernet.h>

#include <netinet/ip.h>

#include <netinet/in.h>

#include <netinet/tcp.h>

#include <arpa/inet.h>

using namespace std;

void packetHandler(u\_char \*userData, const struct pcap\_pkthdr\* pkthdr, const u\_char\* packet);

int main() {

pcap\_t \*descr;

char errbuf[PCAP\_ERRBUF\_SIZE];

// open capture file for offline processing

descr = pcap\_open\_offline("http.cap", errbuf);

if (descr == NULL) {

cout << "pcap\_open\_live() failed: " << errbuf << endl;

return 1;

}

cout<<"Source Ip\t\t"<<"Source Port\t\t"<<"Destination Ip\t\t"<<"Destination port\t\n";

// start packet processing loop, just like live capture

if (pcap\_loop(descr, 0, packetHandler, NULL) < 0) {

cout << "pcap\_loop() failed: " << pcap\_geterr(descr);

return 1;

}

cout << "capture finished" << endl;

return 0;

}

void packetHandler(u\_char \*userData, const struct pcap\_pkthdr\* pkthdr, const u\_char\* packet) {

const struct ether\_header\* ethernetHeader;

const struct ip\* ipHeader;

const struct tcphdr\* tcpHeader;

char sourceIp[INET\_ADDRSTRLEN];

char destIp[INET\_ADDRSTRLEN];

u\_int sourcePort, destPort;

u\_char \*data;

int dataLength = 0,ipflag=0;

string dataStr = "";

ethernetHeader = (struct ether\_header\*)packet;

if (ntohs(ethernetHeader->ether\_type) == ETHERTYPE\_IP) {

ipflag=1;

ipHeader = (struct ip\*)(packet + sizeof(struct ether\_header));

inet\_ntop(AF\_INET, &(ipHeader->ip\_src), sourceIp, INET\_ADDRSTRLEN);

inet\_ntop(AF\_INET, &(ipHeader->ip\_dst), destIp, INET\_ADDRSTRLEN);

if (ipHeader->ip\_p == IPPROTO\_TCP) {

tcpHeader = (tcphdr\*)(packet + sizeof(struct ether\_header) + sizeof(struct ip));

sourcePort = ntohs(tcpHeader->source);

destPort = ntohs(tcpHeader->dest);

data = (u\_char\*)(packet + sizeof(struct ether\_header) + sizeof(struct ip) + sizeof(struct tcphdr));

dataLength = pkthdr->len - (sizeof(struct ether\_header) + sizeof(struct ip) + sizeof(struct tcphdr));

// convert non-printable characters, other than carriage return, line feed,

// or tab into periods when displayed.

for (int i = 0; i < dataLength; i++) {

if ((data[i] >= 32 && data[i] <= 126) || data[i] == 10 || data[i] == 11 || data[i] == 13) {

dataStr += (char)data[i];

} else {

dataStr += ".";

}

}

// print the results

cout << sourceIp<< "\t\t"<< sourcePort << "\t\t" << destIp<<"\t\t" << destPort <<"\n";

if(ipflag==1)

{

cout<<"protocols : IP and TCP";

}

if (dataLength > 0) {

// cout << dataStr << endl;

}

}

}

}

Output:

/\*\*\*\*\* Assignment File of wireshark \*\*\*\*\*/

Part A-Ethernet

#include <iostream>

#include <pcap.h>

#include <net/ethernet.h>

#include <netinet/ip.h>

#include <netinet/in.h>

#include <netinet/tcp.h>

#include <arpa/inet.h>

using namespace std;

void packetHandler(u\_char \*userData, const struct pcap\_pkthdr\* pkthdr, const u\_char\* packet);

int main() {

pcap\_t \*descr;

char errbuf[PCAP\_ERRBUF\_SIZE];

// open capture file for offline processing

descr = pcap\_open\_offline("http.cap", errbuf);

if (descr == NULL) {

cout << "pcap\_open\_live() failed: " << errbuf << endl;

return 1;

}

cout<<"Source Ip\t\t"<<"Source Port\t\t"<<"Destination Ip\t\t"<<"Destination port\t\n";

// start packet processing loop, just like live capture

if (pcap\_loop(descr, 0, packetHandler, NULL) < 0) {

cout << "pcap\_loop() failed: " << pcap\_geterr(descr);

return 1;

}

cout << "capture finished" << endl;

return 0;

}

void packetHandler(u\_char \*userData, const struct pcap\_pkthdr\* pkthdr, const u\_char\* packet) {

const struct ether\_header\* ethernetHeader;

const struct ip\* ipHeader;

const struct tcphdr\* tcpHeader;

char sourceIp[INET\_ADDRSTRLEN];

char destIp[INET\_ADDRSTRLEN];

u\_int sourcePort, destPort;

u\_char \*data;

int dataLength = 0,ipflag=0;

string dataStr = "";

ethernetHeader = (struct ether\_header\*)packet;

if (ntohs(ethernetHeader->ether\_type) == ETHERTYPE\_IP) {

ipflag=1;

ipHeader = (struct ip\*)(packet + sizeof(struct ether\_header));

inet\_ntop(AF\_INET, &(ipHeader->ip\_src), sourceIp, INET\_ADDRSTRLEN);

inet\_ntop(AF\_INET, &(ipHeader->ip\_dst), destIp, INET\_ADDRSTRLEN);

if (ipHeader->ip\_p == IPPROTO\_TCP) {

tcpHeader = (tcphdr\*)(packet + sizeof(struct ether\_header) + sizeof(struct ip));

sourcePort = ntohs(tcpHeader->source);

destPort = ntohs(tcpHeader->dest);

data = (u\_char\*)(packet + sizeof(struct ether\_header) + sizeof(struct ip) + sizeof(struct tcphdr));

dataLength = pkthdr->len - (sizeof(struct ether\_header) + sizeof(struct ip) + sizeof(struct tcphdr));

// convert non-printable characters, other than carriage return, line feed,

// or tab into periods when displayed.

for (int i = 0; i < dataLength; i++) {

if ((data[i] >= 32 && data[i] <= 126) || data[i] == 10 || data[i] == 11 || data[i] == 13) {

dataStr += (char)data[i];

} else {

dataStr += ".";

}

}

// print the results

cout << sourceIp<< "\t\t"<< sourcePort << "\t\t" << destIp<<"\t\t" << destPort <<"\n";

if(ipflag==1)

{

cout<<"protocols : IP and TCP";

}

if (dataLength > 0) {

// cout << dataStr << endl;

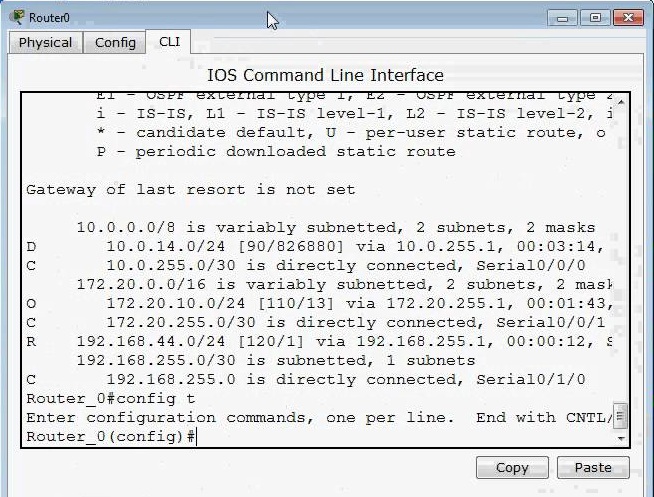
}

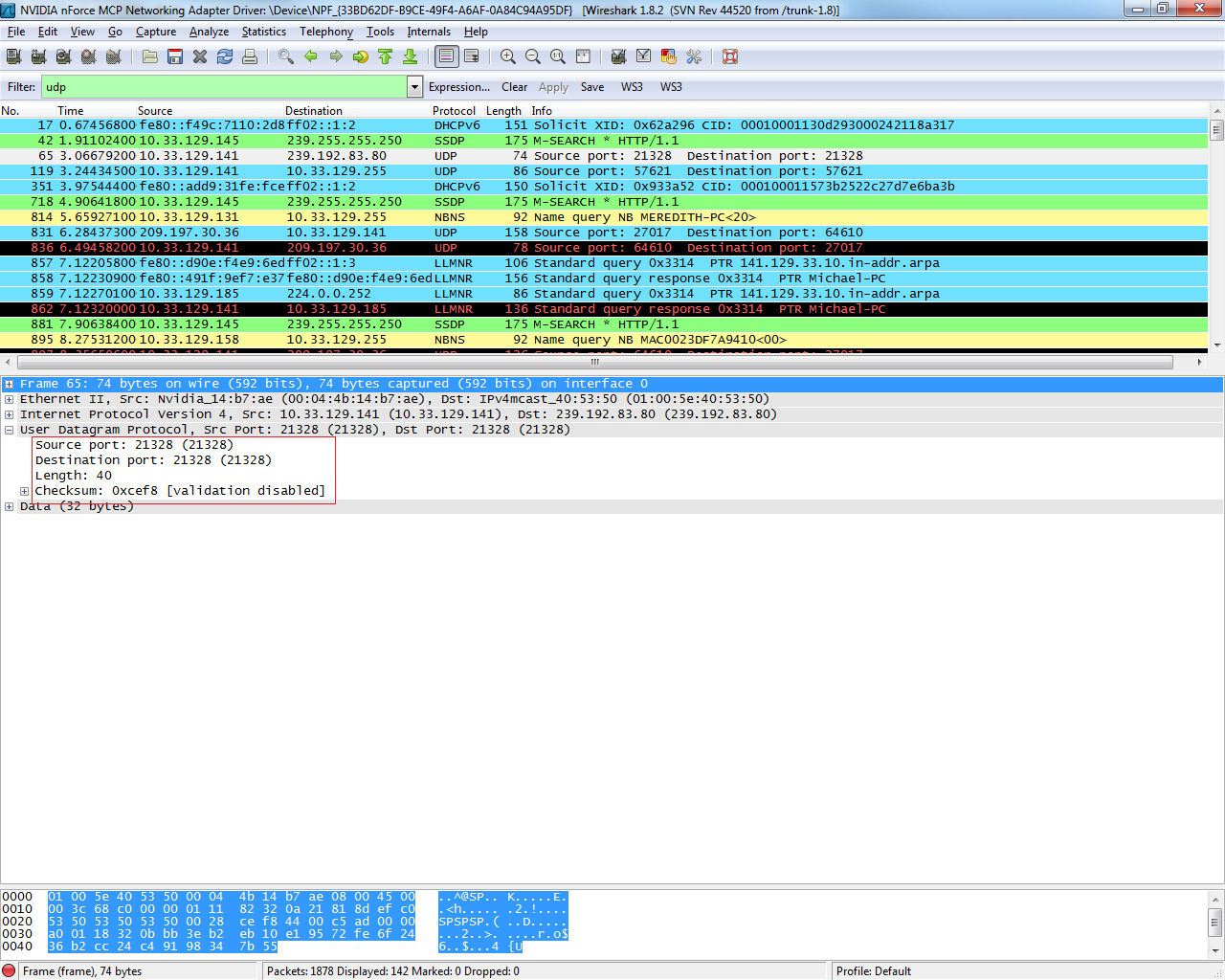
}

}

}

Output:



//Part B-UDP

PartC TCP

