Ministry of Education of Republic of Moldova
Technical University of Moldova
CIM Faculty
Anglophone Department

Report

on NETWORK PROGRAMMING

Laboratory Work Nr. 4

Performed by st. gr. FAF-131 **Boldescu A.**

Verified by lect.univ. **Railean A.**

Chișinău 2016

Objective

Write a Network Sniffer.

Generic requirements

- Learn how to manipulate sockets at a lower level, by setting certain options.
- Write a simple sniffer that can capture an IP packet, parse it and print it on the screen in a human readable form.

Grading policy

Assuming that everything is correct,

- 8 - for capturing a raw IP packet;
- 9 - for properly parsing the packet and displaying it;
- 10 - for extending the program with another feature that uses socket options, or describing in detail how that feature could be implemented.

Code in C#

I decided to try implementing a program in some other language (not Python) so here is my result (a console app for a network sniffer in c#)

IPHeader.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Net;

namespace NetworkSniffer
{
    class IPHeader:UDPHeader
    {
        //IP Header fields
```

```
        private byte byVersionAndHeaderLength;   //Eight bits for version and header
length
        private byte byDifferentiatedServices;    //Eight bits for differentiated
services (TOS)
        private ushort usTotalLength;              //Sixteen bits for total length of
the datagram (header + message)
        private ushort usIdentification;           //Sixteen bits for identification
        private ushort usFlagsAndOffset;           //Eight bits for flags and
fragmentation offset
        private byte byTTL;                       //Eight bits for TTL (Time To Live)
        private byte byProtocol;                  //Eight bits for the underlying
protocol
        private short sChecksum;                   //Sixteen bits containing the
checksum of the header
        //(checksum can be negative so taken as short)
        private uint uiSourceIPAddress;           //Thirty two bit source IP Address
        private uint uiDestinationIPAddress;      //Thirty two bit destination IP
Address
        //End IP Header fields

        private byte byHeaderLength;              //Header length
        private byte[] byIPData = new byte[4096];  //Data carried by the datagram

        public enum Protocol
        {
            TCP = 6,
            UDP = 17,
            Unknown = -1
        };
        public IPHeader() { }
        public IPHeader(byte[] byBuffer, int nReceived)
        {

            try
            {
                //Create MemoryStream out of the received bytes
                MemoryStream memoryStream = new MemoryStream(byBuffer, 0, nReceived);
                //Next we create a BinaryReader out of the MemoryStream
                BinaryReader binaryReader = new BinaryReader(memoryStream);

                //The first eight bits of the IP header contain the version and
                //header length so we read them
                byVersionAndHeaderLength = binaryReader.ReadByte();

                //The next eight bits contain the Differentiated services
                byDifferentiatedServices = binaryReader.ReadByte();

                //Next eight bits hold the total length of the datagram
                usTotalLength =
(ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next sixteen have the identification bytes
                usIdentification =
(ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next sixteen bits contain the flags and fragmentation offset
                usFlagsAndOffset =
(ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next eight bits have the TTL value
                byTTL = binaryReader.ReadByte();

                //Next eight represnts the protocol encapsulated in the datagram
                byProtocol = binaryReader.ReadByte();
```

```
                //Next sixteen bits contain the checksum of the header
                sChecksum = IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next thirty two bits have the source IP address
                uiSourceIPAddress = (uint)(binaryReader.ReadInt32());

                //Next thirty two hold the destination IP address
                uiDestinationIPAddress = (uint)(binaryReader.ReadInt32());

                //Now we calculate the header length

                byHeaderLength = byVersionAndHeaderLength;
                //The last four bits of the version and header length field contain the
                //header length, we perform some simple binary airthmatic operations to
                //extract them
                byHeaderLength <<= 4;
                byHeaderLength >>= 4;
                //Multiply by four to get the exact header length
                byHeaderLength *= 4;

                //Copy the data carried by the data gram into another array so that
                //according to the protocol being carried in the IP datagram
                Array.Copy(byBuffer,
                           byHeaderLength,  //start copying from the end of the header
                           byIPData, 0,
                           usTotalLength - byHeaderLength);
            }
            catch (Exception ex)
            {
               Console.WriteLine(ex.Message, "occured prease fix errors");
            }
        }

        public string Version
        {
            get
            {
                //Calculate the IP version

                //The four bits of the IP header contain the IP version
                if ((byVersionAndHeaderLength >> 4) == 4)
                {
                    return "IP v4";
                }
                else if ((byVersionAndHeaderLength >> 4) == 6)
                {
                    return "IP v6";
                }
                else
                {
                    return "Unknown";
                }
            }
        }

        public string HeaderLength
        {
            get
            {
                return byHeaderLength.ToString();
            }
        }

        public ushort MessageLength
        {
```

```
            get
            {
                //MessageLength = Total length of the datagram - Header length
                return (ushort)(usTotalLength - byHeaderLength);
            }
        }

        public string DifferentiatedServices
        {
            get
            {
                //Returns the differentiated services in hexadecimal format
                return string.Format("0x{0:x2} ({1})", byDifferentiatedServices,
                    byDifferentiatedServices);
            }
        }

        public string Flags
        {
            get
            {
                //The first three bits of the flags and fragmentation field
                //represent the flags (which indicate whether the data is
                //fragmented or not)
                int nFlags = usFlagsAndOffset >> 13;
                if (nFlags == 2)
                {
                    return "Don't fragment";
                }
                else if (nFlags == 1)
                {
                    return "More fragments to come";
                }
                else
                {
                    return nFlags.ToString();
                }
            }
        }

        public string FragmentationOffset
        {
            get
            {
                //The last thirteen bits of the flags and fragmentation field
                //contain the fragmentation offset
                int nOffset = usFlagsAndOffset << 3;
                nOffset >>= 3;

                return nOffset.ToString();
            }
        }

        public string TTL
        {
            get
            {
                return byTTL.ToString();
            }
        }
        public Protocol ProtocolType
        {
            get
            {
                //The protocol field represents the protocol in the data portion
```

```
                //of the datagram
                if (byProtocol == 6)        //A value of six represents the TCP
protocol
                {
                    return Protocol.TCP;
                }
                else if (byProtocol == 17)  //Seventeen for UDP
                {
                    return Protocol.UDP;
                }
                else
                {
                    return Protocol.Unknown;
                }
            }
        }

        public string IPChecksum
        {
            get
            {
                //Returns the checksum in hexadecimal format
                return string.Format("0x{0:x2}", sChecksum);
            }
        }

        public IPAddress SourceAddress
        {
            get
            {
                return new IPAddress(uiSourceIPAddress);
            }
        }

        public IPAddress DestinationAddress
        {
            get
            {
                return new IPAddress(uiDestinationIPAddress);
            }
        }

        public string TotalLength
        {
            get
            {
                return usTotalLength.ToString();
            }
        }

        public string Identification
        {
            get
            {
                return usIdentification.ToString();
            }
        }

        public byte[] IPData
        {
            get
            {
                return byIPData;
            }
        }
```

```
    public void ShowIP()
        {

            Console.WriteLine("An IP packet of size of {0} bytes was reached",
this.TotalLength);
            Console.WriteLine("Parsed data");
            Console.WriteLine("{");
            Console.WriteLine("Ver: " + this.Version);
            Console.WriteLine("Header Length: " + this.HeaderLength);
            Console.WriteLine("Differntiated Services: " + this.DifferentiatedServices);
            Console.WriteLine("Total Length: " + this.TotalLength);
            Console.WriteLine("Identification: " + this.Identification);
            Console.WriteLine("Flags: " + this.Flags);
            Console.WriteLine("Fragmentation Offset: " + this.FragmentationOffset);
            Console.WriteLine("Time to live: " + this.TTL);
            switch (this.ProtocolType)
            {
                case Protocol.TCP:
                    Console.WriteLine("Protocol: " + "TCP");
                    break;
                case Protocol.UDP:
                    Console.WriteLine("Protocol: " + "UDP");
                    break;
                case Protocol.Unknown:
                    Console.WriteLine("Protocol: " + "Unknown");
                    break;
            }
            Console.WriteLine("Checksum: " + this.IPChecksum);
            Console.WriteLine("Source: " + this.SourceAddress.ToString());
            Console.WriteLine("Destination: " + this.DestinationAddress.ToString());
            Console.WriteLine("}");
        }
    }
}
```

UDPheader.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Net;
namespace NetworkSniffer
{
    class UDPHeader
    {
        //UDP header fields
        private ushort usSourcePort;            //Sixteen bits for the source port
number
        private ushort usDestinationPort;       //Sixteen bits for the destination port
number
        private ushort usLength;                //Length of the UDP header
        private short sChecksum;                //Sixteen bits for the checksum
        //(checksum can be negative so taken as short)
        //End UDP header fields
```

```
        private byte[] byUDPData = new byte[4096];  //Data carried by the UDP packet
        public UDPHeader() { }
        public UDPHeader(byte[] byBuffer, int nReceived)
        {
            MemoryStream memoryStream = new MemoryStream(byBuffer, 0, nReceived);
            BinaryReader binaryReader = new BinaryReader(memoryStream);

            //The first sixteen bits contain the source port
            usSourcePort =
(ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

            //The next sixteen bits contain the destination port
            usDestinationPort =
(ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

            //The next sixteen bits contain the length of the UDP packet
            usLength = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

            //The next sixteen bits contain the checksum
            sChecksum = IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

            //Copy the data carried by the UDP packet into the data buffer
            Array.Copy(byBuffer,
                       8,               //The UDP header is of 8 bytes so we start
copying after it
                       byUDPData,
                       0,
                       nReceived - 8);
        }

        public string SourcePort
        {
            get
            {
                return usSourcePort.ToString();
            }
        }

        public string DestinationPort
        {
            get
            {
                return usDestinationPort.ToString();
            }
        }

        public string Length
        {
            get
            {
                return usLength.ToString();
            }
        }

        public string Checksum
        {
            get
            {
                //Return the checksum in hexadecimal format
                return string.Format("0x{0:x2}", sChecksum);
            }
        }

        public byte[] Data
```

```
        {
            get
            {
                return byUDPData;
            }
        }

        public void showUDP()
        {
           Console.WriteLine("UDP datagram");
           Console.WriteLine("{");
           Console.WriteLine("Source Port: " + this.SourcePort);
           Console.WriteLine("Destination Port: " +this.DestinationPort);
           Console.WriteLine("Length: " + this.Length);
           Console.WriteLine("Checksum: " + this.Checksum);
           Console.WriteLine("}");
        }
    }

}
```

Program.cs(main)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net.Sockets;
using System.Net;

namespace NetworkSniffer
{
    class Program : IPHeader
    {
        private Socket mainSocket;                          //The socket which captures
all incoming packets
        private byte[] byteData = new byte[4096];
        private bool bContinueCapturing = false;



        private void ParseData(byte[] byteData, int nReceived)
        {


            //Since all protocol packets are encapsulated in the IP datagram
            //so we start by parsing the IP header and see what protocol data
            //is being carried by it
            IPHeader ipHeader = new IPHeader(byteData, nReceived);


            //Now according to the protocol being carried by the IP datagram we parse
            //the data field of the datagram
            switch (ipHeader.ProtocolType)
            {
                case Protocol.TCP:

                    ipHeader.ShowIP();

                    break;

                case Protocol.UDP:
```

```
                    UDPHeader udpHeader = new UDPHeader(ipHeader.IPData,
//IPHeader.Data stores the data being
                        //carried by the IP datagram

(int)ipHeader.MessageLength);//Length of the data field
                    ipHeader.ShowIP();
                    udpHeader.showUDP();

                    break;

                case Protocol.Unknown:
                    break;
            }

        ;
        }

        private void OnReceive(IAsyncResult ar)
        {
            try
            {
                int nReceived = mainSocket.EndReceive(ar);

                //Analyze the bytes received...

                ParseData(byteData, nReceived);



                if (bContinueCapturing)
                    {
                        byteData = new byte[4096];

                        //Another call to BeginReceive so that we continue to receive
the incoming
                        //packets


                        mainSocket.BeginReceive(byteData, 0, byteData.Length,
SocketFlags.None,
                                new AsyncCallback(OnReceive), null);

                    }


            }
            catch (ObjectDisposedException)
            {
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message, "occured fix errors");
            }
        }

        static void Main(string[] args)
        {

            try
            {
                Program sniffer = new Program();
                if(sniffer.bContinueCapturing==false)
                {
                    //Start capturing the packets...
```

```
                    sniffer.bContinueCapturing = true;

                    //For sniffing the socket to capture the packets has to be a raw
socket, with the
                    //address family being of type internetwork, and protocol being IP
                    sniffer.mainSocket = new Socket(AddressFamily.InterNetwork,
                        SocketType.Raw, System.Net.Sockets.ProtocolType.IP);
                    IPAddress[] ip = Dns.GetHostAddresses("127.0.0.100");
                    //Bind the socket to the selected IP address
                    sniffer.mainSocket.Bind(new IPEndPoint(ip[0], 0));

                    //Set the socket  options
                    sniffer.mainSocket.SetSocketOption(SocketOptionLevel.IP,
//Applies only to IP packets
                                                       SocketOptionName.HeaderIncluded, //Set
the include the header
                                                       true);
//option to true

                    byte[] byTrue = new byte[4] { 1, 0, 0, 0 };
                    byte[] byOut = new byte[4] { 1, 0, 0, 0 }; //Capture outgoing
packets

                    //Socket.IOControl is analogous to the WSAIoctl method of Winsock 2
                    sniffer.mainSocket.IOControl(IOControlCode.ReceiveAll,
//Equivalent to SIO_RCVALL constant
                         //of Winsock 2
                                         byTrue,
                                         byOut);


                    //Start receiving the packets asynchronously
                    sniffer.mainSocket.BeginReceive(sniffer.byteData, 0,
sniffer.byteData.Length, SocketFlags.None,
                         new AsyncCallback(sniffer.OnReceive), null);


                }


            }

            catch (Exception ex)
            {
                Console.WriteLine(ex.Message, "occured fix errors");
            }
            Console.ReadKey();

        }
    }
}
```

Screenshots of the result

The main features of the code were comented inside the listing so here is an example of received output.

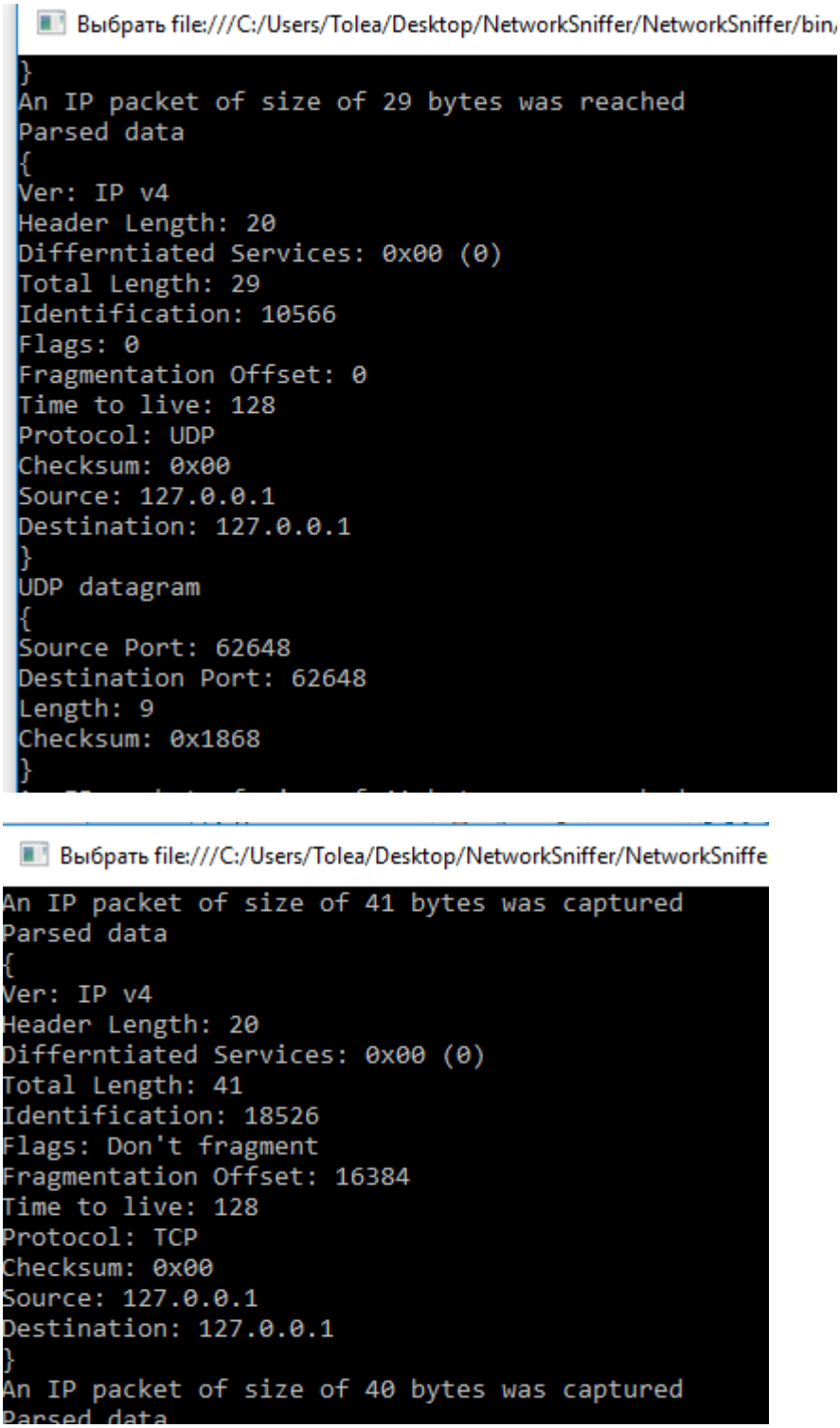

Figure 1-2 Screenshots of the results for UDP datagram and TCP segment

Conclusion

In this laboratory work I studied the basics of implementing Network Sniffer program using C# and also I learned how to parse an IP and UDP headers using C# features.