МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И

ТЕЛЕКОММУНИКАЦИЙ

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Отчёт**

**по лабораторной работе №4**

**по дисциплине**

**«Сетевые технологии»**

Выполнил: студент гр. пм-51 Бакулин А.В. \_\_\_\_\_\_\_\_\_\_\_

Проверил: преподаватель Прозорова Т.Г. \_\_\_\_\_\_\_\_\_\_\_

Киров 2014

**Задание 1.**

Разработать приложение, подсчитывающее входящий и исходящий сетевой трафик без учѐта его природы.

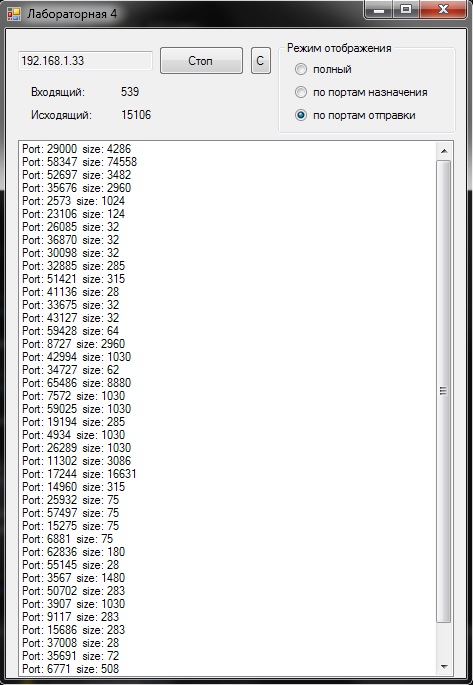
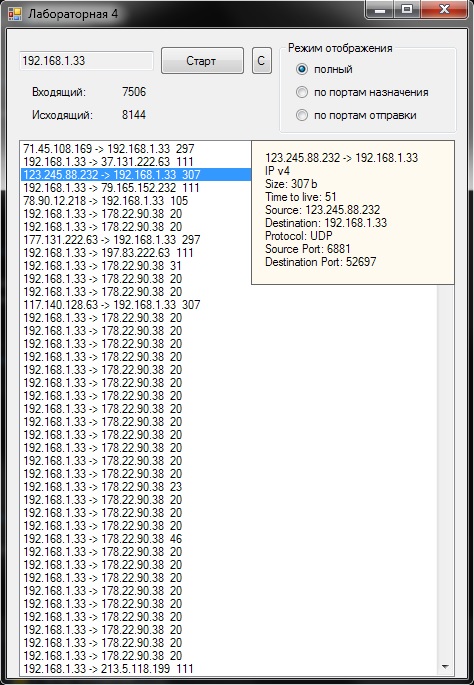
**Задание 2.**

Разработать приложение, подсчитывающее входящий и исходящий сетевой трафик с учѐтом приложения (порта), для которого данный трафик предназначен. Приложение должно формировать отчѐт с разделением трафика по портам назначения/отправки.

**Задание 3.**

Разработать приложение, анализирующее весь сетевой трафик подсети. Приложение должно перехватывать не только трафик, адресованный данному узлу, но и анализировать пакеты, адресованные другим узлам сети.

**Реализация**

Внешний вид приложения приведен на скриншотах:

Приложение поддерживает три режима работы: отображение полного списка всех проходящих пакетов с суммированием входящего и исходящего трафика (при нажатии на пакет в списке выводится подробная информация); отображение суммарного входящего и исходящего трафика по портам.

Листинг

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Net;

using System.IO;

namespace task44

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private Socket socket;

private string myip;

private byte[] data = new byte[4096];

private bool isRunning = false;

private static int insize;

private static int outsize;

private static Dictionary<int, int> inport;

private static Dictionary<int, int> outport;

private void Form1\_Load(object sender, EventArgs e)

{

myip = Dns.GetHostByName(Dns.GetHostName()).AddressList[0].ToString();

textBox1.Text = myip;

insize = outsize = 0;

inport = new Dictionary<int, int>();

outport = new Dictionary<int, int>();

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

if (isRunning) socket.Close();

}

private void Log(Unit u)

{

if (listBox1.InvokeRequired) listBox1.Invoke(new Action<Unit>((s) => listBox1.Items.Add(s)), u);

else listBox1.Items.Add(u);

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

if (!isRunning)

{

button1.Text = "Стоп";

isRunning = true;

timer1.Start();

socket = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.IP);

socket.Bind(new IPEndPoint(IPAddress.Parse(textBox1.Text), 0));

socket.SetSocketOption(SocketOptionLevel.IP, SocketOptionName.HeaderIncluded, true);

byte[] byTrue = new byte[4] { 1, 0, 0, 0 };

byte[] byOut = new byte[4] { 1, 0, 0, 0 };

socket.IOControl(IOControlCode.ReceiveAll, byTrue, byOut);

socket.BeginReceive(data, 0, data.Length, SocketFlags.None, new AsyncCallback(OnReceive), null);

}

else

{

button1.Text = "Старт";

isRunning = false;

timer1.Stop();

socket.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void OnReceive(IAsyncResult ar)

{

try

{

int nReceived = socket.EndReceive(ar);

ParseData(data, nReceived);

if (isRunning)

{

data = new byte[4096];

socket.BeginReceive(data, 0, data.Length, SocketFlags.None, new AsyncCallback(OnReceive), null);

}

}

catch (ObjectDisposedException)

{

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

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

{

IPHeader ipHeader = new IPHeader(byteData, nReceived);

if (radioButton1.Checked)

{

if (ipHeader.SourceAddress.ToString() == myip) outsize += ipHeader.MessageLength;

else insize += ipHeader.MessageLength;

string s = ipHeader.SourceAddress.ToString() + " -> " + ipHeader.DestinationAddress.ToString() + " " + ipHeader.MessageLength;

string f = ipHeader.SourceAddress.ToString() + " -> " + ipHeader.DestinationAddress.ToString() +

"\r\n" + ipHeader.Version +

"\r\nSize: " + ipHeader.MessageLength +

" b\r\nTime to live: " + ipHeader.TTL +

"\r\nSource: " + ipHeader.SourceAddress.ToString() +

"\r\nDestination: " + ipHeader.DestinationAddress.ToString();

switch (ipHeader.ProtocolType)

{

case Protocol.TCP:

TCPHeader tcpHeader = new TCPHeader(ipHeader.Data, ipHeader.MessageLength);

f += "\r\nProtocol: TCP\r\nSource Port: " + tcpHeader.SourcePort + "\r\nDestination Port: " + tcpHeader.DestinationPort;

break;

case Protocol.UDP:

UDPHeader udpHeader = new UDPHeader(ipHeader.Data, (int)ipHeader.MessageLength);

f += "\r\nProtocol: UDP\r\nSource Port: " + udpHeader.SourcePort + "\r\nDestination Port: " + udpHeader.DestinationPort;

break;

case Protocol.Unknown:

f += "\r\nProtocol: Unknown";

break;

}

Log(new Unit(s, f));

}

else

{

int size = ipHeader.MessageLength;

int port = 0;

switch (ipHeader.ProtocolType)

{

case Protocol.TCP:

port = Convert.ToInt32(new TCPHeader(ipHeader.Data, ipHeader.MessageLength).DestinationPort);

break;

case Protocol.UDP:

port = Convert.ToInt32(new UDPHeader(ipHeader.Data, (int)ipHeader.MessageLength).DestinationPort);

break;

case Protocol.Unknown:

return;

}

if (radioButton2.Checked)

{

if (!inport.ContainsKey(port)) inport.Add(port, size);

else inport[port] += size;

}

else

{

if (!outport.ContainsKey(port)) outport.Add(port, size);

else outport[port] += size;

}

}

}

private void label1\_Click(object sender, EventArgs e)

{

panel1.Visible = false;

}

private void listBox1\_Click(object sender, EventArgs e)

{

panel1.Visible = false;

}

private void listBox1\_DoubleClick(object sender, EventArgs e)

{

label1.Text = ((Unit)listBox1.SelectedItem).info;

panel1.Visible = true;

}

private void timer1\_Tick(object sender, EventArgs e)

{

if (radioButton1.Checked)

{

label4.Text = insize.ToString();

label5.Text = outsize.ToString();

}

else

{

listBox1.Items.Clear();

if (radioButton2.Checked)

foreach (KeyValuePair<int, int> kvp in outport) listBox1.Items.Add("Port: " + kvp.Key + " size: " + kvp.Value);

else

foreach (KeyValuePair<int, int> kvp in inport) listBox1.Items.Add("Port: " + kvp.Key + " size: " + kvp.Value);

}

}

private void button2\_Click(object sender, EventArgs e)

{

listBox1.Items.Clear();

}

}

public class IPHeader

{

private byte byVersionAndHeaderLength;

private byte byDifferentiatedServices;

private ushort usTotalLength;

private ushort usIdentification;

private ushort usFlagsAndOffset;

private byte byTTL;

private byte byProtocol;

private short sChecksum;

private uint uiSourceIPAddress;

private uint uiDestinationIPAddress;

private byte byHeaderLength;

private byte[] byIPData = new byte[4096];

public IPHeader(byte[] byBuffer, int nReceived)

{

try

{

MemoryStream memoryStream = new MemoryStream(byBuffer, 0, nReceived);

BinaryReader binaryReader = new BinaryReader(memoryStream);

byVersionAndHeaderLength = binaryReader.ReadByte();

byDifferentiatedServices = binaryReader.ReadByte();

usTotalLength = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usIdentification = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usFlagsAndOffset = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

byTTL = binaryReader.ReadByte();

byProtocol = binaryReader.ReadByte();

sChecksum = IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

uiSourceIPAddress = (uint)(binaryReader.ReadInt32());

uiDestinationIPAddress = (uint)(binaryReader.ReadInt32());

byHeaderLength = byVersionAndHeaderLength;

byHeaderLength <<= 4;

byHeaderLength >>= 4;

byHeaderLength \*= 4;

Array.Copy(byBuffer, byHeaderLength, byIPData, 0, usTotalLength - byHeaderLength);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "MJsniffer", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

public string Version

{

get

{

if ((byVersionAndHeaderLength >> 4) == 4) return "IP v4";

else if ((byVersionAndHeaderLength >> 4) == 6) return "IP v6";

else return "Unknown";

}

}

public ushort MessageLength

{

get

{

return (ushort)(usTotalLength - byHeaderLength);

}

}

public string TTL

{

get

{

return byTTL.ToString();

}

}

public Protocol ProtocolType

{

get

{

if (byProtocol == 6) return Protocol.TCP;

else if (byProtocol == 17) return Protocol.UDP;

else return Protocol.Unknown;

}

}

public IPAddress SourceAddress

{

get

{

return new IPAddress(uiSourceIPAddress);

}

}

public IPAddress DestinationAddress

{

get

{

return new IPAddress(uiDestinationIPAddress);

}

}

public byte[] Data

{

get

{

return byIPData;

}

}

}

public class UDPHeader

{

private ushort usSourcePort;

private ushort usDestinationPort;

private ushort usLength;

private short sChecksum;

private byte[] byUDPData = new byte[4096];

public UDPHeader(byte[] byBuffer, int nReceived)

{

MemoryStream memoryStream = new MemoryStream(byBuffer, 0, nReceived);

BinaryReader binaryReader = new BinaryReader(memoryStream);

usSourcePort = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usDestinationPort = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usLength = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

sChecksum = IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

Array.Copy(byBuffer, 8, byUDPData, 0, nReceived - 8);

}

public string SourcePort

{

get

{

return usSourcePort.ToString();

}

}

public string DestinationPort

{

get

{

return usDestinationPort.ToString();

}

}

public byte[] Data

{

get

{

return byUDPData;

}

}

}

public class TCPHeader

{

private ushort usSourcePort;

private ushort usDestinationPort;

private uint uiSequenceNumber = 555;

private uint uiAcknowledgementNumber = 555;

private ushort usDataOffsetAndFlags = 555;

private ushort usWindow = 555;

private short sChecksum = 555;

private ushort usUrgentPointer;

private byte byHeaderLength;

private ushort usMessageLength;

private byte[] byTCPData = new byte[4096];

public TCPHeader(byte[] byBuffer, int nReceived)

{

try

{

MemoryStream memoryStream = new MemoryStream(byBuffer, 0, nReceived);

BinaryReader binaryReader = new BinaryReader(memoryStream);

usSourcePort = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usDestinationPort = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

uiSequenceNumber = (uint)IPAddress.NetworkToHostOrder(binaryReader.ReadInt32());

uiAcknowledgementNumber = (uint)IPAddress.NetworkToHostOrder(binaryReader.ReadInt32());

usDataOffsetAndFlags = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usWindow = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

sChecksum = (short)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

usUrgentPointer = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

byHeaderLength = (byte)(usDataOffsetAndFlags >> 12);

byHeaderLength \*= 4;

usMessageLength = (ushort)(nReceived - byHeaderLength);

Array.Copy(byBuffer, byHeaderLength, byTCPData, 0, nReceived - byHeaderLength);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public string SourcePort

{

get

{

return usSourcePort.ToString();

}

}

public string DestinationPort

{

get

{

return usDestinationPort.ToString();

}

}

public byte[] Data

{

get

{

return byTCPData;

}

}

}

public enum Protocol

{

TCP = 6,

UDP = 17,

Unknown = -1

};

public class Unit

{

string shortinfo;

string fullinfo;

public string info

{

get

{

return fullinfo;

}

}

public Unit (string s1, string s2)

{

shortinfo = s1;

fullinfo = s2;

}

public override string ToString()

{

return shortinfo;

}

}

**Вывод**

В результате выполнения лабораторной работы исследованы функции работы с трафиком в локальной сети. Написано приложение, анализирующего проходящий трафик пассивными методами.