

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Перехват и протоколирование сетевого трафика

Студент

Басенко К. А.

Руководитель

В. Д. Владимцев

Минск 2022

Министерство образования Республики Беларусь

Учреждение образования

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ**

Факультет КС и С Кафедра Информатики
Специальность 1-40 04 01 Специализация _____

ЗАДАНИЕ

по курсовому проекту студента

Басенко Кирилла Александровича

(фамилия, имя, отчество)

1. Тема проекта: **Перехват и протоколирование сетевого трафика**

2. Срок сдачи студентом законченной работы _____

3. Исходные данные к проекту Тип операционной системы – ОС Microsoft Windows;
Языки программирования – С#;

Цель проекта: разработка сервисов или компонентов ядра, модифицирующие и/или
дополняющие текущие функции ОС (платформа Windows).

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)

1 Введение

2 Формирование требований к сервису

3 Обзор используемых технологий

4 Проектирование сервиса

5 Программная реализация

6 Тестирование приложения

Заключение

Список использованных источников

Приложение А - Текст программы

5. Перечень графического материала (с точным указанием наименования) и обозначения
вида и типа материала)

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта (работы)	Объём этапа в %	Срок выполнения этапа	Примечан ие
Анализ предметной области, разработка технического задания	15-20	11.02–01.03	
Разработка функциональных требований, проектирование архитектуры программы	20-15	02.03–15.03	
Разработка схемы программы, алгоритмов, схемы данных	20-15	16.03–01.05	
Разработка программного средства	15-20	02.05– 10.05	
Тестирование и отладка	10	11.05-12.05	
Оформление пояснительной записки и графического материала	20	13.05-14.05	

Дата выдачи задания _____ Руководитель В. Д. Владимцев

Задание принял к исполнению _____ К. А. Басенко

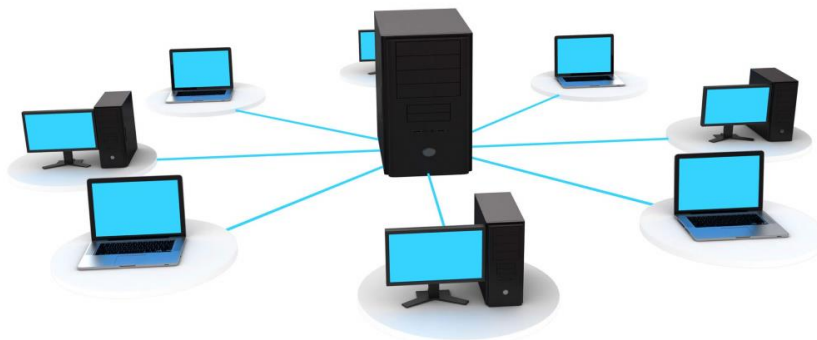
Содержание:

1. Введение.....	5
2. Формирование требований к сервису.....	8
3. Обзор используемых технологий.....	9
4. Проектирование сервиса.....	12
5. Программная реализация.....	18
6. Тестирование приложения.....	21
7. Заключение.....	22
8. Список используемой литературы.....	23
9. Приложение А - текст программы.....	24

Введение

Сетевой трафик - данные, передаваемые через компьютерную сеть за любой период времени. Данными обмениваются приложения. Например, когда мы заходим на сайт с помощью браузера, происходит общение между браузером и сайтом, в котором браузер представляет себя и пользователя сайту и запрашивает доступ к ресурсам, которые сайт и представляет браузеру, которые браузер, в свою очередь, представляет пользователю.

Компьютерная сеть - множество компьютеров, компьютерных устройств, соединенных линиями связи и работающих под управлением специального программного обеспечения. Взаимодействие в компьютерной сети происходит по единым правилам, определенных сетевыми протоколами.



Компьютерная сеть

Для объединения компьютеров и компьютерных устройств в компьютерную сеть применяются аппаратные средства (сетевые адаптеры, установленные на компьютерах, коммутаторы, маршрутизаторы, кабели и др.) и программные средства, которые реализуют правила взаимодействия программных и аппаратных компонентов компьютерной сети, определенные сетевыми протоколами. Сетевые протоколы соответствуют действующим в компьютерной сети сетевым технологиям (комплекс программно-аппаратно реализованных методов, определяющих функционирование компьютерной сети).

По назначению программные и аппаратные составляющие компьютерной сети условно разделяют на три уровня: верхний – прикладные программы (серверы и клиенты электронной почты, веб-серверы и браузеры и др.); средний – программные средства, реализующие сетевые протоколы; нижний – сетевые аппаратные

сред-ст-ва. Ком-пью-те-ры и ком-пью-тер-ные уст-рой-ст-ва в со-ста-ве компьютерных сетей име-ют уни-каль-ные (в пре-де-лах компьютерных сетей се-те-вые ад-ре-са (при-над-ле-жа-щие еди-но-му се-те-во-му ад-рес-но-му про-стран-ст-ву) и на-зы-ва-ют-ся уз-ла-ми компьютерной сети. Фор-мат и ин-тер-пре-та-ция се-те-во-го ад-ре-са оп-ре-де-ля-ют-ся при-ме-няе-мым се-те-вым про-то-ко-лом (напр., ес-ли при-ме-ня-ет-ся IP-про-то-кол, то уз-лы по-лу-ча-ют IP-ад-ре-са).

4 бита Номер версии	4 бита Длина заголовка	8 бит Тип сервиса				16 бит Общая длина																					
		PR	D	T	R																						
16 бит Идентификатор пакета										3 бита Флаги		13 бит Смещение фрагмента															
						D		M																			
8 бит Время жизни				8 бит Протокол верхнего уровня								16 бит Контрольная сумма															
32 бита IP-адрес источника																											
32 бита IP-адрес назначения																											
Параметры и выравнивание																											

Структура ip-пакета

Взаи-мо-дей-ст-вие ме-ж-ду уз-ла-ми осу-ще-ст-в-ля-ет-ся пу-тём об-ме-на со-об-ще-ния-ми, раз-би-ты-ми на не-боль-шие бло-ки оп-ре-де-лён-но-го фор-ма-та (се-те-вые па-ке-ты). Раз-бив-ку на па-ке-ты, их сжа-тие (при не-об-хо-ди-мо-сти) и др. осу-ще-ст-в-ля-ют про-грам-мы, реа-ли-зую-щие се-те-вые про-то-ко-лы. Оп-ти-че-ские или элек-трич. сиг-на-лы, со-от-вет-ст-вую-щие зна-че-ни-ям би-тов в со-ста-ве се-те-вых па-ке-тов, пе-ре-да-ют-ся по мед-ным или оп-то-во-ло-кон-ным ка-бе-лям, а так-же сред-ст-ва-ми бес-про-вод-ной свя-зи (напр., ра-дио-сиг-на-ла-ми).

Протоколировать сетевой трафик нужно для разных целей, будь то диагностика сети, просмотр передаваемых данных, или отслеживание действий пользователей и программ; просто говоря, для его дальнейшего анализа.

Задача анализа сетевого трафика приобретает все большую актуальность в связи с развитием и внедрением новых сетевых технологий (и, как следствие, увеличением объема данных, передаваемых по сети), а также появлением большого количества новых сетевых протоколов прикладного уровня. В качестве наиболее популярных областей практического применения можно выделить:

- анализ трафика с целью выявления проблем в работе сети (в том числе, несанкционированной активности);

- восстановление потоков данных («прослушивание»);
- предотвращение различного рода сетевых атак;
- сбор статистики.

Если говорить о комплексном решении задачи анализа сетевого трафика, то в первую очередь следует разделить ее на три в достаточной степени независимые подзадачи: перехват трафика, его хранение и анализ.

Система анализа должна обеспечивать захват 100% трафика, а также предоставлять эффективные методы анализа и навигации по его результатам. Захват трафика осуществляется посредством снифферов. В общем случае, сниффер – это программа или программно-аппаратное устройство, предназначенное для перехвата трафика. В рамках конкретных продуктов могут быть реализованы дополнительные возможности, например, разбор заголовков сетевых протоколов, фильтрация по заданным критериям, восстановление сессий. Перехват сетевого трафика может осуществляться:

- с помощью «прослушивания» сетевого интерфейса;
- подключением сниффера в разрыв канала;
- посредством анализа побочных электромагнитных излучений;
- через атаку на канальном или сетевом уровне, приводящую к перенаправлению трафика жертвы на сниффер.

В нашем случае сниффер будет установлен на конечном узле сети, компьютере с операционной системой Windows. Т.к. весь трафик должен быть захвачен, то нельзя полагаться на то, что при запуске системы пользователь сам запустит нужное обеспечение для перехвата трафика. В этом случае нам нужно инкапсулировать процесс перехвата трафика.

Одним из таких решений будет служба, которая и будет перехватывать весь трафик. Такой подход позволит перехватывать весь трафик независимо от того, зашел ли пользователь в систему, управляет пользователь системой благодаря графическому пользовательскому интерфейсу, или же посредством терминала. Службы позволяют создавать долговременные исполняемые приложения, которые запускаются в собственных сеансах Windows. Для этих служб не предусмотрен пользовательский интерфейс. Они могут запускаться автоматически при загрузке компьютера, их также можно приостанавливать и перезапускать. Благодаря этому службы идеально подходят для использования на сервере, а также в ситуациях, когда необходимы долго выполняемые процессы, которые не мешают работе пользователей на том же компьютере.

Формирование требований

Данный проект будет представлять из себя сервис, который перехватывает сетевой трафик и хранит его. Он должен перехватывать сетевой исходящий и входящий трафик, передаваемые по протоколу IP, расшифровывать по протоколу TCP или UDP, и записывать его в файл для дальнейшего анализа при необходимости или по желанию.

Обзор используемых технологий

Данный проект написан на языке C# (произносится как "си шарп") — современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. Здесь представлен обзор основных компонентов языка C# 8 и более ранних версий. Если вы хотите изучить язык с помощью интерактивных примеров, рекомендуем поработать с вводными руководствами по C#.

C# — объектно-ориентированный, ориентированный на компоненты язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО. В основном C# — объектно-ориентированный язык. Вы определяете типы и их поведение.

Вот лишь несколько функций языка C#, которые позволяют создавать надежные и устойчивые приложения. Сборка мусора автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них. Лямбда-выражения поддерживают приемы функционального программирования. Синтаксис LINQ создает общий шаблон для работы с данными из любого источника.

Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем. В C# имеется Единая система типов. Все типы C#, включая типы-примитивы, такие как `int` и `double`, наследуют от одного корневого типа `object`. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, C# поддерживает как определяемые пользователями ссылочные типы, так и типы значений. C# позволяет динамически выделять объекты и хранить упрощенные структуры в стеке. C# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. C# предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода.

C# подчеркивает управление версиями чтобы обеспечить совместимость

программ и библиотек с течением времени. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как отдельные модификаторы `virtual` и `override`, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

```
Stack<string> paramsAndArgs = new ();
int index;
foreach (var consoleArg in consoleArgs.Reverse())
{
    if ((index = consoleArg.IndexOf('=', StringComparison.Ordinal)) != -1)
    {
        paramsAndArgs.Push(consoleArg[(index + 1) ..]);
        paramsAndArgs.Push(consoleArg[..index]);
        continue;
    }
    paramsAndArgs.Push(consoleArg);
}
```

Пример программы на языке C#

Для перехвата пакета используется библиотека SharpPcap. Цель SharpPcap - обеспечить основу для захвата, инъекции и анализа сетевых пакетов для .NET приложений.

SharpPcap - это полностью управляемая кросс-платформенная библиотека. Та же сборка работает под Microsoft .NET также как Mono на 32 и 64-битных платформах.

Возможности, которые в настоящее время поддерживаются в SharpPcap:

1. Одна сборка для Microsoft .NET и Mono платформ на Windows (32 или 64-разрядные), Linux (32 или 64 бит) и Mac.
2. Высокая производительность — SharpPcap позволяет захватывать данные до >3MB/s скорости передачи
3. Удаленный захват пакетов
4. Инъекции пакетов, используя отправку очередей.
5. Сбор сетевой статистики по определенному сетевому интерфейсу
6. Поддержка AirPcap
7. Перечисление и отображение подробных сведений о физических и сетевых интерфейсах на Windows-машине.
8. Захват низкоуровневых сетевых пакетов, проходящих через определенный интерфейс.
9. Использование Packet.Net для разбора пакетов
10. Чтение и запись в pcap файлы

```

// Retrieve the device list
var devices = CaptureDeviceList.Instance;

// If no devices were found print an error
if (devices.Count < 1)
{
    Console.WriteLine("No devices were found on this machine");
    return;
}

Console.WriteLine("The following devices are available on this machine:");
Console.WriteLine("-----");
Console.WriteLine();

int i = 0;

// Print out the available devices
foreach (var dev in devices)
{
    Console.WriteLine("{0} {1}", i, dev.Description);
    i++;
}

Console.WriteLine();
Console.Write("-- Please choose a device to send a packet on: ");
i = int.Parse(Console.ReadLine());

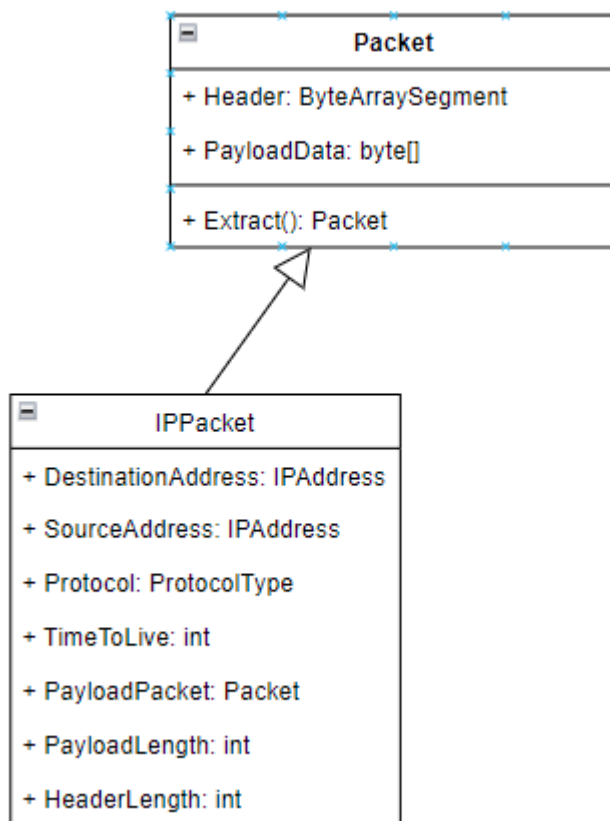
using var device = devices[i];

Console.Write("-- This will send a random packet out this interface, " +
    "continue? [YES|no]");
string resp = Console.ReadLine().ToLower();

```

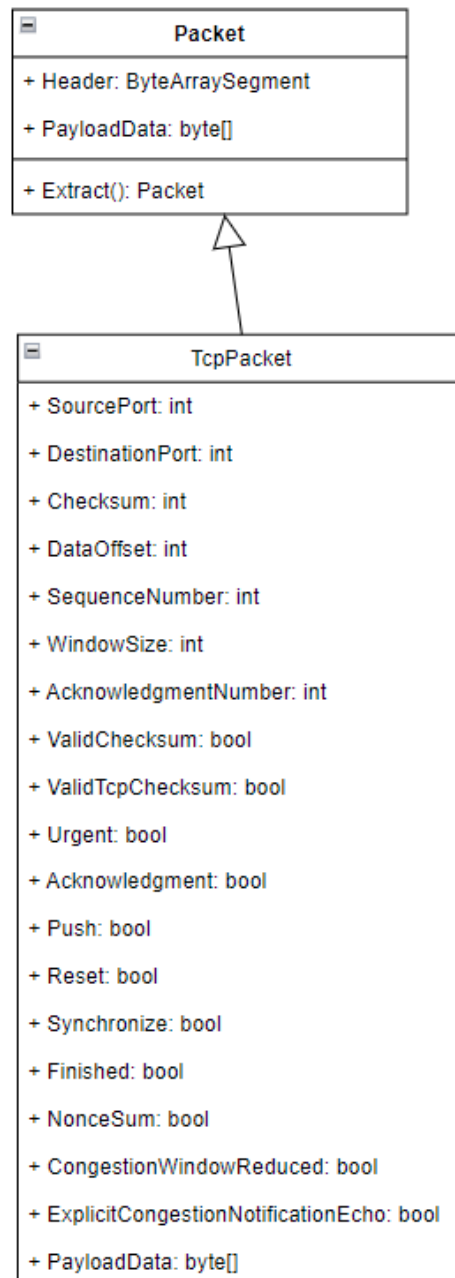
Пример программы с использованием SharpPcap

Проектирование сервиса

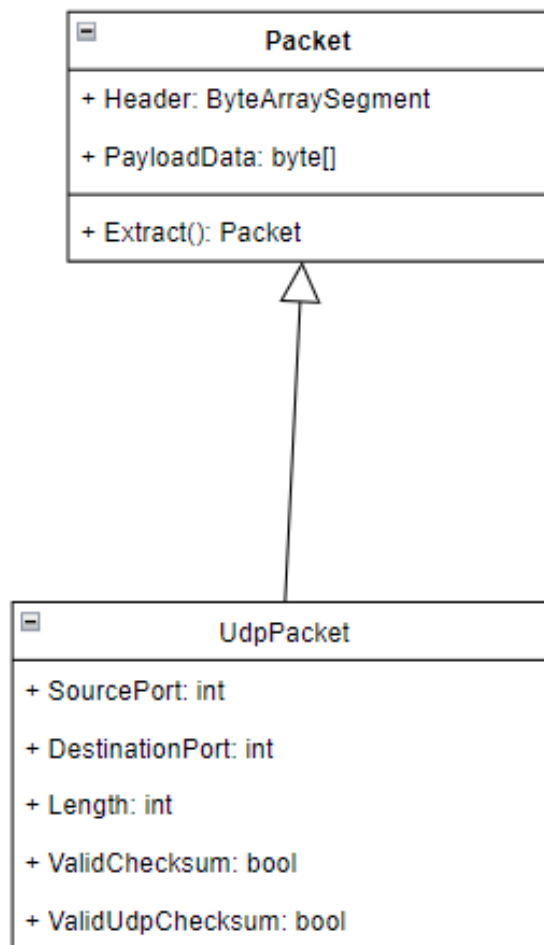


Класс IPpacket

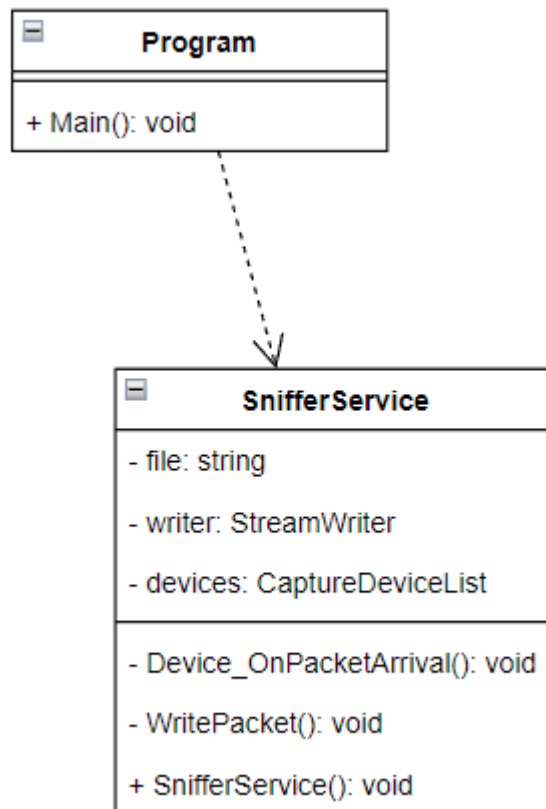
Класс IPpacket представляет IP-пакет, из которого будут извлечены tcp- и udp-пакеты.



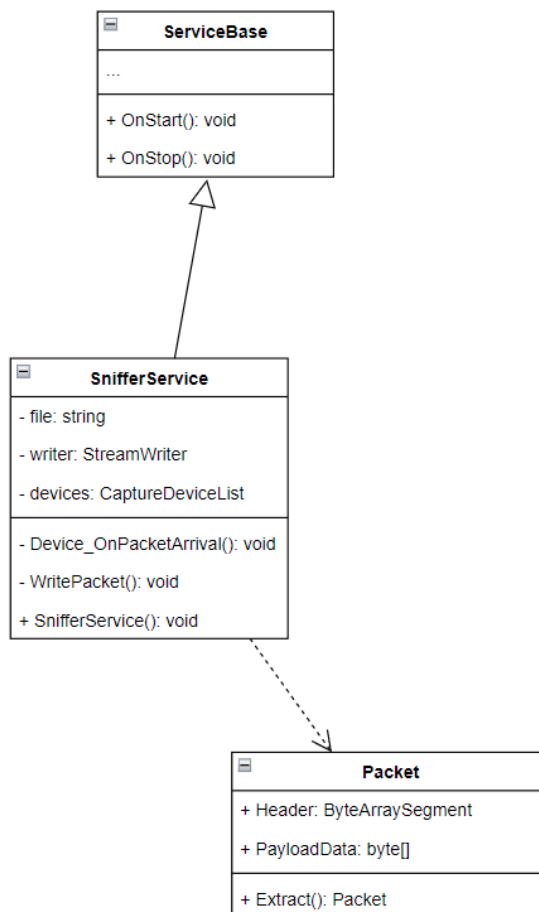
Класс TcpPacket



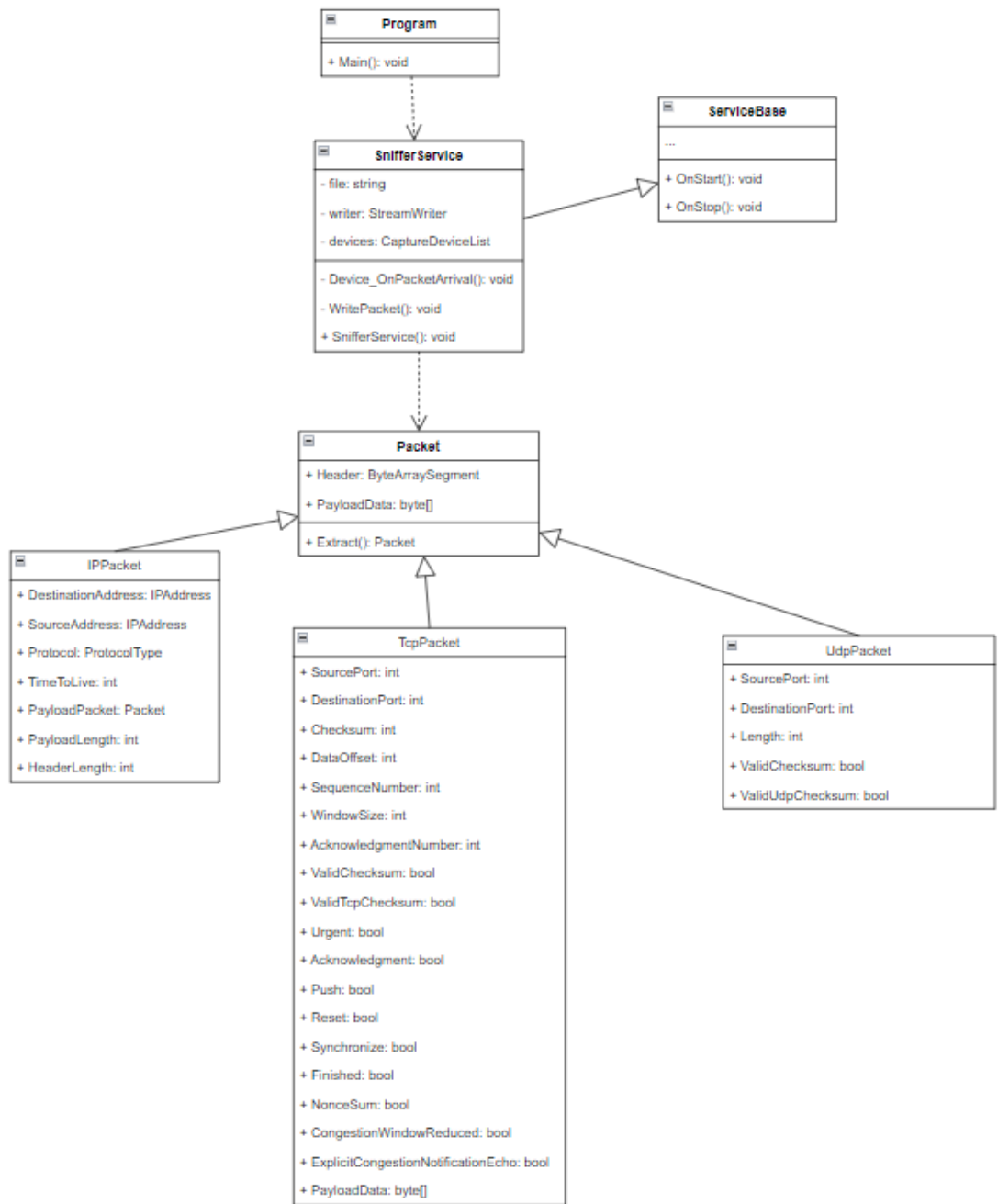
класс UdpPacket



класс Program



класс SnifferService



итоговая UML-диаграмма используемых классов программы

Программная реализация

```
public SnifferService()
{
    InitializeComponent();

    this.file = Path.Combine(
        Environment.GetFolderPath(
            Environment.SpecialFolder.Desktop),
        "captured.txt");

    this.writer = new StreamWriter(
        File.OpenWrite(file));

    this.devices = CaptureDeviceList.Instance;
}
```

При старте сервиса подписываемся на событие *OnPacketArrival* на всех считывающих устройствах и начинаем перехватывать все пакеты.

```
protected override void OnStart(string[] args)
{
    foreach (var device in devices)
    {
        device.Open();
        device.OnPacketArrival += Device_OnPacketArrival;
        device.StartCapture();
    }
}
```

Обработчик события:

```

private void Device_OnPacketArrival(object s, PacketCapture e)
{
    var e_packet = e.GetPacket();

    DateTime time = e_packet.Timeval.Date;
    var time_str = (time.Hour + 1) + ":" + time.Minute + ":" + time.Second + ":" + time.Millisecond;

    var packet = Packet.ParsePacket(e_packet.LinkLayerType, e_packet.Data);

    IPPacket ipPacket = packet.Extract<IPPacket>();

    if (ipPacket != null)
    {
        System.Net.IPAddress srcIp = ipPacket.SourceAddress;
        System.Net.IPAddress dstIp = ipPacket.DestinationAddress;
        var protocol_type = ipPacket.Protocol;
        var sourceIP = srcIp.ToString();
        var destinationIP = dstIp.ToString();
        var payload_packet = ipPacket.PayloadPacket;
        var ttl = ipPacket.TimeToLive;



        WritePacket(packet, protocol_type, sourceIP, destinationIP, payload_packet, time_str, ttl);
        Console.WriteLine("\n");
    }
}

```

WritePacket:

```

private void WritePacket(
    Packet packet, ProtocolType protocol,
    string sourceIP, string destinationIP,
    Packet payload_packet, string time, int ttl)
{
    this.writer.WriteLine($"Time: {time}");
    this.writer.WriteLine($"Protocol type: {protocol.ToString()}");
    this.writer.WriteLine($"Source ip: {sourceIP}");
    this.writer.WriteLine($"Destination ip: {destinationIP}");
    this.writer.WriteLine($"Time to live: {ttl}");

    switch (protocol)
    {
        case ProtocolType.Tcp:
            
        case ProtocolType.Udp:
            
    }

    writer.Flush();
}

```

Tcp:

```

case ProtocolType.Tcp:
{
    var tcpPacket = packet.Extract<TcpPacket>();
    if (tcpPacket != null)
    {
        int srcPort = tcpPacket.SourcePort;
        int dstPort = tcpPacket.DestinationPort;
        var checksum = tcpPacket.Checksum;

        var str =
            "Source port: " + srcPort +
            "\r\nDestination port: " + dstPort +
            "\r\nData offset: " + tcpPacket.DataOffset +
            "\r\nWindow size: " + tcpPacket.WindowSize +
            "\r\nChecksum: " + checksum.ToString() + (tcpPacket.ValidChecksum ? ",valid" : ",invalid") +
            "\r\nTCP checksum: " + (tcpPacket.ValidTcpChecksum ? ",valid" : ",invalid") +
            "\r\nSequence number: " + tcpPacket.SequenceNumber.ToString() +
            "\r\nAcknowledgment number: " + tcpPacket.AcknowledgmentNumber + (tcpPacket.Acknowledgment ? ",valid" : ",invalid") +
            "\r\nUrgent pointer: " + (tcpPacket.Urgent ? "valid" : "invalid") +
            "\r\nACK flag: " + (tcpPacket.Acknowledgment ? "1" : "0") +
            "\r\nPSH flag: " + (tcpPacket.Push ? "1" : "0") +
            "\r\nRST flag: " + (tcpPacket.Reset ? "1" : "0") +
            "\r\nSYN flag: " + (tcpPacket.Synchronize ? "1" : "0") +
            "\r\nFIN flag: " + (tcpPacket.Finished ? "1" : "0") +
            "\r\nNS flag: " + (tcpPacket.NonceSum ? "1" : "0") +
            "\r\nCWR flag: " + (tcpPacket.CongestionWindowReduced ? "1" : "0") +
            "\r\nECE flag: " + (tcpPacket.ExplicitCongestionNotificationEcho ? "1" : "0") +
            "\r\nURG flag: " + (tcpPacket.Urgent ? "1" : "0") +
            $" \r\nPayload data: {string.Join("", payload_packet.PayloadData.Select((o) => o.ToString("x")))}";

        writer.WriteLine(str);
        writer.WriteLine("\n");
        Console.WriteLine(str);
    }
    break;
}

```

Udp:

```

case ProtocolType.Udp:
{
    var udpPacket = packet.Extract<UdpPacket>();
    if (udpPacket != null)
    {
        int srcPort = udpPacket.SourcePort;
        int dstPort = udpPacket.DestinationPort;
        var checksum = udpPacket.Checksum;

        var str =
            "Source port: " + srcPort +
            "\r\nDestination port: " + dstPort +
            "\r\nChecksum: " + checksum.ToString() + " valid: " + udpPacket.ValidChecksum +
            "\r\nValid UDP checksum: " + udpPacket.ValidUdpChecksum +
            $" \r\nPayload data: {string.Join("", payload_packet.PayloadData.Select((o) => o.ToString("x")))}";

        writer.WriteLine(str);
        writer.WriteLine("\n");
        Console.WriteLine(str);
    }
    break;
}

```

Тестирование приложения

Для тестирования приложения на другом устройстве в локальной сети Wi-fi был написан скрипт, который по протоколу UDP отправляет определенный набор чисел через сокет:

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as ss:
    addr = ("192.168.100.8", 8000)

    ss.bind(("192.168.100.5", 8000))

    data = bytes([0, 1, 2, 3, 4, 5, 6, 7, 8])

    ss.sendto(data, addr)
```

```
Time: 1:6:34:150
Protocol type: Udp
Source ip: 192.168.100.5
Source dns name: User-PC.local
Destination ip: 192.168.100.8
Destination dns name: DESKTOP-AHPC276
Time to live: 65
Source port: 8000
Destination port: 8000
Checksum: 25565 valid: True
Valid UDP checksum: True
Payload data: 012345678
```

Заключение

В ходе данной работы была реализована служба-сниффер, которая перехватывает отправляемые и получаемые ip-пакеты, достает из них tcp- и udp-пакеты, и хранит их для дальнейшего анализа.

Список используемой литературы

- docs.microsoft.com/en-us/dotnet/csharp/ [электронный доступ 09.05.2022]
- [Github.com/dotnetcap/sharppcap](https://github.com/dotnet/sharppcap) [электронный доступ 09.05.2022]
- [Wikipedia.org](https://en.wikipedia.org/wiki/SharpCap) [электронный доступ 09.05.2022]

Приложение А - текст программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;

namespace Sniffer
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main()
        {
            ServiceBase[] ServicesToRun;

            ServicesToRun = new ServiceBase[]
            {
                new SnifferService()
            };

            ServiceBase.Run(ServicesToRun);
        }
    }
}
```



```
}
```

```
using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Configuration.Install;  
using System.Linq;  
using System.ServiceProcess;  
using System.Threading.Tasks;
```

```
namespace Sniffer
```

```
{
```

```
    [RunInstaller(true)]
```

```
    public partial class ProjectInstaller : System.Configuration.Install.Installer
```

```
    {
```

```
        private ServiceInstaller serviceInstaller;
```

```
        private ServiceProcessInstaller processInstaller;
```

```
        public ProjectInstaller()
```

```
        {
```

```
            InitializeComponent();
```

```
            // Instantiate installers for process and services.
```

```
            processInstaller = new ServiceProcessInstaller();
```

```

serviceInstaller = new ServiceInstaller();

// The services run under the system account.
processInstaller.Account = ServiceAccount.LocalSystem;

// The services are started manually.
serviceInstaller.StartType = ServiceStartMode.Manual;

// ServiceName must equal those on ServiceBase derived classes.
serviceInstaller.ServiceName = "Sniffer";

// Add installers to collection. Order is not important.
Installers.Add(serviceInstaller);
Installers.Add(processInstaller);
}

private void serviceProcessInstaller1_AfterInstall(object sender,
InstallEventArgs e)
{

}

private void serviceinstaller1_AfterInstall(object sender,
InstallEventArgs e)
{

```

```

    }

    //public ProjectInstaller()
    //{
    //    InitializeComponent();
    //}
}

```

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration.Install;
using System.Linq;
using System.ServiceProcess;
using System.Threading.Tasks;

namespace Sniffer
{
    [RunInstaller(true)]
    public partial class ProjectInstaller : System.Configuration.Install.Installer
    {
        private ServiceInstaller serviceInstaller;
    }
}

```

```

private ServiceProcessInstaller processInstaller;

public ProjectInstaller()
{
    InitializeComponent();

    // Instantiate installers for process and services.
    processInstaller = new ServiceProcessInstaller();
    serviceInstaller = new ServiceInstaller();

    // The services run under the system account.
    processInstaller.Account = ServiceAccount.LocalSystem;

    // The services are started manually.
    serviceInstaller.StartType = ServiceStartMode.Manual;

    // ServiceName must equal those on ServiceBase derived classes.
    serviceInstaller.ServiceName = "Sniffer";

    // Add installers to collection. Order is not important.
    Installers.Add(serviceInstaller);
    Installers.Add(processInstaller);
}

private void serviceProcessInstaller1_AfterInstall(object sender,
InstallEventArgs e)

```

```

    {

    }

    private void serviceinstaller1_AfterInstall(object sender,
InstallEventArgs e)
    {

    }

    //public ProjectInstaller()
    //{
    //    InitializeComponent();
    //}
}

namespace Sniffer
{
    partial class SnifferService
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>

        private System.ComponentModel.IContainer components = null;

```

```

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>

    /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

```

#region Component Designer generated code

```

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>

    private void InitializeComponent()
    {
        components = new System.ComponentModel.Container();
        this.ServiceName = "Service1";
    }

```

```
        #endregion  
    }  
}
```

```
using PacketDotNet;  
using SharpPcap;  
using System;  
using System.Data;  
using System.IO;  
using System.Linq;  
using System.ServiceProcess;
```

```
namespace Sniffer  
{  
    public partial class SnifferService : ServiceBase  
    {  
        private string file;  
  
        private StreamWriter writer;  
  
        private CaptureDeviceList devices;  
  
        public SnifferService()
```

```

{
    InitializeComponent();

    this.file = Path.Combine(
        Environment.GetFolderPath(
            Environment.SpecialFolder.Desktop),
        "captured.txt");

    this.writer = new StreamWriter(
        File.OpenWrite(file));

    this.devices = CaptureDeviceList.Instance;
}

protected override void OnStart(string[] args)
{
    foreach (var device in devices)
    {
        device.Open();
        device.OnPacketArrival += Device_OnPacketArrival;
        device.StartCapture();
    }
}

protected override void OnStop()

```



```

    {
        foreach (var device in this.devices)
        {
            device.Close();
        }

        writer.Close();
    }

    private void Device_OnPacketArrival(object s, PacketCapture e)
    {
        var e_packet = e.GetPacket();

        DateTime time = e_packet.Timeval.Date;

        var time_str = (time.Hour + 1) + ":" + time.Minute + ":" +
time.Second + ":" + time.Millisecond;

        var packet = Packet.ParsePacket(e_packet.LinkLayerType,
e_packet.Data);

        IPPacket ipPacket = packet.Extract<IPPacket>();

        if (ipPacket != null)
        {
            System.Net.IPAddress srcIp = ipPacket.SourceAddress;

            System.Net.IPAddress dstIp = ipPacket.DestinationAddress;

```

```

        var protocol_type = ipPacket.Protocol;
        var sourceIP = srcIp.ToString();
        var destinationIP = dstIp.ToString();
        var payload_packet = ipPacket.PayloadPacket;
        var ttl = ipPacket.TimeToLive;

        WritePacket(packet, protocol_type, sourceIP, destinationIP,
payload_packet, time_str, ttl);

        Console.WriteLine("\n");
    }
}

```

```

private void WritePacket(
    Packet packet, ProtocolType protocol,
    string sourceIP, string destinationIP,
    Packet payload_packet, string time, int ttl)
{
    this.writer.WriteLine($"Time: {time}");
    this.writer.WriteLine($"Protocol type: {protocol.ToString()}");
    this.writer.WriteLine($"Source ip: {sourceIP}");
    this.writer.WriteLine($"Destination ip: {destinationIP}");
    this.writer.WriteLine($"Time to live: {ttl}");

    switch (protocol)
    {

```

```

case ProtocolType.Tcp:
{
    var tcpPacket = packet.Extract<TcpPacket>();
    if (tcpPacket != null)
    {
        int srcPort = tcpPacket.SourcePort;
        int dstPort = tcpPacket.DestinationPort;
        var checksum = tcpPacket.Checksum;

        var str =
            "Source port:" + srcPort +
            "\r\nDestination port: " + dstPort +
            "\r\nData offset: " + tcpPacket.DataOffset +
            "\r\nWindow size: " + tcpPacket.WindowSize +
            "\r\nChecksum:" + checksum.ToString() +
            (tcpPacket.ValidChecksum ? ",valid" : ",invalid") +
            "\r\nTCP checksum: " + (tcpPacket.ValidTcpChecksum ?
            ",valid" : ",invalid") +
            "\r\nSequence number: " +
            tcpPacket.SequenceNumber.ToString() +
            "\r\nAcknowledgment number: " +
            tcpPacket.AcknowledgmentNumber + (tcpPacket.Acknowledgment ? ",valid" :
            ",invalid") +
            "\r\nUrgent pointer: " + (tcpPacket.Urgent ? "valid" :
            "invalid") +
            "\r\nACK flag: " + (tcpPacket.Acknowledgment ? "1" :
            "0") +
            "\r\nPSH flag: " + (tcpPacket.Push ? "1" : "0") +

```

```

        "\r\nRST flag: " + (tcpPacket.Reset ? "1" : "0") +
        "\r\nSYN flag: " + (tcpPacket.Synchronize ? "1" : "0") +
        "\r\nFIN flag: " + (tcpPacket.Finished ? "1" : "0") +
        "\r\nNS flag: " + (tcpPacket.NonceSum ? "1" : "0") +
        "\r\nCWR flag: " +
(tcpPacket.CongestionWindowReduced ? "1" : "0") +
        "\r\nECE flag: " +
(tcpPacket.ExplicitCongestionNotificationEcho ? "1" : "0") +
        "\r\nURG flag: " + (tcpPacket.Urgent ? "1" : "0") +
        $" \r\nPayload data: {string.Join("",
payload_packet.PayloadData.Select((o) => o.ToString("x"))}";

```

```

        writer.WriteLine(str);
        writer.WriteLine("\n");
        Console.WriteLine(str);
    }
    break;
}

case ProtocolType.Udp:
{
    var udpPacket = packet.Extract<UdpPacket>();
    if (udpPacket != null)
    {
        int srcPort = udpPacket.SourcePort;
        int dstPort = udpPacket.DestinationPort;
        var checksum = udpPacket.Checksum;

```

```

        var str =

            "Source port:" + srcPort +

            "\r\nDestination port: " + dstPort +

            "\r\nChecksum:" + checksum.ToString() + " valid: " +
udpPacket.ValidChecksum +

            "\r\nValid UDP checksum: " +
udpPacket.ValidUdpChecksum +

            $" \r\nPayload data: {string.Join("",
payload_packet.PayloadData.Select((o) => o.ToString("x"))}";

        writer.WriteLine(str);

        writer.WriteLine("\n");

        Console.WriteLine(str);

    }

    break;

}

}

writer.Flush();

}

}

}

```