СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc326120634)

[1 ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc326120635)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 15](#_Toc326120636)

[2.1 Структура программного средства для сканирования сети 15](#_Toc326120637)

[2.2 Средства разработки 17](#_Toc326120638)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 21](#_Toc326120639)

[3.1 Выбор протокола для сканирования 21](#_Toc326120640)

[3.2 Алгоритмы сканирования и трассировки 23](#_Toc326120641)

[3.3 Классы межсетевого взаимодействия 24](#_Toc326120642)

[3.4 Файлы 33](#_Toc326120643)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 35](#_Toc326120644)

[4.1 Алгоритм формирования диапазона IP-адресов 35](#_Toc326120645)

[4.2 Алгоритм работы с ARP-протоколом 38](#_Toc326120646)

[4.3 Алгоритм трассировки сети 42](#_Toc326120647)

[4.4 Алгоритм построения графов 45](#_Toc326120648)

[5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 50](#_Toc326120649)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 55](#_Toc326120650)

[6.1 Инсталляция приложения 55](#_Toc326120651)

[6.2 Настройка и запуск сканирования 57](#_Toc326120652)

[7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПРОЕКТА 60](#_Toc326120653)

[8 ОХРАНА ТРУДА. ОБЕСПЕЧЕНИЕ ЗАЩИТЫ РАБОТАЮЩИХ ОТ ИОНИЗИРУЮЩИХ ИЗЛУЧЕНИЙ 71](#_Toc326120654)

[ЗАКЛЮЧЕНИЕ 75](#_Toc326120655)

[ПРИЛОЖЕНИЕ А. Листинг кода класса NetworkScaner 78](#_Toc326120656)

[ПРИЛОЖЕНИЕ Б. Листинг кода класса GraphBuilder 82](#_Toc326120657)

[ПРИЛОЖЕНИЕ В. Спецификация программного дипломного проекта 86](#_Toc326120658)

[ПРИЛОЖЕНИЕ Г. Ведомость документов 87](#_Toc326120659)

# ВВЕДЕНИЕ

Сегодня уже трудно представить себе, как люди жили когда-то без столь удобного и полезного инструмента, как локальные сети. Однако знало человечество и такие времена. Впервые идея связать несколько независимо работающих компьютеров в единую распределенную вычислительную систему посетила светлые головы инженеров еще в середине 60-х годов XX века. А если говорить более конкретно, то первый успешный эксперимент по передаче дискретных пакетов данных между двумя компьютерами провел в 1965 году молодой исследователь из лаборатории Линкольна Массачусетского технологического института Лари Роберте. Алгоритмы передачи данных, предложенные Робертсом, во многом послужили основой для построенной в 1969 году по инициативе американского «Агентства перспективных научных исследований» глобальной вычислительной сети ARPANet. Она впоследствии, объединившись с несколькими другими существовавшими на тот момент сетями, стала фундаментом, на котором вырос современный Интернет. Однако и широко использовавшиеся в те времена многотерминальные системы, в которых пользователям предоставлялся доступ к одному головному многофункциональному компьютеру посредством нескольких конечных устройств удаленного подключения — терминалов — по принципу разделения процессорного времени, и глобальные сети, объединявшие между собой мейнфреймы крупных вычислительных центров и лабораторий, являлись лишь предтечей локальных сетей в их нынешнем понимании. Существенный толчок в направлении развития малых локальных сетей дало бурное развитие во второй половине 70-х годов настольных персональных компьютеров.

Создание компьютерных сетей вызвано практической потребностью совместного использования информации пользователями, работающими на удаленных друг от друга компьютерах. Сети предоставляют пользователям возможность не только быстрого обмена информацией, но и совместного использования принтеров и других периферийных устройств и даже одновременной работы с документами.

Локальная сеть объединяет несколько компьютеров и дает возможность пользователям совместно использовать ресурсы компьютеров, а также подключенных к сети периферийных устройств (принтеров, плоттеров, дисков, модемов и др.).

В небольших локальных сетях все компьютеры обычно равноправны, то есть пользователи самостоятельно решают, какие ресурсы своего компьютера (диски, каталоги, файлы) сделать общедоступными по сети. Такие сети называются одноранговыми. Если к локальной сети подключено большое количество компьютеров, одноранговая сеть может оказаться недостаточно производительной. Для увеличения производительности, а также в целях обеспечения большей надежности при хранении информации в сети некоторые компьютеры специально выделяются для хранения файлов и программных приложений. Такие компьютеры называются серверами, а локальная сеть – сетью на основе сервера.

Сегодня компьютерные сети занимают важную часть не только в личной жизни каждого, но и в бизнесе, медицине и других отраслях. Эффективность сетей зависит не только от их правильного построения, но и от правильной эксплуатации и грамотного сервиса. Таким образом, при достаточно больших сетях возникает необходимость их администрирования и обслуживания. Существует масса программных средств для тестирования и диагностики компьютерных сетей. Всех их объединяет одна задача: предоставить необходимую информацию в кратчайшие сроки в максимально удобном виде.

Существуют различные топологии сетей, у каждой из которых есть свои особенности. Задачей данного дипломного проекта является проектирование и разработка гибкого программного обеспечения для сканирования компьютерной сети и визуализации её топологии, а также проведения базовой диагностики, результаты которой будут показывать задержки доставки пакетов. Весомым плюсом данного решения будет кроссплатформенность. Программу можно будет запустить как на персональном компьютере под управлением операционных систем Windows или Mac OS X, так и на мобильных устройствах под управлением iOs или Android. В данном случае кроссплатформенность приложения играет большую роль в удобстве использования, учитывая, что на сегодняшний день популярность мобильных устройств и планшетных компьютеров растет с каждой секундой.

# **1** ОБЗОР ЛИТЕРАТУРЫ

Разработка системы построения топологии сети и ее диагностики, а также кроссплатформенную реализацию приложения потребовала изучения и применения нижеприведенных информационных технологий.

К настоящему моменту наиболее популярными объектно-ориентированными языками программирования являются C#, Java, C++ и Objective-C. Учитывая преимущества и недостатки каждого из них, был сделан выбор в пользу языка C# и платформы Microsoft .NET Framework.

Microsoft .NET Framework – программная технология, предназначенная для создания как обычных программ, так и веб-приложений. Фактически представляет собой операционную систему внутри операционной системы, высокопроизводительную, основанную на стандартах, многоязыковую среду, которая позволяет интегрировать существующие приложения с приложениями и сервисами следующего поколения, а также решать задачи развертывания и использования интернет-приложений. .NET Framework состоит из трех основных частей - общеязыковой среды выполнения (Common Language Runtime), иерархического множества унифицированных библиотек классов и компонентную версию ASP, называемую ASP.NET.. Основой платформы является [виртуальная машина](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) [Common Language Runtime (CLR)](http://ru.wikipedia.org/wiki/Common_Language_Runtime), способная выполнять как обычные настольные программы, так и веб-приложения. Одной из основных идей Microsoft .NET является совместимость различных служб, написанных на разных языках. Например, служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi; на C# можно написать класс, наследованный от класса, написанного на Visual Basic .NET, а исключение, созданное методом, написанным на C#, может быть перехвачено и обработано в Delphi. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок. Так же как и технология Java, среда разработки .NET создаёт байт-код, предназначенный для исполнения виртуальной машиной. Входной язык этой машины в .NET называется MSIL (Microsoft Intermediate Language), или CIL (Common Intermediate Language, более поздний вариант), или просто IL. Применение байт-кода позволяет получить кроссплатформенность на уровне скомпилированного проекта (в терминах .NET – сборка), а не только на уровне исходного текста, как, например, в С. Перед запуском сборки в среде исполнения CLR, байт-код преобразуется встроенным в среду JIT-компилятором («just in time» ­– компиляция на лету, англ.) в машинные коды целевого процессора. Также существует возможность скомпилировать сборку в родной код для выбранной платформы.

Описание базовых принципов функционирования платформы .NET, системы типов .NET и различных инструментальных средств разработки, используемых при создании приложений, базовые возможности языка программирования С#, включая новые синтаксические конструкции, появившиеся с выходом .NET 4.0, а также синтаксис и семантика языка CIL подробно описаны в [1]. Также там рассматривается формат сборок .NET, библиотеки базовых классов .NET файловый ввод-вывод, возможности удаленного доступа, конструкция приложений Windows Forms, доступ к базам данных с помощью ADO.NET. Там описаны основные объекты модели ADO.NET и рассказано как сохранять, искать, фильтровать и сортировать данные с использованием этих объектов. Книга была очень полезна, так как организована как справочное пособие и позволяет быстро найти ответы на интересующие вопросы. Здесь рассматриваются как подсоединенные, так и отсоединенные объекты, в том числе DataAdapter.DataSet, Connection, Command, Transaction, DataReader и др. Отдельный раздел книги посвящен созданию эффективных Windows- и Web-приложений для доступа к БД с использованием ADO.NET.

Одним из лучших практических руководств по программированию можно назвать «Совершенный код». Оттуда почерпнуты эффективные методики и принципы программирования, которые были использованы при разработке.

Более детальная информация с рекомендациями по использованию конкретных классов приведена в [2]. В ней рассматривается подробное описание внутреннего устройства и функционирования общеязыковой исполняющей среды (CLR) Microsoft .NET Framework версии 2.0, система типов .NET Framework и разъяснены способы управления ими. Представлены концепции программирования с широким использованием библиотеки FCI, относящиеся ко всем языкам, ориентированным на работу с .NET Framework. Особое внимание уделено обобщениям, управлению асинхронными операциями и синхронизации потоков.

В процессе написания любого программного продукта приходится неоднократно сталкиваться с вопросами использования тех или иных функций. В этом случае незаменима электронная документация к .NET Framework [3], предлагаемая разработчиками компании Microsoft и входящая в пакет Visual Studio. Здесь приведены сведения об использовании классов системных библиотек и поведении конкретных методов.

Наравне с документацией [3], использовалась спецификация языка программирования C# [4], которая по праву считается первоисточником для изучения языка С# и платформы .NET Framework.

Так как для разрабатываемого программного обеспечения важна кроссплатформенность, мой выбор остановился на платформе Mono. Это проект по созданию полноценного воплощения системы .Net на базе свободного программного обеспечения. За подробным описанием проекта можно обратится на официальный сайт [5]. В основе проекта лежат спецификации на С# и CLI, переданные Microsoft в организации по стандартизации. Плюс его в том, что почти вся документация и литература, написанная под .Net Framework, прекрасно подходит для изучения этой платформы, в этом можно удостоверится здесь [6]. Проект Mono разрабатывается довольно долгое время — с 2001 года, но до сих пор окончательно не ясен правовой статус этого программного продукта. В 2003 г. корпорация Microsoft получила патент США 20030028685, фактически охватывающий весь каркас .NET Framework. И с тех пор ситуация с патентами на С# и CLI остается неоднозначной. Эта причина сдерживает широкое и повсеместное использование платформы. Но в последнее время она обретает популярность и все больше разработчиков свободного программного обеспечения выбирают именно эту платформу. В [6] можно прочитать о том, что вместо использования виртуальной машины, среда исполнения может быть интегрирована в приложение с целью создания встраиваемой системы, не требующей установки Mono для запуска и обладающей высоким быстродействием. При этом, в приложение включаются только необходимые компоненты среды исполнения. В документации [7] к Mono Touch (платформа для iOs), описана реализация Ahead-Of-Time (AOT) компилятора промежуточного языка. Цель этого компилятора в том, чтобы уменьшить время запуска программы и оптимизировать ее код, путем компиляции промежуточного языка CLI в машинный код. Аналогом в Microsoft .NET Framework является утилита Ngen. Этот метод компиляции будет использован мною для сборки проекта для операционной системы iOs.

Для визуализации топологии сети было рассмотрено множество конкурирующих между собой 3D движков. Но мой выбор был сделан в пользу Unity3D. В [8] подробно описаны возможности данного продукта. Среди факторов, повлиявших на мой выбор, была поддержка сети, кроссплатформенность, быстрая работа на мобильных устройствах, возможность использования C# и Mono библиотек, а также удобство работы в редакторе сцен.

Также я обратил внимание на информацию из [8], в которой говорится о языках программирования, которые использует Unity3D. Платформа поддерживает три языка скриптования: JavaScript, C# и диалект Python'а под названием Boo. Все три одинаково быстры и взаимно совместимы. Все три могут использовать лежащие в основе .NET библиотеки, которые поддерживают базы данных, регулярные выражения, XML, доступ к файлам и работу с сетью. О скриптах часто думают, как о чем то ограниченном и медленном. А в Unity скрипты компилируются в машинный код и работают почти так же быстро, как C++. И в итоге мы получаем быстрое время интеграции и простоту использования.

Современные мобильные устройства предоставляют пользователю совершенно разные способы ввода данных и методы управления: некоторые из них имеют собственную клавиатуру, некоторые – небольшое количество управляющих клавиш и сенсорный экран, взаимодействие с которым происходит при помощи стилуса. Но самые современные представители дают пользователю возможность управлять мобильным устройством посредством сенсорного экрана без использования клавиш либо стилуса – этот способ управления позволяет пользователю быстро и легко производить достаточно сложные операции. Так же, у многих мобильных устройств имеются встроенные фотокамеры, которые позволяют получать достаточно четкие и насыщенные изображения. Именно поэтому, для реализации клиентской части системы поиска изображений по содержимому была выбрана платформа iOS. Источник [9] позволил ознакомиться с основными возможностями мобильного устройства iPhone/iPad.

В iPhone используется multi-touch дисплей, что означает поддержку одновременно нескольких нажатий. Например, для того, чтобы увеличить или уменьшить фотографию, веб-сайт, карту местности или что-то еще, достаточно положить два пальца на дисплей, после чего приблизить или удалить их. В iPhone внедрены революционные технологии, что обеспечивает очень большую точность нажатия. Кроме этого, при печати текста на виртуальной клавиатуре происходит автоматическая коррекция ошибок-промахов, которые могут случаться довольно часто из-того, что палец покрывает сразу несколько виртуальных кнопок. Данное мобильное устройство поддерживает стандарты GSM 850, 900, 1800, 1900, а так же Bluetooth 2.0 EDR, Wi-Fi и EDGE. Это позволяет иметь доступ интернет как при помощи мобильного оператора, так и при помощи Wi-Fi точки доступа.

Во время работы над интерфейсом приложения для получения достоверных данных использовалась литература, предоставленная создателями платформы iPhone OS. Большая ее часть еще не переведена на русский язык, но это не помешало получению нужной информации. Наличие такой документации в сети интернет весьма упростило ознакомление с платформой. Распространение такой литературы выгодно как разработчикам, так и создателям данной платформы: это снижает уровень сложности вхождения в разработку программных средств и увеличивает популярность платформы как среди программистов, так и среди пользователей, т.к. с увеличением количества программистов, использующих эту платформу в качестве рабочего инструмента, увеличивается и количество приложений, которые могут пользоваться спросом среди пользователей. Исключительная особенность данной платформы заключается в том, что все манипуляции с интерфейсом приложений происходят с использованием технологии Multi-Touch, разработка которой началась в 1982 году в Торонтском университете.

Основным источником информации о интерфейсе iPhone-приложений было руководство о проектировании интерфейса приложений, разрабатываемых для iOS [9]. Из него были почерпнуты как начальные, так и расширенные знания о методах функционирования платформы, организации операционной системы и принципах построения интерфейса приложения. В данном руководстве представлено большое количество примеров организации интерфейса приложений, исчерпывающая информация о целях такой организации и ее обоснование.

Так же, во время подготовки к работе мною активно изучалось руководство по проектированию интерфейса и логики приложений, ориентированных на работу в Internet в iOS [10]. Это руководство является незаменимым источником информации по данному вопросу. Из него были получены основы и правила построения интернет-приложений для платформы iPhone, которые имеют ряд отличительных признаков и параметров.

Во время изучения языка Objective-C большое количество информации было получено из [11]. Там было найдено большое количество примеров и советов, которые позволили легко и быстро разобраться в основах использования среды разработки, синтаксисе языка и его специфических свойствах. Т.к. в языке Objective-C используются парадигмы объектно-ориентированного программирование, ознакомление с ним не заняло большого количества времени. Синтаксис языка С-подобный, что упростило его изучение, т.к. полученные в университете знания включают в себя как парадигмы объектно-ориентированного программирования, так и знание языка С.

При проектировании архитектуры многие шаблоны были подчерпнуты из [12]. Основным паттерном является Model-view-controller (MVC, «Модель-представление-поведение», «Модель-представление-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области. Основная цель применения этой концепции состоит в разделении бизнес-логики (модели) от ее визуализации (представления, вида). За счет такого разделения повышается возможность повторного использования. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. Для реализации схемы Model-View-Controller используется достаточно большое число шаблонов проектирования (в зависимости от сложности архитектурного решения), основные из которых Наблюдатель, Стратегия, Компоновщик, Фабрика, о которых подробно написано в [12].

На момент проектирования необходимо было получить знания о базовых методах диагностики сети. В [16] подробно описывается эффективная стратегия «снизу вверх», основанная на изоляции проблем внутри каждого из семи уровней модели OSI. Проанализировав применение сетевых анализаторов для измерения пропускной способности и задержек, выявления `узких мест`, а также исследования времени отклика серверов и клиентов, были сформированы основные требования к подходу диагностики. Кроме того, были получены полезные советы, диаграммы, примеры и файлы отслеживания работы сети, иллюстрирующие наиболее важные концепции и методы анализа сетей. В [16] представлено описание: кабельных сетей и методов их тестирования; технологий кодирования пересылаемых данных; влияния различных типов широковещательного трафика; работы и методов анализа коммутаторов; методов анализа сетей Ethernet и Token Ring; датаграмм и маршрутизации; особенностей протокола IP, включая адресацию, выделение подсетей и использование пакетов ICMP; методов анализа протокола IPX; методов анализа протоколов UDP, TCP, SPX и SPX II; способов поиска ресурсов с помощью NetBIOS, SAP и DNS; регистрации пользователей через различные стеки протоколов; принципов работы протоколов DHCP, SMB, NCP, FTP, HTTP и MS Browse; методов измерения пропускной способности и задержки, включая «клин» запаздывания.

Эндрю Таненбаум в [17] подробно и на примерах описал основные принципы работы современных компьютерных сетей и технологий. Он последовательно изложил основные концепции, определяющие современное состояние и тенденции развития компьютерных сетей. Автор подробнейшим образом объясняет устройство и принципы работы аппаратного и программного обеспечения, рассматривает все аспекты и уровни организации сетей - от физического до уровня прикладных программ. Изложение теоретических принципов дополняется яркими, показательными примерами функционирования Интернета и компьютерных сетей различного типа.

# **2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Целью данной дипломной работы является разработка программного средства сканирования локальной сети, построения ее топологии, а также проведения базовой диагностики.

В результате анализа задачи дипломного проекта были определены основные цели разработки программного средства:

* разработать приложение, позволяющее пользователю произвести сканирование локальной сети и произвести базовую диагностику;
* предоставить пользователю возможность задавать параметры сканирования для своей сети;
* обеспечить удобный и понятный интерфейс пользователя;
* разработать модуль, отвечающий за сетевую часть программного средства;
* предоставить полную и понятную документацию об архитектуре и возможностях, а также руководство пользователя для проекта.

Разрабатываемое программное средство должно поддерживать следующие основные функции:

* ввод пользовательских данных о сканируемой сети, а также выбор режима работы;
* сканирование хостов сети по заданному алгоритму;
* базовая диагностика;
* построение топологии сети;
* отображение результатов в трёхмерном виде.

## **2.1** Структура программного средства для сканирования сети

В процессе проектирования при решении всех требуемых задач проект был разделен на две относительно большие части, взаимодействующие между собой и выполняющие свои особые функции (рисунок 2.1).

Рассмотрим структурные модули:

1. Модуль организации пользовательского интерфейса. При помощи этого модуля пользователь может задать режимы сканирования, задать сканируемый диапазон, а также указать дополнительные опции. Область ответственности этого модуля - обработка ввода информации от пользователя, а также формирование нужных запросов сетевым модулям.



Рис. 2.1 - Модули приложения

Обобщенная структурная схема программного средства изображена на рисунке 2.2.



Рис. 2.2. Обобщенная структурная схема

1. Фреймворк nGUI от компании Tasharen Entertainment используется для отображения элементов пользовательского интерфейса (кнопки, списки, текст и т.д.).
2. Модуль сетевой диагностики. Эта часть программного средства отвечает за проведение базовой диагностики сети, которая доступна только после сканирования сети на предмет хостов. Результаты диагностики будут выведены в виде цветных связей, где каждый цвет будет означать скорость канала. Диагностику можно проводить несколько раз для получения более точных результатов. Её принцип основан на анализе задержек доставки пакетов, описанный в [16].
3. Модуль работы с ARP-протоколом. Данный компонент входит в модуль сетевого взаимодействия и является главным компонентом при сканировании сети на предмет обнаружения хостов. Был выбран именно ARP-протокол вместо простого ICMP потому, что на многих компьютерах установленные брандмауэры могут блокировать ICMP-трафик по желанию пользователя. В то время ARP как протокол более низкого уровня повышает вероятность обнаружения компьютера в сети [17].
4. Модуль сетевой трассировки. Данный компонент также входит в модуль сетевого взаимодействия. Он сканирует хосты за маршрутизатором текущей подсети. Пользователь может включить эту опцию при сканировании.
5. Модуль сериализации и десериализации данных. Он представлен библиотекой JsonFX. Компактная и функциональная библиотека для сериализации данных в формат json. Используется для передачи данных из сетевого модуля в модуль пользовательского интерфейса.
6. Модуль построения графов. Область ответственности – просчет координат расположения элементов сети. Анализирует количество хостов и на основе этой информации строит граф.

## **2.2** Средства разработки

Выбор основных технологий является ключевым моментом при подготовке к созданию программного обеспечения, ведь правильный выбор позволит не только упростить процесс проектирования и разработки, но и облегчить эксплуатацию программного продукта.

Рассматривая современные средства разработки приложений, можно заметить, что их развитие подчиняется определенным закономерностям и тенденциям [12].

Первой из таких тенденций является использование в программировании объектно-ориентированного подхода, позволившее осуществлять повторное использование кода с большей эффективностью, нежели при применении так называемого модульного программирования. Зачатки объектного подхода появились во многих популярных в 80-х годах средствах разработки, таких, например, как Clipper. Однако расцвет объектно-ориентированного программирования начался с появления языка С++.

Следующей немаловажной тенденцией развития средств разработки являлось создание высокопроизводительных компиляторов и стремление использовать скомпилированный код. Известно, что последний обладает существенно более высокой производительностью, чем код интерпретируемый.

Третьей тенденцией развития инструментальных средств являлось создание визуальных средств проектирования пользовательских интерфейсов, что позволило ускорить работу над проектами, облегчить повторное использование кода и в определенной степени привлечь к созданию приложений начинающих программистов. Наиболее ярким примером такого средства явилось появление в середине 90-х годов Visual Basic, имеющего в своем составе элементы VBX, из которых можно было строить интерфейс приложения, просто размещая их на форме, а так же различных средств редактирования ресурсов типа Borland Resorse Workshop. Отметим, однако, что в случае Visual Basic пользователь вынужден был довольствоваться готовыми VBX-элементами, либо создавать их на языке С при помощью других средств разработки.

И, наконец, еще одним немаловажным фактором развития инструментальных средств явилась необходимость масштабируемой поддержки баз данных, так как, во-первых, именно информационные системы стали наиболее часто встречающимся типом разрабатываемых приложений и, во-вторых, именно в конце 90-х годов начался массовый переход от настольных СУБД к архитектуре клиент/сервер. Отметим, однако, что далеко не все средства разработки одинаково хорошо поддерживают все СУБД – нередко имеется явная ориентация на поддержку SQL-сервера того же производителя, что и производитель средства разработки (типичный пример – средства разработки Oracle).

Следует отметить, что Microsoft Visual Studio.NET 2010 представляет собой следствие влияния всех этих тенденций, так как сочетает в себе удобства визуальной среды разработки, объектно-ориентированный подход, разнообразие возможностей повторного использования кода, открытую архитектуру и высокопроизводительный компилятор. А так же масштабируемый доступ к данным, хранящимся в различных СУБД, как настольных, так и серверных.

Microsoft Visual Studio.NET 2010 стал одними из самых популярных на сегодняшний день инструментов для создания как настольных, так и корпоративных информационных систем благодаря уникальному сочетанию удобства разработки пользовательских интерфейсов, компонентной архитектуры, однотипности доступа к разнообразным базам данных, начиная от плоских таблиц и кончая серверными СУБД. Во многом именно наличие таких продуктов стимулировало, достаточно безболезненный, перенос в архитектуру клиент/сервер ряда информационных систем, модернизация которых иными средствами была бы сопряжена с большими трудовыми и материальными затратами.

Платформа .NET поддерживает межъязыковое взаимодействие — это способность C++/CLI работать в тесном сотрудничестве с программами на C#, Visual Basic и других языках, которые компилируются в CLR. Из-за общей базовой платформы, общего IL и общих форматов сборки и метаданных в значительной степени можно использовать сборку C# или Visual Basic так, как другую сборку C++/CLI. Можно сослаться на нее с помощью #using, можно создать экземпляры типов, объявленных в таких сборках, вызывать их методы и так далее. А можно пойти на шаг дальше и создать иерархии наследственности, которые стирают межъязыковые границы, такие как C#-класс, который реализует C++/CLI-интерфейс, или C++/CLI-класс, который является наследником класса, написанного в Visual Basic. Как только эти типы скомпилированы в MSIL, совсем немного указывает на исходный язык, на котором они были созданы.

Следует отметить, что эффективность разработки и отладки приложений достигается не только за счет использования удобных средств визуального проектирования форм (сейчас это не редкость), но и за счет, во-первых, высокой производительности самих компиляторов и, во-вторых, так называемой инкрементной компиляции и компоновки исполняемого модуля (когда перекомпиляции и перекомпоновке подвергаются только те модули, в которые были внесены изменения).

Так как приложение задумано как кроссплатформенное, то написание некоторых участков кода планируется на нативном языке для мобильных устройств на базе операционной системы iOS.

Операционная система iOS 5.1, под которую планируется разрабатывать iPhone приложение, поддерживает различные языки программирования, но компания Apple, создавшая iPhone, настоятельно рекомендует использовать для разработки приложений язык программирования Objective-C. Именно поэтому, для разработки некоторых частей целесообразно использовать именно его.

Язык программирования Objective-C, полностью совместим с языком С и обладает очень мощной и выразительной объектной моделью, заимствованной из языка Smalltalk.

В Objective-C сами классы являются полноценными объектами, поддерживается интроспекция и динамическая обработка сообщений [11]. Язык Objective-C крайне прост - его освоение у С-программиста займет всего несколько дней. Он является именно расширением языка С - в язык С добавлены новые возможности для объектно-ориентированного программирования. При этом любая программа на С является программой и на Objective-C, что не верно для С++.

Еще одной из особенностей языка является то, что он message-oriented [11] в то время как С++ - function-oriented. Это значит, что в нем вызовы метода интерпретируются не как вызов функции, а как посылка сообщения объекту, подобно тому, как это происходит в Smalltalk-е. Такой подход дает целый ряд плюсов - так любому объекту можно послать любое сообщение. Объект может вместо обработки сообщения делегировать его, т.е. переслать его другому объекту для обработки. Именно так можно легко реализовать распределенные объекты – т.е. объекты находящиеся в различных адресных пространствах и даже на разных компьютерах. Привязка сообщения к соответствующей функции происходит непосредственно на этапе выполнения.

Язык Objective-C поддерживает работу с метаинформацией - так у объекта непосредственно на этапе выполнения можно спросить его класс, список методов (с типами передаваемых аргументов) и instance-переменных, проверить, является ли класс потомком заданного и поддерживает ли он заданный протокол и т.п.

В языке есть нормальная поддержка протоколов. Для объектов поддерживается наследование, для протоколов поддерживается множественное наследование. Объект может быть унаследован от другого объекта и поддерживать сразу несколько протоколов.

На данный момент язык Objective-C поддерживается компилятором gcc. Довольно много в языке перенесено на runtime-библиотеку и сильно зависит от нее. Вместе с компилятором gcc поставляется минимальный вариант такой библиотеки.

# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробно функционирование программы. Для этого проведем анализ основных модулей программы и рассмотрим их зависимости. А также проанализируем все классы, которые входят в состав кода программы, и рассмотрим назначение всех методов, свойств и переменных класса.

## **3.1** Выбор протокола для сканирования

Основная задача данного программного средства сводится к получению информации об устройствах, находящихся в сети. Необходимо было выбрать протокол для межсетевого взаимодействия, который бы отвечал всем заявленным требованиям. При проектировании были рассмотрены два официальных протокола модели TCP/IP-ARP (Address Resolution Protocol) и ICMP (Internet Control Message Protocol), которые будучи внимательно изучены, могут использоваться для достижения целей при написании данного программного средства. В качестве основного протокола для получения информации об устройствах в сети был выбран ARP-протокол.

Преимущества над ICMP-протоколом:

* на любой машине в сети ICMP-пакеты могут быть добавлены в исключения в файрволе. В такой ситуации устройство просто не ответит пакетом на эхо-запрос, следовательно, обнаружение устройств в сети становится не самым надежным. Администраторы локальных сетей стараются не блокировать ARP-пакеты для правильного функционирования сети;
* заголовок ARP-пакета содержит в себе МАС-адрес откликнувшегося устройства. При использовании ICMP-протокола не возможно напрямую получить МАС-адрес устройства;
* ICMP-протокол блокируется брандмауэром Windows Vista и Windows Server 2008, а также некоторыми маршрутизаторами и функционально независимыми брандмауэрами по умолчанию. Чего нельзя сказать про ARP-протокол.

ARP (*Address Resolution Protocol*) — протокол канального уровня, предназначенный для определения MAC-адреса по известному IP-адресу [17]. Наибольшее распространение этот протокол получил благодаря повсеместности сетей  IP, построенных поверх  Ethernet, поскольку практически в 100 % случаев при таком сочетании используется ARP. Структура заголовка пакета представлена на рисунке 3.1.

Принцип работы протокола заключается в следующем:

1. Узел, которому нужно выполнить отображение IP-адреса на локальный адрес, формирует ARP запрос, вкладывает его в кадр протокола канального уровня, указывая в нем известный IP-адрес, и рассылает запрос широковещательно.
2. Все узлы локальной сети получают ARP запрос и сравнивают указанный там IP-адрес с собственным.
3. В случае их совпадения узел формирует ARP-ответ, в котором указывает свой IP-адрес и свой локальный адрес и отправляет его уже направленно, так как в ARP запросе отправитель указывает свой локальный адрес.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **+** | **Bits 0 - 7** | **8 - 15** | **16 - 31** | | **0** | Hardware  type (HTYPE) | | Protocol  type (PTYPE) | | **32** | Hardware length (HLEN) | Protocol length (PLEN) | Operation (OPER) | | **64** | Sender hardware address (SHA) | | | | **?** | Sender protocol address (SPA) | | | | **?** | Target hardware address (THA) | | | | **?** | Target protocol address (TPA) | | | |

Рисунок 3.1 – структура ARP-пакета.

Преобразование адресов выполняется путем поиска в таблице. Эта таблица, называемая ARP-таблицей, хранится в памяти и содержит строки для каждого узла сети. В двух столбцах содержатся IP- и Ethernet-адреса. Если требуется преобразовать IP-адрес в Ethernet-адрес, то ищется запись с соответствующим IP-адресом.

Структура пакета по полям:

1. Hardware type (HTYPE). Каждый транспортный протокол передачи данных имеет свой номер, который хранится в этом поле. Например, Ethernet имеет номер 0x0001.
2. Protocol type (PTYPE). Код протокола. Например, для IPv4 будет записано 0x0800.
3. Hardware length (HLEN). Длина физического адреса в байтах. Ethernet адреса имеют длину 6 байт.
4. Protocol length (PLEN). Длина логического адреса в байтах. IPv4 адреса имеют длину 4 байта.
5. Operation. Код операции отправителя: 1 в случае запроса и 2 в случае ответа.
6. Sender hardware address (SHA). Физический адрес отправителя.
7. Sender protocol address (SPA). Логический адрес отправителя.
8. Target hardware address (THA). Физический адрес получателя. Поле пусто при запросе.
9. Target protocol address (TPA). Логический адрес получателя.

Стоит отметить, что для сканирования диапазона IP-адресов, был использован не широковещательный метод рассылки пакетов, а направленный, так как в данном случае в широковещательном запросе нет смысла.

## **3.2** Алгоритмы сканирования и трассировки

*3.2.1 ARP-сканирования сети*

Алгоритм ARP-сканирования сети заключается в следующем:

1. Пользователь задает диапазон сканируемых IP-адресов и маску подсети.
2. Создается пул потоков для выполнения параллельной рассылки ARP-пакетов на разные адреса.
3. Происходит инициализация пула потоков.
4. Запуск одновременно нескольких потоков из пула для параллельной рассылки и ожидания ответа. При этом для каждого запроса происходит сборка ARP-пакета.
5. Если удаленный компьютер ответил на ARP-запрос, то происходит анализ ARP-ответа. Информация об IP- и MAC-адресах добавляется в коллекцию.
6. Ожидание завершения всех потоков из пула.
7. Передача полученной информации на уровень пользовательского интерфейса.

Из плюсов данного алгоритма можно указать параллельное выполнение, что заметно ускоряет время сканирования даже не очень большого диапазона IP-адресов.

*3.2.2 Алгоритм трассировки сети за пределы маршрутизатора*

Для определения промежуточных маршрутизаторов программа отправляет целевому узлу серию ICMP-пакетов (по умолчанию 3 пакета), с каждым шагом увеличивая значение поля TTL («время жизни») на единицу. Это поле обычно указывает максимальное количество маршрутизаторов, которое может быть пройдено пакетом. Первая серия пакетов отправляется с TTL, равным единице, и поэтому первый же маршрутизатор возвращает обратно ICMP-сообщение "Time exceeded in transit", указывающее на невозможность доставки данных. Программа фиксирует адрес маршрутизатора, а также время между отправкой пакета и получением ответа (эти сведения сохраняются и позже сериализуются для передачи на уровень пользовательского интерфейса для построения графа сети). Затем программа повторяет отправку серии пакетов, но уже с TTL равным двум, что заставляет первый маршрутизатор уменьшить TTL пакетов на единицу и направить их ко второму маршрутизатору. Второй маршрутизатор, получив пакеты с TTL равным единице, так же возвращает "Time exceeded in transit". Процесс повторяется до тех пор, пока пакет не достигнет целевого узла или пока не будет достигнута заданная глубина трассировки. При получении ответа от этого узла процесс трассировки считается завершённым. Также процесс может быть окончен при достижении максимального числа узлов, которое задается пользователем.

## **3.3** Классы межсетевого взаимодействия

Для выполнения дипломного проекта была выбрана концепция объектно-ориентированного программирования. Основной единицей данного подхода является *класс*. Ниже приведено описание классов межсетевого взаимодействия.

*3.3.1 Класс NetworkScaner*

Представляет собой класс, реализующий основной функционал при работе с ARP-протоколом, трассировкой сети, отправкой пакетов и прочим сетевым взаимодействием.

Поля:

* traceOut - поле типа bool, содержащее в себе информацию о том, будет ли использован алгоритм сетевой трассировки для обнаружения хостов за пределами маршрутизатора;
* traceHops *–* поле типа int, определяющее глубину сетевой трассировки за пределы маршрутизатора;
* ipRange *–* переменная класса List<IPAddress>, которое содержит в себе информацию о сканируемом диапазоне IP-адресов;
* workerLocker *–* поле типа object, служащее для защиты общих переменных при доступе к ним из разных потоков;
* rununingWorkers *–* поле типа int, определяющее количество одновременно запущенных потоков;
* arpResults *–* переменная класса ARPList, содержащая в себе информацию о результатах ARP-сканирования;
* traceResults *–* переменная класса TraceList, содержащая в себе информацию о результатах трассировки сети за пределы маршрутизатора.

Методы:

* NetworkScaner(string args) – конструктор класса. Параметр args содержит в себе аргументы для сканирования;
* bool InitializeArgs() – метод, анализирующий переданные параметры. Также этот метод на основе полученной информации выставляет требуемые режимы функционирования. Возвращает истинное значение, если все аргументы были правильно заданы и инициализированы, и ложное в противном случае;
* void InitializeThreading(int maxThreadCount) – метод, инициализирующий пул потоков. Аргумент maxThreadCount используется для указания максимально доступного для использования количества потоков;
* void Scan() – метод, запускающий сканирование локальной сети с проиниализированными заранее параметрами. Если параметры заранее не установлены, будет произведена попытка их инициализации. При неудаче

будет сгенерировано исключение;

* void SendArpPackage(ARPPackage package) – метод, отправляющий заранее сформированный ARP-пакет на указанный в нем IP-адрес. Параметр package хранит в себе информацию о собранном ARP-пакете;
* ARPPackage ARPPackageAssembler(IPAddress ipAddr) – метод, собирающий ARP-пакет с указанным в параметре ipAddr IP-адресом. При этом инициализируются только необходимые поля заголовка ARP-пакета;
* void TraceRoute(int maxHops) – метод, реализовывающий трассировку сети за пределы маршрутизатора. Параметр maxHops отвечает за глубину сканирования;
* void SaveData() – метод, сохраняющий данные сканирования сети при помощи класса DataSerializer.

*3.3.2 Класс GraphBuilder*

Представляет собой класс, реализующий компонент, строящий графы локальной сети по заданным данным. Анализируя предоставленную информацию, объект данного класса рассчитывает количество подсетей и их размеры. После чего расставляет устройства, помечая их IP- и MAC-адресом, и объединяет в общую топологию. Если опция трассировки сети включена, то GraphBuilder располагает маршрутизаторы провайдеров с заданной глубиной.

Поля:

* gatewayPosition - поле типа Vector3, содержащее в себе информацию о расположении маршрутизатора по умолчанию. Значение этого поля вычисляется исходя из размера самой большой подсети, а также от взаимного расположения других подсетей;
* subNetworks – поле (коллекция-обобщение) типа Dictionary<string, ARPList>, содержащее информацию о подсетях в отсортированном и сгруппированном виде. При анализе списка устройств для нанесения на карту сети учитывается маска подсети и на её основе заполняется данное поле;
* circleRadiuses – поле (коллекция-обобщение) типа Dictionary<float, Vector3>, содержащее информацию о радиусах окружностей подсетей. Данное поле служит для предотвращения коллизий при построении графов, а также для вычисления gatewayPosition;
* uiManager - поле типа UIManager, содержащее в себе ссылку на объект менеджера пользовательского интерфейса. В данном классе используется для отправления сообщений уровню пользовательского интерфейса;
* camera - поле типа Transform, содержащее в себе ссылку на главную камеру. Используется для перемещения положения камеры при построении графа;
* earthModel - поле типа GameObject, содержащее в себе ссылку на 3д модель земного шара. Эта модель обозначает конечную точку трассировки сети - Интернет;
* routerModels - поле типа GameObject[], содержащее в себе массив ссылок на 3д модели маршрутизаторов разных типов;
* colors - поле типа Color[], содержащее в себе массив различных цветов, для раскрашивания графа;
* lineMaterial - поле типа Material, содержащее в себе ссылку на материал для отрисовки линий связей в графе;
* circleOffset - поле типа float, содержащее информацию о расстоянии между двумя графами-окружностями. Задается на стадии проектирования разработчиком;
* elementSize - поле типа float, содержащее информацию о площади, занимаемой одним элементом графа. Задается на стадии проектирования разработчиком;
* computerModel - поле типа GameObject, содержащее в себе ссылку на 3д модель компьютера, являющимся основным узлом графа сети.

Методы:

* void GraphBuilder() – конструктор класса;
* void Initialize() – метод, инициализирующий построитель графов. Устанавливает необходимые параметры, а также отсылает уровню пользовательского интерфейса сообщение о начале сканирования;
* void BuildNetwork() – метод, последовательно запускающий корутины по построению графов сети на основе ARP-сканирования и трассировки сети, а также расстановки главного маршрутизатора и прочих элементов;
* void OnApplicationQuit() – обработчик события класса Monobehavour, отрабатывающее при выходе из приложения. Служит для правильного завершения процесса построения;
* void BuildArpThree() – метод, строящий граф сети, который отображает её топологию. В качестве информации для построения служат полученные данные на этапе ARP-сканирования;
* void BuildTraceThree() – метод, строящий граф сети, который отображает маршрутизаторы за пределами подсети. В качестве данных для построения служит полученная информация на этапе трассировки сети;
* void PlaceMainRouter() – метод, располагающий на карте сети главный маршрутизатор (Gateway);
* void AnalyzeARPData(ARPList arpList) – метод, анализирующий ARP данные о хостах и выделяющий на основе маски подсети необходимое количество графов для построения. Параметр arpList содержит информацию о просканированной сети;
* void BuildArpCircle(ARPList nodes, Vector3 center, float radius, string key) – метод, расставляющий устройства из списка nodes в виде окружности с центром в center, радиусом radius и ключом key;
* ARPList ARPDataReader() – метод, читающий информацию о просканированных хостах с диска и возвращающий её в виде ARPList;
* TraceList TraceDataReader() – метод, читающий информацию о просканированных при помощи трассировки маршрутизаторах с диска и возвращающий её в виде TraceList.

*3.3.3 Класс UIManager*

Представляет собой класс, реализующий менеджер слоя пользовательского интерфейса программы. В его задачи входит обновление элементов по событиям, генерация собственных событий, обработка и валидация введенной пользователем информации. Он является ключевым классом при общении с пользователем.

Поля:

* ipAddr01 - поле типа UILabel, содержащее в себе ссылку на вводимый пользователем первый из диапазона IP-адрес;
* ipAddr02 - поле типа UILabel, содержащее в себе ссылку на вводимый пользователем второй из диапазона IP-адрес;
* subnetMask - поле типа UIPopupList, содержащее в себе ссылку на выпадающий список для выбора маски подсети;
* maxHops - поле типа UIPopupList, содержащее в себе ссылку на выпадающий список для выбора количества маршрутизаторов при трассировке сети;
* trace - поле типа UICheckbox, содержащее в себе ссылку на элемент чекбокса для выбора опции трассировки;
* panel01 - поле типа GameObject, содержащее в себе ссылку на первую панель элементов пользовательского интерфейса;
* panel02 - поле типа GameObject, содержащее в себе ссылку на вторую панель элементов пользовательского интерфейса;
* procStarter - поле типа ProcessStarter, содержащее в себе ссылку на объект, запускающий процесс сканирования.

Методы:

* void UIManager – конструктор класса.
* void Scan() – метод-обработчик, запускающий сканирование сети;
* void OnIpChanged01() – метод, заполняющий первый IP-адрес из диапазона;
* void OnIpChanged02() – метод, заполняющий второй IP-адрес из диапазона;
* void OnMaskChanged() – метод, заполняющий маску подсети;
* void OnHopsChanged() – метод, заполняющий максимальное количество проходимых маршрутизаторов при трассировке;
* void OnTraceChecked() – метод, устанавливающий опцию трассировки сети.

*3.3.4 Класс Context*

Представляет собой статический класс, реализующий контекст приложения, в котором хранится вся необходимая информация. Любой объект в любое время может получить доступ к этой информации. Класс хранит только общедоступную информацию.

Поля:

* ipAddr01 – статическое поле типа string, содержащее в себе первый из диапазона IP-адрес;
* ipAddr02 – статическое поле типа string, содержащее в себе второй из диапазона IP-адрес;
* subnetMask - статическое поле типа string, содержащее в себе маску подсети сканируемого диапазона;
* maxHops - статическое поле типа int, содержащее в себе максимальное количество маршрутизаторов при трассировке сети;
* trace - поле типа bool, содержащее в себе выбор опции трассировки.

Методы:

* void OnApplicationQuit() – обработчик события класса Monobehavour, отрабатывающее при выходе из приложения. Служит для сохранения контекста при выходе из приложения;

*3.3.5 Класс ProcessStarter*

Представляет собой класс, реализующий менеджер запуска процесса сканирования. В его задачи входит подготовка аргументов, создание нового процесса и ожидание его завершения.

Поля:

* graphBuilder - поле типа GraphBuilder, содержащее в себе ссылку на компонент построения графов. По завершении процесса сканирования этому компоненту передается все необходимые данные.

Методы:

* void ProcessStarter() – конструктор класса;
* string ArgsCreator() – метод, анализирующий введенную пользователем информацию и подготавливающий необходимые аргументы для процесса сканирования;
* void Start() – метод, создающий новый процесс сканирования, инициализирующий его всеми необходимыми данными;
* void Update() – метод, дожидающийся окончания процесса и стартующий построение графов сети. Является событием класса Monobehavour.

*3.3.6 Класс IPRange*

Представляет собой класс, анализирующий и конвертирующий строку в специальном формате («x-x.x-x.x-x.x-x», где х – числа от 0 до 255) в диапазон IP-адресов.

Поля:

* beginIp - поле типа byte[], содержащее в себе начальный IP-адрес из диапазона.
* endIp - поле типа byte[], содержащее в себе конечный IP-адрес из диапазона.

Методы:

* void IPRange(string ipRange) – конструктор класса;
* IEnumerable<IPAddress> GetAllIP() – метод, возвращающий коллекцию всех IP-адресов из заданного диапазона;
* bool TryParseSimpleRange(string ipRange) – метод, анализирующий указанный ipRange диапазон. Возвращает истинное значение в случае успеха, в противном случае - ложное;
* bool TryParseCIDRNotationRange(string ipRange) – метод, анализирующий указанный ipRange диапазон, заданный в CIDR (Classless Inter-Domain Routing) нотации. Возвращает истинное значение в случае успеха, в противном случае – ложное.

*3.3.7 Класс Options*

Представляет собой класс, содержащий настройки программы, а также всю логику, необходимую для их изменения пользователем, сохранение в постоянное запоминающее устройство и чтение их оттуда при запуске программы. В случае, если файл с настройками отсутствует или поврежден, имеется возможность заполнить настройки значениями по умолчанию. Класс построен с применением паттерна проектирования «Одиночка». Это означает, что он не может иметь более одной копии объекта. Данный факт позволяет единожды созданный объект настроек использовать все классам, которым необходимо иметь доступ к ним.

Поля:

* static \_OptionsList – статическая переменная, являющаяся уникальным объектом класса Options. Данное поле реализует паттерн проектирования «Одиночка» ;
* TraceOption – поле типа string, хранящее опции трассировки сети;
* RangeOption – поле типа string, хранящее опции диапазона сканируемых IP-адресов;
* TraceHost – поле типа string, хранящее опции хоста трассировки;
* MaskOption – поле типа string, хранящее опции маски подсети.

*3.3.8 Класс CameraController*

Представляет собой класс, управляющий камерой. Основные функции: считывание ввода пользователя, преобразование его в удобный формат, перемещение камеры вперед/назад, поворот камеры, контроль поворота.

Поля:

* lookSpeed – поле типа float, содержащее информацию о скорости поворота камеры;
* moveSpeed – поле типа float, содержащее информацию о скорости поворота камеры;
* rotationX – поле типа float, содержащее информацию о повороте камеры по оси Х;
* rotationY – поле типа float, содержащее информацию о повороте камеры по оси Y.

Методы:

* void CameraController() – конструктор класса;
* void Initialize() – метод, инициализирующий стартовое положение камеры и переменных;
* void Update() – метод, считывающий ввод пользователя, анализирующий его и управляющий камерой на основе полученных данных.

*3.3.9 Класс DataSerializer*

Представляет собой класс, сериализующий информацию в JSON формат и десериализующий из него. Класс статический.

Методы:

* void SerializeARP(ARPList list) – метод, сериализующий list, содержащий в себе результаты ARP-сканирования сети, в JSON-файл;
* void SerializeTrace(TraceList list) – метод, сериализующий list, содержащий в себе результаты трассировки сети, в JSON-файл;
* ARPList DeserializeARP(string path) – метод, десериализующий JSON-файл, находящийся по пути path и содержащий в себе результаты ARP-сканирования сети;
* TraceList DeserializeTrace(string path) – метод, десериализующий JSON-файл, находящийся по пути path и содержащий в себе результаты трассировки сети.

## **3.4** Файлы

Для работы приложение широко используются файлы, для хранения данных в них, преимущественно используется формат JSON, а так же обычный текстовый формат.

*3.4.1 Файлы с результатами ARP-сканирования сети*

Данный тип файлов представляет текстовый файл в формате JSON для хранения информации о результатах сканирования локальной сети. Для чтения и разбора этого типа файлов предназначен класс DataSerializer. Формат файла выглядит следующим образом:

{

"List":

[

{

"IpAddress": "192.168.1.2",

"MacAddress": "74-EA-3A-81-0B-30"

},

{

"IpAddress": "192.168.1.4",

"MacAddress": "00-24-1D-7E-A2-9F"

},

{

"IpAddress": "192.168.1.1",

"MacAddress": "08-18-1A-CE-82-3B"

},

{

"IpAddress": "192.168.1.6",

"MacAddress": "14-5A-05-DE-71-92"

}

]

}

Файл содержит список записей по каждому устройству в сети, которое откликнулось в процессе сканирования. Информация о каждой записи представлена двумя полями: IpAddress и MacAddress. В них находится информация об IP- и MAC-адресах соответственно.

*3.4.2 Файлы с результатами трассировки сети*

Данный тип файлов представляет текстовый файл в формате JSON для хранения информации о результатах трассировки за пределы локальной сети. Для чтения и разбора этого типа файлов предназначен класс DataSerializer. Формат файла выглядит следующим образом:

{

"List":

[

{

"Number": 1,

"ReplyDelay": 239,

"IpAddress": "192.168.1.1"

},

{

"Number": 2,

"ReplyDelay": 115,

"IpAddress": "178.120.0.1"

},

{

"Number": 3,

"ReplyDelay": 217,

"IpAddress": "93.84.80.106"

}

]

}

Файл содержит информацию о маршрутизаторах провайдера. Каждая запись содержит три поля: Number, ReplyDelay и IpAddress. Number – это порядковый номер. Он нужен при нанесении маршрутизаторов на карту сети. Номер увеличивается в сторону хоста, до которого производится трассировка. ReplyDelay содержит данные о времени доставки пакета. А IpAddress – IP-адрес откликнувшегося устройства.

# **4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке программного средства сканирования и трехмерного моделирования локальной сети потребовалась реализация основных алгоритмов работы с сетевыми протоколами, как, например, ARP- и ICMP-протоколы, а также некоторых вспомогательных алгоритмов. Подробнее рассмотрим их ниже.

## **4.1** Алгоритм формирования диапазона IP-адресов

Первым делом необходимо получить список IP-адресов для сканирования. Основная задача этого алгоритма – это преобразование специально подготовленной строки в диапазон адресов. Исходную строку можно представить не только в виде «a-b.c-d.e-f.g-h», где a,b,c,d,e,f,g,h – число в диапазоне от 0 до 255, но и в CIDR-нотации (192.168.0.0/16).

Рассмотрим первый случай. Исходная строка разбивается на подстроки по символу точки.

String[] ipParts = ipRange.Split('.');

Таким образом получаем массив строк вида «X-Y». Далее запускаем цикл от 1 до 4, который будет анализировать наши подстроки.

beginIP = new byte[4];

endIP = new byte[4];

for (int i = 0; i < 4; i++)

{

string[] rangeParts = ipParts[i].Split('-');

if (rangeParts.Length < 1 ||

rangeParts.Length > 2)

return false;

beginIP[i] = byte.Parse(rangeParts[0]);

endIP[i] = (rangeParts.Length == 1) ? beginIP[i] : byte.Parse(rangeParts[1]);

}

В теле цикла производится разбиение строки, только на этот раз по символу '-'. После чего проверяется, чтобы полученный массив строк содержал либо одно, либо два значения. В противном случае алгоритм прерывается. Затем заполняются соответствующие октеты начального и конечного IP- адресов формируемого диапазона.

Теперь рассмотрим случай, когда исходная строка представлена в CIDR-нотации. Первым делом делается разбиение строки по символу обратного слеша '/'.

string[] x = ipRange.Split('/');

if (x.Length != 2)

return false;

Если полученный массив строк содержит число элементов отличное от двух, алгоритм завершает работу, т.к. данный случай является ошибкой ввода исходных данных.

Маска подсети сохраняется:

byte bits = byte.Parse(x[1]);

Далее происходит формирование IP-адреса сети. Происходит разбиение строки по точке. Запускается цикл от одного до четырех, в каждой итерации которого значение заносится в адрес и сдвигается влево на 8 бит.

uint ip = 0;

String[] ipParts0 = x[0].Split('.');

for (int i = 0; i < 4; i++)

{

ip = ip << 8;

ip += uint.Parse(ipParts0[i]);

}

После чего происходит формирование первого адреса из диапазона с учетом её маски, а также проверка на правильную маску. Если IP-адреса совпадают, алгоритм возвращает ложное значение и прекращает выполнение.

byte shiftBits = (byte)(32 - bits);

uint ip1 = (ip >> shiftBits) << shiftBits;

if (ip1 != ip)

return false;

Формирование второго адреса из диапазона представлено циклом, в котором на каждой итерации добавляется значимая часть адреса.

uint ip2 = ip1 >> shiftBits;

for (int k = 0; k < shiftBits; k++)

{

ip2 = (ip2 << 1) + 1;

}

После чего запускается последний цикл, который формирует результирующие значения IP-адресов для диапазона:

beginIP = new byte[4];

endIP = new byte[4];

for (int i = 0; i < 4; i++)

{

beginIP[i] = (byte)((ip1 >> (3 - i) \* 8) & 255);

endIP[i] = (byte)((ip2 >> (3 - i) \* 8) & 255);

}

return true;

В конце всех оперцаций возвращается истинное значение, которое служит маркером успешного преобразования для конструктора.

В самом конце алгоритма запускается четыре вложенных цикла для прохода по всему диапазону и формируется каждый IP-адрес.

int capacity = 1;

for (int i = 0; i < 4; i++)

capacity \*= endIP[i] - beginIP[i] + 1;

List<IPAddress> ips =

new List<IPAddress>(capacity);

for (int i0 = beginIP[0]; i0 <= endIP[0]; i0++)

{

for (int i1 = beginIP[1]; i1 <= endIP[1]; i1++)

{

for (int i2 = beginIP[2]; i2 <= endIP[2]; i2++)

{

for (int i3 = beginIP[3]; i3 <= endIP[3]; i3++)

{

ips.Add(new IPAddress(new byte[] { (byte)i0, (byte)i1, (byte)i2, (byte)i3 }));

}

}

}

}

Таким образом результатом работы данного алгоритма является коллекция IP-адресов. Именно на основе этой коллекции в дальнейшем происходит рассылка ARP-запросов при сканировании локальной сети для обнаружение активных устройств.

## **4.2** Алгоритм работы с ARP-протоколом

Алгоритм ARP-сканирования сети является основным сетевым алгоритмом данного программного средства. Блок-схема алгоритма сканирования сети изображена на рисунке 4.1.

Первым делом происходит инициализация переданных аргументов. Если аргументы пустые, метод возвращает ложное значение и показывает сообщение об ошибке и справку.

if (args.Length == 0)

{

Help();

inited = false;

return;

}

Иначе запускается цикл для анализа введенных аргументов. Проверяются совпадения с ключевыми словами и, если они были найдены, происходит формирование значений по данным ключам и их сохранение.

for (int i = 0; i < args.Length; i++)

{

switch (args[i])

{

case Options.TraceOption:



Рисунок 4.1 – Блок-схема алгоритма сканирования

traceOut = true;

inited = true;

traceHops = int.Parse(args[i + 1]);

i++;

break;

case Options.RangeOption:

range = true;

inited = true;

rangeValue = args[i + 1];

IPRange r = new IPRange(rangeValue);

ipRange = r.GetAllIP();

i++;

break;

case Options.HelpOption:

Help();

inited = false;

return;

}

}

Если все аргументы были переданы верно, то после их обработки наступает этап инициализации пула потоков.

ThreadPool.SetMaxThreads(10, 10);

После чего происходит запуск потоков через пул. Пул потоков сам отслеживает текущее количество запущенных потоков и не дает запускать их больше, чем заданное количество.

runningWorkers = ((List<IPAddress>)ipRange).Count;

int maxWorkers = runningWorkers;

for (int i = 0; i < runningWorkers; i++)

{

ThreadPool.QueueUserWorkItem(ARPSend,

((List<IPAddress>)ipRange)[i]);

}

Затем идет ожидание выполнения всех потоков, после чего данные сканирования сериализуются на диск в виде json-файла. В качестве синхронизации потоков выбран простой locker.

lock (workerLocker)

{

while (runningWorkers > 0)

{

Monitor.Wait(workerLocker);

}

}

WriteDataToDisk();

Рассмотрим посылку и обработку ARP-запросов подробнее.

IPAddress addr = (IPAddress)ip;

byte[] ab = new byte[6];

int len = ab.Length;

int r = SendARP((int)addr.Address, 0, ab, ref len);

Первым делом преобразовываем IP-адрес к нужному типу. После чего формируем аргументы для метода SendARP и посылаем ARP-запрос на указанный адрес. Метод SendARP блокирующий, поэтому текущий поток ждет ответа от удаленной машины. При получении ответа поток продолжает выполнять инструкции. Если удаленное устройство ответило на запрос, то происходит сохранение его IP- и MAC-адресов.

string mac = BitConverter.ToString(ab, 0, 6);

lock (workerLocker)

{

if (mac != MACAddress.Empty)

{

lock (ArpResults)

{

ARPData data = new ARPData { IpAddress =

((IPAddress)ip).ToString(), MacAddress = mac };

ArpResults.List.Add(data);

Monitor.Pulse(ArpResults);

}

}

runningWorkers--;

Monitor.Pulse(workerLocker);

}

В данном случае потокобезопастность достигается при помощи блокировка объектов ArpResults и workerLocker. Можно быть уверенным в том, что все данные будут обработаны и сохранены без потерь.

## **4.3** Алгоритм трассировки сети

Рассмотрим подробнее алгоритм трассировки сети за пределы маршрутизатора. Блок-схема алгоритма изображена на рисунке 4.2.

Первым делом происходит формирование ICMP-пакета размером 32 байта. Пакет должен быть не фрагментируемым. Параметр TTL устанавливаем в единицу.

PingReply pingReply = null;

Ping pingSender = new Ping();

PingOptions pingOptions = new PingOptions();

Stopwatch stopWatch = new Stopwatch();

byte[] bytes = new byte[32];

pingOptions.DontFragment = true;

pingOptions.Ttl = 1;

После всех необходимых инициализаций запускается цикл, который следит за тем, чтобы максимальное число «прыжков» не было превышено. В нем происходит отсылка ICMP-пакетов.

for (int i = 1; i < maxHops + 1; i++)

{

stopWatch.Reset();

stopWatch.Start();

pingReply = pingSender.Send(ipAddressOrHostName, 1000, new byte[32], pingOptions);

stopWatch.Stop();



Рисунок 4.2 – Блок-схема алгоритма трассировки

Метод Send блокирующий, поэтому основной поток дожидается прихода ответа от удаленного устройства. После чего происходит анализ ответа. Если статус ответа не IPStatus.TtlExpired и IPStatus.Success, то фиксируем ошибку. Иначе производится разбор ответа. Если статус пришедшего ответа IPStatus.Success, то происходит завершение цикла, так как алгоритм достиг конечного хоста.

if (pingReply.Status != IPStatus.TtlExpired && pingReply.Status != IPStatus.Success)

{

NetworkError();

}

else

{

IPHostEntry @ipHost = Dns.Resolve(pingReply.Address.ToString());

TraceData data;

if (ipHost.HostName != pingReply.Address.ToString())

{

data = new TraceData { Number = i, ReplyDelay = (int)stopWatch.ElapsedMilliseconds, IpAddress = pingReply.Address.ToString()

};

}

else

{

data = new TraceData { Number = i, ReplyDelay = (int)stopWatch.ElapsedMilliseconds, IpAddress = pingReply.Address.ToString() };

}

Traceresults.List.Add(data);

}

if (pingReply.Status == IPStatus.Success)

{

break;

}

pingOptions.Ttl++;

При анализе пришедшего ответа IP-адрес откликнувшегося устройства, порядковый номер и время отклика заносятся в коллекцию, которая в будущем будет сериализована на диск. Также после каждой итерации значение TTL увеличивается на единицу. Это позволяет постепенно от проследить маршрут, по которому проходит пакет.

## **4.4** Алгоритм построения графов

Для того чтобы начать рассмотрение алгоритма построения графов сети, необходимо пояснить, что такое корутина в среде Unity3D.

Корутина – это метод, в котором можно использовать инструкцию yield. Это особенный тип возврата значения из функции. При следующем вызове данного метода выполнение инструкций начнется не сначала, а с того места, где в прошлый раз был выход по yield return. Этот механизм удобен, если необходимо подождать каких-либо действий. Весь алгоритм построения графов построен на корутинах.

Алгоритм начинается с запуска главной корутины, которая поочередно стартует корутины различных его частей. При этом она дожидается выполнения каждой из запущенных корутин.

private IEnumerator BuildNetwork()

{

yield return

StartCoroutine(InitializeArpBuilder(ARPDataReader()));

yield return StartCoroutine(PlaceMainRouter());

yield return StartCoroutine(BuildTrace(TraceDataReader()));

}

Первым шагом алгоритма является запуск корутины, которая инициализирует построитель графов. В этом методе идет анализ коллекции IP-адресов и с учетом маски сети происходит формирование списка подсетей для нанесения их на карту.

foreach (var node in arpList.List)

{

string[] ipParts = node.IpAddress.Split('.');

string key = string.Empty;

for (int i = 0; i < Context.subNetParts; i++)

{

key += ipParts[i] + ".";

}

if (!subNets.ContainsKey(key))

{

subNets.Add(key, new ARPList());

subNets[key].List = new List<ARPData>();

}

subNets[key].List.Add(node);

}

После чего запускается цикл, в котором вычисляется радиус графа каждой подсети. Затем запускается корутина для построения графа текущей подсети. Также на каждой итерации цикла происходит расчет сдвига графов относительно друг друга. Сдвиг служит для избегания коллизий при построении.

Vector3 localOffset = new Vector3();

foreach (var item in subNets)

{

var subNet = item.Value;

float r = (subNet.List.Count \* (elementSize)) / (2 \* Mathf.PI);

localOffset += new Vector3(0, 0, circleOffset + r);

yield return StartCoroutine(BuildArpCircle(subNet, localOffset, r, key));

localOffset += new Vector3(0, 0, circleOffset + r);

}

Подробно рассмотрим корутину, отвечающую за построение графа одной подсети. Сразу вычисляется угловое смещение, которое будет использовано при размещении моделей устройств.

float deltaAng = 360f / nodes.List.Count;

После чего происходит инстансирование 3д-модели маршрутизатора по заданной позиции. Роутер подписывается ключом подсети. Также устанавливаются координаты расположения камеры в центр строящейся окружности, для удобного наблюдения за прогрессом построения.

GameObject router = Instantiate(routers[0], center, routers[0].transform.rotation) as GameObject;

cam.position = new Vector3(router.transform.position.x, router.transform.position.y + 12, router.transform.position.z + 25);

router.GetComponent<Router>().label.text = "Network " + key;

router.name = key;

Далее в цикле для всех устройств в подсети происходит инстансирование 3д-модели компьютера по заданной позиции. Под моделью располагается текст с IP- и MAC-адресами. После чего задаются две координаты для соединительной линии. Настраивается отображаемый материал, толщина и цвет.

for (int i = 0; i < nodes.List.Count; i++)

{

GameObject go = Instantiate(computer, new Vector3(radius \* Mathf.Cos((deltaAng \* i) / 57) + center.x, 0, radius \* Mathf.Sin((deltaAng \* i) / 57) + center.z), Quaternion.identity) as GameObject;

go.GetComponent<Computer>().AdressLabel.text = string.Format("IP {0}\nMAC {1}", nodes.List[i].IpAddress, nodes.List[i].MacAddress);

LineRenderer line = go.AddComponent<LineRenderer>();

line.SetWidth(0.3f, 0.3f);

line.SetPosition(0, go.transform.position);

line.SetPosition(1, center);

line.material = lineMaterial;

line.material.SetColor("\_TintColor", pingColor);

yield return new WaitForSeconds(2f / nodes.Count);

}

После построения всех окружностей, алгоритм переходит к расположению главного маршрутизатора (Gateway). Его позиция вычисляется следующим образом: находится окружность с самым большим радиусом и к по Z координате прибавляется фиксированное смещение. Таким образом маршрутизатор будет находиться всегда за пределами окружностей.

float maxR = 0;

foreach (var item in circleRadiuses.Keys)

{

if (maxR < item)

{

maxR = item;

}

}

Vector3 routerPosition = new Vector3(circleRadiuses[maxR].x + maxR + circleOffset \* 3, circleRadiuses[maxR].y, circleRadiuses[maxR].z - 20);

GameObject go = Instantiate(routers[1], routerPosition, routers[1].transform.rotation) as GameObject;

Далее центр каждой окружности соединяется линией с координатами поставленного маршрутизатора. Настраивается толщина, отображаемый материал и цвет линий. Позиция маршрутизатора сохраняется.

foreach (var item in circleRadiuses)

{

Vector3 center = item.Value;

GameObject line = new GameObject("Line");

LineRenderer lr = line.AddComponent<LineRenderer>();

lr.SetWidth(0.7f, 0.7f);

lr.material = lineMaterial;

lr.material.SetColor("\_TintColor", Color.white);

lr.SetVertexCount(3);

lr.SetPosition(0, center);

lr.SetPosition(1, new Vector3(routerPosition.x, 0, center.z));

lr.SetPosition(2, routerPosition);

}

gatewayPosition = routerPosition;

Заключительным этапом построения графа сети является расположение маршрутизаторов, полученных с помощью трассировки сети. Для построения запускается цикл по коллекции данных о маршрутизаторах, в котором инстансируются 3д-модели роутеров относительно позиции главного маршрутизатора, соединяются линиями, настраиваются материалы для отображения, ширина и их цвет.

for (int i = 0; i < traceList.List.Count; i++)

{

GameObject go = Instantiate(routers[1], new Vector3(gatewayPosition.x, gatewayPosition.y, gatewayPosition.z - 30 \* (i + 1)), routers[1].transform.rotation) as GameObject;

go.GetComponent<Router>().label.text = traceList.List[i].IpAddress;

GameObject line = new GameObject("Line");

LineRenderer lr = line.AddComponent<LineRenderer>();

lr.SetWidth(0.7f, 0.7f);

lr.material.SetColor("\_TintColor", Color.white);

lr.SetPosition(0, go.transform.position);

lr.SetPosition(1, gatewayPosition);

yield return new WaitForSeconds(0.4f);

}

Граф сети построен. После чего программа удаляет временные файлы и предлагает пользователю в деталях ознакомиться с картой сети.

# **5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения – процесс исследования программного обеспечения с целью получения информации о качестве продукта. Тестирование является одним из важных этапов разработки, поскольку при написании программного кода невозможно предусмотреть все случаи, и ошибки в работе программы неизбежны. Кроме того, возможно появление ошибок в случае внесения дополнительного функционала в программу даже в уже отлаженном рабочем коде.

Тестирование программы направлено на доказательства её соответствия заявленным требованиям. Таким образом, основной целью тестирования является выделение несоответствия функционирования программы заявленным требованиям и устранение их, для повышения качества программного продукта.

Тестирование программы проводилось в два этапа:

* поэтапное тестирование отдельно каждого модуля в процессе написания программного кода;
* полное тестирование программы после окончания процесса написания программного кода.

Оба этапа являются достаточно важными, ни один из них невозможно исключить. Например, без модульного тестирования, при анализе работы программы в целом, будет происходить достаточное количество сбоев, выявить и локализовать которые может оказаться достаточно сложным заданием, в то время как при анализе работы одного модуля неисправность оказывается достаточно очевидной. И обратный случай, работоспособность каждого компонента в отдельности не гарантирует корректное поведение всей программы в целом.

Тестирование программы проводилось на следующих компьютерах:

* AMD Phenom II X4 965 4 ядра по 3,4 ГГц, оперативная память 6Гб, видеокарта nVidia GTX260 до 896 Мб. Операционная система Windows 7 Professional x64 Service Pack 1;
* Intel Core i3 540 2 ядра по 3,07 ГГц, оперативная память 4ГБ, видеокарта Intel HD Graphics до 1696 МБ. Операционная система Windows 7 Enterprise x64;
* Intel Celeron M420 1,6 ГГц, оперативная память 512 МБ, видеокарта Radeon 64 МБ. Операционная система Windows XP Professional x32 Service Pack 3.
* AMD Athlon 64 X2 7550 2 ядра по 2,5 ГГц, оперативная память 1ГБ, видеокарта nVidia GTS8800Pro до 512 МБ. Операционная система Windows 8 Consumer Preview x32;

В программе предусмотрен механизм анализа введенной информации. При неправильном вводе пользователю сообщается о тех ошибках, которые он допустил. На рисунке 5.1 показана ошибка ввода неверного IP-адреса.



Рисунок 5.1 – Ошибка при вводе IP-адреса

Начать сканирование невозможно, пока пользователь не введет правильные данные. В главном меню только поле IP-адреса вводится вручную. Остальные поля сделаны в виде выпадающих списков с подготовленными заранее значениями. Преимущество таких элементов пользовательского интерфейса в том, что пользователь никогда не допустит ошибку. Соответственно отлавливать и обрабатывать в этих случаях нечего.

Если по какой-либо причине программа не может получить доступ к файловой системе на диске для записи файла, то пользователь увидит соответствующее сообщение, изображенное на рисунке 5.2. В данном случае сканирование завершится, но из-за ошибки граф построен не будет.

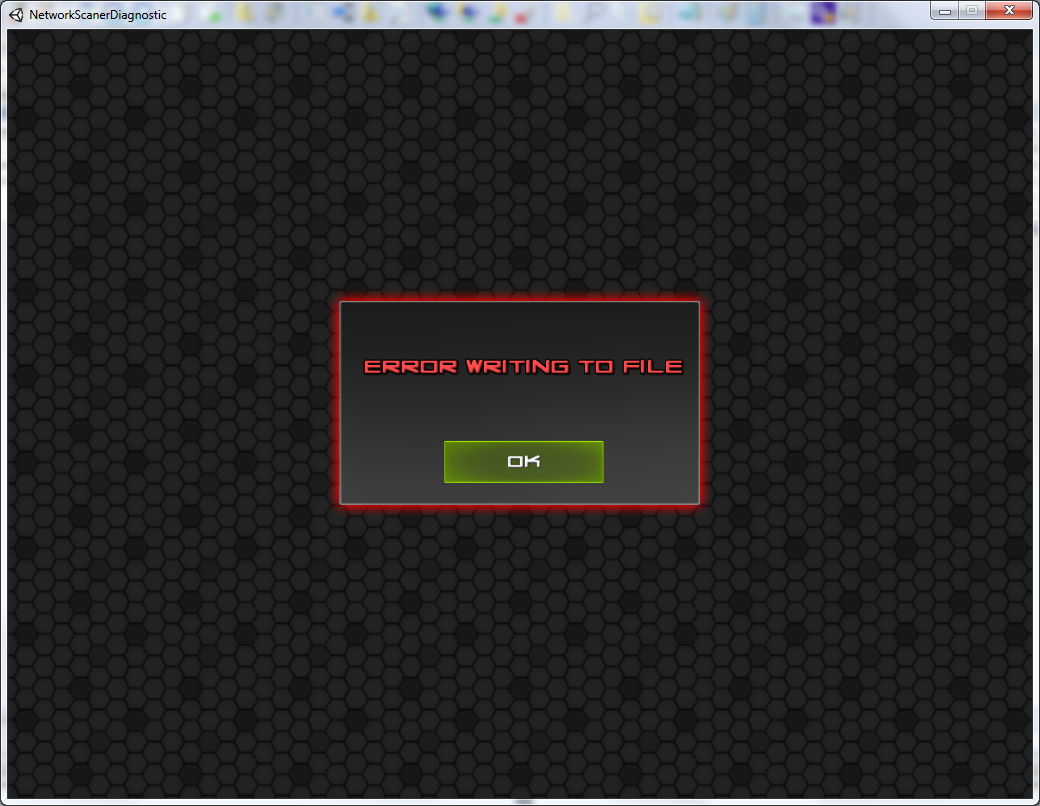


Рисунок 5.2 – Ошибка при записи файла

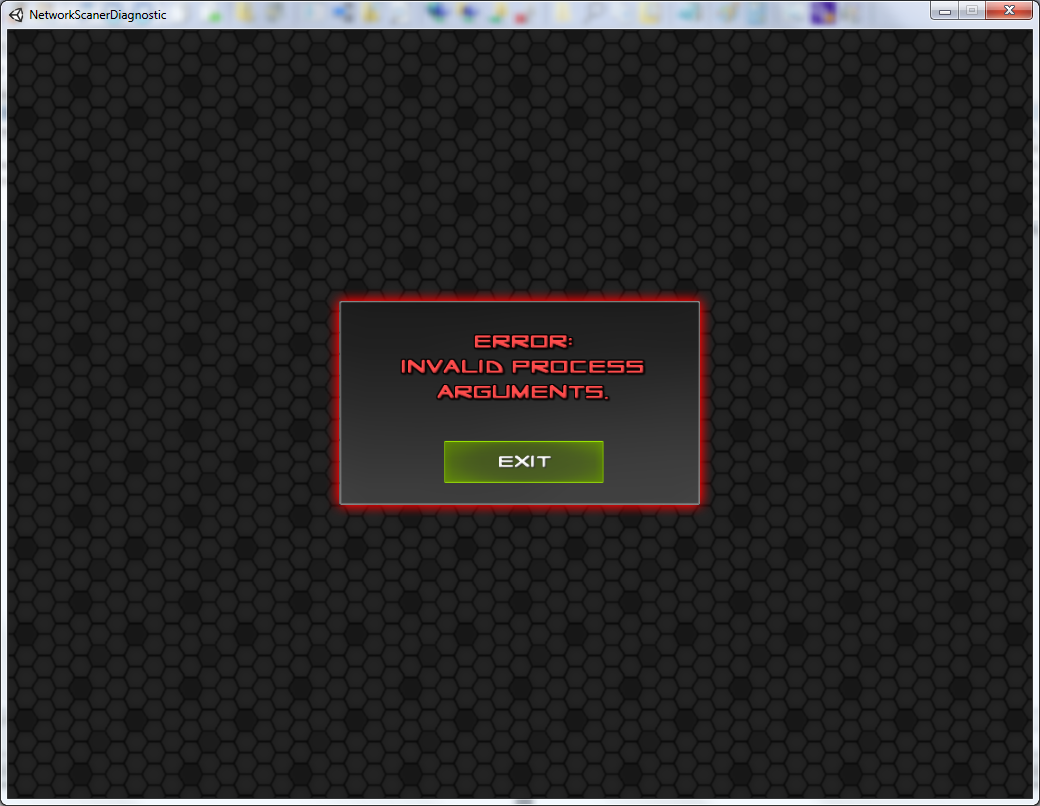


Рисунок 5.3 – Ошибка при формировании аргументов для процесса сканирования

Если возможна ситуация, в которой аргументы для процесса сканирования будут сформированы не правильно, то сбоя в программе не будет. При запуске процесса сканирования идет строгий анализ аргументов и если хотя бы один аргумент задан не верно, процесс завершается, не начав сканирование. При этом пользователь видит сообщение, изображенное на рисунке 5.3.

При выходе из приложения программа удаляет временные файлы, которые хранят информацию о просканированной сети. Если пользователь сам удалит эти файлы, сбоя программы не будет. При удалении проверяется существование файлов.

if (File.Exists(Common.CommonData.TraceFile))

{

File.Delete(Common.CommonData.TraceFile);

}

if (File.Exists(Common.CommonData.ArpFile))

{

File.Delete(Common.CommonData.ArpFile);

}

Такая же проверка на существование файлов производится построителем графов перед началом построения. Если необходимые файлы не были найдены, процесс построения не начинается.

Для обнаружения остальных неприятных моментов проводилось функциональное тестирование. Тестирование уже готовой программы проводилось сразу несколькими пользователями на разных компьютерах с разной производительностью и разными операционными системами. Тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| Меню программы | Содержание теста | Ожидаемый результат | Тест пройден |
| 1 | 2 | 3 | 4 |
| Главное | Попытка изменить первый IP-адрес из диапазона | IP-адрес меняется на введённый. | да |
| Главное | Попытка изменить второй IP-адрес из диапазона | IP-адрес меняется на введённый. | да |
| Продолжение таблицы 5.1 | | | |
| 1 | 2 | 3 | 4 |
| Главное | Нажатие на флажок выбора опции трассировки | Значение флажка меняется на противоположное. | да |
| Главное | Нажатие по полю выбора маски подсети | Появляется выпадающий список и при выборе определенного элемента из этого списка маска подсети устанавливается на выбранную. | да |
| Главное | Нажатие на флажок выбора опции трассировки | Значение флажка меняется на противоположное. | да |
| Главное | Нажатие по полю выбора глубины трассировки | Появляется выпадающий список и при выборе определенного элемента из этого списка значение глубины трассировки устанавливается на выбранное. | да |
| Главное | Попытка ввести заведомо неверный IP-адрес | Вывод диалогового окна с текстом ошибки. | да |
| Главное | Реакция программы на нажатие кнопки «Scan» | Начало сканирования. Отображение процесса в виде надписи «Please Wait». | да |
| Карта сети | Реакция программы на нажатия клавиш WASD | Перемещение камеры вперед, влево, вправо и назад. | да |
| Карта сети | Реакция программы движение мыши при зажатой правой клавише (для мобильных устройств движением пальца по экрану) | Поворот камеры вокруг своей оси. | да |

Как видно из таблицы, приложение хорошо справилось с тестами, что говорит о высокой его работоспособности.

# **6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## **6.1** Инсталляция приложения

Данное программное средство доступно в виде приложения под следующие операционные системы: Microsoft Windows, Mac OS X, Android, iOs. Рассмотрим способы инсталляции для iOs и Microsoft Windows.

*6.1.1 Инсталляция приложения на операционной системе Windows*

Для установки приложения следует запустить NetScanDiagSetup.exeи проследовать инструкции установщика. Стартовое окно установщика изображено на рисунке 6.1. Процедура установки стандартная как для большинства Windows приложений. Указываем папку установки, после чего выбираем создание необходимых ярлыков. Затем подтверждаем выбранные опции и приступаем непосредственно к инсталляции.

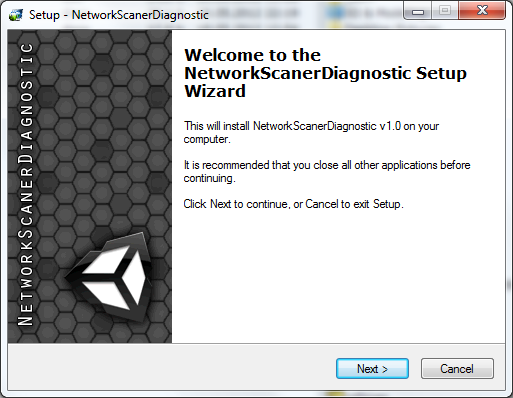


Рисунок 6.1 – Стартовое окно инсталлятора

*6.1.2 Инсталляция приложения на iPhone*

Для установки приложения на мобильное устройств существует два пути: установка приложения через iTunes и скачка и установка приложения через AppStore (скриншот магазина изображен на рисунке 6.2). Мы рассмотрим оба пути инсталляции.

Установка через iTunes:

* Запустить приложение iTunes в операционной системе компьютера;
* Подключить iPhone или iPod Touch к компьютеру;
* Добавить приложение NetworkScanerDiagnostic в список приложений на мобильном устройстве;
* Синхронизировать устройство и iTunes;

Установка через AppStore:

* Запустить приложение AppStore на мобильном устройстве;
* Ввести в поиске «Network Scaner Diagnostic»;
* Добавить приложение NetworkScanerDiagnostic в список приложений на мобильном устройстве;
* Дождаться окончания загрузки приложения;

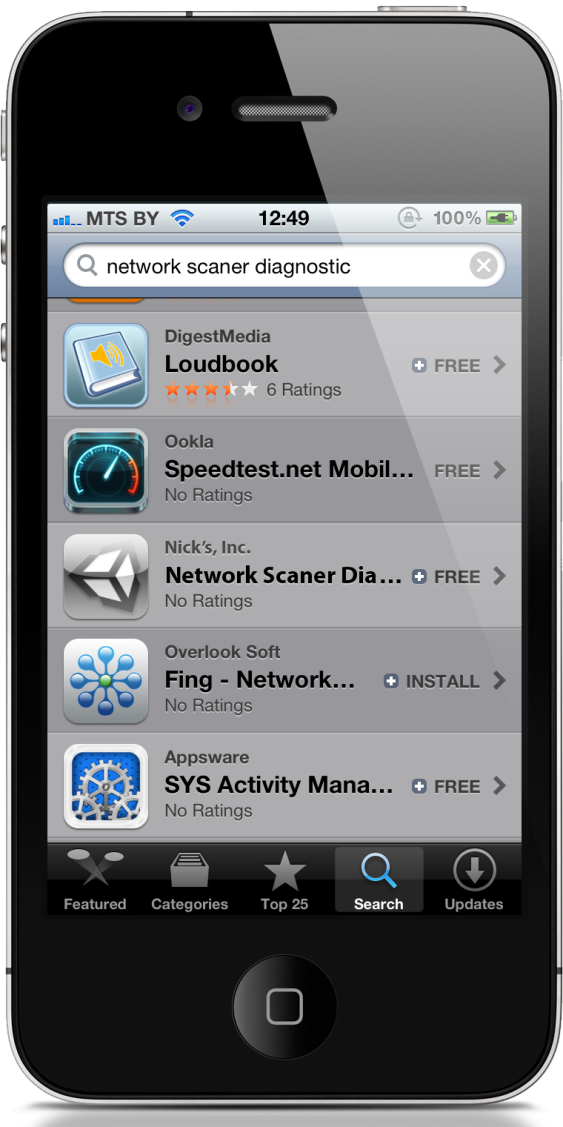


Рисунок 6.2 – Приложение в Apple Appstore

## **6.2** Настройка и запуск сканирования

При запуске программы мы попадаем в главное меню, изображенное на рисунке 6.3. Подробнее рассмотрим элементы пользовательского интерфейса этого меню:

* Поле ввода диапазона IP-адресов. Представлено в виде двух полей для ввода адресов. Разрешается вводить адреса из разных подсетей.
* Поле ввода маски подсети. Выпадающий список из трех значений. Можно выбрать «255.255.255.0», «255.255.0.0», «255.0.0.0». Эти значения были выбраны по причине их повсеместного распространения в современных локальных сетях.
* Флажок «Trace Route». При включенной опции будет произведена трассировка маршрута за пределы маршрутизатора.
* Поле «Max Hops». Служит для указания глубины трассировки. Выпадающий список со значениями в 5, 10, 15, 20, 30 хостов.
* Кнопка «Scan». Запускает сканирование. После чего отображается процесс, изображенный на рисунке 6.4.

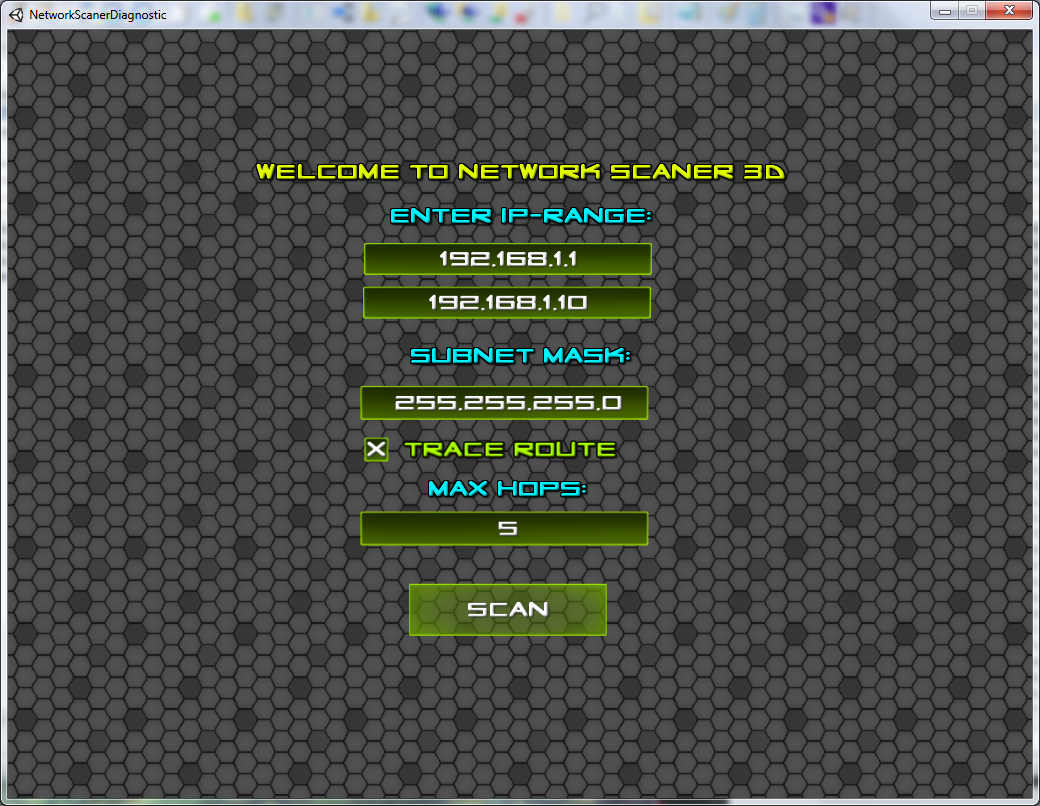


Рисунок 6.3 – Главное меню

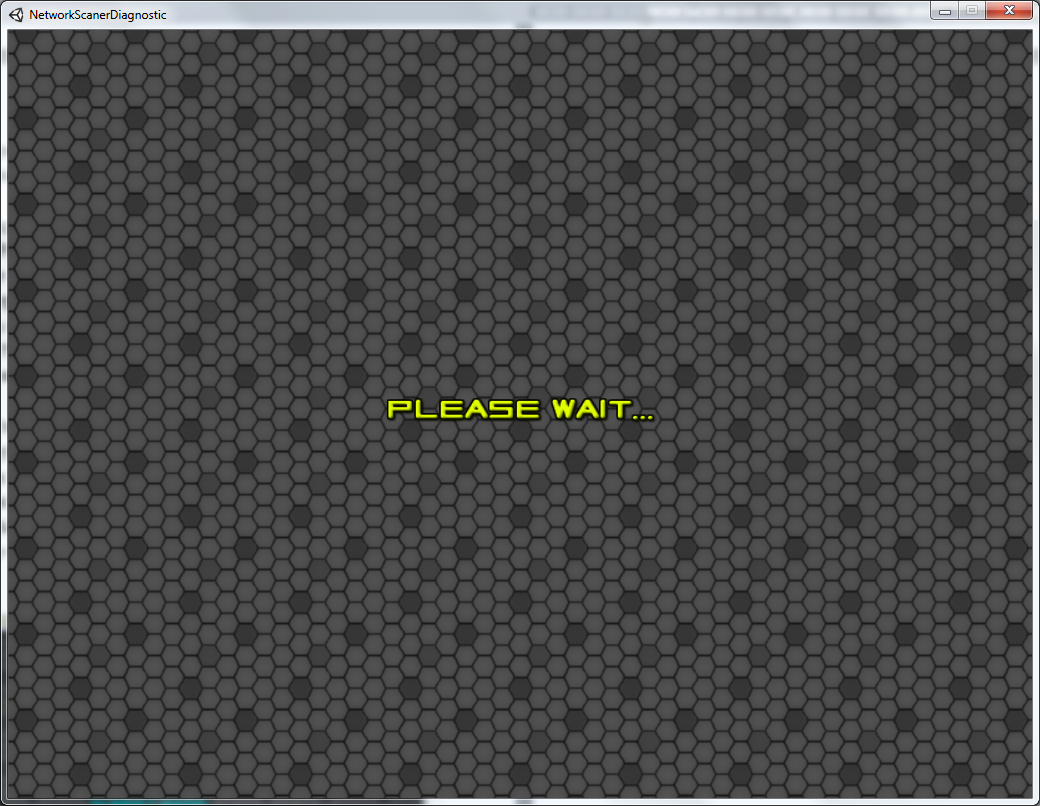


Рисунок 6.4 – Поведение программы во время сканирования сети

После завершения процесса сканирования мы увидим, как строится карта сети (рисунок 6.5). Если подсетей несколько, то камера покажет построение каждой из них. После того, как граф будет полностью построен, его можно исследовать. На мобильных устройствах в нижних углах экрана есть небольшие области, которые ответственны за перемещение камеры в пространстве. Правая ответственна за перемещение вперед и назад, левая – право и лево. На персональном компьютере перемещение камеры осуществляется при помощи клавиш W, A, S, D. Клавиша W – для перемещения камеры вперед, A – влево, S – назад, D – вправо. Поворот камеры на мобильных устройствах реализован следующим образом: для вращения камеры вокруг своей оси необходимо коснутся одним пальцем экрана и вести его в ту сторону, поворот в которую должен быть осуществлен. На персональном компьютере вращение камеры осуществляется при помощи мыши. Для активации режима вращения необходимо зажать правую клавишу мыши и двигать ее в ту сторону, поворот в которую должен быть осуществлен.

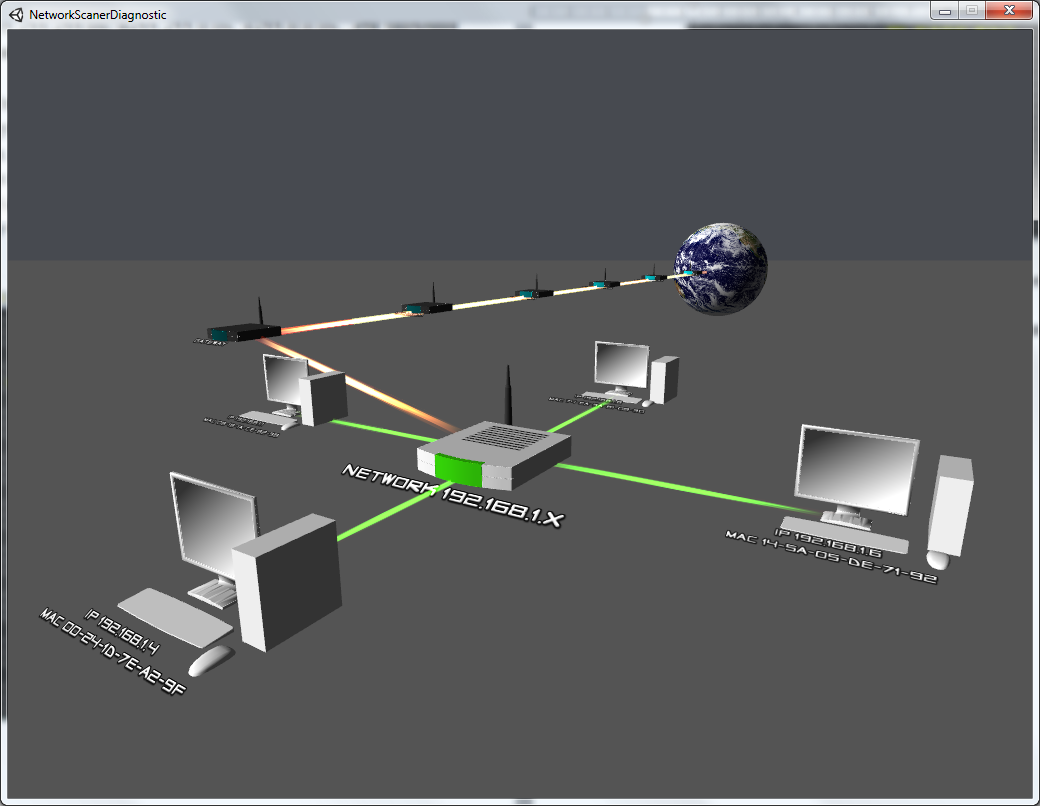


Рисунок 6.5 – Окно с результатами работы программы

Для выхода из программы на персональном компьютере достаточно закрыть главное окно. На мобильных устройствах этот процесс отличается от вышеописанного. Так как данное приложение поддерживает режим мультизадачности мобильных операционных систем iOs и Android, то при нажатии на Home-кнопку, приложение просто свернется. Для того, чтобы закрыть его полностью, необходимо воспользоваться диспетчером задач операционной системы.

Для успешной работы программы ей потребуется процессор Intel или AMD с частотой более 1000 Гц, 1 ГБ оперативной памяти, установленный графический адаптер, минимум 40 МБ свободного места на винчестере, совместимая звуковая карта, мышь, клавиатура.

Список поддерживаемых мобильных устройств: Apple iPod Touch 3G, Apple iPod Touch 4, Apple iPhone 3G, Apple iPhone 3GS, Apple iPhone 4, Apple iPhone 4S, Apple iPad, Apple iPad 2, Apple iPad 4G, Samsung Galaxy S I, Samsung Galaxy S II, Samsung Galaxy Note.

# **7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПРОЕКТА

Программная система сканирования и трехмерного моделирования локальной сети – это полнофункциональный инструмент для сканирования сети, построения её топологии в 3D, а также проведения базовой диагностики. Пользователями данного программного продукта являются не только системные администраторы, но и обычные пользователи локальных сетей. Это стало возможным благодаря грамотному проектированию интерфейса. В итоге мы получаем простую в использовании программу для широкого круга пользователей.

Программный продукт позволяет упростить и ускорить процесс сканирования и построения топологии сети, а также провести простейшую диагностику. Для удобства каждый пользователь может использовать ту версию программы, которая совместима с его устройством (мобильный телефон, персональный компьютер).

В настоящем технико-экономическом обосновании рассчитывается эффективность разработки и использования приложения путем подсчета суммы расходов на разработку.

Базой для расчета плановой сметы затрат и цены на разработку программного обеспечения является объем программного обеспечения.

Стоимостная оценка программного средства у разработчиков предполагает составление сметы затрат. *Смета затрат* – это полный расчет расходов предприятия на производство и реализацию продукции за определенный календарный период.

Смета затрат включает следующие статьи расходов:

* заработную плату (основную и дополнительную) исполнителей ( и );
* отчисления на социальные нужды ();
* материалы и комплектующие изделия ();
* спецоборудование ();
* машинное время ();
* расходы на научные командировки ();
* прочие прямые расходы ();
* накладные расходы ().

На основании сметы затрат по всем статьям (Ср) рассчитывается себестоимость и отпускная цена программного средства.

Таблица 7.1 – Перечень и объем функций

|  |  |  |  |
| --- | --- | --- | --- |
| Номер функции | Наименование (содержание) | Объем функции, LOC | |
| по каталогу (Vi) | уточненный (Vуi) |
| 102 | Контроль, предварительная обработка и ввод информации | 450 | 440 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 320 | 315 |
| 110 | Организация ввода/вывода информации с сети терминалов | 3200 | 3350 |
| 111 | Управление вводом/выводом | 2400 | 2325 |
| 403 | Формирование служебных таблиц | 1070 | 1050 |
| 501 | Монитор ПО (управление работой компонентов) | 740 | 725 |
| 703 | Расчет показателей | 460 | 460 |
| 707 | Графический вывод результатов | 480 | 470 |
| 709 | Изменение состояния ресурсов в интерактивной системе | 480 | 480 |
| Итого |  | 9600 | 9615 |

Общий объем функций рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.1) |

Уточненный объем функций равен:

|  |  |
| --- | --- |
|  | (7.2) |

По уточненному объему программного обеспечения и нормативам затрат труда в расчете на единицу объема определяется нормативная и общая трудоемкость разработки программного обеспечения.

На основании принятого к расчету уточненного объема (Vу) и категории сложности определяется нормативная трудоемкость (Тн) = 307 человеко-часов. Нормативная трудоемкость служит основой для определения общей трудоемкости.

|  |  |
| --- | --- |
|  | (7.3) |

где Кс – коэффициент, учитывающий повышение сложности программного обеспечения, который определяется по формуле:

|  |  |
| --- | --- |
|  | (7.4) |

где Кi – коэффициент, соответствующий степени повышения сложности программного обеспечения за счет конкретных характеристик,

n – количество учитываемых характеристик,

Кт – поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей.

Степень использования в разрабатываемом программном обеспечении стандартных модулей определяется их удельным весом в общем объеме проектируемого средства (60% стандартных модулей, отсюда, Кт = 0,6).

Коэффициент Кн, учитывающий степень новизны программного обеспечения (так как категория новизны программы – В, то Кн =0,7).

Таблица 7.2 – Значение по характеристике программного обеспечения

|  |  |
| --- | --- |
| Характеристика ПО | Значение |
| Интерактивный доступ | 0,06 |
| Наличие у программы одновременно двух определенных характеристик. | 0,12 |

Найдем , исходя из имеющихся данных:

|  |  |
| --- | --- |
|  | (7.5) |

Итак, в результате находим значение общей трудоемкости.

|  |  |
| --- | --- |
| чел/дн. | (7.6) |

После определения общей трудоемкости разработки программного обеспечения, рассчитывается число исполнителей по формуле:

|  |  |
| --- | --- |
|  | (7.7) |

где То – общая трудоемкость проекта,

Тр – срок директивной разработки проекта (9 месяцев или 0,75 года),

Фэф – эффективный фонд времени работы одного исполнителя, рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.8) |

где Дк – количество календарных дней в году (366 дн.);

Дв – количество выходных дней (52 × 2 = 104 дн.);

Дп – количество праздничных дней в году, не совпадающих с выходными (7 дн);

До – количество дней отпуска (21 дн.).

|  |  |
| --- | --- |
|  | (7.9) |

Отсюда численность рабочих получаем:

|  |  |
| --- | --- |
|  | (7.10) |

Общая трудоемкость, плановая численность и плановые сроки разработки являются базой для расчета основной заработной платы исполнителей.

Оплата труда устанавливается либо на основе Единой тарифной сетки Республики Беларусь (ЕТС), в которой даны тарифные разряды и тарифные коэффициенты, либо на основе системы оплаты труда, принятой на конкретном предприятии (организации). Месячная тарифная ставка каждого исполнителя определяется путём умножения действующей месячной тарифной ставки 1-го разряда на тарифный коэффициент (Тк), соответствующий установленному тарифному разряду.

В соответствии со штатным расписанием, в проекте будут заняты:

Таблица 7.3 – Лица, занятые в проекте

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнители | Чел/дн. занятости | Разряд | Тарифный коэффициент |
| Руководитель проекта | 22,2 | 17 | 3,98 |
| Программист 1-й категории | 130 | 13 | 3,04 |

Рассчитаем месячную тарифную заработную плату исполнителей по формуле:

|  |  |
| --- | --- |
|  | (7.11) |

где Тмi – тарифная ставка 1-го разряда,

Ктi – тарифный коэффициент, соответствующий i-ому разряду.

Примем тарифную ставку 1-го разряда равной 600 000 руб. и рассчитаем месячные тарифные ставки руководителя:

|  |  |
| --- | --- |
|  | (7.12) |
|  |  |

и программиста:

|  |  |
| --- | --- |
|  | (7.13) |

Дневная тарифная ставка рассчитывается путем деления месячной тарифной заработной платы каждого исполнителя на количество дней в месяце (22).

|  |  |
| --- | --- |
|  | (7.14) |

|  |  |
| --- | --- |
|  | (7.15) |

Основная заработная плата исполнителей на конкретное программное средство определяется по формуле:

|  |  |
| --- | --- |
|  | (7.16) |

где n – количество исполнителей на конкретное программное средство,

Тднi – дневная тарифная зарплата i-го исполнителя (из таблицы 6.3),

Фэi – эффективный фонд рабочего времени i-го исполнителя,

Кп – коэффициент премий (Кп = 1,4).

|  |  |
| --- | --- |
| 18 463 016 | (7.17) |

Дополнительная заработная плата (Зд) включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов подростков и другие) и определяется по нормативу в процентах к основной заработной плате:

|  |  |
| --- | --- |
|  | (7.18) |

где Нд – норматив дополнительной заработной платы, % (Нд = 15%).

|  |  |
| --- | --- |
|  | (7.19) |

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,5%) в процентах от основной и дополнительной заработной платы.

|  |  |
| --- | --- |
|  | (7.20) |

По статье «Материалы» отражаются расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки ПО. Норма расхода материалов в суммарном выражении (Нм) определяются либо в расчете на 100 строк исходного кода, либо в процентах к основной заработной плате разработчиков (3%):

|  |  |
| --- | --- |
|  | (7.21) |

Расходы по статье «Спецоборудование» (Рсо) включает затраты средств на приобретение технических и программных средств специального назначения:

|  |  |
| --- | --- |
|  | (7.22) |

где Цоi – цена конкретного вида оборудования,

n – количество применяемых видов оборудования.

Однако так как спецоборудование не приобретается, данная статья не рассчитывается.

Расходы по статье «Машинное время» (Рм) включают оплату машинного времени, необходимого для разработки и отладки программного средства. Они определяются в машино-часах по нормативам на 100 строк исходного кода (Нмв) машинного времени в зависимости от характера решаемых задач и типа ПС (Нмв = 12).

|  |  |
| --- | --- |
|  | (7.23) |

где Цмв – цена одного машино-часа (5 тыс.руб.),

Vо – общий объем программного средства,

Нмв – норматив расхода машинного времени на отладку 100 строк кода, машино-часов.

|  |  |
| --- | --- |
|  | (7.24) |

Расходы на научные командировки берутся либо в процентах от основной заработной платы исполнителей:

|  |  |
| --- | --- |
|  | (7.25) |

где Нк – норматив командировочных расходов по отношению к основной заработной плате исполнителей (Нк = 10%).

|  |  |
| --- | --- |
|  | (7.26) |

Расходы по статье «Прочие затраты» включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются в процентах к основной заработной плате.

|  |  |
| --- | --- |
|  | (7.27) |

где Нпз – норматив прочих затрат в целом по организации, % (Нпз =15%).

|  |  |
| --- | --- |
|  | (7.28) |

Затраты по статье «Накладные расходы» связаны с содержанием аппарата управления, вспомогательных хозяйств и опытных производств, а также с расходами на общехозяйственные нужды. Определяются по нормативу в процентах к основной заработной плате:

|  |  |
| --- | --- |
|  | (7.29) |

где Ннак – норматив накладных расходов в целом по научной организации, (Ннак = 20%).

|  |  |
| --- | --- |
|  | (7.30) |

Общая сумма расходов по всем статьям на программное обеспечение представляет полную его себестоимость:

|  |  |
| --- | --- |
| Сп = Рм + Зо + Зд + Рсоц + Рсо + Рм + Рк + Рпр + Рнак =  = 43 179 918,9 (руб) | (7.31) |

Для определения цены программного обеспечения необходимо рассчитать плановую прибыль.

Прибыль рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.32) |

где По – плановая прибыль от реализации программного обеспечения,

Ур – уровень рентабельности программного обеспечения, (Ур = 30),

Сп – общая сумма расходов по всем статьям.

|  |  |
| --- | --- |
|  | (7.33) |

Рентабельность и прибыль создаваемого программного обеспечения определяется исходя из результатов анализа рыночных условий, переговоров с заказчиком и согласования с ним отпускной цены.

После расчета прибыли от реализации определяется прогнозируемая цена программного обеспечения без налогов:

|  |  |
| --- | --- |
| = =  = 56 133 894,6 (руб). | (7.34) |

Отпускная цена программного обеспечения включает налог на добавленную стоимость:

|  |  |
| --- | --- |
| . | (7.35) |

|  |  |
| --- | --- |
|  | (7.36) |

где Ндс – ставка налога на добавленную стоимость, (в настоящее время

Ндс = 20%).

|  |  |
| --- | --- |
|  | (7.37) |

Прогнозируемая отпускная цена, руб:

|  |  |
| --- | --- |
| . | (7.38) |
|  |  |

Прибыль от реализации программного средства за вычетом налога на прибыль (Нп) остается организации разработчику и представляет собой экономический эффект от создания нового программного средства. Это и есть ключевая цифра в расчётах, которая является хорошим показателем рентабельности разработки.

|  |  |
| --- | --- |
| , | (7.39) |
|  |  |

где Нп – ставка налога на прибыль (Нп = 18%).

Все расчеты себестоимости и прибыли от реализации программного средства можно свести в таблицу 7.4.

Таблица 7.4 – Результаты и формулы расчетов.

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статей | Усл. обозн. | Значение (руб) | Методика расчета |
| 1 | 2 | 3 | 4 |
| Материалы и комплектующие | Рм | 553 890 | Определяются на основании расчетов |
| Основная заработная плата исполнителей | Зо | 18 463 016 | Определяются на основании расчетов |
| Дополнительная заработная плата исполнителей | Зд | 2 769 452 |  |
| Отчисления в фонд социальной защиты населения | Рсоц | 7 325 201 |  |
| Машинное время | Рмв | 5 760 000 | Определяются на основании расчета. Цена 1 машино-часа рыночная |
| Расходы на научные командировки | Рнк | 1 846 301 |  |
| Прочие прямые расходы | Рпз | 2 769 452 |  |
| Накладные расходы | Рнак | 3 692 603 |  |
| Полная себестоимость | Сп | 43 179 918 | Сп = Рм + Зо + Зд + Рсоц + + Рм + Рнп + Рпр + Рн |
| Прогнозируемая прибыль | По | 12 953 975 |  |
| Прогнозируемая цена без налогов (цена предприятия) | Цп | 56 133 894 |  |
| Налог на добавленную стоимость (НДС) | НДС | 11 226 778 |  |
| Прогнозируемая отпускная цена | Цот | 67 360 673 |  |
| Месячная тарифная ставка 1-го разряда | Тм1 | 600 000 |  |
| Ставка налога на прибыль | Нп | Нп=18% |  |

Таким образом, после проведения расчетов экономической эффективности стала известна прогнозируемая отпускная цена продукта, равная 67 360 673 рублям. Она значительно меньше цены аналогов на рынке сегодня, что должно хорошо сказаться на продажах. Сокращение расходов при создании программного средства позволит улучшить этот показатель. Заложенная высокая рентабельность гарантирует большой доход от реализации программного продукта.

# **8** ОХРАНА ТРУДА. ОБЕСПЕЧЕНИЕ ЗАЩИТЫ РАБОТАЮЩИХ ОТ ИОНИЗИРУЮЩИХ ИЗЛУЧЕНИЙ

В данной дипломной работе представлено программное средство сканирования локальной сети и построения ее топологии. Непосредственная разработка производится на ЭВМ, компонентом которой является монитор. И он же является единственным возможным источником ионизирующего излучения для рабочих.

Выбор данной темы раздела охраны труда связан с тем, что в последнее время заметно повышение мирового интереса к проблеме радиационной безопасности. К тому же, для населения Республики Беларусь данная проблема на протяжении многих лет остаётся актуальной.

Хоть современные мониторы не являются источником ионизирующих излучений, никто не гарантирует, что разрабатываемое программное средство не будет использоваться на ЭВМ со старыми ЭЛТ мониторами. Поэтому стоит рассмотреть воздействие ионизирующих излучений ЭЛТ монитора на работающих.

ЭЛТ мониторы компьютеров являются источником рентгеновского, бета- и гамма-излучений. Рентгеновское излучение присутствует только при работе монитора. Оно возникает при торможении пучка электронов и как характеристическое излучение атомов материалов кинескопа. Спектр рентгеновского излучения является непрерывным с набором моноэнергетических линий. Максимальная энергия спектра ~20 кэВ. Бета-, гамма-излучения присутствуют и при включенном и при выключенном мониторе. Источником этих излучений является радиоактивный распад ядер семейств урана и тория, а также ядер калия-40. Спектральный состав гамма-излучения преимущественно состоит из набора моноэнергетических линий. Бета - излучение монитора определяется главным образом радиоактивным распадом ядер калия-40; спектральный состав бета-излучения непрерывен, а его максимальная энергия ~1.3 мэВ. При определенных условиях эти ионизирующие излучения способны причинить вред здоровью человека, в частности, вызвать помутнение хрусталика глаза [15].

Далее приведены некоторые характеристики конкретных видов ионизирующих излучений мониторов.

Бета-излучение представляет собой поток электронов β- или протонов β+. При распаде ядер β-активного радионуклида, в отличие от α-распада, различные ядра данного радионуклида испускают β-частицы различной энергии, поэтому энергетический спектр β-частиц непрерывен. По сравнению с α-частицами β-излучение имеет существенно меньшую ионизирующую способность и значительно большую проникающую. Расчет потоков бета -излучения - трудная задача, однако эти потоки могут быть легко измерены бета-счетчиком. Подобные измерения показывают, что на расстоянии 5 см от экрана монитора плотность потока бета-излучения может составлять 0.2 - 0.5 част/с×см2 [15].

 Уровень гамма-излучения зависит от концентраций естественных радионуклидов в стекле монитора, которые для калия-40 составляют 3-10%, для тория - (0.3-1)×10-4%, для урана- (1-3)×10-4%. Исходя из этого можно показать, что на расстоянии 5 см от экрана монитора мощность дозы гамма-излучения ничтожна(~0.03-0.1 мкР/час) и составляет 0.5% от мощности дозы фона [15].

Измерения уровней рентгеновского излучения наиболее трудны. Для таких измерений обычно применяются сцинтилляционные спектрометры с тонкими кристаллами NaI(Tl) или CsI(Tl) и с достаточно большой поверхностью. Полученные с их помощью результаты показывают, что максимальная мощность дозы рентгеновского излучения на расстоянии 5 см от экрана монитора сравнима с фоном и не превышает 5-15 мкР/час.

Исходя из этого, мощность эквивалентной дозы излучений для неблагоприятного случая, когда глаза оператора компьютера расположены на расстоянии 5 см от экрана монитора составит 0.3-0.4 мкЗв/час [15].

Степень воздействия ионизирующих излучений на организм человека, его реакция зависят от дозы излучения, ее мощности, плотности ионизации излучения, вида облучения, продолжительности воздействия, индивидуальной чувствительности, психофизиологического состояния организма и др.

Наиболее чувствительными к действию радиации являются клетки постоянно обновляющихся тканей и органов (костный мозг, половые железы, селезенка и др.) [15].

Согласно нормативной документации устанавливаются две категории облучаемых лиц – персонал и все население.

Для указанных категорий облучаемых лиц устанавливаются три класса нормативов:

* основные дозовые пределы;
* допустимые уровни для одного радионуклида или одного вида внешнего облучения, пути поступления, являющиеся производными от основных дозовых пределов: пределы годового поступления; допустимые среднегодовые объемные активности (ДОА); удельные активности (ДУА) и другие;
* контрольные уровни (дозы), устанавливаемые администрацией учреждений по согласованию с органами Госсанэпиднадзора (таблица 8.1).

Годовая эффективная доза облучения принимается равной сумме эффективной дозы внешнего облучения, накопленной за календарный год, и ожидаемой эффективной дозы внутреннего облучения, обусловленной поступлением в организм радионуклидов за этот же период. Интервал времени для определения величины ожидаемой эффективной дозы устанавливается равным 50 лет для лиц из персонала и 70 лет — для лиц из населения.

Таблица 8.1 – Контрольные уровни, устанавливаемые администрацией учреждений по согласованию с органами Госсанэпиднадзора [15]

|  |  |  |
| --- | --- | --- |
| Нормируемая величина | Дозовый предел (мЗв) | |
| Персонал | Население |
| Эффективная доза | 20 в год в среднем за любые последовательные 5 лет, но не более 50 в год | 1 в год в среднем за любые последовательные 5 лет, но не более 5 в год |
| Эквивалентная доза за год:  в хрусталике; | 150 | 15 |
| в коже; | 500 | 50 |
| в кистях и стопах | 500 | 50 |

Для расчёта годовой эквивалентной дозы излучения монитора при самом неблагоприятном случае (глаза разработчика расположены на расстоянии 15 см от экрана монитора), можно использовать следующую формулу:

|  |  |
| --- | --- |
| , | (8.1) |

где  *–* эквивалентнаядоза, полученная за час работы,

– количество рабочих часов в день,

– количество рабочих дней в году.

Подставив максимальное значение, полученное ранее для эквивалентной дозы за час работы, восьмичасовой рабочий день и среднее количество рабочих дней в году, равное 237 дням, в формулу 8.1, получим следующий результат:

|  |  |
| --- | --- |
|  | (8.2) |

Таким образом, годовая эквивалентная доза, поглощённая хрусталиком глаза программиста под воздействием ионизирующего излучения монитора оказывается намного меньше максимально установленной дозы, как для персонала, так и для населения.

Дополнительно хотелось отметить, что для уменьшения вредного действия ионизирующих излучений в ЭЛТ мониторах было снижено анодное напряжение, а в стекло мониторов добавлен свинец. Также излучение экранируется специальными ферритовыми экранами-чашками [15].

В случае если пользователь или разработчик будет использовать устаревший ЭЛТ монитор, хотелось бы привести некоторые рекомендации по дополнительной защите от ионизирующего излучения:

* организационные мероприятия (регулярные перерывы в работе по 15 минут каждые 2 часа);
* инженерно-технические методы и средства. Защита расстоянием: 40-60 см. от экрана монитора. (также защита временем, применение средств индивидуальной защиты, защитное экранирование и др.).
* медико-профилактические мероприятия (сокращенный рабочий день до 4-6 ч, дополнительный отпуск до 24 рабочих дней, медицинские осмотры через 6-12 месяцев и другие).

Таким образом, современные мониторы имеют очень хорошую защиту пользователя от ионизирующего излучения. Рассчитанная ранее эквивалентная годовая доза для устаревших ЭЛТ мониторов также позволяет работать над разрабатываемым дипломным проектом полный рабочий день и на полной ставке. Приведённые выше меры дополнительной предосторожности сводят практически к нулю негативное влияние ионизирующих излучений.

# ЗАКЛЮЧЕНИЕ

В процессе работы над дипломным проектом были изучены особенности работы с сетевыми протоколами при сканировании сети. По результатам этого, было создано приложение, не имеющее полноценных аналогов. Благодаря этому, пользователь при использовании данного приложения получает следующие возможности:

* сканирования сети по заданному диапазону адресов на предмет обнаружения хостов;
* трассировки сети за пределы маршрутизатора;
* детального изучения карты просканированной сети в трехмерном пространстве;
* базовой диагностики сети;
* установки приложения не только на персональный компьютер, но и на популярные мобильные устройства, в том числе и планшетные ПК.

Таким образом, из достоинств программы следует выделить:

* уникальность в своем роде означает то, что до сих пор программ с подобным функционалом не создавалось;
* гибкость при настройке под конкретную сеть;
* быстродействие алгоритмов, которое достигается распараллеливанием выполняемых операций;
* простота использования включает в себя интуитивно понятный интерфейс;
* кроссплатформенность.

Из недостатков следует отметить следующие факторы:

* при сканировании больших сетей на ноутбуках возможны небольшие «подтормаживания» в режиме навигации;
* поддержка узкого списка мобильных устройств.

В качестве усовершенствования программы можно рассматривать графическую оптимизацию для видеокарт прошлых поколений, различные схемы управления, продвинутая диагностика.

СПИСОК ЛИТЕРАТУРЫ

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Дж. Рихтер. – СПб. : Питер, 2012. – 928 с.
2. Мак-Дональд, М. Microsoft ASP.NET 2.0 с примерами на С# 2005 для профессионалов / М. Мак-Дональд, М. Шпушта. – М. : Вильямс, 2006. – 1405 с.
3. Библиотека MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа: http://msdn.microsoft.com/ru-ru/library/default.aspx
4. C# Version 4.0 Specification [Электронный ресурс] Specification / Microsoft Corporation. – Электронные данные. – Режим доступа: C# Language Specification 4.0.doc.
5. «Official site of Mono project» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.mono-project.com/Main\_Page
6. «Основные принципы Mono» [Электронный ресурс]. – Электронные данные. – Режим доступа:

http://www.ibm.com/developerworks/ru/library/os\_mono1/index.html

1. «Mono Touch» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://xamarin.com/monotouch
2. «Unity3D Official Site» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://unity3d.com/unity/
3. «Apple iPhone Human Interface Guide» [Электронный ресурс]. – Режим доступа: http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/
4. «Apple iPhone Web Application Guide» [Электронный ресурс]. – Режим доступа: http://developer.apple.com/safari/library/documentation/InternetWeb/Conceptual/iPhoneWebAppHIG/
5. «The Objective-C Programming Language» [Электронный ресурс]. – Режим доступа: http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/
6. Гамма, Э. Приемы объектно-ориентированного проектирования / Э. Гамма, Р. Хелм. – СПб. : Питер, 2011. – 368с.
7. Глецевич, И. И. Методические указания по дипломному проектированию для студентов специальности 40 02 01 «Вычислительные машины, системы и сети» / И. И. Глецевич, В. А. Прытков, А. В. Отвагин. – Минск : БГУИР, 2009. – 99с.
8. Михнюк, Т. Ф. Охрана труда и основы экологии: Учебное пособие / Т. Ф. Михнюк. – Минск: Высшая школа, 2007.
9. Девисилов, В. Охрана труда: учебник / В. А. Девисилов. 2-ое изд. испр. и доп. – Москва: Форум, ИНФРА – Москва, 2006.
10. Хогдал, С. Анализ и диагностика компьютерных сетей / С. Хогдал. – М.: Лори, 2007. – 354 с.
11. Таненбаум, Э. Компьютерные сети / Э. Таненбаум. – СПб.: Питер, 2012. – 960 с.

# ПРИЛОЖЕНИЕ А.

# Листинг кода класса NetworkScaner

(NetworkScaner.cs)

class NetworkScaner

{

static bool traceOut = false;

static bool range = false;

static bool inited = false;

static int traceHops = 0;

static string rangeValue;

static IEnumerable<IPAddress> ipRange;

static object workerLocker = new object();

static int runningWorkers;

static ARPList ArpResults = new ARPList();

static TraceList Traceresults = new TraceList();

static string MACEmpty { get; set; }

public static void Main(string[] args)

{

InitializeArgs(args);

if (!inited)

return;

InitializeThreading();

if (range)

{

runningWorkers = ((List<IPAddress>)ipRange).Count;

int maxWorkers = runningWorkers;

for (int i = 0; i < runningWorkers; i++)

{

ThreadPool.QueueUserWorkItem(ARPSend, ((List<IPAddress>)ipRange)[i]);

}

lock (workerLocker)

{

while (runningWorkers > 0)

{

Monitor.Wait(workerLocker);

}

}

}

if (traceOut)

{

TraceRoute(Options.TraceHost, traceHops);

}

WriteDataToDisk();

}

private static void WriteDataToDisk()

{

if (ArpResults.List.Count > 0)

{

JsonWriter jw = new JsonWriter(Common.CommonData.ArpFile);

jw.Settings.PrettyPrint = true;

jw.Write(ArpResults);

jw.TextWriter.Close();

jw.TextWriter.Dispose();

}

if (Traceresults.List.Count > 0)

{

JsonWriter jw = new JsonWriter(Common.CommonData.TraceFile);

jw.Settings.PrettyPrint = true;

jw.Write(Traceresults);

jw.TextWriter.Close();

jw.TextWriter.Dispose();

}

File.Create(Common.CommonData.DoneFile);

}

private static void InitializeThreading()

{

ThreadPool.SetMaxThreads(10, 10);

}

private static void InitializeArgs(string[] args)

{

if (args.Length == 0)

{

Help();

inited = false;

return;

}

for (int i = 0; i < args.Length; i++)

{

switch (args[i])

{

case Options.TraceOption:

traceOut = true;

inited = true;

traceHops = int.Parse(args[i + 1]);

i++;

break;

case Options.RangeOption:

range = true;

inited = true;

rangeValue = args[i + 1];

IPRange r = new IPRange(rangeValue);

ipRange = r.GetAllIP();

i++;

break;

case Options.HelpOption:

Help();

inited = false;

return;

}

}

}

private static void ARPSend(object ip)

{

IPAddress addr = (IPAddress)ip;

byte[] ab = new byte[6];

int len = ab.Length;

int r = SendARP((int)addr.Address, 0, ab, ref len);

string mac = BitConverter.ToString(ab, 0, 6);

lock (workerLocker)

{

if (mac != MACEmpty)

{

lock (ArpResults)

{

ARPData data = new ARPData { IpAddress = ((IPAddress)ip).ToString(), MacAddress = mac };

ArpResults.List.Add(data);

Monitor.Pulse(ArpResults);

}

}

runningWorkers--;

Monitor.Pulse(workerLocker);

}

}

private static void TraceRoute(string ipAddressOrHostName, int maxHops)

{

{

PingReply pingReply = null;

using (Ping pingSender = new Ping())

{

PingOptions pingOptions = new PingOptions();

Stopwatch stopWatch = new Stopwatch();

byte[] bytes = new byte[32];

pingOptions.DontFragment = true;

pingOptions.Ttl = 1;

for (int i = 1; i < maxHops + 1; i++)

{

stopWatch.Reset();

stopWatch.Start();

pingReply = pingSender.Send(ipAddressOrHostName, 1000, new byte[32], pingOptions);

stopWatch.Stop();

if (pingReply.Status != IPStatus.TtlExpired && pingReply.Status != IPStatus.Success)

{

Console.WriteLine(string.Format("{0} \t{1}", i, pingReply.Status.ToString()));

}

else

{

IPHostEntry @ipHost = Dns.Resolve(pingReply.Address.ToString());

TraceData data;

if (ipHost.HostName != pingReply.Address.ToString())

{

data = new TraceData { Number = i, ReplyDelay = (int)stopWatch.ElapsedMilliseconds, IpAddress = pingReply.Address.ToString() };

}

else

{

data = new TraceData { Number = i, ReplyDelay = (int)stopWatch.ElapsedMilliseconds, IpAddress = pingReply.Address.ToString() };

}

Traceresults.List.Add(data);

}

if (pingReply.Status == IPStatus.Success)

{

break;

}

pingOptions.Ttl++;

}

}

}

}

private static void Help()

{

Console.WriteLine("-t [MaxHops] - TraceRoute with max hops parameter\n-r [Range] - Set IP Range to scan. For Example 192.168.1-5.1-254\n-h - This Help page");

}

}

# ПРИЛОЖЕНИЕ Б.

# Листинг кода класса GraphBuilder

(GraphBuilder.cs)

public class GraphBuilder

{

public GameObject computer;

public float elementSize;

public float circleOffset;

public Material lineMaterial;

public Color[] colors;

public GameObject[] routers;

public GameObject earth;

public Transform cam;

public UIManager manager;

private Dictionary<float, Vector3> circleRadiuses = new Dictionary<float, Vector3>();

private Dictionary<string, ARPList> subNets = new Dictionary<string, ARPList>();

private Vector3 gatewayPosition;

public void Build()

{

manager.panel02.transform.position = new Vector3(100000, 0, 0);

StartCoroutine(BuildNetwork());

}

private IEnumerator BuildNetwork()

{

yield return StartCoroutine(InitializeArpBuilder(ARPDataReader()));

yield return StartCoroutine(PlaceMainRouter());

yield return StartCoroutine(BuildTrace(TraceDataReader()));

}

private void OnApplicationQuit()

{

if (File.Exists(Common.CommonData.TraceFile))

{

File.Delete(Common.CommonData.TraceFile);

}

if (File.Exists(Common.CommonData.ArpFile))

{

File.Delete(Common.CommonData.ArpFile);

}

}

private IEnumerator BuildTrace(TraceList traceList)

{

for (int i = 0; i < traceList.List.Count; i++)

{

GameObject go = Instantiate(routers[1], new Vector3(gatewayPosition.x, gatewayPosition.y, gatewayPosition.z - 30 \* (i + 1)), routers[1].transform.rotation) as GameObject;

go.GetComponent<Router>().label.text = traceList.List[i].IpAddress;

GameObject line = new GameObject("Line");

LineRenderer lr = line.AddComponent<LineRenderer>();

lr.SetWidth(0.7f, 0.7f);

lr.material = lineMaterial;

lr.material.SetColor("\_TintColor", Color.white);

lr.SetPosition(0, go.transform.position);

lr.SetPosition(1, gatewayPosition);

yield return new WaitForSeconds(0.4f);

}

Instantiate(earth, new Vector3(gatewayPosition.x, gatewayPosition.y, gatewayPosition.z - 30 \* (traceList.List.Count + 1)), earth.transform.rotation);

GameObject line0 = new GameObject("Line");

LineRenderer lr0 = line0.AddComponent<LineRenderer>();

lr0.SetWidth(0.7f, 0.7f);

lr0.material = lineMaterial;

lr0.material.SetColor("\_TintColor", Color.white);

lr0.SetPosition(0, new Vector3(gatewayPosition.x, gatewayPosition.y, gatewayPosition.z - 30 \* (traceList.List.Count + 1)));

lr0.SetPosition(1, gatewayPosition);

}

private IEnumerator PlaceMainRouter()

{

float maxR = 0;

foreach (var item in circleRadiuses.Keys)

{

if (maxR < item)

{

maxR = item;

}

}

Vector3 routerPosition = new Vector3(circleRadiuses[maxR].x + maxR + circleOffset \* 3, circleRadiuses[maxR].y, circleRadiuses[maxR].z - 20);

GameObject go = Instantiate(routers[1], routerPosition, routers[1].transform.rotation) as GameObject;

go.GetComponent<Router>().label.text = "Gateway";

foreach (var item in circleRadiuses)

{

Vector3 center = item.Value;

GameObject line = new GameObject("Line");

LineRenderer lr = line.AddComponent<LineRenderer>();

lr.SetWidth(0.7f, 0.7f);

lr.material = lineMaterial;

lr.material.SetColor("\_TintColor", Color.white);

if (center.z - routerPosition.z < 20.5f)

{

lr.SetPosition(0, center);

lr.SetPosition(1, routerPosition);

}

else

{

lr.SetVertexCount(3);

lr.SetPosition(0, center);

lr.SetPosition(1, new Vector3(routerPosition.x, 0, center.z));

lr.SetPosition(2, routerPosition);

}

}

gatewayPosition = routerPosition;

yield return null;

}

private IEnumerator InitializeArpBuilder(ARPList arpList)

{

if (arpList == null)

yield break;

foreach (var node in arpList.List)

{

string[] ipParts = node.IpAddress.Split('.');

string key = string.Empty;

for (int i = 0; i < Context.subNetParts; i++)

{

key += ipParts[i] + ".";

}

key = key.Remove(key.Length - 1);

if (!subNets.ContainsKey(key))

{

subNets.Add(key, new ARPList());

subNets[key].List = new List<ARPData>();

}

subNets[key].List.Add(node);

}

}

private IEnumerator BuildArpCircle(ARPList nodes, Vector3 center, float radius, string key)

{

circleRadiuses.Add(radius, center);

float deltaAng = 360f / nodes.List.Count;

GameObject router = Instantiate(routers[0], center, routers[0].transform.rotation) as GameObject;

cam.position = new Vector3(router.transform.position.x, router.transform.position.y + 12, router.transform.position.z + 25);

router.GetComponent<Router>().label.text = "Network " + key;

router.name = key;

for (int i = 0; i < nodes.List.Count; i++)

{

GameObject go = Instantiate(computer, new Vector3(radius \* Mathf.Cos((deltaAng \* i) / 57) + center.x, 0, radius \* Mathf.Sin((deltaAng \* i) / 57) + center.z), Quaternion.identity) as GameObject;

go.transform.parent = router.transform;

go.name = nodes.List[i].IpAddress;

go.GetComponent<Computer>().AdressLabel.text = string.Format("IP {0}\nMAC {1}", nodes.List[i].IpAddress, nodes.List[i].MacAddress);

LineRenderer line = go.AddComponent<LineRenderer>();

line.SetWidth(0.3f, 0.3f);

line.SetPosition(0, go.transform.position);

line.SetPosition(1, center);

line.material = lineMaterial;

line.material.SetColor("\_TintColor", colors[Random.Range(0, 69)]);

yield return new WaitForSeconds(2f / nodes.List.Count);

}

}

private ARPList ARPDataReader()

{

if (File.Exists(Common.CommonData.ArpFile))

{

FileStream stream = File.Open(Common.CommonData.ArpFile, FileMode.Open);

TextReader txtReader = new StreamReader(stream);

JsonReader jr = new JsonReader(txtReader.ReadToEnd());

return jr.Deserialize<ARPList>();

}

else

{

return null;

}

}

private TraceList TraceDataReader()

{

if (File.Exists(Common.CommonData.TraceFile))

{

FileStream stream = File.Open(Common.CommonData.TraceFile, FileMode.Open);

TextReader txtReader = new StreamReader(stream);

JsonReader jr = new JsonReader(txtReader.ReadToEnd());

return jr.Deserialize<TraceList>();

}

else

{

return null;

}

}

# ПРИЛОЖЕНИЕ В.

# Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ Г.

# Ведомость документов