|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования МОсковский государственный технический университет имени баумана (Мгту им. н. э. баумана) |

|  |
| --- |
| Приложение - анализатор трафика  для Windows  "MyWireshark" |

Пояснительная записка

Автор: Стаселович Олег

Группа: ИУ8-33

Москва, 2015

Приложение "MyWireshark" является программой, носящей познавательный характер.

Актуальность заключается в том, что сегодня сетевые технологии все больше входят в нашу жизнь, и умение работать с сетью очень важно для любого, кто работает в IT-сфере.

Цель: создание приложения для операционной системы Windows на языке C# с возможностью перехвата сетевого трафика.

Решения, присутствующие на рынке:

1. Wireshark
2. tcpdump
3. CommView
4. IRIS

Плюсы:

* Интуитивное управление
* Удобный интерфейс
* Дополнительный функционал в виде возможности определения MAC-адреса посылкой ARP-запроса

Минусы:

* Отсутствие продвинутого функционала, имеющегося у других подобных программ
* Возможность использования в преступных целях
* Поддержка одной платформы

Задачи:

1) Формирование требований к программе;

2) Разработка концепции программы;

3) Создание блок-схемы, демонстрирующей игровой процесс;

4) Разработка технического проекта;

5) Создание документации к программе;

6) Ввод программы в действие.

Ход работы:

1) Формирование требований к программе. Основные требования к программе - понятный для пользователя интерфейс и необходимый функционал.

2) Разработка концепции программы. Во время разработки было решено, что программа будет иметь одну главную форму и одну вспомогательную для отправки ARP-запроса. Главная форма будет включать в себя список возможных устройств для захвата пакетов, список захваченных пакетов, а также разбор выбранного пакета.

3) Создание блок-схемы, описывающей рабочий процесс. Блок-схема представлена в одноименном файле. Было решено, что разбор захваченных пакетов будет происходить до транспортного уровня включительно (то есть канальный, сетевой, транспортный уровни).

4) Разработка технического проекта. Основной частью создания приложения является процесс написания кода. В качестве используемой библиотеки для захвата пакетов была выбрана библиотека SharpPcap. SharpPcap — библиотека для .NET, которая позволяет перехватывать пакеты. По сути, это обертка над библиотекой Pcap, которая используется во многих популярных продуктах (например, в Wireshark). Вместе с SharpPCap также поставляется библиотека для парсинга пакетов — Packet.Net

Packet.Net поддерживает следующий протоколы:

* Ethernet
* LinuxSLL
* Ip (IPv4 and IPv6)
* Tcp
* Udp
* ARP
* ICMPv4 и ICMPv6
* IGMPv2
* PPPoE
* PTP
* Link Layer Discovery Protocol (LLDP)
* Wake-On-LAN (WOL)

Для написания кода приложения на языке C# была выбрана интерактивная среда разработки Microsoft Visual Studio 2013. К ее преимуществам можно отнести:

* редактор исходного кода с поддержкой технологии IntelliSense;
* возможность простейшего рефакторинга кода.
* быстрое создание высококачественного кода приложений для платформы Windows.

4.1) Немного теории сетей и сетевых протоколов.

Каждый раз, когда данные передаются по сети, они как-то перетекают от одного компьютера к серверу или другому компьютеру. Это происходит с помощью множества различных протоколов.

В соответствии с сетевой моделью взаимодействия открытых систем (OSI - Open Systems Interconnection), созданной Международной организацией по стандартам, сетевое взаимодействие делится на семь уровней:

• *Физический уровень.* - передача битов по физическим каналам (коаксиальный кабель, витая пара, оптоволоконный кабель). Здесь определяются характеристики физических сред и параметры электрических сигналов.

• *Канальный уровень* - передача кадра данных между любыми узлами сетях типовой топологии или соседними узлами произвольной топологии. В качестве адресов на канальном уровне используются MAC-адреса.

• *Сетевой уровень* - доставка пакета любому узлу в сетях произвольной топологии. На этом уровне нет никаких гарантий доставки пакета.

• *Транспортный уровень* - доставка пакета любому узлу с любой топологией сети и заданным уровнем надёжности доставки. На этом уровне имеются средства для установления соединения, буферизации, нумерации и упорядочивания пакетов.

• *Сеансовый уровень* - управление диалогом между узлами. Обеспечена возможность фиксации активной на данный момент стороны.

• *Уровень представления* - здесь возможно преобразование данных (шифрование, компрессия).

• *Прикладной уровень* - набор сетевых сервисов (FTP, E-mail и др.) для пользователя и приложения.

Более подробно про модель OSI можно почитать здесь:

<https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI>

Как работает протокол по этой модели? Все начинается с прикладного уровня. Пакет попадает на этот уровень, и к нему добавляется заголовок. После этого прикладной уровень отправляет этот пакет на следующий уровень (уровень представления). Здесь ему также добавляется свой собственный заголовок, и пакет отправляется дальше. И так до физического уровня, который занимается непосредственно передачей данных и отправляет пакет в сеть.

Другая машина, получив пакет, начинает обратный отсчет. Пакет с физического уровня попадает на канальный. Канальный уровень убирает свой заголовок и поднимает пакет выше (на уровень сети). Уровень сети убирает свой заголовок и поднимает пакет выше. Так пакет поднимается до уровня приложения, где остается чистый пакет без служебной информации, которая была прикреплена на исходном компьютере перед отправкой пакета.

Передача данных не обязательно должна начинаться с седьмого уровня. Если используемый протокол работает на четвертом уровне, то процесс передачи начнется с него, и пакет будет подниматься вверх до физического уровня для отправки. Количество уровней в протоколе определяет его потребности и возможности при передаче данных.

Чем ниже находится протокол (ближе к прикладному уровню), тем больше у него возможностей и больше накладных расходов при передаче данных (длиннее и сложнее заголовок). Рассматриваемые в данной книге протоколы будут находиться на разных уровнях, поэтому будут иметь разные возможности.

*Протокол IP*

Из модели OSI видно, что протокол IP находится на сетевом уровне. Из этого можно сделать вывод, что IP выполняет сетевые функции — доставка пакета любому узлу в сетях произвольной топологии.

Протокол IP при передаче данных не устанавливает виртуального соединения и использует датаграммы (пакеты данных) для отправки информации от одного компьютера к другому. Это значит, что по протоколу IP пакеты просто отправляются в сеть без ожидания подтверждения о получении данных, а значит, без гарантии доставки пакетов и, соответственно, без гарантии целостности данных. Если хотя бы один пакет из 100 необходимых не дойдет до адресата, то данные нарушатся, и собрать их в единое целое будет невозможно. Все необходимые действия по подтверждению и обеспечению целостности данных должны обеспечивать протоколы, работающие на более высоком уровне. Каждый IP-пакет содержит адреса отправителя и получателя, идентификатор протокола, TTL (время жизни пакета) и контрольную сумму для проверки целостности пакета. Как видите, здесь есть контрольная сумма, которая все же позволяет узнать целостность пакета. Но об этом узнает только получатель. Когда компьютер-получатель принял пакет, то он проверяет контрольную сумму только для себя. Если сумма сходится, то пакет обрабатывается, иначе просто игнорируется. А компьютер-отправитель не сможет узнать об ошибке, возникшей в пакете, и повторить посылку. Именно поэтому соединение по протоколу IP нельзя считать надежным.

*Транспортные протоколы.*

На транспортном уровне имеется два основных протокола: UDP и TCP. Оба они работают поверх IP. Это значит, что, когда пакет TCP или UDP опускается на уровень ниже для отправки в сеть, он попадает на уровень сети к протоколу IP. Здесь пакету добавляется сетевой адрес, TTL и другие атрибуты протокола IP. После этого пакет идет дальше вниз для физической отправки в сеть. Голый пакет TCP не может быть отправлен в сеть, потому что он не имеет информации о получателе, эта информация добавляется к пакету с IP-заголовком на уровне сети.

*UDP*

Как и IP, протокол UDP для передачи данных не устанавливает соединения с сервером. Данные просто выбрасываются в сеть, и протокол даже не заботится о доставке пакета. Если данные на пути к серверу испортятся или вообще не дойдут, то отправляющая сторона об этом не узнает. Так что по этому протоколу, как и по голому IP, не желательно передавать очень важные данные.

Благодаря тому, что протокол UDP не устанавливает соединения, он работает очень быстро (в несколько раз быстрее TCP). Из-за высокой скорости его очень удобно использовать там, где не нужно заботиться о целостности данных. Таким примером могут служить радиостанции в Интернете. Звуковые данные просто выплескиваются в глобальную сеть, и если слушатель не получит одного пакета, то максимум, что он заметит — небольшое заикание в месте потери. Но если учесть, что сетевые пакеты имеют небольшой размер, то эта задержка будет практически незаметна.

Большая скорость — большие проблемы с безопасностью. Так как нет соединения между сервером и клиентом, то нет никакой гарантии в достоверности данных. Протокол UDP больше подвержен спуфингу (подмена адреса отправителя), поэтому построение на нем защищенных сетей затруднено.

Итак, UDP очень быстр, но его можно использовать только там, где данные не имеют высокой ценности (возможна потеря отдельных пакетов) и не секретны (UDP больше подвержен взлому).

*TCP*

Протокол TCP лежит на одном уровне с UDP и работает поверх IP, который используется для отправки данных. Именно поэтому протоколы TCP и IP неразрывно связаны и их часто объединяют одним названием TCP/IP.

В отличие от UDP-протокол TCP устраняет недостатки своего транспорта (IP). В этом протоколе заложены средства установления связи между приемником и передатчиком, обеспечение целостности данных и гарантии их доставки.

Когда данные отправляются в сеть по TCP, то на отправляющей стороне включается таймер. Если в течение определенного времени приемник не подтвердит получение данных, то будет предпринята еще одна попытки отправки данных. Если приемник получит испорченные данные, то он сообщит об этом источнику и попросит снова отправить испорченные пакеты. Благодаря этому обеспечивается гарантированная доставка данных.

Когда нужно отправить сразу большую порцию данных, не вмещающихся в один пакет, то они разбиваются на несколько TCP-пакетов. Пакеты отправляются порциями по несколько штук (зависит от настроек стека). Когда сервер получает порцию пакетов, то он восстанавливает их очередность и собирает данные вместе (даже если пакеты прибыли не в том порядке, в котором они отправлялись).

Из-за лишних накладных расходов на установку соединения подтверждение доставки и повторную пересылку испорченных данных протокол TCP намного медленней UDP. Зато TCP можно использовать там, где нужна гарантия доставки и большая надежность. Хотя надежность нельзя назвать сильной (нет шифрования, сохраняется возможность взлома), но она приемлемая и намного больше, чем у UDP. По крайней мере, тут спуфинг не может быть реализован так просто, как у UDP.

*Протокол ARP*

Протокол сетевого уровня ARP (Address Resolution Protocol, протокол определения адреса) предназначен для определения аппаратного (MAC) адреса компьютера в сети по его IP-адресу. Прежде чем данные могут быть посланы на какой-нибудь компьютер по локальной сети, отправитель должен знать аппаратный адрес получателя. Именно для этого и предназначен ARP.

Когда компьютер посылает ARP-запрос на поиск аппаратного адреса, то протокол сначала ищет этот адрес в локальном кэше. Если уже были обращения поданному IP-адресу, то информация о MAC-адресе должна сохраниться в кэше. Если ничего не найдено, то в сеть посылается широковещательный запрос с вопросом, чей это адрес, который получают все компьютеры сети. Они получат этот пакет и проверят адрес. Тот, кому принадлежит искомый IP, ответит на запрос, указав свой MAC-адрес. Так как этот адрес должен быть уникальным (прошивается в сетевом устройстве на заводе-изготовителе), то и ответ должен быть один. Но нужно учитывать, что есть средства подделки MAC-адресов, и может возникнуть ситуация, когда ответ придет от двух машин. В приложении существует возможность получить MAC-адрес устройства путем отправки ARP-запроса с указанием IP-адреса.

4.2) (BackgroundThreadFunc) - ссылка на метод синхронизации интерфейса(который будет выполняться в отдельном потоке).

4.2) Класс MessageService.

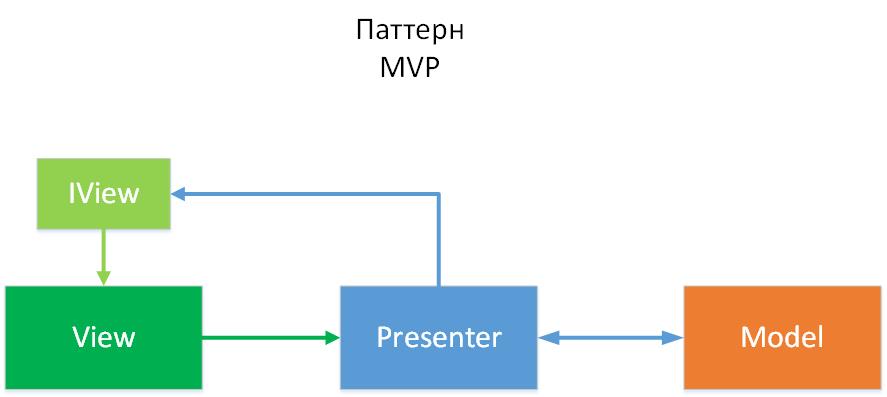
4.3) Для отправки ARP-запроса в приложении была создана форма ARPResolveForm, которая вызывается из пункта меню Tools. В коде формы создается класс библиотеки SharpPcap ARP. У класса есть конструктор, принимающий [SharpPcap.LibPcap.LibPcapLiveDevice](http://sharppcap.sourceforge.net/htmldocs/SharpPcap.LibPcap/LibPcapLiveDevice.html), который создает новый ARP "распознаватель".

Далее используется метод класса ARP.Resolve, принимающий IP-адрес, который распознает MAC-адрес устройства по принятому IP-адресу. Если устройство по такому IP-адресу найдено его MAC-адрес выводится в текстовую форму.

4.4) В приложении использован шаблон проектирования C#-приложений MVP (Model-View-Presenter).

Любой шаблон проектирования представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования. В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению свое имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Model-View-Presenter - это шаблон, который используется в основном для построения пользовательского интерфейса. На следующей блок-схеме показана упрощенная структура MVP

****

Создаваемый код состоит из трех больших частей:

* *Модель* (*Model*) — предоставляет *данные* для пользовательского интерфейса, реализует в себе бизнес-логику приложения
* *Представление* (*View*) — реализует *отображение* данных (Модели) и маршрутизацию пользовательских команд или событий Presenter-у, это все элементы графического интерфейса - формы, сообщения, любое взаимодействие с пользователем осуществляется через Представление.
* *Presenter* — управляет Моделью и Представлением. Например извлекает данные из Модели и форматирует их для отображения в Представлении.

Особенностью реализации является то, что View кроме как взаимодействия с пользователем и отрисовкой элементов управления больше ничем не занимается, он является пассивной частью, уведомляет управляющий код (Presenter), что с ней взаимодействовал пользователь и дает необходимые данные, в обратную сторону отображает те элементы управления, которые ей дает управляющий код, но само View ничего не решает и не содержит никакой логики.

Presenter - часть, которая содержит управляющий код, Presenter управляет как Моделью, так и Представлением, реализует в себе все процессы, происходящие в приложении, Presenter соединяет воедино бизнес-логику и Представление, которое реализует графический интерфейс. Model реализует ту основу, ради которой создается приложение, его ядро. По шаблону MVP Presenter обращается к Модели, Модель не знает о существовании Presenter'а, Presenter использует ее методы и команды, Модель ничего не решает и выполняет только те функции, которые требует от нее Presenter. View реализует интерфейс IView, в который вносятся все те члены типа, которые нужны Presenter'у и только, Presenter получает ссылку на этот интерфейс, и через него взаимодействует с View. Так достигается удобное разбиение кода на логически части. Если в дальнейшем будут изменения, легко увидеть ту часть, в которую нужно вносить изменения. Presenter обращается как к Модели так и к Представлению.

Объявляется интерфейс IMainForm в пространстве имен GUI. Presenter имеет доступ к FilePath, который возвращает тот путь к файлу который выбрал пользователь, оно только для чтения, так как презентер не может менять его. Метод SetSymbolCount - устанавливает количество символов, которое содержит файл, далее - 3 события, через которые форма уведомляет презентер, что с ней произошло. во-первых, она скажет презентеру, что пользователь нажал на кнопку открытия файла, далее - сохранения файла, в -третьих, что пользователь внес изменение в содержимое файла.

Начнем реализацию интерфейса. Для начала объявим те свойства, через которые презентер будет взаимодействовать с формой. По сути они являются экранированиям обращения к элементам управления. и это правильно, презентер не должен знать что он обращется к каким-то элементам управления, он полностью отвязан от графического слоя. Свойства ... делают то-то Закончим реализацию интерфейса, объявив события.

Далее при инициализации у конпки butFileOpen выбираем событие Click и через += создаем обработчик события и привязываем его к этому событию. как только пользователь кликнет на кнопку ОткрытьФайл мы попадем в butOpenFile\_Click, в этот элемент кода. Здесь вызываем событие FileOpenClick, чтобы о нем узнал внешний код. Это делается вот так if (FileOpenClick != null) FileOpenClick(this, EventArgs.Empty); .

Пользователь кликает на кнопку, мы попадаем в обработчик этого клика. В нем мы проверяем, есть ли подписчики у нашего собственного события FileOpenClick, если есть, мы его вызываем(пробрасываем). Остальные аналогично. Затем мы транслируем наружу события открытия, сохраниения и изменения содержимого.

Далее мы будем реализовывать собственный код формы. Далее мы должны обработать события клика по кнопке. Мы обрабатываем клик по кнопке Выбрать Файл. После чего привяжем к собственному событию Клик

Класс MessageService в пространстве имен GUI - это своеобразная обертка над классом MessageBox, которая создана для того, чтобы любой код мог получить возможность отображать сообщения вне зависимости от части, в которой он находится. Также объявлен интерфейс IMessageService. Любой класс, который будет манипулировать интерфейсом не будет знать, как именно реализован вывод сообщений (он может быть реализован, например, с помощью MessageBox'ов, как в программе, или в специальном поле для сообщений). Этот класс нужен для того чтобы Presenter мог вывести пользователю какое-либо сообщение через графический интерфейс. Так как Presenter не имеет доступа к графическому слою, тут ему и понадобится класс MessageService и интерфейс IMessageService.

Перейдем к классу FileManager. Здесь мы тоже должны сделать некоторую абстракцию, то есть объявить интерфейс IFileManager и заставить класс его реализовывать. Зачем. Презентеру все равно как осуществляется сохранение и получение содержимого. Ничто не мешает хранить файл в базе данных, тогда будет другой файловый менеджер, который будет работать с базой данных, но тоже будет реализовывать интерфейс IFileManager. Также мы можем разместить все это на удаленном веб сервере, у нас будет веб-клиент, который будет работать с текстом, но также реализовывать IFileManager и все его методы. Мы абстрагируемся от конкретной реализации таким способом.

Создадим Презентер, который соединит все вместе.

Мы создаем конструктор презентера и передаем ссылки на все 3 наших интерфейса, мы не можем создать презентре без ссылки на реализации наших интерфейсом. Это обеспечит гибкость создаваемого приложения, мы можем передать любую реализацию нашего интерфейса.

Далее мы устанавливаем параметр количества символов по умолчанию равным нулю. Далее мы реализуем подписку на все события, которые нам доступны(3 события). Создаем обработчик ContentChanged. Как только у нас возникает событие ContentChanged, оно возбуждается на стороне формы, оно сообщает всем подписчикам(презентеру), что содержимое изменилось. Презентер через свойство Content формы получает это содержимое. Далее через ссылку на менеджер с помощью метода GetSymbolCount получает количество символов в содержимом. И с помощью метода SetSymbolCount устанавливает это количество на форме.

Объявляем следующее событие FileOpenClick. Оно возбуждается формой когда пользователь желает открыть указанный файл. Далее проверяем существует ли этот файла в нашей системе. Если файла не обнаруживается, мы прибегаем к нашему сервису сообщений и выдаем предупреждение и прекращаем выполнение обработчика. Если существует то делаем то то, то то и то то

Объявляем строковую переменную currentfilepath. Она нужна чтобы хранить ссылку на наш открытый файл, чтобы знать куда сохранять. Весь управляющий код находится в презентере. Если мы получаем ошибку(например файл закрыт для чтения) и выдаем ошибку.

Последний метод - метод сохранения файла. Подписываемся на событие. View реализуется формой. Filesave click обработчик реализуется так... Когда идет сохранение файла презентер обращается к методу сэйв контент и передает путь который мы занесли заранее. кодировка по умолчанию Windows-кодировка. После сообщаем пользователю, что файл сохранен.

Осталось запустить весь наш код. Переходим в класс програм.кс. Нас интересует 3 строчка. Она запускает цикл обработки сообщений виндовс на нужной форме до тех пор, пока форма, передаваемая как аргумент. не закроется. Мы объявляем 3 экземпляра: формы сервиса и файлового менеджера. Далее создаем Presenter, куда и передаем все наши3 экземпляра и запускаем с заранее созданной формы.

5) Создание документации к программе. После завершения работы над техническим заданием была начата работа над документацией. В неё входят:

* Руководство пользователя
* Пояснительная записка
* Блок-схема работы программы

6) Ввод программы в действие. Программа была введена в действие после успешной компиляции кода.

**Список литературы**

1. MSDN <https://msdn.microsoft.com/ru-ru/>
2. SharpPcap Tutorial <http://www.codeproject.com/Articles/12458/SharpPcap-A-Packet-Capture-Framework-for-NET>
3. Код библиотеки SharpPcap <http://sourceforge.net/projects/sharppcap/files/>
4. Код библиотеки Packet.Net <http://sourceforge.net/projects/packetnet/files/?source=navbar>
5. Интернет-портал <https://stackoverflow.com/>
6. Интернет-портал <http://rsdn.ru/>
7. Интернет-энциклопедия <https://ru.wikipedia.org/>
8. Гугл <https://www.google.com/>
9. В. Олифер, Н. Олифер - Компьютерные сети (2-е издание)
10. Г. Шилдт - Полный справочник по C#