Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

АНАЛИЗАТОР СЕТЕВОГО ТРАФИКА

БГУИР КП 1-40 02 01 120 ПЗ

Студент: гр. 150501 Почебут А.С.

Руководитель: старший преподаватель кафедры ЭВМ Поденок Л.П.

Минск 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____Б. В. Никульшин
(подпись)
_____ 2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Почебут Анастасии Сергеевне*

1. Тема проекта *Анализатор сетевого трафика*
2. Срок сдачи студентом законченного проекта *17 мая 2023 г.*
3. Исходные данные к проекту *Язык программирования C, ОС Linux*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке) *Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Результаты работы. Заключение. Список использованных источников.*
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) *1. Схема структурная 2. Схема алгоритма print ethernet header 3. Схема алгоритма print udp packet.*
6. Консультант по проекту *Поденок Л.П.*
7. Дата выдачи задания *24 февраля 2023 г.*
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

выбор задания, раздел1 к 1 марта 2023 г. – 15 %;
разделы 2,3 к 1 апреля 2023 г. – 50 %;
разделы 4,5 к 1 мая 2023 г. – 80 %;
оформление пояснительной записки и графического материала к 15 мая 2023 г. – 100 %
Защита курсового проекта с 29 мая 2023 г. по 10 июня 2023 г.

РУКОВОДИТЕЛЬ_____ *Л.П. Поденок*
(подпись)

Задание принял к исполнению _____ *А.С. Почебут*
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В терминологии системных администраторов и специалистов по информационной безопасности часто встречается понятие – «анализаторы трафика». Под анализатором трафика понимается устройство или программа, которая перехватывает трафик и затем его анализирует. Такие продукты полезны в тех случаях, когда нужно извлечь из потока данных какие-либо сведения (например, пароли) или провести диагностику сети. Современные коммерческие анализаторы выпускаются как в виде программных решений, так и в виде аппаратных устройств и служат для комплексного анализа производительности крупных информационных сетей, а также пользовательских приложений.

Анализаторы сетевого трафика – это новая категория продуктов безопасности, которые используют сетевые коммуникации в качестве основного источника данных для обнаружения и расследования угроз безопасности, аномального или вредоносного поведения в сети. Анализаторы, или так называемые снифферы (от англ. `to sniff` – нюхать), передают всю необходимую информацию о событиях, происходящих внутри сети, в центры мониторинга и реагирования.

Есть несколько веских причин для мониторинга общего трафика в сети. Информация, полученная с помощью инструментов мониторинга сетевого трафика, позволяет администратору решать следующие задачи:

- находить проблемы в работе сети (задержки в передаче информации) и быстро их устранять;
- обнаруживать шпионские программы, взломы и другую постороннюю активность (несанкционированный доступ злоумышленников);
- анализировать работоспособность пользовательских приложений,
- собирать статистику;
- определять ведомственную пропускную способность.

Сниффер может анализировать только то, что проходит через его сетевую карту. Внутри одного сегмента сети Ethernet все пакеты рассылаются всем машинам, из-за этого возможно перехватывать чужую информацию. Между сегментами информация передаётся через коммутаторы. Использование коммутаторов и их грамотная конфигурация уже является защитой от прослушивания. Коммутация пакетов – форма передачи, при которой данные, разбитые на отдельные пакеты, могут пересылаться из исходного пункта в пункт назначения разными маршрутами. Так что если кто-то в другом сегменте посылает внутри него какие-либо пакеты, то в ваш сегмент коммутатор эти данные не отправит.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Понятие анализатора сетевого трафика

Анализатор трафика – это программа или устройство, которое позволяет анализировать сетевой трафик, проходящий через компьютерную сеть. Анализаторы трафика, или снифферы, используются для мониторинга и отслеживания сетевых активностей, обнаружения проблем и уязвимостей в сети, а также для решения различных задач в области безопасности и администрирования сети.

Основные составляющие анализатора трафика могут включать:

- захват и анализ трафика: анализаторы трафика могут захватывать и анализировать сетевой трафик, передаваемый между компьютерами в сети;

- фильтрация трафика: анализаторы трафика могут фильтровать трафик, чтобы отображать только определенные типы сетевой активности, например, только HTTP-запросы;

- декодирование протоколов: снифферы могут декодировать различные протоколы, используемые в сети, чтобы обеспечить более подробное представление о том, какие данные передаются между компьютерами;

- графический интерфейс пользователя: анализаторы трафика могут предоставлять графический интерфейс пользователя (GUI) для удобного просмотра и анализа данных;

- возможность сохранения и экспорта данных: анализаторы трафика могут сохранять данные в различных форматах и экспортировать их для дальнейшего анализа или использования в других приложениях.

Анализаторы трафика являются неотъемлемой частью современных сетей и позволяют обеспечивать безопасность, производительность и эффективность работы сети. Они используются для мониторинга сетевых активностей, обнаружения и предотвращения кибератак, анализа производительности и оптимизации сетевых ресурсов. Снифферы могут быть полезными как для крупных корпораций, так и для небольших организаций и даже домашних пользователей, которые желают обеспечить безопасность своих домашних сетей и улучшить их производительность.

Особенности анализаторов трафика:

- безопасность: анализаторы трафика могут использоваться для обнаружения уязвимостей и атак в сети, а также для мониторинга соответствия сетевой политики и правил безопасности;

- администрирование сети: снифферы могут помочь в администрировании сети, позволяя управлять трафиком, оптимизировать производительность и обеспечивать соответствие сетевой политике:

- разработка и тестирование сетевых приложений: анализаторы трафика могут быть использованы для разработки и тестирования сетевых приложений,

позволяя анализировать и отлаживать сетевой трафик, который передается между приложениями;

- решение проблем в сети: анализаторы трафика могут быть использованы для решения различных проблем в сети, таких как снижение производительности, отказы в работе и другие сбои;

- мониторинг сети: снифферы могут использоваться для мониторинга сетевых активностей, включая управление сетевой пропускной способностью, контроль за использованием ресурсов, управление сетевым доступом и т.д.;

- поддержка соответствия: анализаторы трафика могут помочь организациям удостовериться в соответствии различным регулирующим стандартам, например, стандартам PCI DSS, HIPAA и другим;

- улучшение производительности: анализаторы трафика могут помочь улучшить производительность сети, обеспечивая оптимальное использование ресурсов и предоставляя информацию для определения узких мест и проблем в сети;

- отладка сети: снифферы могут использоваться для отладки сетевых проблем, позволяя идентифицировать, где именно происходят ошибки и сбои в сети.

В целом, анализаторы трафика представляют собой важный инструмент для мониторинга, анализа и управления сетевыми активностями, обеспечивая безопасность, производительность и надежность сети. Основные функции и возможности анализаторов трафика могут варьироваться в зависимости от конкретных потребностей и сценариев использования, но в общем случае они могут включать в себя следующие аспекты:

- способность обрабатывать различные протоколы: анализаторы трафика должны иметь возможность обрабатывать различные протоколы, такие как HTTP, FTP, SMTP, TCP/IP, UDP, DNS и другие;

- функции фильтрации и поиска: снифферы должны иметь функции фильтрации и поиска, чтобы помочь пользователям находить необходимые данные и информацию в больших объемах трафика;

- режим захвата пакетов: анализаторы трафика должны иметь возможность захвата пакетов, передаваемых в сети, чтобы анализировать их содержимое и характеристики;

- отображение данных: анализаторы трафика должны предоставлять различные способы отображения данных, включая таблицы, графики и диаграммы, чтобы помочь пользователям визуализировать данные и информацию;

- удобный интерфейс: снифферы должны быть легкими в использовании и предоставлять удобный интерфейс для пользователя, чтобы снизить время обучения и повысить производительность;

- масштабируемость: анализаторы трафика должны иметь возможность масштабироваться для обработки больших объемов трафика и управления растущими сетевыми активностями;

- безопасность: анализаторы трафика должны быть безопасными для использования и обеспечивать конфиденциальность и защиту данных, передаваемых в сети;

- поддержка производительности: анализаторы трафика должны быть производительными и обеспечивать быстрое обнаружение и решение проблем в сети;

В зависимости от сценария использования, снифферы могут иметь дополнительные функции, такие как анализ потока данных, декодирование шифрованного трафика, определение топологии сети и др.

Принцип работы анализатора сетевого трафика заключается в захвате и анализе пакетов, передаваемых в сети. Анализатор сетевого трафика может быть реализован как программное или аппаратное средство и работать на уровне сетевого интерфейса. Когда пакеты проходят через интерфейс, анализатор сетевого трафика захватывает их и передает для анализа. После захвата пакетов, анализатор сетевого трафика разбирает их на составляющие, такие как источник и назначение, протокол, порт и другие атрибуты. Затем он анализирует содержимое каждого пакета и использует специальные алгоритмы для обнаружения аномальных и нежелательных пакетов. Общая схема работы анализатора сетевого трафика представлена на рисунке 1.1.

Рисунок 1.1 – Принцип работы анализатора сетевого трафика

Сниффер может также использоваться для отображения информации о сетевых узлах, скорости передачи данных, объеме передаваемых данных и других метриках производительности сети.

Анализаторы сетевого трафика обладают рядом преимуществ, которые делают их необходимыми инструментами для работы сетевых администраторов и специалистов в области информационной безопасности. Это мощный

инструмент, который помогает эффективно управлять сетями и защищать их. Снифферы обеспечивают надежную и безопасную работу сетей, а также позволяют оптимизировать использование сетевых ресурсов и повысить производительность приложений. Благодаря анализаторам сетевого трафика, сетевые администраторы и специалисты по кибербезопасности могут быстро реагировать на угрозы и проблемы в сети, что позволяет предотвратить серьезные последствия для бизнеса и пользователя.

1.2 Анализ существующих аналогов

Анализаторы сетевого трафика считаются незаменимыми инструментами для анализа и контроля сетей. Некоторые из этих инструментов предоставляют функции анализа трафика с помощью программного обеспечения, в то время как другие предлагают аппаратное обеспечение для мониторинга и контроля сети.

Несмотря на разнообразие анализаторов сетевого трафика и их специфические функции, все они имеют некоторые общие черты. Одна из главных – это возможность мониторинга и анализа трафика на различных уровнях OSI модели, начиная с физического и заканчивая прикладным уровнем. Кроме того, все анализаторы трафика обеспечивают возможность записи и воспроизведения трафика для дальнейшего анализа и отладки сетевых проблем. Также они обычно предоставляют различные виды отчетности и графические интерфейсы, которые позволяют быстро и удобно анализировать данные о трафике в режиме реального времени и на основе сохраненных данных. В этом контексте рассмотрим некоторые из аналогов анализатора сетевого трафика и их возможности.

1.2.1 Colasoft Capsa

Colasoft Capsa – это анализатор трафика локальной сети, который позволяет идентифицировать и отслеживать более 300 сетевых протоколов, создавать настраиваемые отчеты. Он включает в себя мониторинг электронной почты и диаграммы последовательности TCP-синхронизации, все это собрано в одной настраиваемой панели.

Приложение обладает широким набором функций, включая, например, отслеживание DoS/DDoS-атак, активности червей и обнаружение ARP-атак. Также доступны декодирование и захват пакетов, автоматическая диагностика, статистические данные о каждом хосте в сети, контроль обмена пакетами и расширенный анализ протоколов.

Программное обеспечение может быть развернуто в нескольких сценариях и потребностях использования, например, для устранения различных проблем, связанных с сетью, анализа производительности вашей сети и выявления узких мест, обнаружения вредоносных действий в сети, например, наличия вирусов

или червей, как отладка других подобных проблем. Окно программы представлено на рисунке 1.2.

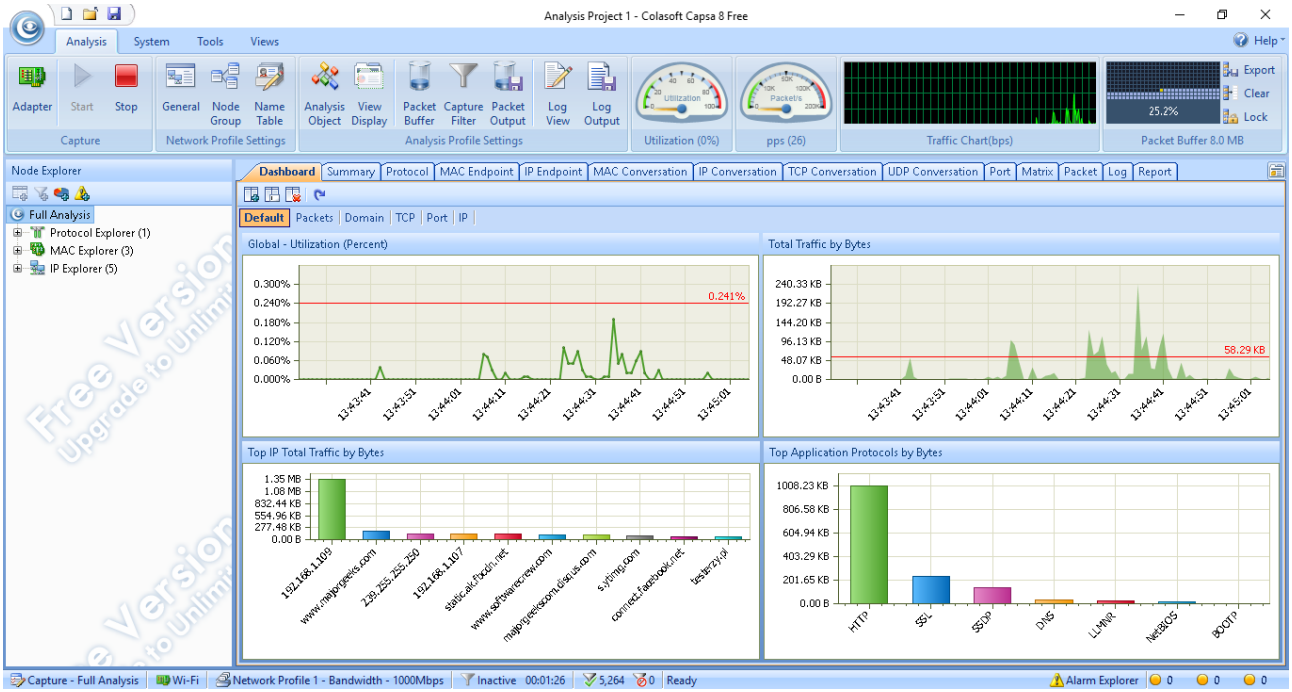


Рисунок 1.2 – Окно программы Colasoft Capsa

Основные характеристики программы Colasoft Capsa:

- поддержка ОС: все 32-битные и 64-битные версии Windows XP;
- минимальные системные требования: 2 Гб оперативной памяти и процессор 2,8 ГГц;
- хорошо работает со всеми проводными и беспроводными интерфейсами, включая Bluetooth, Wi-Fi и Ethernet (совместимый с NDIS 3 или выше).

1.2.2 SolarWinds NetFlow Analyzer

SolarWinds NetFlow Analyzer является одним из наиболее популярных программных инструментов для анализа сетевого трафика и мониторинга пропускной способности, который поддерживает работу с различными протоколами, включая: NetFlow, J-Flow, sFlow, IPFIX и NetStream. Он дает возможность сортировать, помечать и отображать данные различными способами. Это позволяет удобно визуализировать и анализировать сетевой трафик. Инструмент отлично подходит для мониторинга сетевого трафика по типам и периодам времени, а также для определения того, сколько трафика потребляют различные приложения.

Таким образом, с помощью SolarWinds NetFlow Analyzer можно изучать и устранять неполадки и спады в функционировании сети, а также определять пользователей, приложения и устройства, которые используют больше всего ресурсов сети. Окно программы представлено на рисунке 1.3.

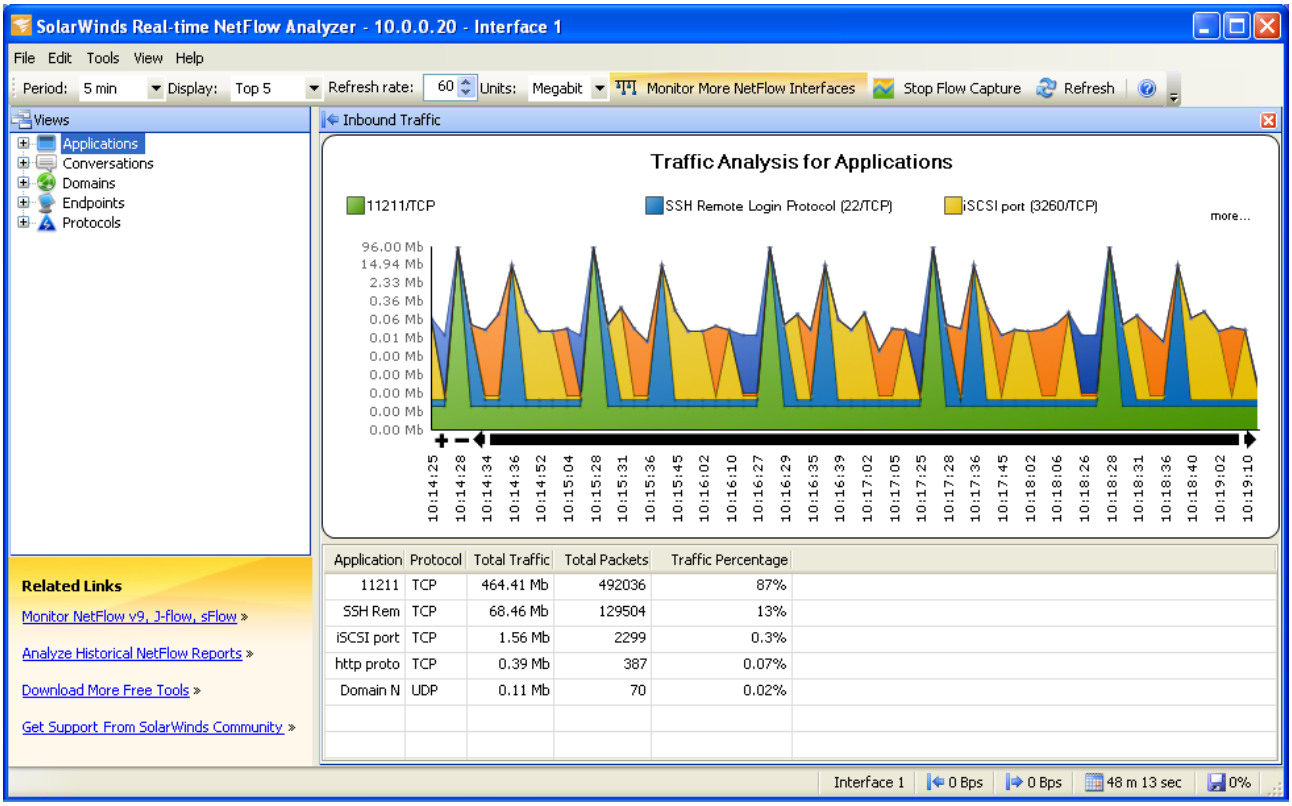


Рисунок 1.3 – Окно программы SolarWinds NetFlow Analyzer

Основные характеристики программы SolarWinds NetFlow Analyzer:

- использование баз данных SQL;
- поддержка ОС: Windows 10, 8, 7, Vista, XP;
- рекомендации скорости процессора, объему оперативной памяти и скорости сетевого адаптера варьируются в зависимости от количества отслеживаемых сетевых элементов.

1.2.3 Ntopng

Ntopng – это веб-инструмент с открытым исходным кодом для мониторинга и анализа сетевого трафика на основе данных о потоке и статистики, извлеченной из наблюдаемого трафика. Через зашифрованный интерфейс поступают данные о переданных пакетах. Программа анализирует их и предоставляет различную полезную информацию о функционировании вашей сети. Окно программы представлен на рисунке 1.4.

С помощью ПО Ntopng вы можете:

- сортировать информацию о сетевом трафике по многим параметрам, включая IP-адреса, порты, протоколы, пропускную способность и т. д.;
- в режиме реального времени отслеживать наиболее активные хосты и приложения, требующие больше всего пропускной способности;

- отслеживать информацию о переданных байтах, а также количество отправленных, полученных и потерянных пакетов;
- сохранять на диск историю сетевого трафика для последующего анализа;
- отображать задержку и статистику TCP (например, потерю пакетов)
- определять географическое местоположение и строить схемы соединения хостов на карте местности.

Поскольку Ntopng является открытым исходным кодом, существует множество возможностей для его расширения. Данные можно экспортировать в MySQL, ElasticSearch и LogStash, где они могут быть объединены в отчеты, хранящиеся на сервере Syslog.

Программа ntopng может подключаться к nProbe, который является коллектором NetFlow/IPFIX. В этом тандеме роли распределены следующим образом: nProbe собирает данные о потоках и отправляет эту информацию в Ntopng, который, в свою очередь, анализирует ее и представляет в удобном для восприятия виде.

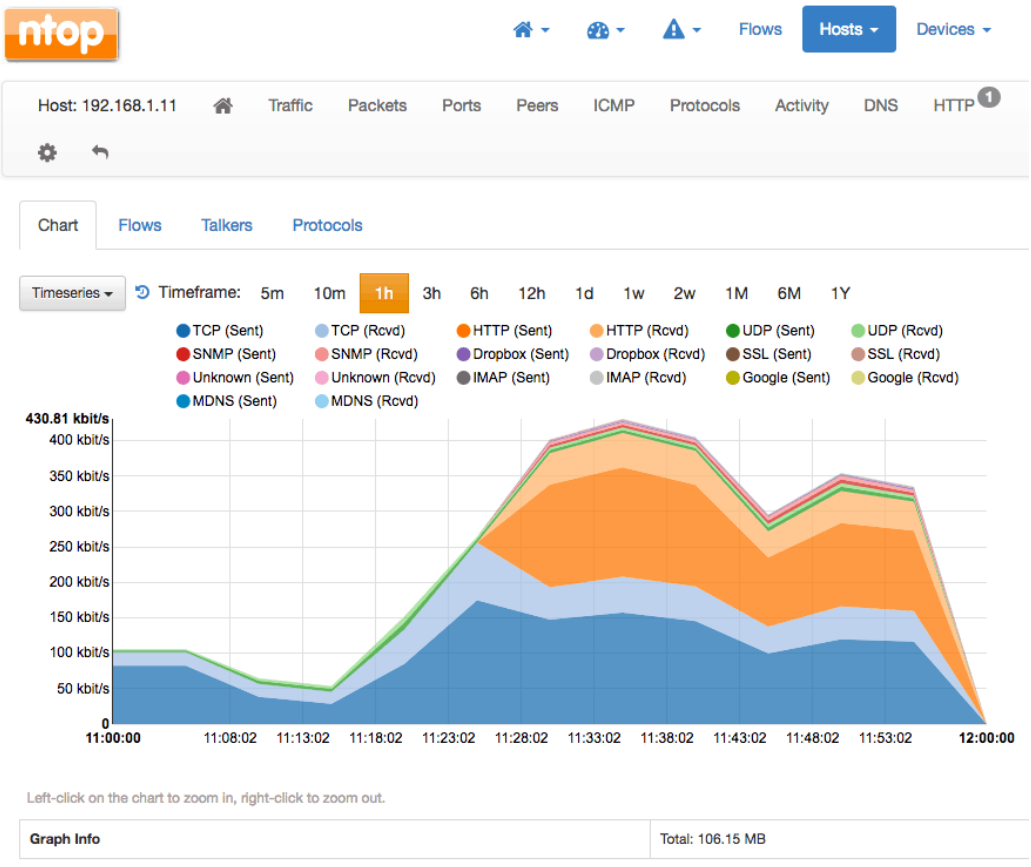


Рисунок 1.4 – Окно программы Ntopng

Основные характеристики:
- поддержка ОС: Linux, MacOS и Windows x64;
- полная поддержка IPv4 и IPv6;
- поддержка TLS/HTTPS;

- доступно через любой веб-браузер, поддерживающий HTML5.

1.3 Постановка задачи

Проанализировав множество аналогов программных инструментов, которые предоставляют пользователю возможность мониторить и анализировать сетевой трафик, можно сделать вывод, что сниффер требует достаточно много ресурсов для того, чтобы предоставлять такое количество информации. Также необходимо учитывать тот факт, что полученную информацию необходимо сохранять, сортировать и предоставлять пользователю в удобной для ознакомления и исследования форме.

Учитывая все вышеприведенные условия и особенности, в рамках данного курсового проекта будет реализован анализатора сетевого трафика, или сниффер, способный мониторить и анализировать сетевой трафик. Анализатор должен обладать функциями анализа и сохранения трафика, а также генерации отчетов на основе полученных данных. Будет рассмотрена возможность визуализации данных в удобной для пользователя форме. Важными аспектами для разработки также являются эффективность работы и поддержка различных протоколов, таких как TCP/IP, UDP, ICMP и другие.

В результате работы должен быть создан инструмент, который позволит эффективно мониторить и анализировать сетевой трафик в режиме реального времени. Основная функциональность программы разработана с использованием языка высокого уровня Си. В качестве операционной системы была выбрана Ubuntu 20.04, которая базируется на ядре Linux версии 5.4. Сетевое взаимодействие основано на сокетах, являющихся разновидностью сокетов Беркли, в частности raw сокеты (от англ. raw – сырой), с помощью которых можно получить доступ к протоколам сетевого уровня. В рамках данного курсового проекта сокеты работают с протоколами транспортного уровня TCP/UDP, а с помощью возможностей raw сокета так же анализируют протоколы сетевого уровня IP/ICMP.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Установка требований к функционалу разрабатываемой в рамках курсового проекта программе позволяет провести разделение всего алгоритма работы приложения на функциональные блоки. Функциональные блоки – это блоки программного компонента, которые ответственны за определенную задачу, а совокупность функциональных блоков позволяет реализовать полноценную работу программы. Наличие функциональных блоков позволит уменьшить количество времени на понимание внутреннего устройства программы, обеспечит гибкость и масштабируемость приложения с целью последующей возможной доработки путем добавления дополнительных программных блоков.

Программа анализатора сетевого трафика может быть разбита на модули, каждый из которых отвечает за определенную функцию. В рамках данного курсового проекта программа разделена на три основных модуля: модуль расшифровки, модуль анализа пакетов, модуль сохранения и визуализации данных. Каждый модуль, входящий в состав программы, играет важную роль в работе анализатора сетевого трафика и выполняет определенную функцию, которая влияет на точность, эффективность и удобство использования программы в целом. Все модули взаимодействуют между собой, чтобы улучшить работу сниффера и удовлетворить все запросы пользователя.

2.1 Модуль захвата, идентификации и подсчета пакетов

Программный модуль, выполняющий захват пакетов и идентификацию их типов. Также модуль реализует подсчет общего количества захваченных пакетов, что позволяет увидеть промежуточные данные.

В рамках модуля программа захватывает пакет при помощи сокета, определяет типа пакета в зависимости от протокола, которому принадлежит пакет, и увеличивает счетчик соответствующего типа, который затем выводит полученные данные в консоль.

2.2 Модуль расшифровки заголовков пакетов

Программный модуль, реализующий расшифровку `IP Header` и расшифровку `Ethernet Header` для последующего парсинга пакетов и анализа полученной информации.

В рамках модуля с помощью сокета открывается доступ к данным, хранящимся внутри вышеуказанных структур, что, в свою очередь, предоставляет возможность получения данных, необходимых для дальнейшей работы программы.

В результате в модуле расшифровки можно выделить два подмодуля:

- модуль расшифровки `IP Header`;

- модуль расшифровки `Ethernet Header`.

2.3 Модуль анализа протоколов

Программный модуль, анализирующий все данные протокола пакета, поступающие из трафика, и отбирающий только необходимые для работы программы параметры, используя определенные критерии. Данные критерии варьируются от пакета к пакету в зависимости от протокола.

Выделенный модуль выполняет декодирование сетевых протоколов, извлечение данных из заголовков пакетов и анализ содержимого пакетов для мониторинга трафика, выявления проблем в сети или аномалий в поведении приложений.

Каждый протокол имеет свои поля данных, поэтому сниффер считывает их в соответствии с заданными критериями, установленными непосредственно программой.

В результате в модуле анализа протоколов можно выделить три подмодуля:

- модуль анализа `TCP` протокола;
- модуль анализа `UDP` протокола;
- модуль анализа `ICMP` протокола;

2.4 Модуль сохранения и визуализации данных

Программный модуль, позволяющий отображать полученную информацию в удобном для пользователя формате, например, в виде графиков, таблиц и диаграмм. В рамках данного курсового проекта программа автоматически сохраняет информацию в текстовый файл в виде логов.

Таким образом, информация предоставляется в удобном пользователю виде. Он может отследить все пакеты, которые были проанализированы программы, изучить их заголовки и другие параметры обнаруженных протоколов.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается структура разрабатываемой в рамках курсового проекта программы с точки зрения описания данных и обрабатывающих их подпрограмм – функций.

3.1 Описание основных структур данных программы

Разработанная программа включает в себя следующие структуры данных:

- `struct sockaddr` – системная структура данных, описывающая сокет. Содержит следующие поля:
  - `u_short sa_family` – код семейства адресов, указывающий на тип адреса в зависимости от протокола;
  - `char sa_data` – массив, содержащий остальную часть адреса, необходимую для использования в конкретных сетевых протоколах.
- `struct sockaddr_in` – системная структура данных, описывающая сокет в домене `AF_INET`. Содержит следующие поля:
  - `short sin_family` – семейство адресов (по умолчанию устанавливается на значение `AF_INET`);
  - `u_short sin_port` – номер порта, на который должен быть отправлен пакет;
  - `struct in_addr sin_addr` – структура, содержащая `IP` адрес, по которому должен быть отправлен пакет;
  - `char sin_zero` – массив, используемый для дополнения структуры до размера структуры sockaddr;
- `struct iphdr` – системная структура данных, хранящая параметры `IP` заголовка. Содержит следующие поля:
  - `unsigned int ihl` – длина `IP` заголовка в 32-битных словах;
  - `unsigned int version` – версия протокола `IP`;
  - `u_int8_t tos` – тип сервиса (указания приоритета трафика);
  - `u_int16_t tot_len` – общая длина `IP` пакета в байтах, включая заголовок и данные;
  - `u_int16_t id` – идентификатор `IP` пакета;
  - `u_int16_t frag_off` – смещение фрагмента `IP` пакета в байтах от начала пакета;
  - `u_int8_t ttl` – время жизни (количество маршрутизаторов, которые пакет может проходить до его отбрасывания);
  - `u_int8_t protocol` – номер протокола верхнего уровня, который передает данные в IP пакете;
  - `u_int16_t check` – контрольная сумма заголовка `IP` пакета (проверка целостности пакета);
  - `u_int32_t saddr` – `IP` адрес отправителя пакета;
  - `u_int32_t daddr` – `IP` адрес получателя пакета.

- struct ethhdr – системная структура данных, хранящая параметры Ethernet заголовка. Содержит следующие поля:
    - unsigned char h_dest[ETH_ALEN] – массив байт длиной ETH_ALEN, содержащий MAC-адрес получателя Ethernet фрейма;
    - unsigned char h_source[ETH_ALEN] – массив байт длиной ETH_ALEN, содержащий MAC-адрес источника Ethernet фрейма;
    - be16 h_proto – двухбайтовое значение, определяющее тип протокола, который используется внутри данных пакета.
- struct tcphdr – системная структура данных, хранящая параметры TCP протокола. Содержит следующие поля:
    - u_int16_t source – порт отправителя;
    - u_int16_t dest – порт получателя;
    - u_int32_t seq – номер последовательности для передачи данных;
    - u_int32_t ack_seq – подтверждение получения последовательности;
    - u_int16_t doff – размер заголовка TCP протокола в 32-битных словах;
    - u_int16_t fin – флаг указывает, что отправитель закончил передачу данных;
    - u_int16_t syn – флаг указывает, что установлена новое соединение;
    - u_int16_t rst – флаг указывает на необходимость прервать соединение;
    - u_int16_t psh – флаг указывает, что следует отправить данные без ожидания заполнения буфера.;
    - u_int16_t ack – флаг указывает, что подтверждение номера последовательности действительно;
    - u_int16_t urg – флаг указывает, что присутствует важная информация, которая требует приоритетной обработки;
    - u_int16_t window – размер окна, который определяет количество данных, которые могут быть переданы отправителем до получения подтверждения;
    - u_int16_t check – контрольная сумма заголовка TCP протокола (проверка целостности данных);
    - u_int16_t urg_ptr – указатель на полезную нагрузку в TCP-сегменте, которая требует приоритетной обработки.
- struct udphdr – системная структура данных, хранящая параметры UDP протокола. Содержит следующие поля:
    - u_int16_t source – порт отправителя;
    - u_int16_t dest – порт получателя;
    - u_int16_t len – общая длина заголовка UDP протокола и данных пакета в байтах;

- u_int16_t check – контрольная сумма заголовка UDP протокола (проверка целостности данных);

- struct icmphdr – системная структура данных, хранящая параметры ICMP протокола. Содержит следующие поля:

- u_int8_t type – тип ICMP-сообщения;

- u_int8_t code – уточнения типа сообщения;

- u_int16_t checksum – контрольная сумма заголовка ICMP протокола (проверка целостности данных);

- u_int16_t id – идентификатор запроса (устанавливается отправителем, используется для передачи пакетов эхо-запроса и эхо-ответа);

- u_int16_t sequence – порядковый номер запроса (устанавливается отправителем, используется для передачи пакетов эхо-запроса и эхо-ответа);

- u_int32_t gateway – адрес шлюза (для сообщений, отправляемых через шлюз);

- u_int16_t mtu – максимальный размер пакета, который может быть передан по маршруту.

3.3 Описание основных функций программы

Разработанная программа состоит из следующих основных функций:

- void process_packet(unsigned char* buffer, int size) – функция определения протокола пакета, вызова соответствующей функции протокола и счетчика протоколов. Принимает буфер данных пакета buffer и размер принятых из пакета данных size;

- void print_ethernet_header(unsigned char* buffer, int size) – функция расшифровки Ethernet заголовка пакета. Принимает буфер данных пакета buffer и размер принятых из пакета данных size;

- void print_ip_header(unsigned char* buffer) – функция расшифровки IP заголовка пакета. Принимает буфер данных пакета buffer;

- void print_tcp_packet(unsigned char* buffer, int size) – функция считывания параметров TCP протокола пакета. Принимает буфер данных пакета buffer и размер принятых из пакета данных size;

- void print_udp_packet(unsigned char* buffer, int size) – функция считывания параметров UDP протокола пакета. Принимает буфер данных пакета buffer и размер принятых из пакета данных size;

- void print_icmp_packet(unsigned char* buffer, int size) – функция считывания параметров ICMP протокола пакета. Принимает буфер данных пакета buffer и размер принятых из пакета данных size;

- void print_data(unsigned char* data, int size) – функция записи полученных из пакетов данных в файл с результатами. Принимает данные пакета data и размер принятых из пакета данных size.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе представлены схемы алгоритмов и алгоритмы по шагам основных функций разработанной в рамках курсового проекта сетевого программы.

4.1 Разработка схем алгоритмов

Функция `print_ethernet_header(unsigned char* buffer, int size)` реализует расшифровку `Ethernet` заголовка пакета. Схема алгоритма данной функции показана в приложении Б.

Функция `print_udp_packet(unsigned char* buffer, int size)` реализует считывание параметров `UDP` протокола пакета. Схема алгоритма данной функции показана в приложении В.

4.2 Разработка алгоритмов

4.2.1 Алгоритм расшифровки IP заголовка

Функция `print_ip_header(unsigned char* buffer, int size)` – на вход поступает массив данных пакета и размер массива. Шаги алгоритма:

1) Начало.

2) Вызов функции `print_ethernet_header()` для вывода `Ethernet` заголовка пакета.

3) Определение переменной `iphdrlen` типа `unsigned short` для хранения размера `IP` заголовка пакета.

4) Инициализация указателя на структуру `IP` заголовка, используя указатель на начало буфера `buffer` и размер структуры `Ethernet` заголовка.

5) Определение длины `IP` заголовка путем умножения поля `ihl` на 4 (каждое значение поля `ihl` представляет длину в 32-битных словах).

6) Обнуление структуры `source` и заполнение поля `sin_addr.s_addr` значением поля `saddr` из `IP` заголовка.

7) Обнуление структуры `dest` и заполнение поля `sin_addr.s_addr` значением поля `daddr` из `IP` заголовка.

8) Вывод значений полей `IP` заголовка пакета в файл `logfile` с помощью функции `fprintf()`. Каждое поле выводится на новой строке.

9) Конец.

4.2.2 Алгоритм анализа TCP протокола

Функция `print_tcp_header(unsigned char* buffer, int size)` – на вход поступает массив данных пакета и размер массива. Шаги алгоритма:

1) Начало.

2) Определение переменной iphdrlen типа unsigned short для хранения размера IP заголовка пакета.

3) Инициализация указателя на структуру IP заголовка, используя указатель на начало буфера buffer и размер структуры Ethernet заголовка.

4) Определение длины IP заголовка путем умножения поля ihl на 4 (каждое значение поля ihl представляет длину в 32-битных словах).

5) Инициализация указателя на структуру заголовка TCP пакета, используя вычисленную длину IP заголовка, указатель на начало буфера buffer и размер структуры Ethernet заголовка.

6) Вычисление размера заголовка TCP пакета, используя размер структуры Ethernet заголовка, размер IP заголовка и размер поля doff структуры tcph.

7) Вызов функции print_ip_header() для вывода IP заголовка.

8) Вывод значений полей заголовка TCP пакета в файл logfile с помощью функции fprintf(). Каждое поле выводится на новой строке.

9) Вывод дампа памяти всех заголовков в файл logfile, используя функцию print_data() и размеры заголовков.

10) Конец.

5 РЕЗУЛЬТАТЫ РАБОТЫ

5.1 Тестирование

Запуск программы производится через терминал из директории, в которой находится исполняемый файл:

```
$ ./main
```

Сразу после запуска программа запускает процесс захвата и анализа пакетов. Об этом сигнализирует строка консоли. В процессе выполнения в консоль выводится промежуточное количество захваченных пакетов:

```
Starting...
TCP: 434  UDP: 77   ICMP: 0   Others: 71Total: 582
```

О завершении программы сигнализирует соответствующая строка консоли, которая выводится сразу после вывода конечного количества захваченных пакетов:

```
Starting...
TCP: 582  UDP: 101  ICMP: 0   Others: 76Total: 759
Finished
```

Результаты работы программы сохраняются в текстовый файл, который создается автоматически. Данные в файле отформатированы так, чтобы пользователю было удобно их проанализировать. Каждый пакет имеет заголовок, в котором указывается протокол. Далее размещаются IP и Ethernet заголовки с описанными внутренними параметрами, после чего расписаны параметры непосредственно протокола. В конце приводится блок данных, которые находились в заголовках пакета в момент его захвата. Фрагмент файла с результатами выглядит следующим образом:

```
**********TCP Packet**********

Ethernet Header
 |-Destination Address : CC-03-FA-62-83-7D
 |-Source Address : EC-2E-98-54-E2-75
 |-Protocol : 8

IP Header
 |-IP Version : 4
 |-IP Header Length : 5 DWORDS or 20 Bytes
 |-Type Of Service : 0
```

```
 |-IP Total Length : 205 Bytes (Size of Packet)
 |-Identification : 7213
 |-TTL : 64
 |-Protocol : 6
 |-Checksum : 8311
 |-Source IP : 192.168.0.27
 |-Destination IP : 149.154.167.41

TCP Header
 |-Source Port : 42178
 |-Destination Port : 443
 |-Sequence Number : 3223488899
 |-Acknowledge Number : 2709835259
 |-Header Length : 8 DWORDS or 32 BYTES
 |-Urgent Flag : 0
 |-Acknowledgement Flag : 1
 |-Push Flag : 1
 |-Reset Flag : 0
 |-Synchronise Flag : 0
 |-Finish Flag : 0
 |-Window : 4680
 |-Checksum : 14390
 |-Urgent Pointer : 0

DATA Dump
IP Header
  CC 03 FA 62 83 7D EC 2E 98 54 E2 75 08 00 45 00  ...b.}...T.u..E.
  00 CD 1C 2D                                      ...-
TCP Header
  40 00 40 06 20 77 C0 A8 00 1B 95 9A A7 29 A4 C2  @.@. w.......)..
  01 BB C0 22 89 83 A1 84 CD FB 80 18 12 48 38 36  ..."......Ъ..H86
Data Payload
  93 0D 55 41 36 8E 67 F8 F4 DD 7E 12 00 35 37 C0  ..UA6.g...~..57.
  D4 2F 16 8A 99 AC 8B 75 72 BD 55 06 06 A7 25 2D  ./.....ur.U...%-
  9F AC 0E 6A 3E EB 34 62 F4 2C 5A B3 D4 E4 61 81  ...j>.4b.,Z...a.
  B8 ED 19 3E F6 93 AD 8E 97 E3 6C 0C 5E C1 F0 84  ...>......l.^...
  C7 3F 1E 3C BB C2 6B B5 E9 97 E9 20 72 68 C8 2F  .?.<..k.... rh./
  9B 70 CD DF 37 54 F6 71 F0 97 1E AB 93 D2 07 52  .p..7T.q.......R
  00 4A 9F 93 5D 1A A4 C0 FD 30 EC 2A 21 AB 6B 3F  .J..]....0.*!.k?
  C7 46 63 AB DB 23 12 0E 27 19 C6 DF 6F BC 62 3C  .Fc..#..'...o.b<
  84 79 2F D6 8F A8 24 B2 9A 5C 9F 72 56 C4 23 96  .y/...$..\.rV.#.
  E2 C8 54 74 E2 DE 8F 90 17                       ..Tt.....
```

В работе программы предусмотрены возможные исключения. В случае ошибки при создании файла, в который будут заноситься данные, в консоль

будет выведено соответствующее сообщение, и программа будет экстренно завершена:

```
Starting…
Unable to create log.txt file
```

В случае неудачной попытки открыть сокет в консоль будет выведено соответствующее сообщение об ошибке, и программа будет экстренно завершена:

```
Starting…
Socket Error: Bad file descriptor
```

В случае неудачной попытки чтения пакета в консоль будет выведено соответствующее сообщение об ошибке, и программа будет экстренно завершена:

```
Starting…
Recvfrom error , failed to get packets
```

5.2 Код программы

Код программы представлен в приложении Г.

ЗАКЛЮЧЕНИЕ

Анализатор сетевого трафика является важным инструментом для мониторинга сетевой активности, анализа безопасности и отладки сетевых протоколов. В рамках курсового проекта была разработана программа, которая способна анализировать сетевой трафик на основе протоколов `Ethernet`, `IP/TCP`, `UDP`, `ICMP` и др.

В процессе разработки были рассмотрены основные этапы анализа сетевого трафика, такие как захват, декодирование и анализ пакетов. Были изучены различные протоколы, их структуры и поля. Также были изучены библиотеки для работы с сетевым трафиком.

В результате работы над проектом была создана программа, которая позволяет захватывать и анализировать сетевой трафик на основе протоколов `Ethernet`, `IP/TCP`, `UDP`, `ICMP` и др. Она способна выводить на экран основные характеристики пакетов, такие как адреса источника и назначения, номера портов, флаги и т.д. Также программа выводит содержимое пакетов.

В заключение можно отметить, что разработанный анализатор сетевого трафика может быть использован для различных задач, связанных с анализом сетевой активности и безопасности. Он может быть дополнен и усовершенствован для поддержки других протоколов и функциональных возможностей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Брайан Керниган, Деннис Ритчи. Язык программирования Си. Издательство: «Вильямс», 2019 г.

[2] Ричард Стивенс, Стивен Раго. UNIX. Профессиональное программирование. Издательство: «Вильямс», 2017 г.

[3] Robert Love. Linux System Programming: Talking Directly to the Kernel and C Library. Издательство: O'Reilly Media, 2013 г.

[4] Андрей Робачевский. Программирование на языке Си в UNIX. Издательство: БХВ-Петербург, 2015 г.

[5] Андрей Алексеев. Программирование на языке Си в среде Linux. Издательство: БХВ-Петербург, 2015 г.

[6] Майкл Керниган, Брайан В. Керниган. UNIX. Руководство системного программиста. Издательство: «Вильямс», 2018 г.

[7] Ричард Стивенс, Стивен Раго. Разработка приложений для UNIX. Издательство: «Питер», 2011 г.

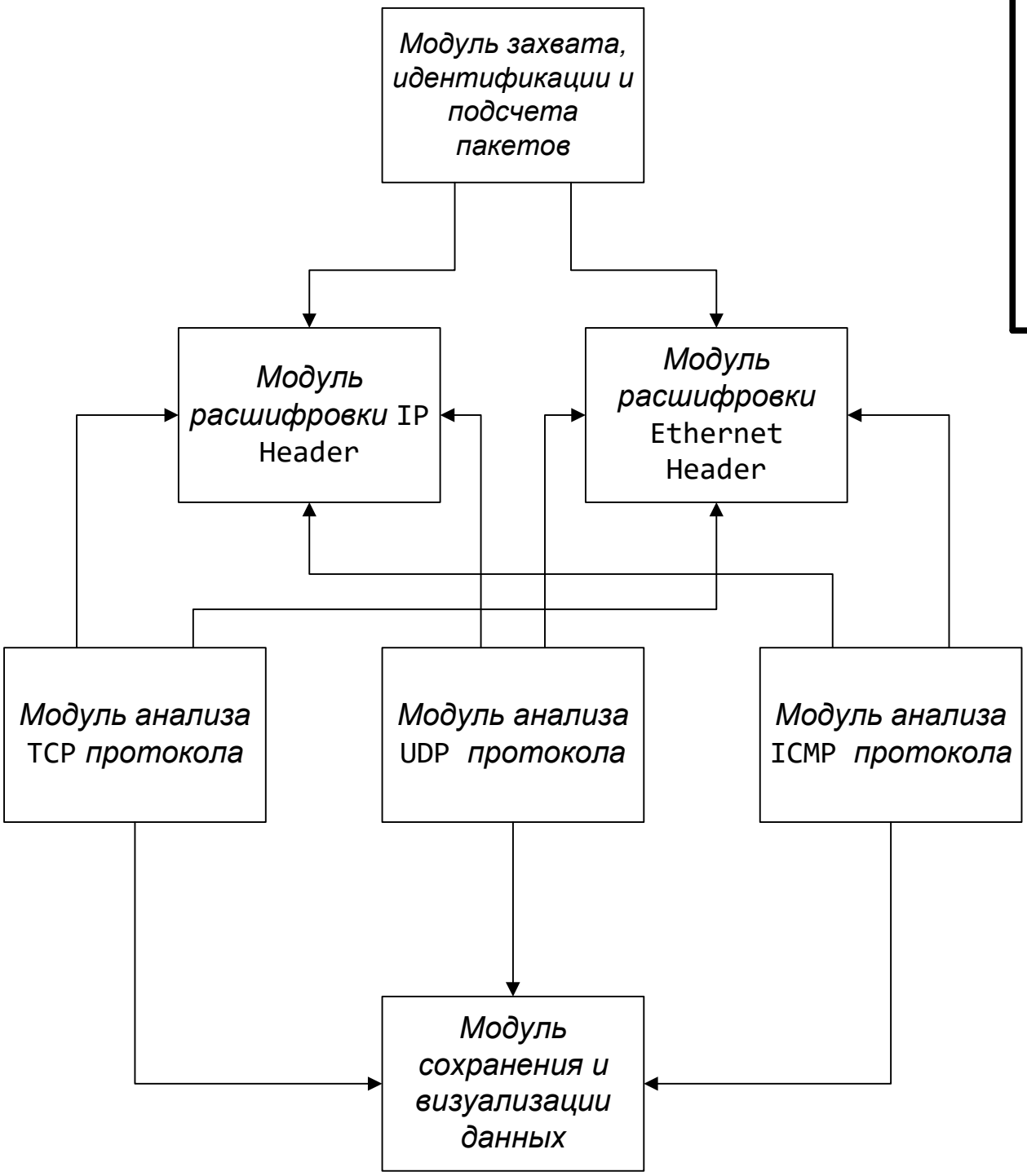
[8] The C Programming Language. Издательство: Prentice Hall, 1988 г.

[9] Стивенс У. Р., Феннер Б., Рудофф Э. М. UNIX Разработка сетевых приложений. Издательство: «Питер», 2007г.

ПРИЛОЖЕНИЕ А
(Обязательное)

Схема структурная

ГУИР.400201.120 С1



Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Почебут		
Пров.		Поденок		

ГУИР.400201.120 С1

Анализатор сетевого трафика
Структурная схема

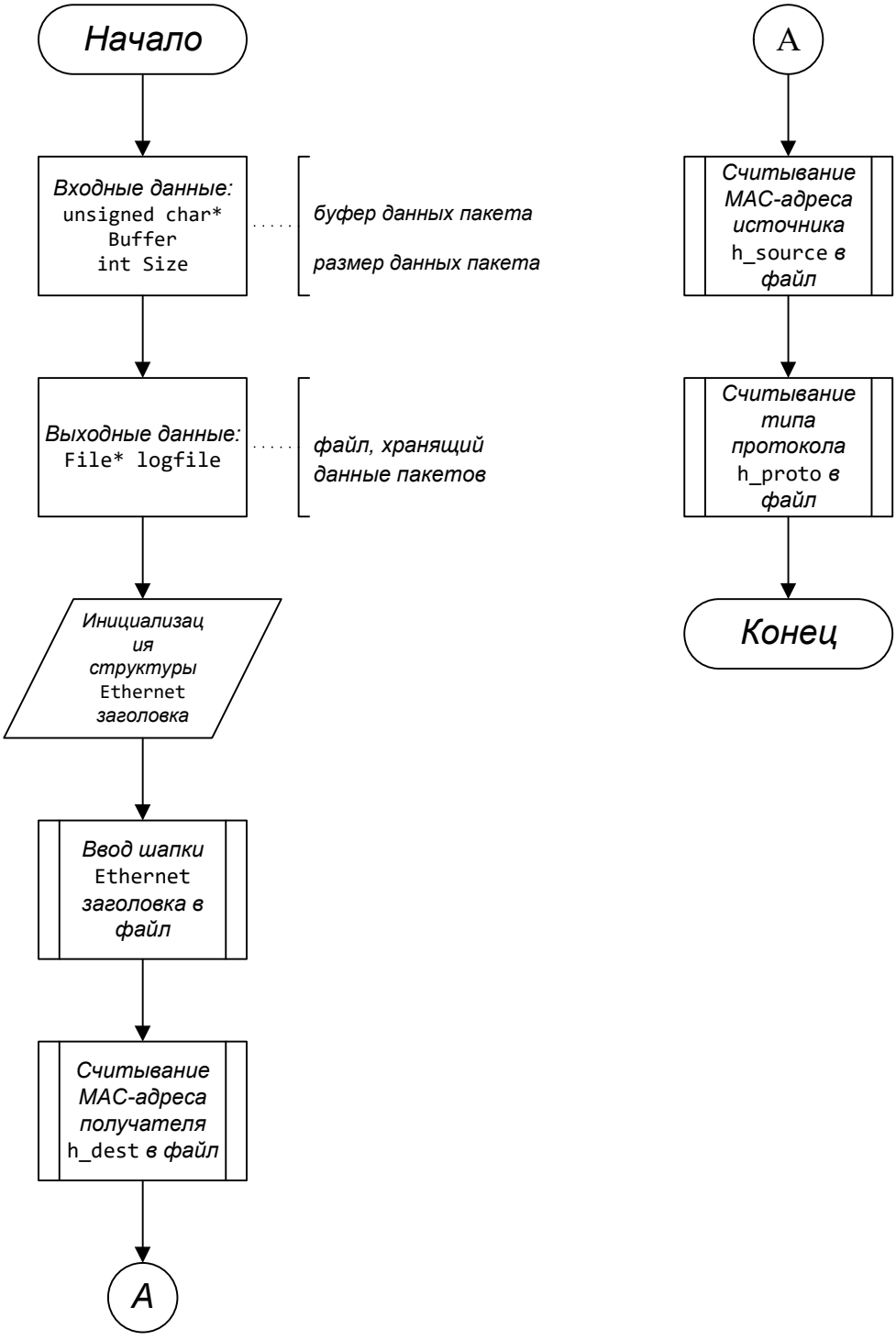
Лит.	Масса	Масштаб
У		

Лист | Листов 1

ЭВМ, гр.150501

ПРИЛОЖЕНИЕ Б
(Обязательное)

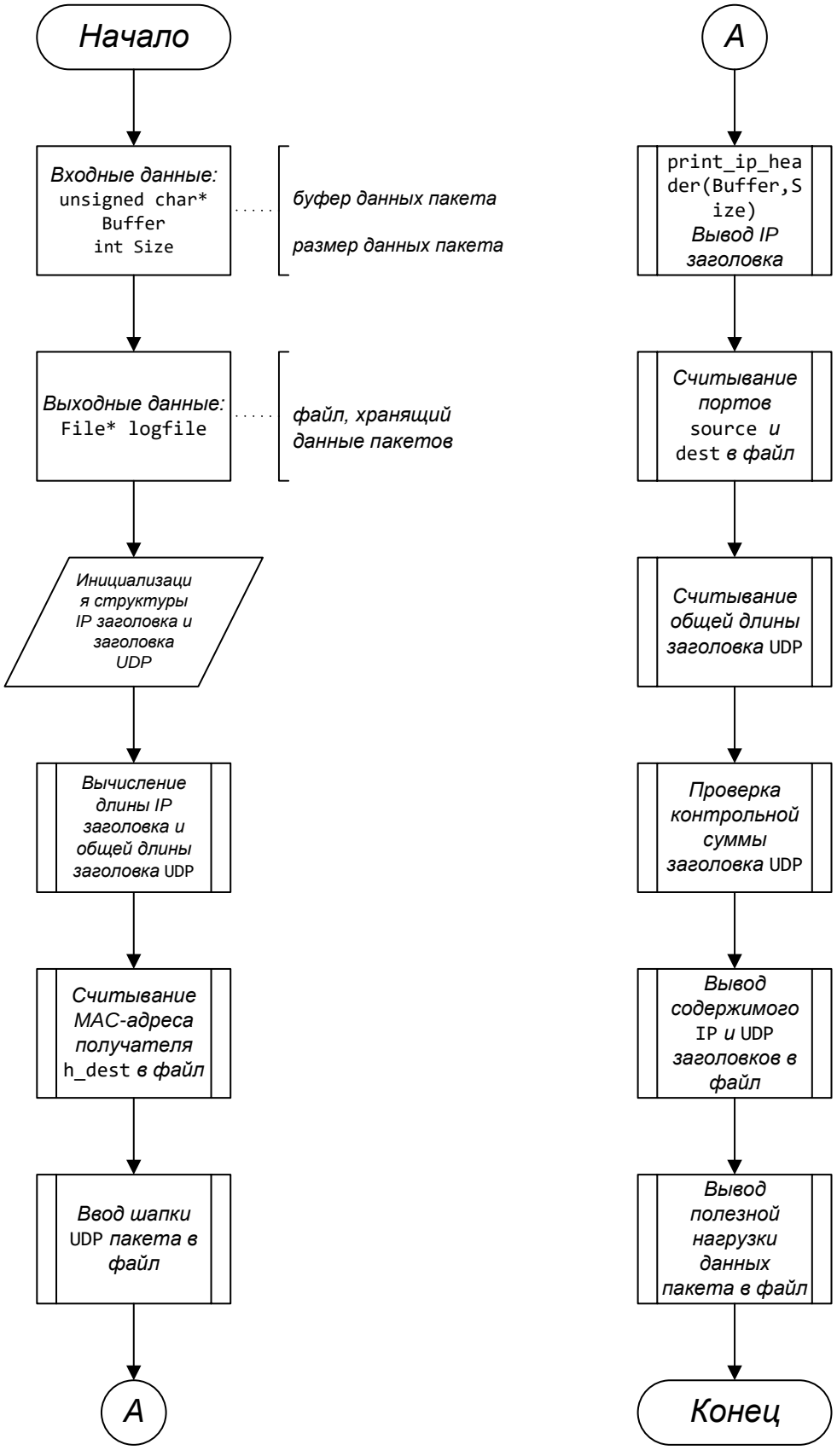
Схема алгоритма `print_ethernet_header()`



					ГУИР.400201.120 ПД.1	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Почебут			Схема алгоритма print_ethernet_header()	Лит. У / Масса / Масштаб
Пров.		Поденок				Лист / Листов 1
						ЭВМ, гр.150501

ПРИЛОЖЕНИЕ В
(Обязательное)

Схема алгоритма `print_udp_packet()`



					ГУИР.400201.120 ПД.2	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Почебут			Схема алгоритма print_udp_packet()	Лит. У / Масса / Масштаб
Пров.		Поденок				Лист / Листов 1
						ЭВМ, гр.150501

ПРИЛОЖЕНИЕ Г
(Обязательное)

Листинг кода

```
#include<netinet/in.h>
#include<stdio.h>     //For standard things
#include<stdlib.h>    //malloc
#include<string.h>    //strlen
#include<netinet/ip_icmp.h> //Provides declarations for icmp header
#include<netinet/udp.h>     //Provides declarations for udp header
#include<netinet/tcp.h>     //Provides declarations for tcp header
#include<netinet/ip.h>      //Provides declarations for ip header
#include<net/ethernet.h>    //For ether_header
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>

void process_packet(unsigned char*, int);
void print_ip_header(unsigned char*);
void print_tcp_packet(unsigned char *, int );
void print_udp_packet(unsigned char *, int );
void print_icmp_packet(unsigned char*, int );
void print_data(unsigned char*, int);

FILE *logfile;
struct sockaddr_in source, dest;
int tcp = 0, udp = 0, icmp = 0, others = 0, total = 0;

int main() {
    int saddr_size , data_size,  count_packets = 0;
    struct sockaddr saddr;
    unsigned char *buffer = (unsigned char *) malloc(65536);
    logfile = fopen("log.txt","w");
    if(logfile == NULL) {
        printf("Unable to create log.txt file\n");
    }
    printf("Starting...\n");
    int sock_raw = socket( AF_PACKET , SOCK_RAW , htons(ETH_P_ALL));
    if(sock_raw < 0) {
//Print the error with proper message
        perror("Socket Error");
        return 1;
    }
    while(count_packets <= 1000) {
        saddr_size = sizeof saddr;
//Receive a packet
        data_size = recvfrom(sock_raw, buffer, 65536, 0, &saddr,
                             (socklen_t*) &saddr_size);
        if(data_size < 0) {
            printf("Recvfrom error , failed to get packets\n");
            return 1;
        }
//Now process the packet
        process_packet(buffer, data_size);
```

```
        count_packets++;
    }
    close(sock_raw);
    printf("Finished\n");
    return 0;
}

void process_packet(unsigned char* buffer, int size) {
//Get the IP Header of this packet, excluding the ethernet header
    struct iphdr *iph = (struct iphdr*)(buffer+sizeof(struct ethhdr));
    ++total;
    switch (iph->protocol) //Check protocol and do accordingly...
    {
        case 1: //ICMP Protocol
            ++icmp;
            print_icmp_packet(buffer, size);
            break;
        case 6: //TCP Protocol
            ++tcp;
            print_tcp_packet(buffer, size);
            break;
        case 17: //UDP Protocol
            ++udp;
            print_udp_packet(buffer, size);
            break;
        default: //Some Other Protocol like ARP etc.
            ++others;
            break;
    }
    printf("TCP: %d\tUDP: %d\tICMP: %d\tOthers: %d\tTotal: %d\r",
tcp, udp, icmp, others, total);
}

void print_ethernet_header(unsigned char* Buffer) {
    struct ethhdr *eth = (struct ethhdr *)Buffer;
    fprintf(logfile, "\n");
    fprintf(logfile, "Ethernet Header\n");
    fprintf(logfile, " |-Destination Address : %.2X-%.2X-%.2X-%.2X-
%.2X-%.2X \n", eth->h_dest[0], eth->h_dest[1], eth->h_dest[2],
eth->h_dest[3], eth->h_dest[4], eth->h_dest[5]);
    fprintf(logfile, " |-Source Address : %.2X-%.2X-%.2X-%.2X-%.2X-
%.2X \n", eth->h_source[0], eth->h_source[1], eth->h_source[2],
eth->h_source[3], eth->h_source[4], eth->h_source[5] );
    fprintf(logfile, " |-Protocol : %u \n",
(unsigned short)eth->h_proto);
}

void print_ip_header(unsigned char* Buffer) {
    print_ethernet_header(Buffer);
    struct iphdr *iph = (struct iphdr*)(Buffer+sizeof(struct ethhdr));
```

```
    memset(&source, 0, sizeof(source));
    source.sin_addr.s_addr = iph->saddr;
    memset(&dest, 0, sizeof(dest));
    dest.sin_addr.s_addr = iph->daddr;
    fprintf(logfile, "\n");
    fprintf(logfile, "IP Header\n");
    fprintf(logfile, " |-IP Version : %d\n",
(unsigned int)iph->version);
    fprintf(logfile, " |-IP Header Length : %d DWORDS or %d Bytes\n",
(unsigned int)iph->ihl, ((unsigned int)(iph->ihl))*4);
    fprintf(logfile, " |-Type Of Service : %d\n",
(unsigned int)iph->tos);
    fprintf(logfile, " |-IP Total Length: %d Bytes(Size of Packet)\n",
ntohs(iph->tot_len));
    fprintf(logfile, " |-Identification : %d\n", ntohs(iph->id));
    fprintf(logfile, " |-TTL : %d\n",(unsigned int)iph->ttl);
    fprintf(logfile, " |-Protocol: %d\n", (unsigned int)iph->protocol);
    fprintf(logfile, " |-Checksum : %d\n", ntohs(iph->check));
    fprintf(logfile, " |-Source IP: %s\n", inet_ntoa(source.sin_addr));
    fprintf(logfile, " |-Destination IP: %s\n",
inet_ntoa(dest.sin_addr));
}

void print_tcp_packet(unsigned char* Buffer, int Size) {
    unsigned short iphdrlen;
    struct iphdr *iph = (struct iphdr *)( Buffer + sizeof(struct
ethhdr) );
    iphdrlen = iph->ihl*4;
    struct tcphdr *tcph = (struct tcphdr*)(Buffer + iphdrlen +
sizeof(struct ethhdr));
    int header_size = sizeof(struct ethhdr) + iphdrlen + tcph-
>doff*4;
    fprintf(logfile, "\n\n**********************TCP Packet
*************************\n");
    print_ip_header(Buffer);
    fprintf(logfile, "\n");
    fprintf(logfile, "TCP Header\n");
    fprintf(logfile, " |-Source Port : %u\n", ntohs(tcph->source));
    fprintf(logfile, " |-Destination Port: %u\n",ntohs(tcph->dest));
    fprintf(logfile, " |-Sequence Number : %u\n", ntohl(tcph->seq));
    fprintf(logfile, " |-Acknowledge Number: %u\n",
ntohl(tcph->ack_seq));
    fprintf(logfile, " |-Header Length : %d DWORDS or %d BYTES\n",
(unsigned int)tcph->doff, (unsigned int)tcph->doff*4);
    fprintf(logfile, " |-Urgent Flag: %d\n",(unsigned int)tcph->urg);
    fprintf(logfile, " |-Acknowledgement Flag : %d\n",
(unsigned int)tcph->ack);
    fprintf(logfile, " |-Push Flag : %d\n",(unsigned int)tcph->psh);
    fprintf(logfile, " |-Reset Flag: %d\n",(unsigned int)tcph->rst);
```

```
    fprintf(logfile, " |-Synchronise Flag : %d\n",
(unsigned int)tcph->syn);
    fprintf(logfile, " |-Finish Flag: %d\n", (unsigned int)tcph->fin);
    fprintf(logfile, " |-Window : %d\n", ntohs(tcph->window));
    fprintf(logfile, " |-Checksum : %d\n", ntohs(tcph->check));
    fprintf(logfile, " |-Urgent Pointer : %d\n", tcph->urg_ptr);
    fprintf(logfile, "\n");
    fprintf(logfile, " DATA Dump ");
    fprintf(logfile, "\n");
    fprintf(logfile, "IP Header\n");
    print_data(Buffer, iphdrlen);
    fprintf(logfile, "TCP Header\n");
    print_data(Buffer + iphdrlen, tcph->doff*4);
    fprintf(logfile, "Data Payload\n");
    print_data(Buffer + header_size , Size - header_size );
    fprintf(logfile,
"\n#########################################################");
}

void print_udp_packet(unsigned char *Buffer, int Size) {
    unsigned short iphdrlen;
    struct iphdr *iph = (struct iphdr*)(Buffer+sizeof(struct ethhdr));
    iphdrlen = iph->ihl*4;
    struct udphdr *udph = (struct udphdr*)(Buffer + iphdrlen +
sizeof(struct ethhdr));
    int header_size = sizeof(struct ethhdr)+iphdrlen+sizeof udph;
    fprintf(logfile, "\n\n**********************UDP Packet
*************************\n");
    print_ip_header(Buffer);
    fprintf(logfile, "\nUDP Header\n");
    fprintf(logfile, " |-Source Port : %d\n", ntohs(udph->source));
    fprintf(logfile, " |-Destination Port: %d\n", ntohs(udph->dest));
    fprintf(logfile, " |-UDP Length : %d\n", ntohs(udph->len));
    fprintf(logfile, " |-UDP Checksum : %d\n", ntohs(udph->check));
    fprintf(logfile, "\n");
    fprintf(logfile, "IP Header\n");
    print_data(Buffer, iphdrlen);
    fprintf(logfile, "UDP Header\n");
    print_data(Buffer + iphdrlen, sizeof udph);
    fprintf(logfile, "Data Payload\n");
//Move the pointer ahead and reduce the size of string
    print_data(Buffer + header_size, Size - header_size);
    fprintf(logfile,
"\n#########################################################");
}

void print_icmp_packet(unsigned char* Buffer, int Size) {
    unsigned short iphdrlen;
    struct iphdr *iph = (struct iphdr*)(Buffer+sizeof(struct ethhdr));
    iphdrlen = iph->ihl * 4;
```

```
    struct icmphdr *icmph = (struct icmphdr*)(Buffer + iphdrlen +
sizeof(struct ethhdr));
    int header_size = sizeof(struct ethhdr)+iphdrlen+sizeof icmph;
    fprintf(logfile, "\n\n********************ICMP Packet
**********************\n");
    print_ip_header(Buffer);
    fprintf(logfile, "\n");
    fprintf(logfile, "ICMP Header\n");
    fprintf(logfile, " |-Type : %d", (unsigned int)(icmph->type));
    if((unsigned int)(icmph->type) == 11) {
        fprintf(logfile, " (TTL Expired)\n");
    }
    else if((unsigned int)(icmph->type) == ICMP_ECHOREPLY) {
        fprintf(logfile, " (ICMP Echo Reply)\n");
    }
    fprintf(logfile, " |-Code : %d\n", (unsigned int)(icmph->code));
    fprintf(logfile, " |-Checksum : %d\n", ntohs(icmph->checksum));
    fprintf(logfile, "\n");
    fprintf(logfile, "IP Header\n");
    print_data(Buffer, iphdrlen);
    fprintf(logfile, "UDP Header\n");
    print_data(Buffer + iphdrlen, sizeof icmph);
    fprintf(logfile, "Data Payload\n");
//Move the pointer ahead and reduce the size of string
    print_data(Buffer + header_size, (Size - header_size));
    fprintf(logfile,
"\n###########################################################");
}

void print_data (unsigned char* data, int Size) {
    int i, j;
    for(i = 0; i < Size; i++) {
        if(i != 0 && i%16 == 0)
        {
            fprintf(logfile, " ");
            for(j = i-16; j < i; j++) {
                if(data[j] >= 32 && data[j] <= 128)
                    fprintf(logfile, "%c", (unsigned char)data[j]);
                else
                    fprintf(logfile, ".");  //otherwise print a dot
            }
            fprintf(logfile, "\n");
        }
        if(i%16 == 0) fprintf(logfile, " ");
        fprintf(logfile , " %02X",(unsigned int)data[i]);
        if( i == Size-1) //print the last spaces
        {
            for(j = 0; j < 15-i%16; j++) {
                fprintf(logfile, " ");      //extra spaces
            }
```

```
            fprintf(logfile, " ");
            for(j = i-i%16; j <= i; j++) {
                if(data[j] >= 32 && data[j] <= 128) {
                    fprintf(logfile, "%c", (unsigned char)data[j]);
                }
                else {
                    fprintf(logfile, ".");
                }
            }
            fprintf(logfile, "\n" );
        }
    }
}
```

ПРИЛОЖЕНИЕ Д
(Обязательное)

Ведомость документов

Обозначение	Наименование	Примечание
	Графические документы	
ГУИР.400201.120 С1	Схема структурная	А4
ГУИР.400201.120 ПД.1	Схема алгоритма	А4
	`print_ethernet_header()`	
ГУИР.400201.120 ПД.2	Схема алгоритма	А4
	`print_udp_packet()`	
	Текстовые документы	
БГУИР КР 1-40 02 01 120 ПЗ	Пояснительная записка к курсовой	29 с.
	Листинг кода	6 с.
ГУИР.400201.120-01 12	Текст программы	

Изм	Лист	№ документа	Подпись	Дата	ГУИР.400201.120 Д1		
Разраб.		Почебут			Анализатор сетевого трафика Ведомость документов	Лит.	Лист	Листов
Пров.		Поденок				У		1
						ЭВМ, гр. 150501		