

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.02 –

Специализация -

Информационная безопасность
телекоммуникационных систем
Разработка защищенных
телекоммуникационных систем

ДИПЛОМНЫЙ ПРОЕКТ

на тему

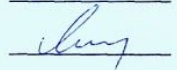
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПЕРЕХВАТА СЕТЕВОГО ТРАФИКА НА
УСТРОЙСТВАХ ПОД УПРАВЛЕНИЕМ ОС ANDROID

Студент



Медведева А.В.

Руководитель к.т.н., доцент



Липилин О.В.

Нормоконтролёр



Фатеев А.Г.

Рецензент ведущий инженер АО «ПНИЭИ»



Горячев Д.В.

Работа допущена к защите (протокол заседания кафедры от 23.01.18 № 9)

Заведующий кафедрой



С.Л. Зефирова

Работа защищена с отметкой хорошо (протокол заседания ГЭК
от 15.01.2018 № 19)

Секретарь ГЭК



Фатеев А.Г.

*Проверка ВКР
по Бачков*


Пенза 2018

Оглавление

Введение.....	3
1 Теоретическая часть.....	6
1.1 Постановка задачи.....	6
1.2 Анализ литературы.....	7
1.3 Обзор аналогов программного обеспечения.....	9
1.4 Обзор существующих методов решения поставленной задачи.....	11
1.5 Техническая спецификация протоколов передачи данных.....	14
1.6 Обзор архитектуры ОС Android.....	22
1.7 Обзор структуры Android-приложения.....	26
2 Практическая часть.....	31
2.1 Формализация задачи.....	31
2.2 Используемые шаблоны проектирования.....	32
2.3 Структура программного обеспечения.....	35
2.4 VPN-компонент.....	42
2.5 Компонент логирования сетевого трафика.....	44
2.6 Компонент просмотра подробной информации.....	46
2.7 Компонент фильтрации.....	47
2.8 Руководство пользователя.....	49
3 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.....	55
Заключение.....	58
Список используемых источников.....	59
Приложение А Код программы.....	62

Приложение Б	Протокол проверки ВКР в системе	
«Антиплагиат».....		101

Введение

Мобильные телефоны давно перестали использоваться исключительно в качестве средства связи – в настоящее время они обладают широким функционалом, который не ограничивается лишь совершением звонков и отправкой SMS-сообщений. Данные устройства обеспечивают доступ в сеть Интернет, предоставляют возможность синхронизации контактов, сообщений, звонков, аккаунтов, настроек, фотографий и прочей информации с почтовым аккаунтом, обладают базовыми средствами для работы с документами, нередко используются для хранения паролей, платежных данных и прочей конфиденциальной информации. Благодаря высокой производительности, быстродействию и большому объему памяти, мобильные телефоны по праву можно считать настоящими карманными компьютерами, среднее время использования которых, согласно статистическим исследованиям, составляет около пяти часов в сутки.

Помимо использования в личных целях, мобильные телефоны нередко используются в корпоративных сетях, поскольку они предоставляют практически мгновенный доступ к контактам или документам из любой части планеты. Так, по результатам исследования компании «Код безопасности», около 60% сотрудников используют мобильные устройства в рабочих целях, например, для чтения корпоративной электронной почты, хранения конфиденциальной информации и т.д.

Таким образом, на мобильных устройствах хранится не меньше персональных данных, чем на настольных или портативных компьютерах, а их утечка, в свою очередь, может привести к значительному ущербу как для обычного пользователя, так и для целых компаний.

В настоящее время более 80% используемых мобильных устройств базируются на операционной системе Android. Исследование компании

«Лаборатория Касперского» показывает, что мобильные приложения, установленные на этой платформе, ежедневно используют по несколько десятков мегабайт трафика, даже в том случае, если пользователь их не запускал. Согласно этому исследованию, 96 из 100 Android приложений работают в фоновом режиме, а 83 из 100 имеют доступ к конфиденциальным пользовательским данным (контакты, сообщения, звонки, файлы и т.д.) [1].

Кроме того, большинство приложений, установленных на Android-устройствах, взаимодействуют с сетью без ведома пользователя, и это, в свою очередь, может использоваться злоумышленником для передачи конфиденциальных пользовательских данных, например, контактов, сообщений, списка звонков, файлов, IMEI установленной сим-карты и т.д.

Помимо умышленной кражи пользовательских данных, нередко имеет место быть непреднамеренная утечка информации, например, из-за ошибок разработчиков может произойти утечка метаданных или данных о местоположении. Кроме того, некоторые Google-приложения (например, GoogleContactsSyncAdapter или GoogleBackupTransport), по-умолчанию установленные на Android-устройствах, собирают определенные виды информации – от списка установленных приложений и запросов в браузере до списка контактов и паролей.

Проблемой, решаемой в рамках данной выпускной квалификационной работы, является мониторинг сетевой активности приложений на устройствах под управлением ОС Android. Поскольку встроенный в Android анализатор сетевого трафика обладает ограниченным функционалом, в частности, позволяет отслеживать только количество используемого тем или иным приложением трафика, для решения этой проблемы требуется разработать программное обеспечение, предназначенное для прослушивания и анализа сетевого трафика в режиме реального времени на Android-устройствах.

Таким образом, целью данной выпускной квалификационной работы является разработка программного обеспечения, позволяющего перехватывать сетевой трафик под управлением ОС Android.

Задачами, решаемыми в рамках выпускной квалификационной работы, являются:

- обзор, сравнение и выявление недостатков у существующих аналогов разрабатываемого программного обеспечения;
- анализ существующих способов программной реализации поставленной задачи;
- изучение технической спецификации используемых протоколов передачи данных;
- обзор архитектуры операционной системы Android и структуры Android-приложения;
- разработка архитектуры и алгоритмов работы приложения;
- реализация перехвата и анализа сетевого трафика на устройствах под управлением ОС Android версии 4.0 и выше;
- реализация фильтрации перехваченного сетевого трафика;
- разработка руководства пользователя.

Помимо этого, в рамках выпускной квалификационной работы должны быть рассмотрены вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

1 Теоретическая часть

1.1 Постановка задачи

Одним из первых этапов разработки любого программного обеспечения является анализ требований к проекту. На данном этапе формулируются ответы на следующие вопросы:

- для кого разрабатывается система;
- каким функционалом она должна обладать;
- где и как она может быть использована;
- какими системными требованиями и ограничениями обладает разрабатываемая система [2].

В рамках данной выпускной квалификационной работы требуется разработать программное обеспечение, которое обладает следующим функционалом:

- перехват сетевого трафика в режиме реального времени;
- просмотр подробной информации о перехваченных пакетах;
- фильтрация перехваченного трафика по заданным пользователем параметрам.

Программное обеспечение может быть использовано для реализации сетевого мониторинга на устройствах под управлением ОС Android с целью анализа и обнаружения подозрительного трафика, предотвращения утечки конфиденциальной информации и выявления вредоносного ПО.

Разрабатываемое программное обеспечение должно соответствовать следующим системным требованиям:

- программное обеспечение должно функционировать на устройствах под управлением ОС Android 4 и выше;
- для корректной работы программного обеспечения не должен требоваться root-доступ (т.е. права суперпользователя).

1.2 Анализ литературы

При проектировании программного обеспечения перехвата сетевого трафика на устройствах под управлением ОС Android требуется рассмотреть следующие вопросы. Во-первых, необходимо иметь представление о структуре и основных принципах работы сети Интернет, в частности, знать о работе сетевой модели TCP/IP, пакетной передаче данных, технологии «клиент-сервер» и т.д. Во-вторых, требуются знания о языках и технологиях программирования, в частности, под Android устройства. В-третьих, необходимо рассмотреть особенности того или иного метода реализации перехвата сетевого трафика под выбранную операционную систему.

Таким образом, для реализации программного обеспечения перехвата сетевого трафика на устройствах под управлением ОС Android может потребоваться литература по следующим вопросам:

- принципы, технологии и протоколы компьютерных сетей;
- алгоритмы и структуры данных;
- шаблоны проектирования;
- средства разработки программного обеспечения для ОС Android (языки программирования, интегрированные среды разработки, Android API, библиотеки).

Базовые знания о сетевых технологиях рассматриваются в книге «Компьютерные сети» под авторством В. Олифер и Н. Олифер [3]. В указанной книге представлено описание принципов построения и особенностей технологий локальных и глобальных сетей, протоколов передачи данных, затронуты вопросы сетевой безопасности, приведены классификация и примеры компьютерных сетей.

Стандарты Глобальной сети представлены в серии пронумерованных информационных документов RFC (Request For Comment) [4]. В указанной

серии документов рассматриваются основные аспекты компьютерных сетей, технические спецификации и стандарты, протоколы, концепции работы, информация об экспериментальных исследованиях, а также рекомендации по реализации стандартов.

Подробное описание алгоритмов и структур данных представлено в книге автора Адитья Бхаргава «Грожаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих» [5]. В первых трех главах этой книги доступно рассказывается об основных алгоритмах и скорости их работы, структурах данных и рекурсии. Последующие главы посвящены методам решений задач, хэш-таблицам, графам и другим алгоритмам.

Информацию и наиболее часто встречающихся шаблонах проектирования можно найти в книге «Паттерны проектирования» Эрика и Элизабет Фримен [6]. В данной книге приводится описание важнейших шаблонов проектирования, рассказывается о преимуществах их использования в рамках решения конкретных задач и приводятся примеры на языке Java.

Базовые сведения об языке программирования Java, а также о принципах объектно-ориентированного программирования представлены в книге К. Сьерра, Б. Бертс «Изучаем Java» [7]. Помимо указанной книги, следует использовать книгу Б. Эккеля «Философия Java» [8], в которой приводится подробное описание таких терминов, как абстрактные классы и интерфейсы, коллекции, контейнеры, итераторы, многозадачность, потоки ввода-вывода, обработка ошибок и других понятий, составляющих ядро языка программирования Java.

В качестве ресурса, посвященного разработке программного обеспечения под ОС Android, можно использовать сайт developer.android.com [9], на котором можно найти подробную информацию об Android API, архитектурных компонентах, библиотеках поддержки, описание классов и примеры их

использования и т.д. В качестве интегрированной среды разработки используется IDE Android Studio, документацию по которой можно найти на том же сайте [10].

1.3 Обзор аналогов программного обеспечения

Для решения проблемы мониторинга сетевой активности приложений на устройствах под управлением ОС Android выполнен поиск и изучение существующих программных анализаторов сетевого трафика для рассматриваемой операционной системы, а также выделены их достоинства и недостатки.

Наиболее популярным приложением, предназначенных для этих целей, является Packet Capture. Данное приложение позволяет перехватывать и хранить сетевые пакеты, просматривать их в текстовом и шестнадцатеричном виде, а также фильтровать перехваченный трафик по приложению-отправителю. Главным преимуществом рассматриваемого приложения является возможность работы на устройствах без root-доступа, удобный пользовательский интерфейс и представление данных в различном виде. Недостатками является ограниченная фильтрация, которая может осуществляться лишь по одному параметру, а также отсутствие просмотра подробной информации о пакете.

Следующим рассмотренным аналогом является программа tPacketCapture. Приложение перехватывает сетевой трафик на мобильном устройстве и сохраняет данные в файлы формата pcap, списки которых выводятся на экран. Однако данное приложение обладает существенными недостатками: отсутствует возможность настроить фильтрацию и просмотреть перехваченный трафик с мобильного устройства (т.к. открытие pcap файла не поддерживается рассматриваемым приложением).

Аналогичным программным обеспечением является приложение Wicar. Несмотря на широкие функциональные возможности, включающие в себя перехват сетевого трафика LTE/WiFi сетей, просмотр подробной информации о пакетах, включающей такие параметры, как тип протокола, длина заголовка, флаги, контрольная сумма и т.д., просмотр данных в текстовом и шестнадцатеричном виде, общая статистика перехваченного трафика по каждому из протоколов, приложение обладает существенным недостатком – для его корректной работы требуется root-доступ, который по умолчанию отсутствует на android-устройствах. Помимо этого, в приложении отсутствует настраиваемая фильтрация и просмотр информации о приложении-отправителе.

Основные функции рассмотренных приложений представлены в таблице 1.

Таблица 1 – Сравнительная таблица средств перехвата трафика на устройствах под управлением ОС Android

Функции	Packet Capture	tPacketCapture	Wicar
Перехват трафика в режиме реального времени	+	+	+
Фильтрация трафика по параметрам	-	-	-
Просмотр всего трафика непосредственно из приложения	+	-	+

Просмотр детальной информации о перехваченном пакете	+	-	+
Хранение перехваченного трафика	+	+	+
Root-доступ	Не требуется	Не требуется	Требуется

Помимо вышеперечисленных приложений рассмотрены и другие аналоги, однако большинство из них обладают крайне ограниченным функционалом и рядом таких существенных недостатков, как, например, отсутствие пользовательского интерфейса или блокирование доступа в Интернет.

Таким образом, на основе результатов, полученных в процессе изучения существующих аналогов разрабатываемого программного обеспечения, выявлены их недостатки, которые обуславливают потребность в разработке новой системы.

1.4 Обзор существующих методов решения поставленной задачи

Способы перехвата и анализа сетевого трафика на устройствах под управлением ОС Android в зависимости от требований к окружению можно условно разделить на три группы:

- перехват сетевого трафика с помощью десктопных устройств;
- перехват сетевого трафика на устройствах с root-доступом;
- перехват сетевого трафика на устройствах без root-доступа.

Наиболее распространенным и простым методом, относящимся к первой группе, является подключение Android-устройства к WiFi-точке и мониторинг его сетевого трафика на десктопном устройстве через любую программу-анализатор (например, Wireshark).

Одной из возможных реализаций перехвата сетевого трафика на Android устройствах с root-доступом является разработка программного обеспечения, использующего утилиту tcpdump, которая позволяет прослушивать сетевой трафик и по умолчанию входит в большинство Unix-дистрибутивов.

Несмотря на простоту реализации метода, необходимость наличия root-доступа на устройстве является существенным недостатком. Как известно, ОС Android работает на ядре Linux, однако по умолчанию из-за соображений безопасности root-доступ в ней отсутствует. Получение такого доступа, или «рутирование» устройства аналогично получению прав суперпользователя, а это, в свою очередь, влечёт за собой обход ограничений безопасности, установленных ОС Android. Так, например, в случае, если вирусное приложение получит root-права, то оно может нанести куда более существенный вред устройству, чем без них. Поэтому данный способ решения поставленной задачи не используется в рамках данной выпускной квалификационной работы.

Наиболее распространенными способами перехвата сетевого трафика на Android устройствах без root-доступа являются установка локальных Прoxy- или VPN-серверов. Первый метод используется в большинстве антивирусов под Android. Для его реализации требуется:

- разработать программное обеспечение, в котором реализована поддержка HTTP-прокси стандарта RFC 2068;
- в настройках WiFi сети устройства установить Прoxy сервер;
- в настройках мобильной сети устройства установить APN (Access Point Name).

Однако у данный метод обладает существенным недостатком. В случае, если пользователь не запустит приложение с Прoxy-сервером, ему придется вручную удалить WiFi-прокси или APN из настроек, иначе ни одно приложение

не сможет соединиться с сетью. Кроме того, необходимо отслеживать появление новых LTE или WiFi сетей и устанавливать для них Proxu или APN [24].

Второй способ заключается в разработке VPN-клиента, с помощью которого весь TCP/IP трафик приложений, установленных на устройстве, будет перенаправлен на TUN-интерфейс. Под TUN-интерфейсом понимается сетевой туннель, который работает на сетевом уровне модели OSI и оперирует IP пакетами. Для анализа трафика и обработки соединений используется стек TCP/IP, который должен быть включен в VPN-клиент. Данный способ можно представить в виде схемы, представленной на рисунке 1.

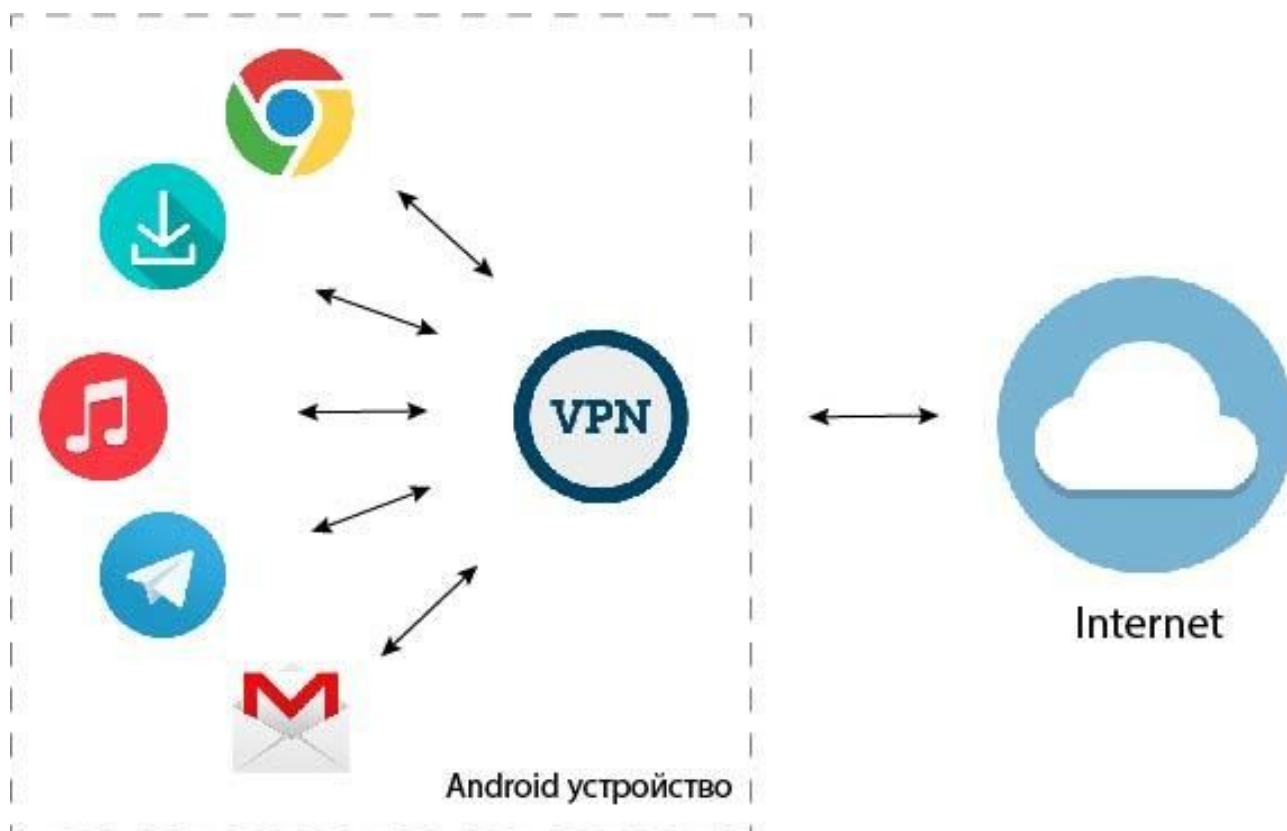


Рисунок 1 – Механизм перехвата сетевого трафика на устройствах под управлением ОС Android с использованием VPN-клиента В рамках данной выпускной квалификационной работы для реализации функции перехвата сетевого трафика в разрабатываемом программном обеспечении выбран последний способ, потому что он:

- использует VPNService API, которое предоставляет все необходимые функции для разработки VPN-клиента и доступно на Android устройствах версии 4.0 и выше;
- не требует root-доступа, что позволяет использовать приложение на любом Android устройстве версии 4.0 и выше;
- не требует дополнительных настроек устройства.

У данного способа, конечно, есть недостаток. Для работы VPN в ядре Linux требуется поддержка TUN-интерфейса, поэтому при её отсутствии, программа не будет корректно работать. Однако эта проблема может присутствовать лишь на Android устройствах с неофициальными прошивками (например, на основе LineageOS), в случае, если поддержка данного интерфейса была убрана из ядра.

1.5 Техническая спецификация протоколов передачи данных

Как было сказано выше, для обработки и анализа трафика используется стек TCP/IP – набор сетевых протоколов, регулирующий сетевые соединения между устройствами в Интернете. Так, рассматриваемый стек протоколов описывает, как информация должна быть представлена, отправлена и получена, т.е. способ передачи данных от отправителя к получателю. Стек TCP/IP включает в себя четыре уровня [3], представленные в таблице 2.

Таблица 2 – Уровни стека TCP/IP

Уровень	Примеры протоколов
Прикладной (Application layer)	HTTP, FTP, DNS, SMTP
Транспортный (Transport layer)	TCP, UDP

Уровень межсетевого взаимодействия (Internet layer)	IP
Уровень межсетевых интерфейсов (Network access layer)	Ethernet, IEEE 802.11 WLAN, Token Ring

Стержнем всего стека TCP/IP является уровень межсетевого взаимодействия. Функциями данного уровня является управление, упаковка, фрагментация, логическая адресация и маршрутизация трафика. Основным протоколом рассматриваемого уровня является протокол IP, техническая спецификация которого определяется документом RFC 791 [12].

Функцией протокола IP является доставка дейтаграмм от отправителя к получателю, в частности, их адресация и фрагментация/сборка. Рассматриваемый протокол не обеспечивает гарантированной и надежной доставки дейтаграммы до получателя, в частности, дейтаграммы могут придти не в том порядке, в котором были отправлены, могут быть повреждены, продублированы или не дойти до получателя вовсе.

Для идентификации перехваченного трафика необходимо иметь представление о структуре заголовка IP-пакета. Согласно пункту 3.1 Формат заголовка (Internet Header Format) документа RFC 791 [12], заголовок дейтаграммы IP имеет вид, представленный на рисунке 2.

Поле Version (4 бита) указывает на версию протокола, а также является определяющим для формата заголовка. В рассматриваемой спецификации описывается версия 4.

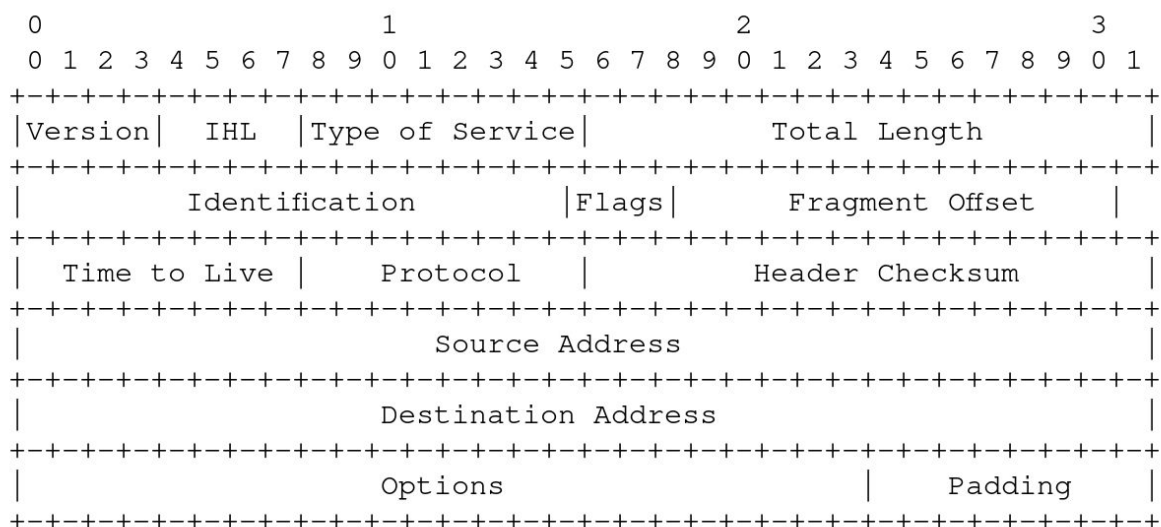


Рисунок 2 – Формат заголовка дейтаграммы IP

В поле IHL (4 бита) содержится размер заголовка IP. Помимо этого, данное поле является указателем на начало данных.

Значение поля Type of Service (ToS, 8 битов) указывает на абстрактные параметры желаемого качества обслуживания, которые впоследствии используются при определении реальных параметров обслуживания при передаче дейтаграммы. Формат данного поля представлен на рисунке 3.

Биты 0 – 1 предназначены для значения предпочтений (Precedence) и могут принимать следующие значения:

- 111 – управление сетью;
- 110 — межсетевое управление;
- 101 — CRITIC/ECR;
- 100 — сверхсрочно;
- 011 — срочно;
- 010 — незамедлительно;
- 001 — приоритетный;
- 000 — обычный.

Бит 3 представляет собой флаг задержки (Delay) и может принимать значения 0 (обычная задержка) и 1 (малая задержка).

Бит 4 – это флаг пропускной способности (Throughput), принимающий значения 0 и 1 для обычной и высокой пропускной способности соответственно.

Бит 5 предназначен для установки флага надежности (Reliability), который может принимать значения 0 (обычная надежность) и 1 (высокая надежность).

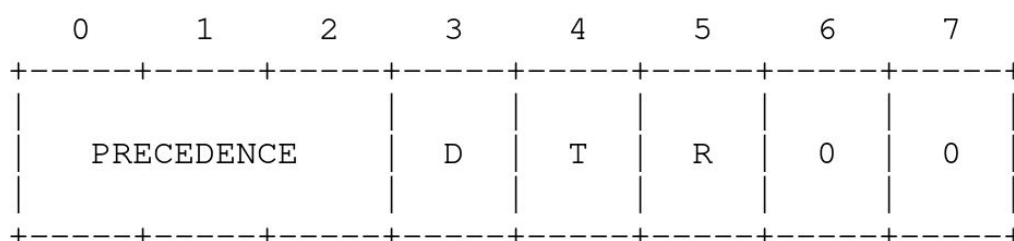


Рисунок 3 – Формат поля ToS

Использование вышеперечисленных флагов может привести к увеличению стоимости обслуживания, поэтому рекомендуется использовать не более двух флагов из трёх возможных.

Поле Total Length (16 битов) содержит общий размер дейтаграммы, включающий размер заголовка и данных. Типичный размер заголовка IP составляет 20 октетов, в то время как максимальный – 60 октетов.

Поле Identification (16 битов) представляет собой идентификационный номер и предназначено для правильной сборки фрагментов дейтаграммы.

Поле Flags (3 бита) представляет набор флагов управления. Возможные значения рассматриваемого поля представлены в таблице 3.

Таблица 3 – Набор флагов управления

Бит	Значение
0	Зарезервирован (по умолчанию, значение 0)

1	0 – фрагментация возможна	1 – фрагментация недопустима
2	0 – фрагмент является последним	1 – фрагмент не является последним

Поле Fragment Offset (13 битов) указывает на положение фрагмента в исходной дейтаграмме.

Поле TTL (8 битов) определяет максимальный срок жизни дейтаграммы. Так, если значение рассматриваемого поля равно нулю, дейтаграмма уничтожается.

Значение поле Protocol (8 битов) представляет собой идентификатор протокола следующего уровня, который содержится в поле данных дейтаграммы. Идентификаторы протокола определяются документом RFC 790 Assigned Numbers [25].

Так как некоторые поля заголовка, например, TTL, меняются при передаче дейтаграммы, в поле Header Checksum содержится значение контрольной суммы для всего заголовка, которое проверяется и пересчитывается при каждой обработке заголовка.

Поля Source address (32 бита) и Destination address (32 бита) представляют собой адреса отправителя и получателя соответственно.

Поле Options (переменная длина) и Padding (переменная длина) являются необязательными полями.

Рассмотрев описание IP дейтаграммы, можно выделить поля, являются наиболее важными в условиях поставленной задачи:

- поле Source Address (Адрес отправителя), которое позволит понять, откуда был отправлен дейтаграмма;

- поле Destination Address (Адрес получателя), которое позволит понять, кому предназначается дейтаграмма;
- поле Protocol (Протокол), определяющее протокол, который использовался для передачи данных.

Согласно пункту 2.1 Связь с другими протоколами («Relation to Other Protocols») документа RFC 791 [12] схема взаимодействия протокола IP с другими протоколами имеет вид, представленный на рисунке 4.

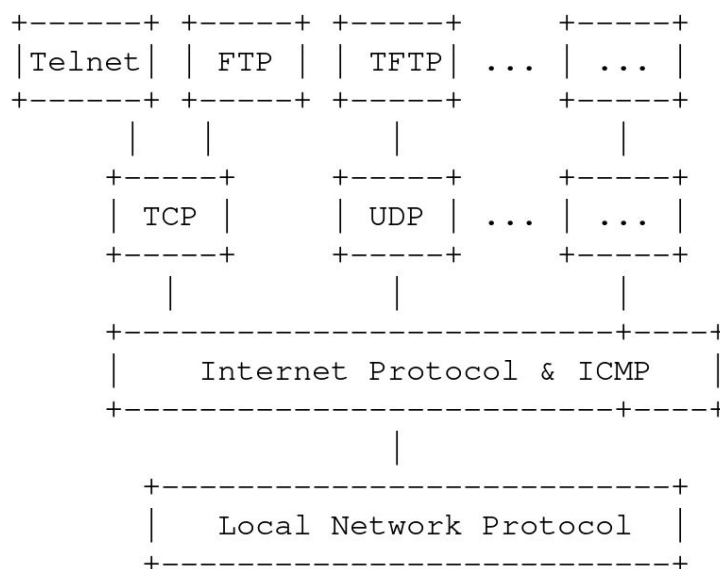


Рисунок 4 – Взаимодействие протокола IP с другими протоколами

Как видно из рисунка, протокол IP взаимодействует с протоколами вышележащего транспортного уровня и уровнем межсетевых интерфейсов.

Транспортный уровень представляет собой интерфейс сетевых приложения для доступа к сети. Основными функциями рассматриваемого уровня являются сегментация сообщений, управление трафиком, мультиплексирование сеанса (предоставление средств приема данных из разных приложений и их передачи в приложение-получатель), а также обнаружение и исправление ошибок.

Транспортный уровень включает в себя два основных протокола: протокол управления передачей (TCP, Transmission Control Protocol) и протокол

пользовательских дейтаграмм (UDP, User Datagram Protocol). Рассмотрим каждый из них.

Техническая спецификация протокола TCP определяется документом RFC 793 [13]. В указанном документе описываются основные функции, которыми обладает протокол TCP, его программные реализации, а также интерфейсы, которым требуется TCP сервис.

Согласно указанному документу, данный протокол предназначен для организации надежного и потенциально безопасного обмена данными между приложениями, т.е. обмена данными, требующими гарантированной доставки. Гарантия доставки обеспечивается за счет использования порядковых номеров и передачи подтверждений. Протокол TCP обладает поддержкой таких функций, как:

- базовый обмен данными (Basic Data Transfer) – поддержка непрерывного потока октетов в каждом направлении между отправителем и получателем;
- надежность (Reliability) – восстановление данных в случае их дублирования, повреждения, утраты, доставки в неверном порядке;
- управление потоком данных (Flow Control) – управление передаваемыми отправителем данными с помощью возврата с каждым установленным флагом ACK значения поля Window, которое показывает диапазон допустимых номеров октетов, которые могут быть отправлены до получения следующего разрешения;
- мультиплексирование (Multiplexing) – одновременная обработка нескольких потоков данных;
- поддержка соединений (Connections) – поддержка информации о состоянии для каждого потока данных, включающая в себя сокет, порядковые номера и размер окна;

– предпочтения и безопасность (Precedence and Security) – возможность установки пользователем предпочтений и режима безопасности для соединений.

Согласно пункту 3.1 Формат заголовка (Header Format) документа RFC 793 [13], заголовок дейтаграммы TCP имеет вид, представленный на рисунке 5.

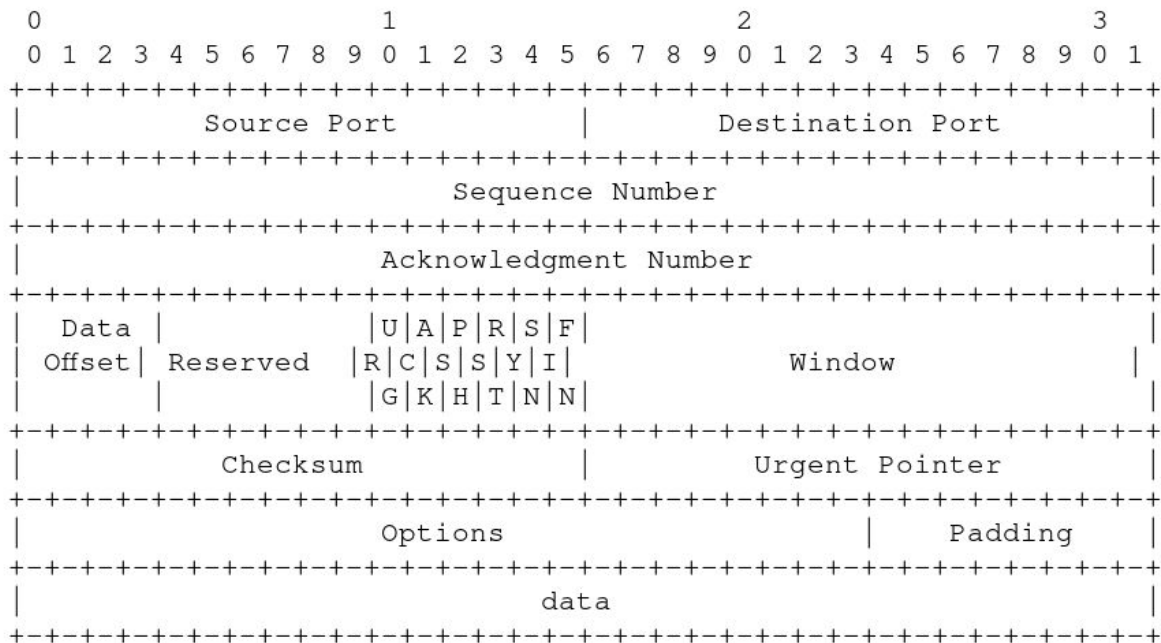


Рисунок 5 – Формат заголовка TCP-дейтаграммы

Поля Source Port (32 бита) и Destination Port (32 бита) содержат значения портов отправителя и получателя соответственно.

Значение поля Sequence Number (32 бита) представляет собой порядковый номер первого октета данных в сегменте (в случае, если в рассматриваемом сегменте присутствует бит SYN, первый октет данных имеет значение, равное значению поля плюс 1).

Значение поля Acknowledgment Number (32 бита) представляет собой следующий порядковый номер, который отправитель должен получить (в случае, если бит ACK установлен).

Поле Data Offset (4 бита) указывает на начало данных в сегменте.

Поле Control Bits (6 битов) содержит значения следующих битов управления:

- URG (указывает на значимость поля Urgent Point);
- ACK (указывает на значимость поля Acknowledgment Number);
- PSH (функция Push);
- RST (сброс соединения);
- SYN (синхронизация порядковых номеров);
- FIN (у отправителя больше нет данных).

Значение поля Window (16 битов) представляет количество октетов данных, которые отправитель ожидает получить.

Поле Checksum (16 битов) содержит значение контрольной суммы заголовка.

Поле Urgent Point (16 битов) представляет собой указатель на срочные данные, т.е. смещение их начала от порядкового номера текущего сегмента.

Протокол UDP представляет собой минимальный механизм передачи сообщений между прикладными программами без установления соединения. В отличие от протокола TCP, рассматриваемый протокол не гарантирует доставки, защиту от дублирования и повреждения дейтаграмм. Техническая спецификация протокола UDP определяется документом RFC 768 [14]. Согласно рассматриваемой спецификации, UDP-заголовок имеет вид, представленный на рисунке 6.

Аналогично TCP пакету, поля Source Port (16 бит) и Destination Port (16 бит) содержат значения портов отправителя и получателя, значение поля Length (16 бит) представляет собой длину общую дейтаграммы (как заголовка, так и данных), а поле Checksum (16 бит) представляет собой контрольную сумму дейтаграммы.

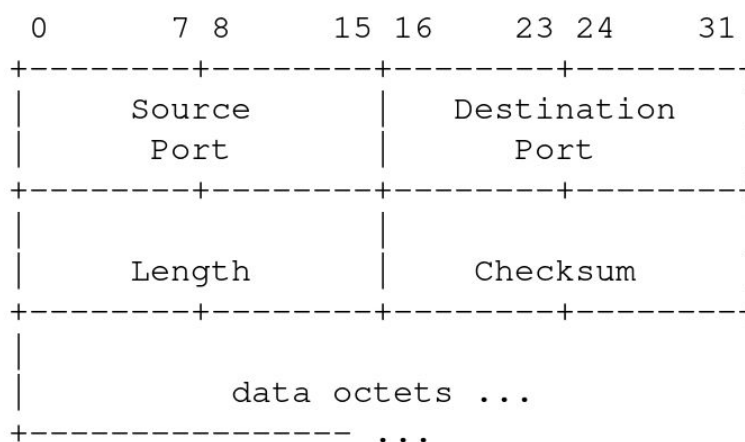


Рисунок 6 – Формат UDP заголовка

В условиях данной задачи наибольший интерес представляют поля TCP и UDP дейтаграмм, определяющие номера портов отправителя и получателя, а также значения битов управления (для TCP дейтаграммы).

1.6 Обзор архитектуры ОС Android

Android – это операционная система с открытым исходным кодом, основанная на ядре Linux и разрабатываемая корпорацией Google. Рассматриваемая операционная система используется на миллионах устройств – от мобильных телефонов до наручных часов и телевизоров – в более чем 190 странах по всему миру.

Android представляет собой программный стек, состоящий из нескольких компонентов. Основные компоненты рассматриваемой операционной системы можно представить в виде иерархии, представленной на рисунке 7.

Как было сказано выше, ОС Android базируется на урезанном ядре Linux, что позволяет рассматриваемой операционной системе использовать ключевые функции безопасности, а также разработчикам создавать аппаратные драйвера для популярного ядра. Помимо этого, к функциям ядра относятся управление памятью, оборудованием, питанием и задачами, потоковая обработка, предоставление драйверов и т.д.

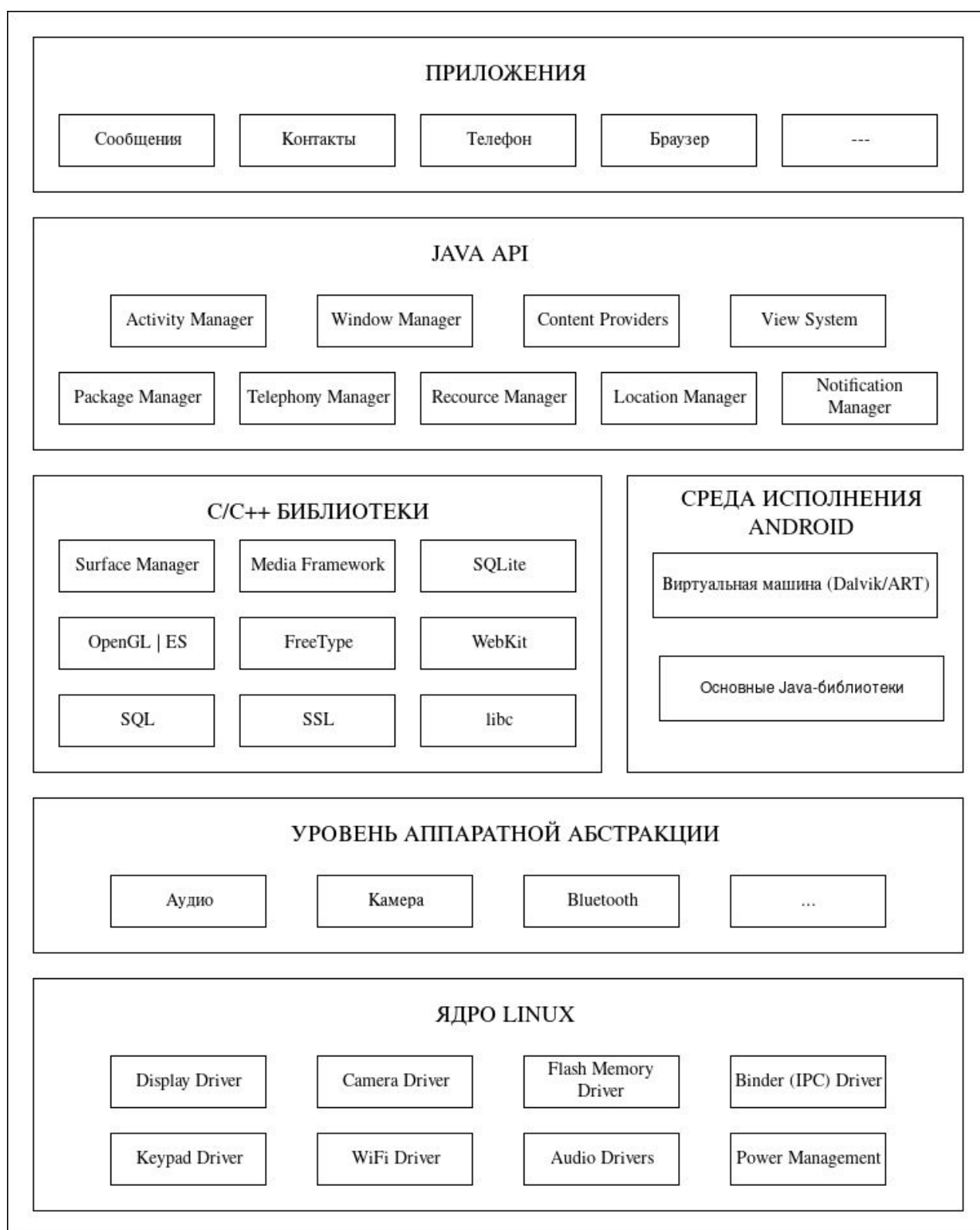


Рисунок 7 – Архитектура ОС Android

Следующим уровнем программного стека ОС Android является уровень аппаратной абстракции. Данный уровень предоставляет собой совокупность

нескольких библиотечных модулей, каждый из которых представляет собой интерфейс определенного аппаратного компонента (например, аудио, камера или Bluetooth) и который загружается в случае, когда API-интерфейсу требуется доступ к аппаратным ресурсам.

Среда исполнения Android включает в себя виртуальные машины Dalvik и ART (Android Runtime). Dalvik и ART (Android Runtime) представляют собой управляемые среды выполнения, которые используются приложениями и некоторыми системными службами Android. Работа приложений изолирована от ОС и ограничена пределами виртуальной машины, что обеспечивает защиту ядра от каких-либо воздействий. Таким образом, приложение запускается в собственном экземпляре виртуальной машины, которая, в свою очередь, ограничивается процессом, управляемым ядром Linux. Данный механизм представлен на рисунке 8.



Рисунок 8 – Механизм работы Android приложения

Виртуальная машина Dalvik является предшественником ART (Android Runtime) и используется на устройствах под управлением ОС Android ниже версии 5.0. Однако, если разрабатываемое приложение работает со средой выполнения ART, то оно может работать и со средой выполнения Dalvik, в то время как выполнение обратного не гарантируется. Помимо виртуальных

машин, среда исполнения Android включает в себя набор основных Java-библиотек, которые обеспечивают большую часть функциональности языка программирования Java.

Многие базовые компоненты и сервисы Android (например, ART или HAL) требуют нативные библиотеки, написанные на языках C и C++. Данные библиотеки включены в промежуточный уровень программного обеспечения и предоставляют необходимый функционал для приложений. Например, за запись и воспроизведение аудио и видео отвечает Media Framework, поддержку 2D- и 3D-графики обеспечивает OpenGL ES, для работы с базами данных используется SQLite и т.д. В случае, если разрабатываемому приложению требуется код, написанный на языках C/C++, доступ к некоторым из нативных библиотек можно получить непосредственно из кода, используя Android NDK.

Следующим уровнем программного стека является совокупность API, написанных на языке Java. С помощью Java API осуществляется доступ к основному функционалу и библиотекам ОС Android. Рассматриваемый уровень включает в себя такие компоненты как Resource Manager (предоставляет доступ к ресурсам – графике, локализованным строкам, аудио и т.д.), Content Providers (позволяет приложениям получать доступ к данным из других приложений или предоставлять собственные данные), View System (используется для создания пользовательского интерфейса) и др.

Верхний уровень программного стека представляет собой совокупность приложений – как пользовательских, так и системных. Все разрабатываемые приложения работают на этом уровне. Устройство под управлением ОС Android поставляется с набором основных приложений (контакты, SMS-сообщения, электронная почта, браузер и т.д.). Системные приложения предоставляют ключевой функционал, которым разработчики могут использовать в своем приложении. Например, если разрабатываемому

приложению необходимо отправить SMS-сообщение или совершить звонок, не нужно самостоятельно создавать эту функцию – можно использовать любое приложение для SMS-сообщений или звонков, уже установленное на устройстве.

1.7 Обзор структуры Android-приложения

Android представляет собой мощную среду разработки для создания приложений и игр, которая основывается на сочетании языка программирования Java и расширяемого языка разметки XML.

Внешний вид каждого из экранов Android-приложения определяется с помощью макетов (layouts), которые описываются на языке разметки XML и представляют собой совокупность компонентов графического пользовательского интерфейса (GUI) – кнопок, текстовых полей, списков и т.д.

Язык программирования Java используется для реализации функционала программы. Специальные java-классы, так называемые активности (activity), используют указанные макеты и определяют, какие действия должны происходить при определенных условиях. Например, если макет содержит кнопку, в java-классе необходимо описать, что должно происходить при нажатии на неё.

Activity – это один из фундаментальных компонентов приложения на платформе Android, который предоставляет экран, с помощью которого пользователи могут взаимодействовать с приложением, например, набрать номер телефона, сделать фотографию, просмотреть карту и т.д. Как правило, Android-приложение может состоять из нескольких activity, слабо связанных друг с другом. Для каждой activity указывается макет, который используется для прорисовки окна графического интерфейса.

Для разработки стабильных и надежных приложений необходимо учитывать следующие основные состояния activity:

а) возобновлена (activity находится в стадии выполнения, пользователь видит activity);

б) приостановлена (пользователь не видит activity, поскольку она перекрывается другой activity, однако она по-прежнему содержит информацию о компонентах и находится в памяти);

в) остановлена (activity по-прежнему содержит информацию о компонентах и находится в памяти, но никак не связана с диспетчером окон) [16].

Обработка переходов из одного указанного состояния в другое реализуется через ряд методов, в совокупности составляющий жизненный цикл activity. Указанный жизненный цикл схематично представлен на рисунке 9.

Реализация каждого программного компонента, обозначенного на предыдущих стадиях данной выпускной квалификационной работы, основана на фрагментах – классе `Fragment`, представляющем собой поведение или часть пользовательского интерфейса в activity. Одна activity может взаимодействовать с неограниченным количеством фрагментов. Так, на примере разрабатываемого приложения, класс `MainActivity` взаимодействует с фрагментами `FragmentPackets` (отображение списка перехваченных сетевых пакетов), `FragmentDetails` (просмотр подробной информации о сетевом пакете), `FragmentFilters` (установка фильтров для перехваченного сетевого трафика).

Фрагмент обладает собственным жизненным циклом, однако на него напрямую влияет жизненный цикл activity, т.е. в случае, если activity приостановлена, то в этом же состоянии находятся все фрагменты, которые встроены в неё. Жизненный цикл фрагмента и его взаимосвязь с состоянием, в котором находится activity, представлен на рисунке 10.

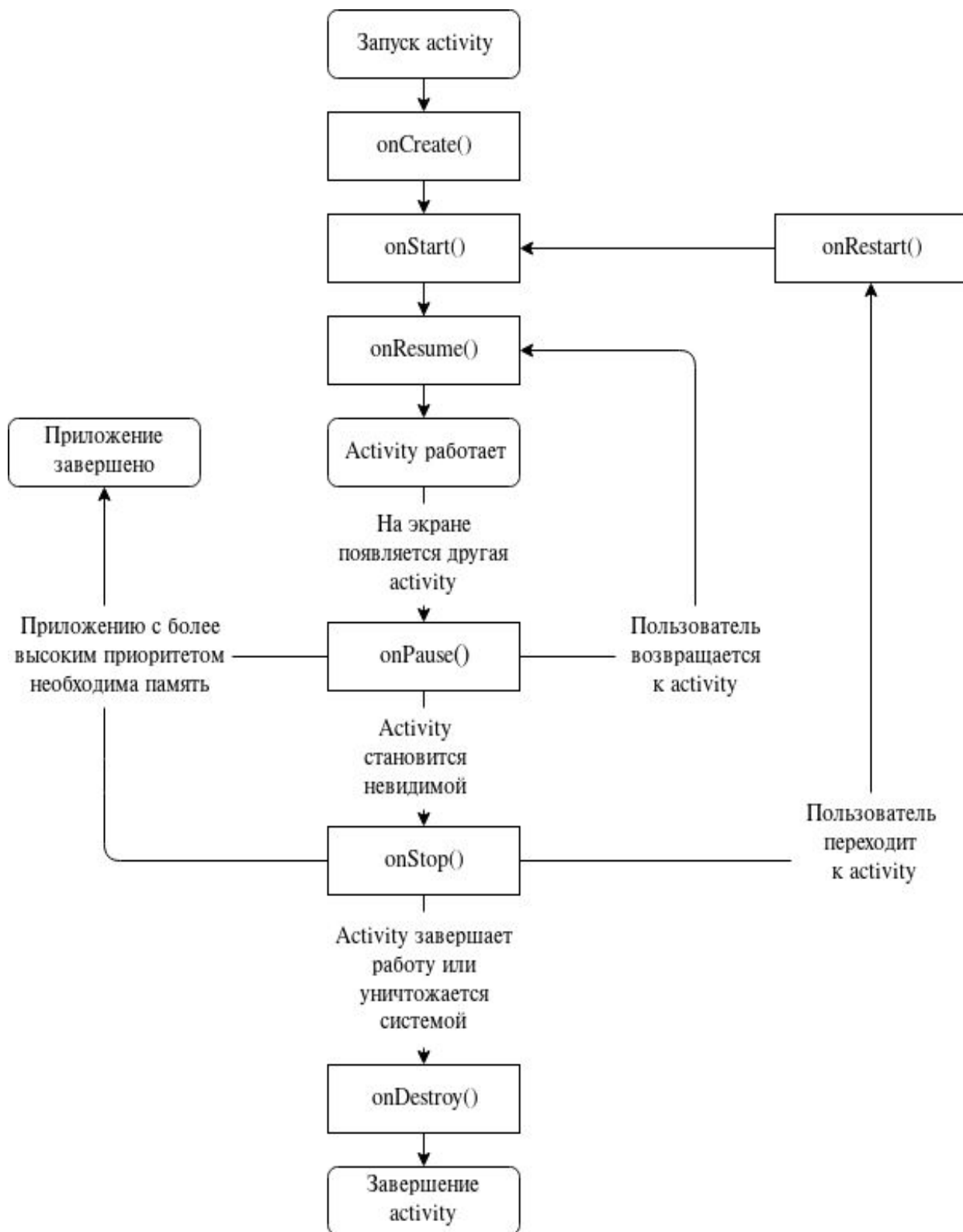


Рисунок 9 – Жизненный цикл activity

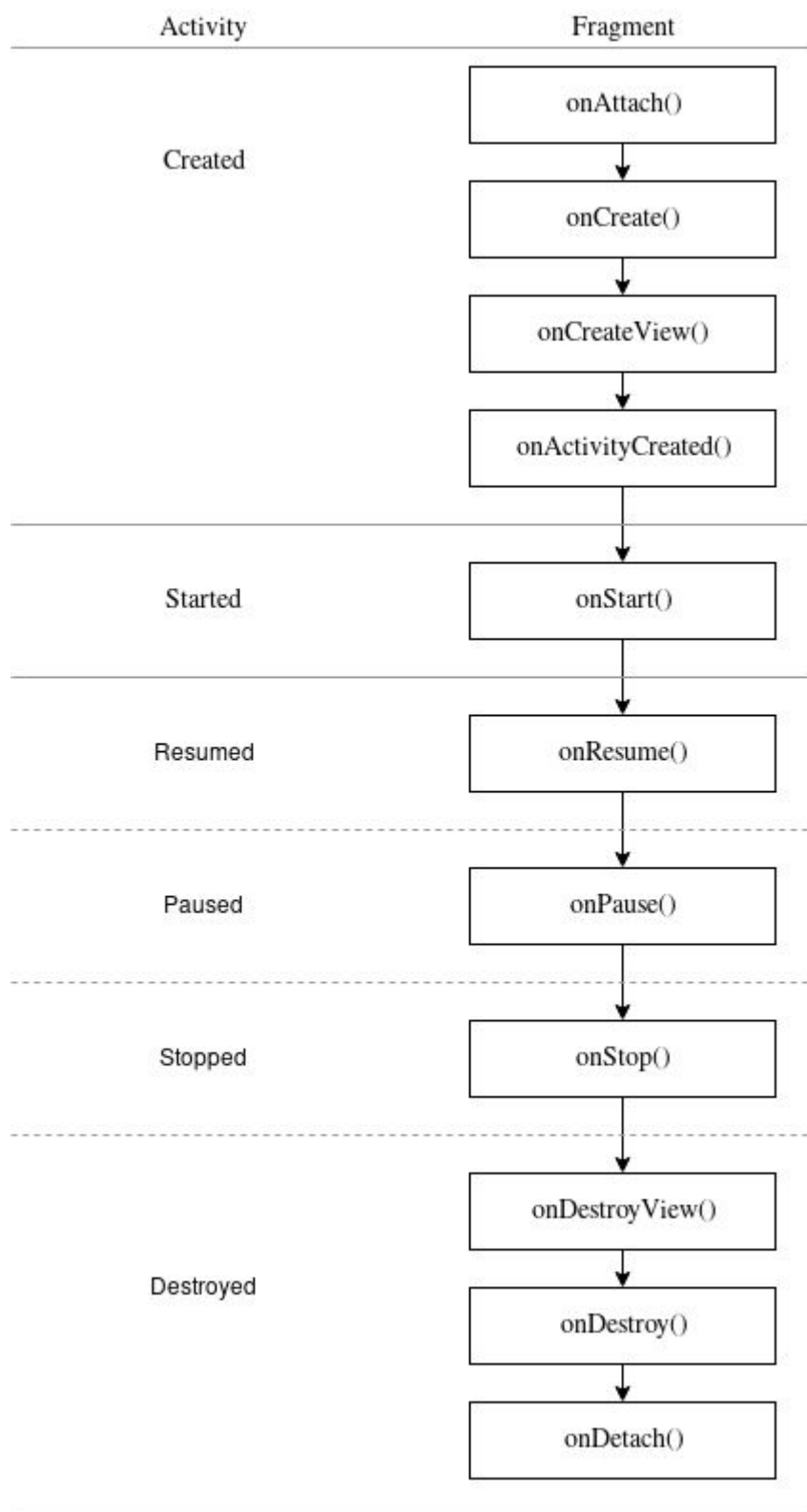


Рисунок 10 – Жизненный цикл фрагмента и его взаимосвязь с состоянием activity

При создании фрагмента, как правило, необходимо реализовать следующие методы жизненного цикла:

а) `onCreate()` – метод, который вызывается системой при создании фрагмента. В реализации этого метода следует инициализировать ключевые компоненты фрагмента;

б) `onCreateView()` – метод, который вызывается системой при первом отображении пользовательского интерфейса на экране, следовательно, в его реализации следует инициализировать компоненты пользовательского интерфейса;

в) `onPause()` – метод, который вызывается системой в случае, если пользователь покидает фрагмент, например, при переходе с одного экрана на другой [17].

Помимо Java-кода и макетов Android-приложениям необходимы дополнительные ресурсы, в частности, графические и аудио-файлы, локализованные строки и т.д.

Кроме того, необходимой частью любого Android-приложения является так называемый манифест – файл `AndroidManifest.xml`, который хранится в корневом каталоге приложения и содержит необходимую информацию приложения, в частности, список компонентов, необходимых библиотек и разрешений, требуемых приложению для его корректной работы и выполнения определенного функционала (например, разрабатываемому приложению требуется разрешение на использование VPN-соединения и доступ в Интернет).

2 Практическая часть

2.1 Формализация задачи

На предыдущем этапе выполнения выпускной квалификационной работы сформулированы основные функции разрабатываемого программного обеспечения, в частности:

- перехват сетевого трафика в режиме реального времени;
- просмотр подробной информации о перехваченных пакетах;
- фильтрация перехваченного трафика по заданным пользователем параметрам.

Для дальнейшей реализации приложения выполнена детализация каждой из вышеперечисленных функций. Так, перехват сетевого трафика должен осуществляться:

- с использованием локального VPN сервера;
- в реальном времени;
- не влияя на работоспособность приложений, отправляющих/принимающих сетевой трафик;
- в фоновом режиме.

Пользователь должен иметь возможность просмотра подробной информации о перехваченном пакете, в частности:

- информацию о приложении, выполняющем обработку пакета (название, идентификатор процесса);
- ip-адреса и используемый порт источника и получателя пакета;
- доменные имена источника/получателя пакета;
- название протокола;
- версия протокола;
- флаги (при наличии);
- данные (при наличии);

- время перехвата пакета.

Фильтрация перехваченного трафика должна быть доступна по следующим параметрам:

- приложение, выполняющее обработку пакета;
- ip-адрес источника пакета;
- ip-адрес получателя пакета;
- тип протокола.

2.2 Используемые шаблоны проектирования

При разработке приложения под Android устройства нередко весь функциональный код, а также обработка взаимодействий с пользователем и работа с асинхронными задачами помещается в методы жизненного цикла Activity или Fragment, тем самым усложняя поддержку, расширение и тестирование приложения. Для решения этой проблемы для каждого из компонентов разрабатываемого программного обеспечения использовался шаблон MVP (Model-View-Presenter). Данный шаблон предоставляет возможность отделить логику от уровня представления, т.е. то, как работает интерфейс, от того, как он представляется на экране, что, в свою очередь, делает фоновые процессы независимым от большинства событий, связанных с жизненным циклом. Таким образом, шаблон MVP позволяет разделить обязанности приложения на следующие уровни: model, view, presenter [18].

Модель (Model) представляет собой интерфейс, ответственный за доступ и управление данными. К основным функциям рассматриваемого слоя относится использование API удаленного сервера, кэширование данных, управление базами данных и т.д.

Посредник (Presenter) представляет собой посредника между графическим представлением (View) и данными (Model). Он извлекает данные из модели, выполняет их обработку и передает их View. Помимо этого, он

отвечает за обновление модели и содержит методы, отвечающие за взаимодействие пользователя через интерфейс приложения.

Представление (View) отвечает за представление данных. Помимо этого, он вызывает методы посредника (Presenter) при каждом взаимодействии пользователя с графическим интерфейсом (например, нажатие кнопок).

Таким образом, преимуществами использования данного шаблона являются:

- разделение обязанностей между компонентами, что, в свою очередь, облегчает понимание, расширение и поддержку кода;
- возможность изменения графического интерфейса без повреждения логики приложения;
- облегчение автоматизированного тестирования приложения, т.к. из-за четких границ между компонентами, становится намного проще тестировать каждый компонент в отдельности.

Графическая структура шаблона MVP представлена на рисунке 11.

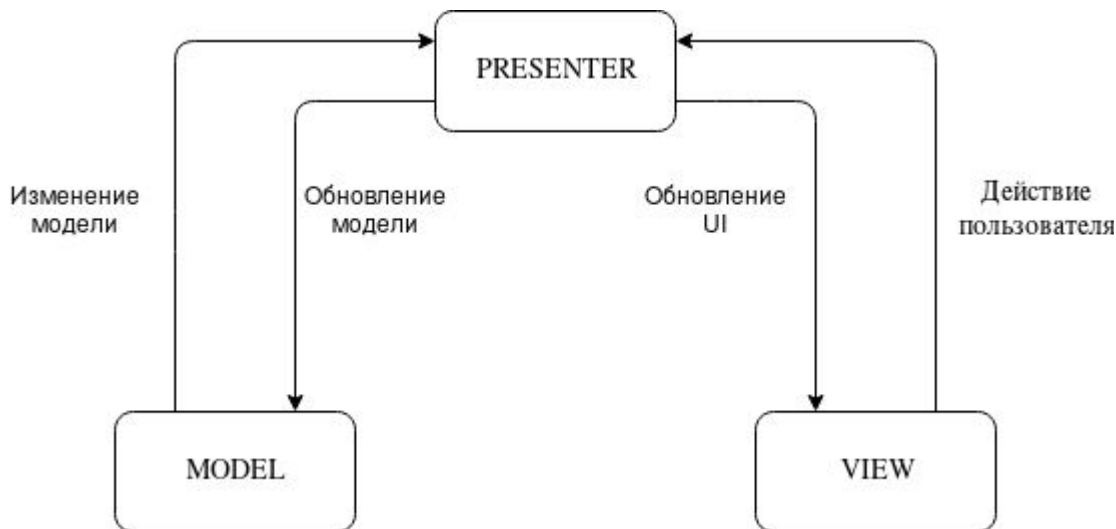


Рисунок 11 – Структура шаблона MVP

Другим архитектурным шаблоном, используемым при разработке программного обеспечения в данной выпускной квалификационной работе, является шаблон Внедрение зависимости (Dependency Injection, DI) – процесс

предоставления внешней зависимостей, необходимых для компонента, который возлагается на внешний, специально предназначенный для этого общий модуль. Данный шаблон является «специфичной формой инверсии управления» (Inversion of control), и его использование позволяет решить следующие проблемы:

- изменение одного программного компонента не ведет к изменениям других программных компонентов;
- изменение одного программного компонента не приводит к возникновению неконтролируемых ошибок в других программных компонентах;
- программный компонент легко изолировать от остальной части приложения для повторного использования или тестирования.

В случае, если разрабатываемое приложение содержит в себе множество классов, каждый из которых обладает своими зависимостями, их ручное внедрение через точку входа приложения (например, функция `main()` или метод `onCreate()`) является затруднительной задачей. Поэтому для реализации рассматриваемого шаблона в разрабатываемом приложении использовалась библиотека с открытым исходным кодом Dagger2, который автоматически генерирует весь код для внедрения зависимостей. Основными преимуществами использования этой библиотеки являются:

- простая конфигурация сложных зависимостей;
- код, генерируемый данной библиотекой, легко понять и отладить;
- Dagger2 предоставляет простой способ получить ссылки на общие экземпляры;
- простота тестирования модулей и интеграция;
- малые размеры библиотеки [19].

Для упрощения взаимодействия (обмена данными и событиями) между компонентами приложения использовался поведенческий шаблон проектирования Издатель-Подписчик (Pub-Sub). Связь между объектами осуществляется через шину событий (Event Channel). Издатель отправляет событие в шину событий, а подписчик, в свою очередь, подписывается на нужное событие и слушает его на шине [20]. Схематично рассматриваемый шаблон представлен на рисунке 12.

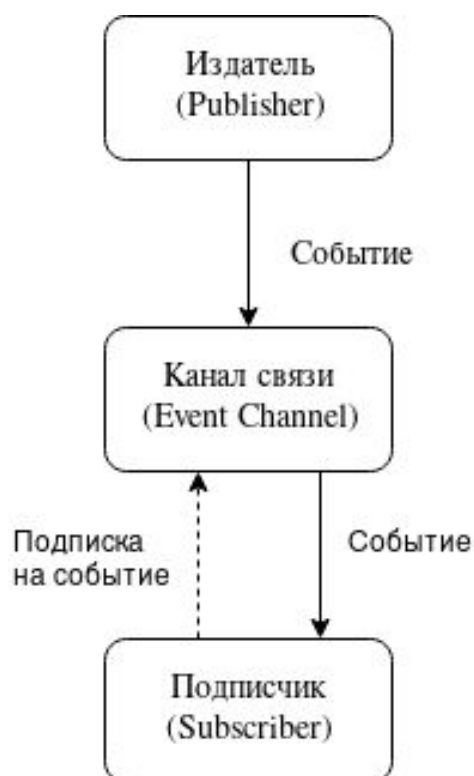


Рисунок 12 - Шаблон Издатель-подписчик

Для упрощения реализации данного шаблона в рамках разрабатываемого программного обеспечения используется библиотека EventBus. Указанная библиотека упрощает связь между издателями и подписчиками, а также хорошо работает с фрагментами и фоновыми потоками, избегая сложных зависимостей. Кроме этого, плюсом этой библиотеки является небольшой размер.

2.3 Структура программного обеспечения

При разработке архитектуры программного обеспечения целесообразно выполнить декомпозицию программы на отдельные функциональные или

логические компоненты, которые содержат четко определенные методы и события. Это позволяет обеспечить более высокий уровень абстракции и гибкости системы, а также облегчает задачу тестирования и поддержки программного обеспечения.

Каждый компонент представляет собой программную составляющую, предназначенную для решения одной из задач разрабатываемого программного обеспечения и представляющую для данной цели весь необходимый функционал. Помимо этого, каждый компонент характеризуется набором входящих данных, необходимых для выполнения обозначенных функций [11].

Каждый из компонентов должен обладать следующими свойствами:

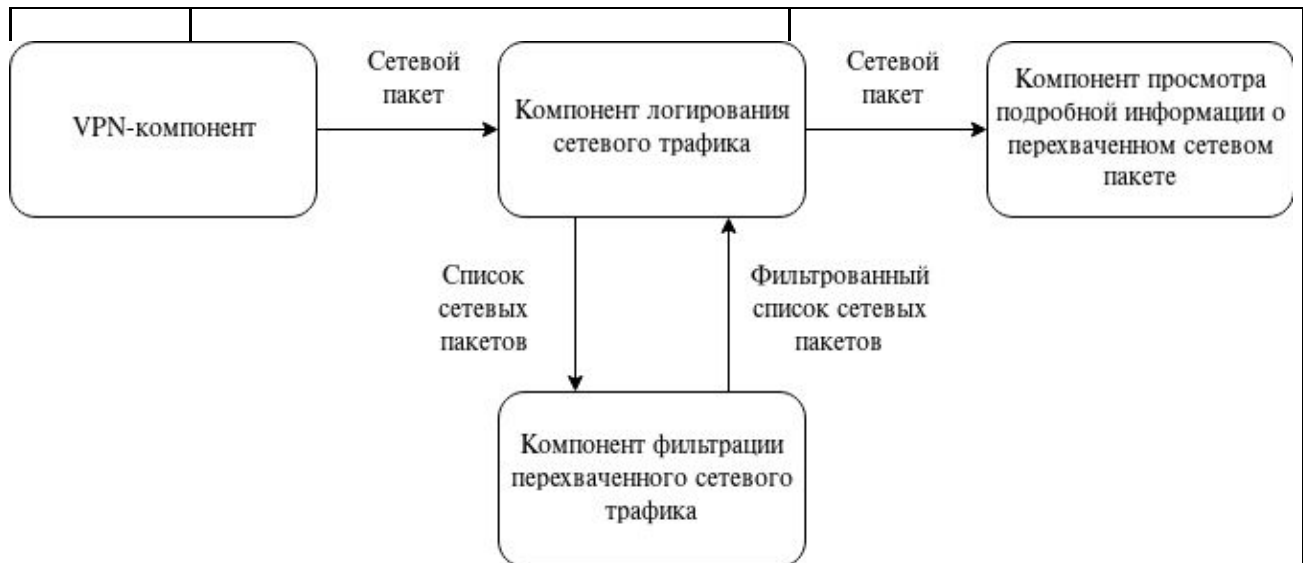
- работа программного компонента не должна зависеть от работы других компонентов (только от входящих данных);
- программный компонент должен обладать функциональной целостностью, т.е. решать только одну задачу и предоставлять для её решения весь необходимый функционал.

Таким образом, на основе анализа функционала разрабатываемого программного обеспечения, его можно представить в виде следующих программных компонентов:

- VPN-компонент;
- компонент логирования сетевого трафика;
- компонент просмотра подробной информации о пакете;
- компонент фильтрации перехваченного сетевого трафика.

После определения основных компонентов системы выполнена идентификация их взаимодействия между друг другом, а также входные данные для каждого из них. Результат представлен в виде схемы на рисунке 13.

На основании разработанной схемы взаимодействия разработана структура Java-кода, которая представлена в таблице 4. Код программы представлен в приложении А.



Каталог	Файл	Описание
adapters/	PacketsListAdapter.java	Представляет собой адаптер, который используется для отображения списка перехваченных пакетов
common/	Constants.java	Содержит список общих константных переменных, в частности, сетевые адреса и протоколы, маршруты и т.д.
	Utils.java	Включает в себя общие методы, например, получение текущей даты, списка

		используемых приложений и др.
di/	application/AppComponent.java	Реализация шаблона проектирования Dependency Injection
	application/AppModule.java	
	details/DetailsComponent.java	
	details/DetailsModule.java	
	filters/FiltersComponent.java	
	filters/FiltersModule.java	
	packets/PacketsComponent.java	
	packets/PacketsModule.java	
events/	AppInfosEvent.java	Событие, отправляемое при получении информации об установленных приложениях
	DetailsEvent.java	Событие, отправляемое при запросе подробной информации о пакете
	UpdatePacketsEvent.java	Событие, отправляемое при обновлении списка перехваченных пакетов

fragments/	details/presenter/DetailsPresenter.java	Реализация компонента просмотра подробной информации о пакете
	details/presenter/IDetailsPresenter.java	
	details/view/FragmentDetails.java	
	details/view/IDetailsView.java	
	filters/presenter/FiltersPresenter.java	Реализация компонента фильтрации перехваченных пакетов
	filters/presenter/IFiltersPresenter.java	
	filters/FragmentFilters.java	
	filters/IFiltersView.java	
	packets/presenter/PacketsPresenter.java	Реализация компонента логирования сетевого трафика
	packets/presenter/IPacketsPresenter.java	
	packets/view/FragmentPackets.java	
	packets/view/IPacketsView.java	
models/	Packet.java	Модель сетевого пакета
	PacketFilter.java	Модель фильтра

service/	handlers/CommandHandler.java	Обработчик команд установки/деактивации VPN-соединения
	handlers/PacketHandler.java	Обработчик нового перехваченного сетевого пакета
	MintferVpnService.java	Программная реализация VPN сервера
/	MainActivity.java	Главная активность, которая отвечает за запуск приложения

Рисунок 13 – Взаимодействие компонентов разрабатываемого программного обеспечения

Таблица 4 – Структура приложения

Как видно из таблицы 4, в разрабатываемом программном обеспечении используются две модели – для описания структуры пакета и фильтра. Подробное описание модели сетевого пакета представлено в таблице 5, а фильтра – в таблице 6.

Таблица 5 – Модель сетевого пакета

Поле	Тип	Описание
version	int	Версия протокола IP
protocol	int	Тип протокола
flags	int	Флаги (при наличии)
sourceAddress	String	Адрес источника
sourcePort	int	Порт источника
destinationAddress	String	Адрес получателя
destinationPort	int	Порт получателя
uid	int	Уникальный идентификатор
data	String	Данные
applicationInfo	ApplicationInfo	Информация о приложении, которое обрабатывает пакет

date	String	Время и дата перехвата пакета
sourceDomainName	String	Доменное имя источника
destinationDomainName	String	Доменное имя получателя

Таблица 6 – Модель фильтра

Алгоритм работы приложения заключается в следующем. Работа программы начинается с запуска MainActivity. При запуске активности открывается экран логирования сетевого трафика, а также вызывается метод получения списка всех установленных на телефоне приложений, который затем сохраняется их в виде списка для дальнейшего использования. Помимо этого, указанная активность обладает следующим основным функционалом:

Поле	Тип	Описание
sourceAddress	String	Адрес отправителя
destinationAddress	String	Адрес получателя
protocol	Constants.TransportProtocol	Тип протокола
applicationInfo	ApplicationInfo	Информация о приложении, которое обрабатывает пакет
isFiltering	boolean	Использование фильтрации пакетов

- обработка пользовательского нажатия на кнопку запуска/остановки перехвата трафика;
- обработка пользовательского нажатия на кнопку фильтрации трафика;
- генерация зависимостей компонентов.

При нажатии на кнопку запуска/остановки перехвата сетевого трафика активность взаимодействует с VPN-компонентом, вызывая его соответствующие методы.

2.4 VPN-компонент

Локальный VPN-сервер является основным компонентом разрабатываемого программного обеспечения. Рассматриваемый компонент реализует следующие функции:

- запуск и остановка локального VPN-сервера;
- перехват и обработка пакета;
- обработка ошибок, возникающих при работе VPN-сервера.

Для программной реализации рассматриваемого компонента использовался интерфейс VPNService [22], доступный для устройств под управлением ОС Android версии 4.0 и выше. Данный класс использует протокол IP, позволяет создавать виртуальный сетевой интерфейс и настраивать адресацию и правила маршрутизации. При работе с указанным интерфейсом необходимо учитывать следующее:

- при первом подключении пользователя требуется получить разрешение от пользователя;
- одновременно может работать только одно VPN-соединение, т.е. при создании нового соединения, предыдущее разъединяется;
- в течение всего времени использования VPN-соединения системой отображается уведомление о его использовании.

Класс VPNService содержит два основных метода:

- prepare(Context) – останавливает VPN-соединение, созданное другим приложением, если такое имеется;
- establish() – устанавливает новое VPN-соединение, используя параметры, предоставленные классом-помощником VPNService.Builder.

Класс VPNService.Builder [23] всегда используется только в рамках VPNService. Основные методы, используемые для установки VPN-соединения в рамках разрабатываемого программного обеспечения, и их краткое описание представлены в таблице 7.

Таблица 7 – Основные методы класса VPNService.Builder

Метод	Параметры	Описание метода
addAddress(String address, int prefixLength)	String address – сетевой адрес, int prefixLength – длина префикса	Устанавливает сетевой адрес VPN-интерфейса
addDnsServer(String address)	String address – DNS-адрес	Используется для настройки DNS
addRoute(String address, int prefixLength)	String address – сетевой адрес, int prefixLength – длина префикса	Используется для настройки маршрутизации
setMtu(int mtu)	int mtu – значение MTU	Устанавливает максимальный объём данных, который

		может быть передан за одну итерацию
addAllowedApplication(String packageName)	String packageName – название пакета (приложения)	Разрешает определенному приложению доступ к VPN-соединению
addDisallowedApplication(String packageName)	String packageName – название пакета (приложения)	Запрещает определенному приложению доступ к VPN-соединению
setSession(String session)	String session – имя сессии	Устанавливает имя сессии
establish()	—	Создает VPN-соединение, используя установленные параметры

Алгоритм работы рассматриваемого компонента заключается в следующем. При нажатии пользователя на кнопку старта перехвата сетевого трафика вызывается метод `prepare(Context)`. Данный метод возвращает так называемое намерение (`Intent`) – механизм, который в рассматриваемом случае является объявлением о намерении запустить сервис. Если указанное возвращаемое значение не является нулевым, вызывается метод запуска сервиса. В данном методе задаются параметры `VPNService.Builder` и создается VPN интерфейс, на который перенаправляются все сетевые пакеты. Каждый сетевой пакет анализируется и передается обработчику `PacketHandler`, который, в свою очередь, по полученному идентификатору процесса определяет приложение, которое является источником или получателем пакета, а затем генерирует событие `UpdatePacketsList`.

2.5 Компонент логирования сетевого трафика

Функцией компонента логирования является просмотр перехваченного сетевого трафика, в частности:

- отображение списка перехваченных пакетов;
- обновление данного списка в режиме реального времени;
- очистка данного списка;
- обработка ошибок, возникающих при работе данного компонента.

Диаграмма классов рассматриваемого компонента представлена на рисунке 14.

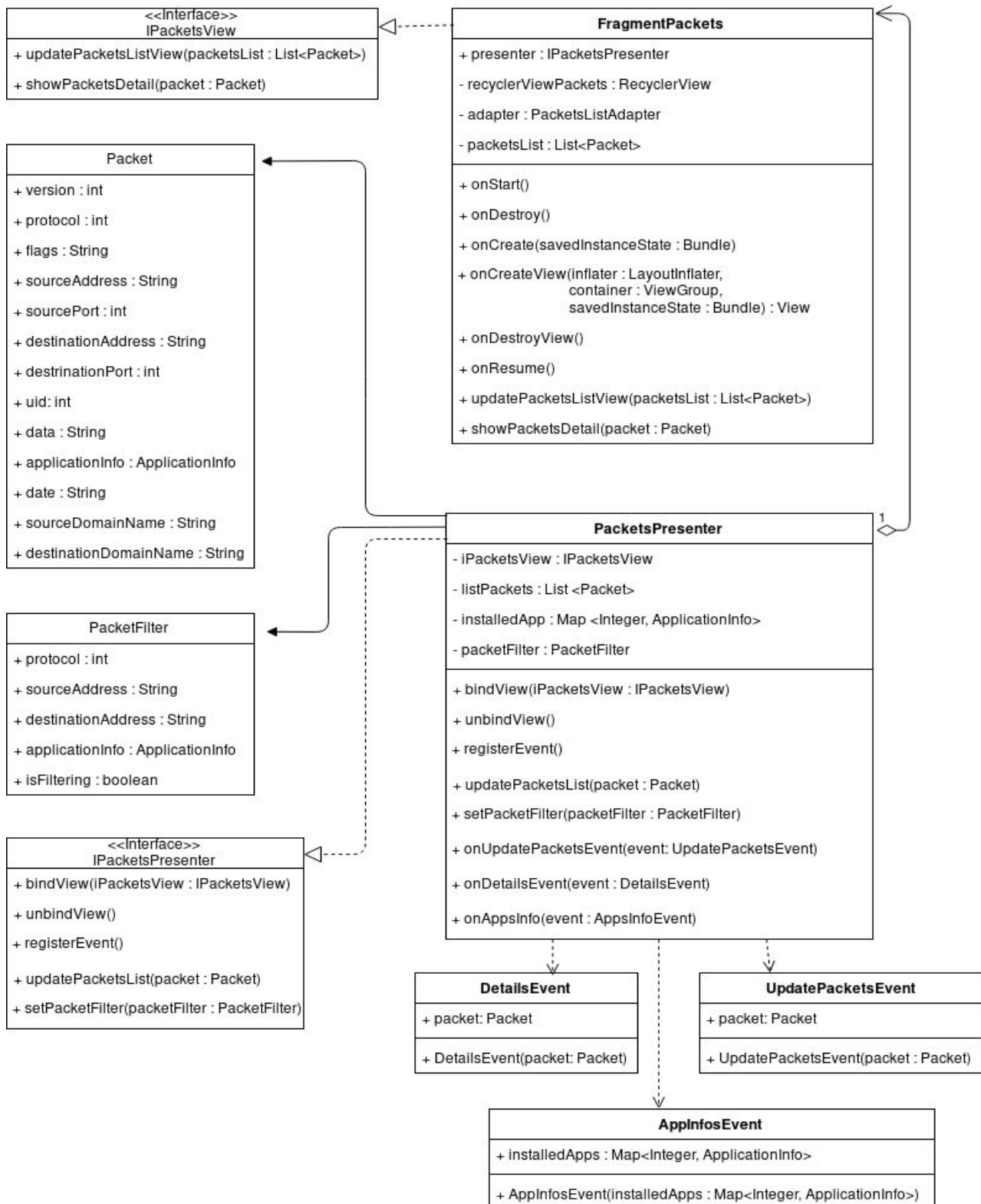


Рисунок 14 – Диаграмма классов компонента логирования сетевого трафика

Как было сказано выше, экран логирования сетевого трафика открывается при запуске приложения. При запуске компонента вызывается метод `registerEvent()`, в котором выполняется подписка на событие `UpdatePacketsEvent`. После того, как VPN-компонент генерирует указанное событие, в методе `onUpdatePacketsEvent(UpdatePacketsEvent event)` извлекается перехваченный пакет и добавляется в список, после чего вызывается метод `updatePacketsList(List<Packets> packetsList)`, который обновляет отображаемый список пакетов.

При нажатии на любой элемент из списка перехваченных пакетов генерируется событие `DetailsEvent(Packet packet)`, входящим параметром которого является модель указанного элемента, после чего открывается экран просмотра подробной информации о пакете.

2.6 Компонент просмотра подробной информации

Компонент просмотра подробной информации о пакете предоставляет собой пользовательский интерфейс просмотра следующих параметров указанного пакета:

- информацию о приложении, выполняющем обработку пакета (название, идентификатор процесса);
- ip-адреса и используемый порт источника и получателя пакета;
- доменные имена источника/получателя пакета;
- название протокола;
- версия протокола;
- флаги (при наличии);
- данные (при наличии);
- время перехвата пакета.

Эти данные извлекаются из переданной модели пакета и отображаются в виде списка. Диаграмма классов указанного компонента представлена на рисунке 15.

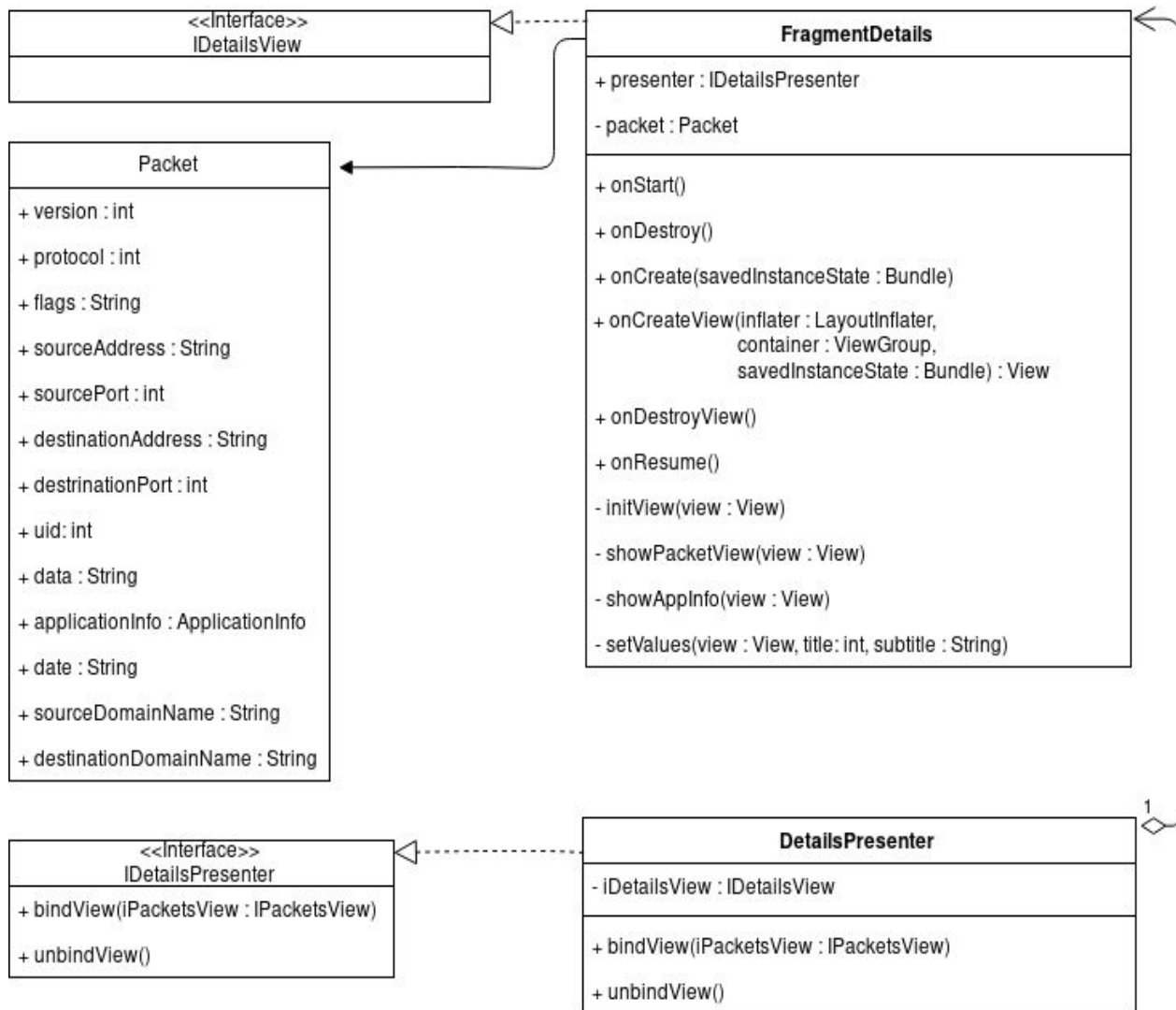


Рисунок 15 – Диаграмма компонента просмотра подробной информации о сетевом пакете

2.7 Компонент фильтрации

Компонент фильтрации перехваченного сетевого трафика по заданным пользователем параметрам обладает следующим функционалом:

- установка одного или нескольких фильтров для списка перехваченных пакетов;
- сброс фильтров для данного списка;

- обработка ошибок, возникающих при работе данного компонента.

Фильтрация списка перехваченных пакетов может осуществляться по следующим параметрам:

- название приложения, выполняющего обработку пакета;
- ip-адрес отправителя пакета;
- ip-адрес получателя пакета;
- тип протокола.

Диаграмма классов компонента фильтрации списка перехваченных пакетов представлена на рисунке 16.

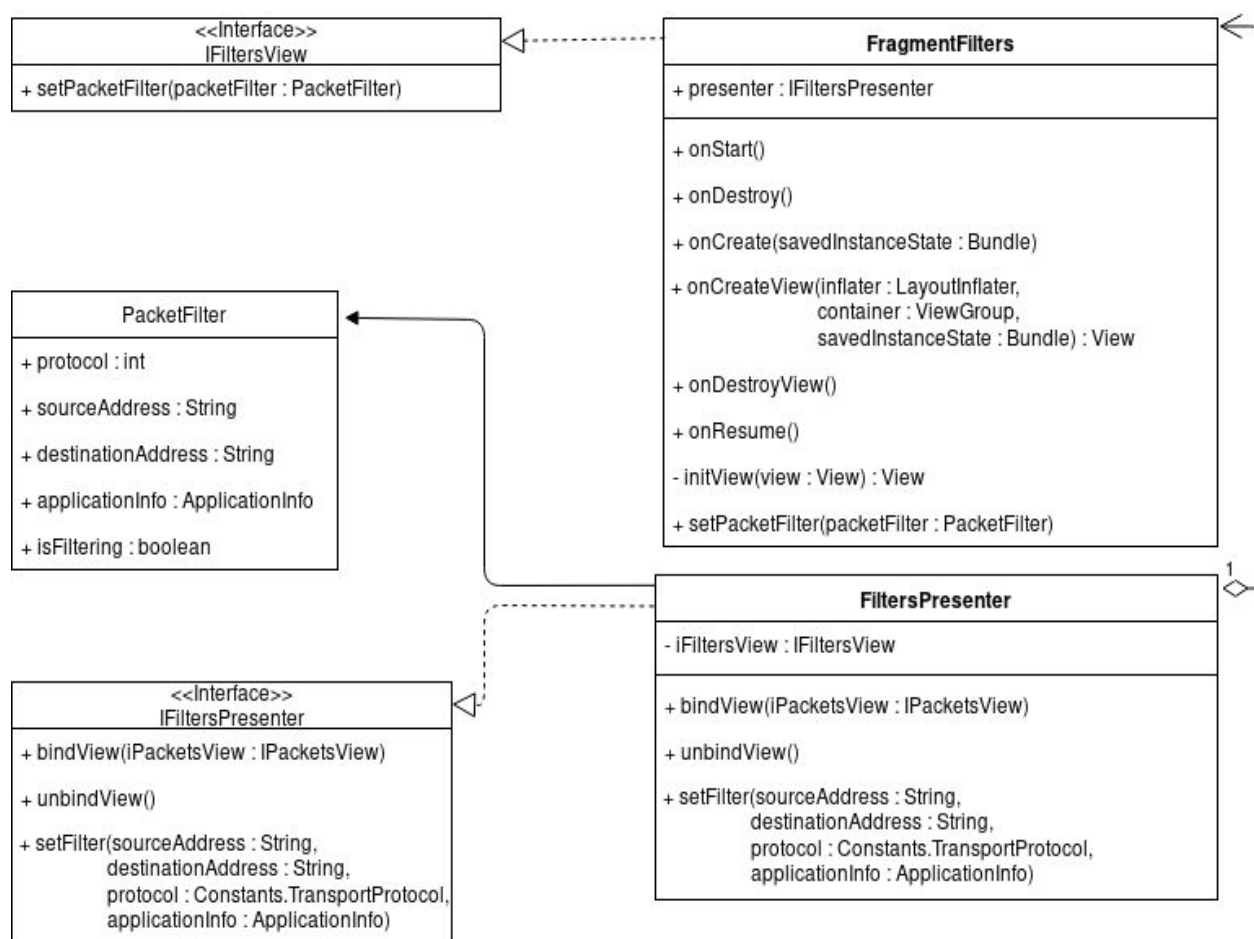


Рисунок 16 – Диаграмма компонента фильтрации сетевого трафика

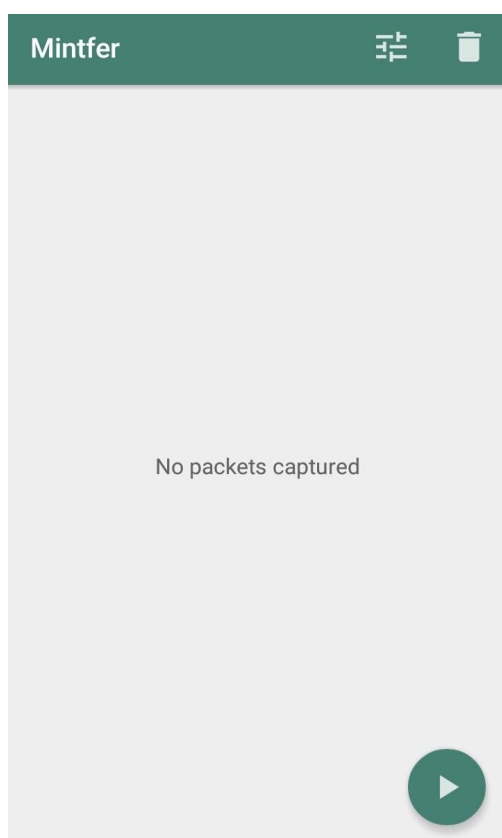
Экран фильтрации списка перехваченных пакетов открывается после нажатия на соответствующую кнопку в верхнем меню. Он представляет собой текстовые поля для ввода пользовательских значений и кнопку сохранения,

после нажатия на которую создается модель PacketFilter и происходит автоматический переход на предыдущий экран со списком перехваченных пакетов, которые фильтруются по заданной модели PacketFilter. Для сброса установленных фильтров необходимо нажать на соответствующую кнопку на экране фильтрации.

2.8 Руководство пользователя

Программное обеспечение, разработанное в ходе выполнения данной выпускной квалификационной работы, предназначено для мониторинга сетевой активности на устройствах под управлением ОС Android. Указанное средство позволяет в режиме реального времени отслеживать сетевой трафик установленных на устройстве приложений, а также проводить анализ и фильтрацию сетевых пакетов. Для использования данного приложения не требуется каких-либо специальных навыков или знаний.

Для работы с приложением, разработанным в ходе выпускной квалификационной работы, необходимо иметь устройство на базе ОС Android версии 4.0 и выше, на который требуется установить apk-файл приложения.



Стартовый экран приложения представлен на рисунке 17. Рисунок 17 –

Стартовый экран приложения

На верхней панели находятся кнопки «Фильтр» и «Корзина». При нажатии на кнопку «Фильтр» откроется экран установки параметров фильтрации. Для того, чтобы очистить текущий список перехваченных пакетов, необходимо нажать на кнопку «Корзина».

Для запуска или остановки перехвата сетевого трафика необходимо нажать на кнопку в правом нижнем углу. При первом запуске приложения появится диалоговое окно с запросом на подключение к сети VPN,

представленное на рисунке 18. Для продолжения работы требуется нажать на



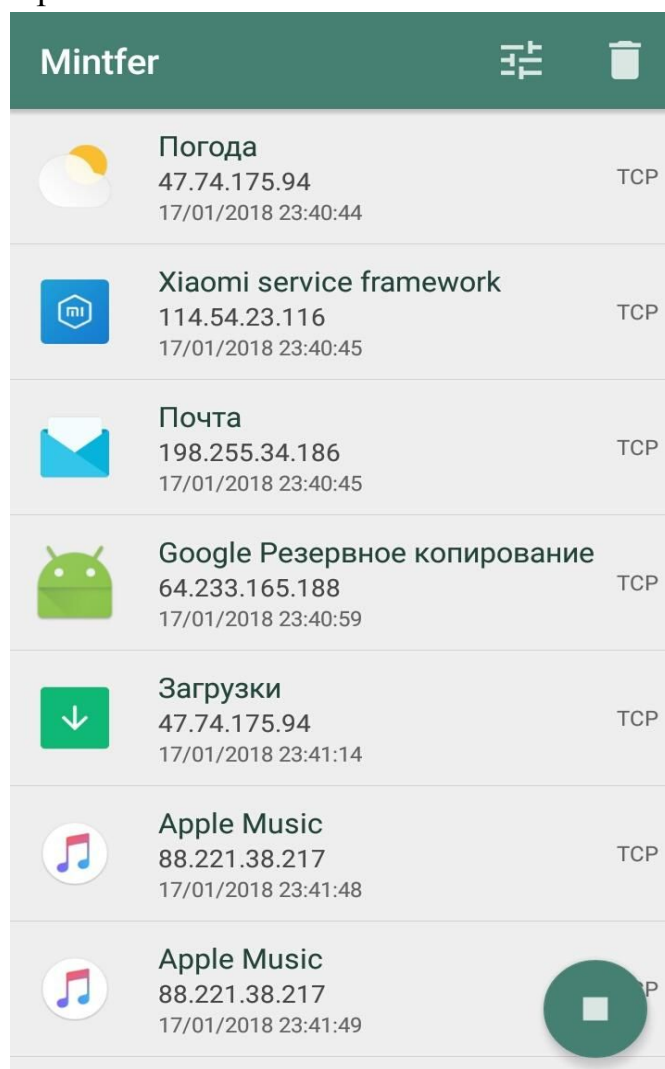
кнопку «ОК».

Рисунок 18 – Запрос на подключение к сети VPN

После начала перехвата сетевого трафика, на главном экране отобразиться динамически обновляемый список перехваченных пакетов, как показано на рисунке 19. Для каждого сетевого пакета отображается:

- название приложения, которое обрабатывает пакет;
- иконка приложения;
- ip-адрес получателя сетевого пакета;
- тип используемого протокола;

– время перехвата сетевого пакета.



The screenshot shows the Mintfer application interface. At the top is a dark green header with the name 'Mintfer' on the left, a list icon in the center, and a trash icon on the right. Below the header is a list of seven intercepted network packets, each with an icon, an application name, an IP address, a timestamp, and a protocol type. The last item in the list has a large green circular button with a white square icon overlaid on it.

Icon	Application	IP Address	Timestamp	Protocol
	Погода	47.74.175.94	17/01/2018 23:40:44	TCP
	Xiaomi service framework	114.54.23.116	17/01/2018 23:40:45	TCP
	Почта	198.255.34.186	17/01/2018 23:40:45	TCP
	Google Резервное копирование	64.233.165.188	17/01/2018 23:40:59	TCP
	Загрузки	47.74.175.94	17/01/2018 23:41:14	TCP
	Apple Music	88.221.38.217	17/01/2018 23:41:48	TCP
	Apple Music	88.221.38.217	17/01/2018 23:41:49	TCP

Рисунок 19 – Отображение списка перехваченных пакетов

Для просмотра подробной информации о каком-либо сетевом пакете необходимо нажать на любой элемент списка, после чего откроется экран, представленный на рисунке 20.

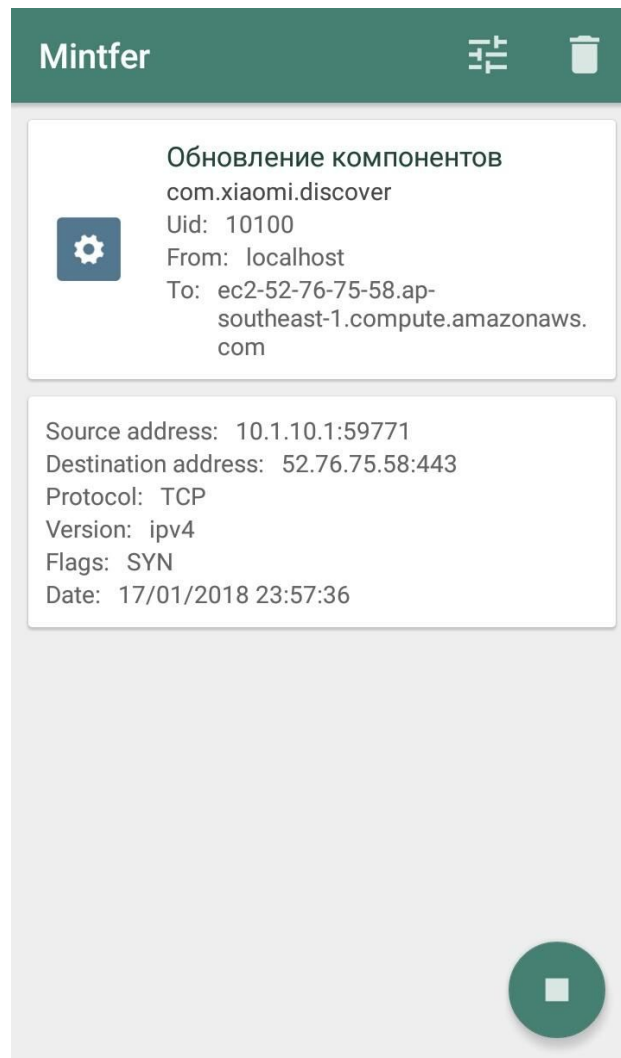


Рисунок 20 – Экран просмотра подробной информации о пакете

В верхней части экрана отображается следующая информация:

- название приложения, которое обрабатывает пакет;
- имя пакета;
- уникальный идентификатор;
- иконка приложения;
- доменное имя отправителя;
- доменное имя получателя.

В нижнем блоке содержатся детальная информация о выбранном сетевом пакете, в частности:

- IP-адрес и порт отправителя;

- IP-адрес и порт получателя;
- тип используемого протокола;
- версия протокола;
- флаги (при наличии);
- дата перехвата пакета.

При нажатии на кнопку «Фильтры» в верхнем меню открывается экран, представленный на рисунке 21.

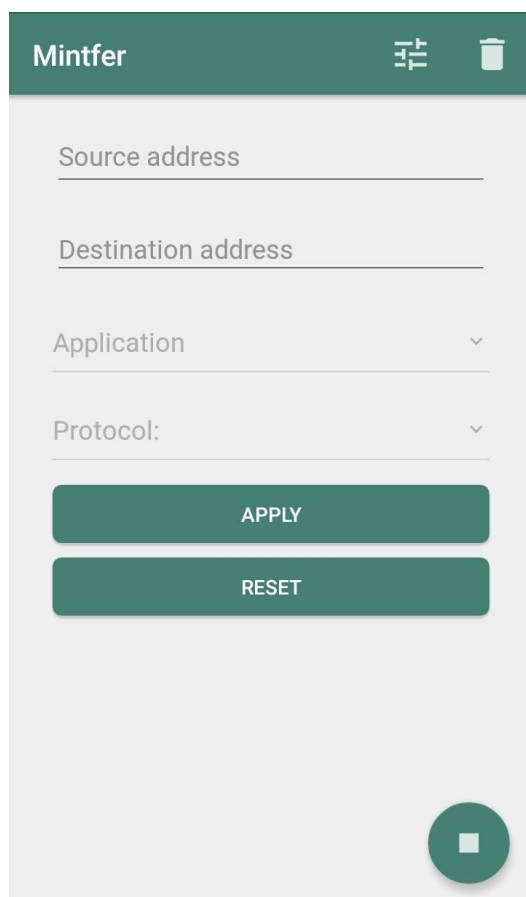


Рисунок 21 – Экран установки фильтров

На данном экране можно установить параметры фильтрации для перехваченного трафика. Для этого в поля «Source address» и «Destination address» необходимо ввести IP-адреса отправителя и получателя, соответственно. При нажатии на поле «Application» откроется список установленных на устройстве приложений. Для выбора приложения

необходимо нажать на его название. Аналогично с полем «Protocol» – из списка доступных протоколов необходимо выбрать нужный.

Можно указать как все параметры фильтрации, так и некоторые, заполнив соответствующие поля. После ввода данных необходимо нажать на кнопку «Apply», после чего откроется главный экран приложения, на котором будет отображен список перехваченных пакетов, отфильтрованный по заданным параметрам. Для сброса установленных фильтров необходимо нажать на кнопку «Reset».

Пример возможных параметров фильтрации представлен на рисунке 22.

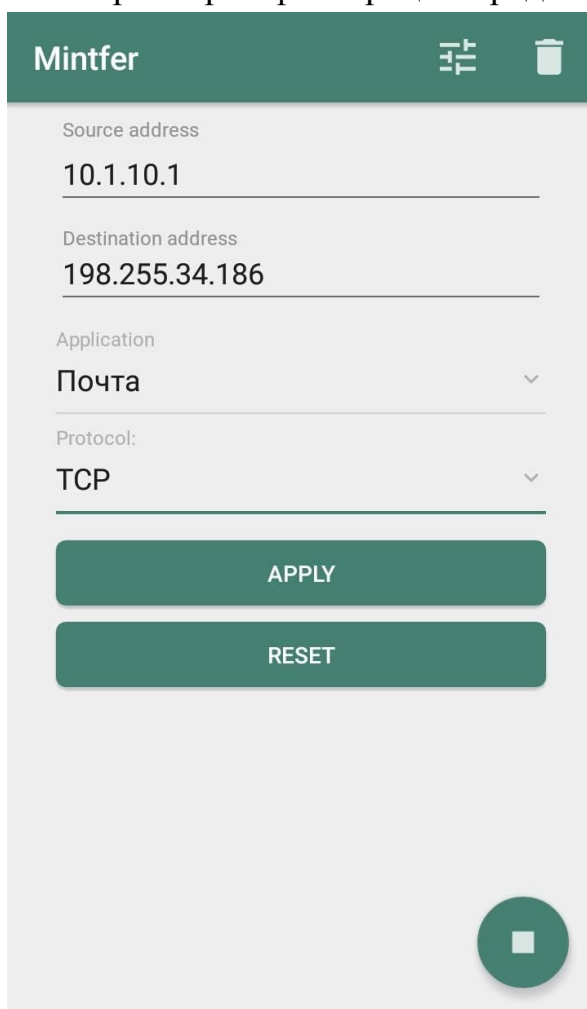
The image shows a mobile application interface for 'Mintfer'. At the top is a dark green header with the app name 'Mintfer' on the left, and two icons (a list icon and a trash icon) on the right. Below the header, there are four input fields: 'Source address' with the value '10.1.10.1', 'Destination address' with the value '198.255.34.186', 'Application' with a dropdown menu showing 'Почта', and 'Protocol' with a dropdown menu showing 'TCP'. Below these fields are two large green buttons: 'APPLY' and 'RESET'. At the bottom right of the screen is a circular green button with a white square icon inside.

Рисунок 22 – Пример установленных параметров фильтрации

3 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Нервная система является одним из основных механизмов поддержания человеческого организма в равновесии. Она регулирует деятельность и взаимодействие всех органов и систем человека, в то время, как заболевания этой системы оказывают серьезное влияние на организм в целом, могут привести к инвалидности или даже к летальному исходу (например, сосудисто-мозговая недостаточность или инсульт). Причинами заболеваний нервной системы являются различные факторы – от несбалансированного питания до инфекционных заболеваний, и различные психофизиологические факторы не являются исключением.

Психофизиологические факторы, которые негативно влияют на работоспособность и эффективность человека, можно подразделить на две группы:

- нервно-психические перегрузки;
- физические перегрузки.

Влияние на нервную систему специалиста по защите информации оказывает как первая группа, так и вторая группа. Нервно-психические перегрузки негативно сказываются на центральной нервной системе (т.е. на функциях головного мозга), в то время, как физические перегрузки оказывают влияние на периферическую нервную систему. Рассмотрим подробнее каждую из них.

К нервно-психическим перегрузкам можно отнести:

- а) перегрузки из-за монотонности трудового процесса – постоянное повторение однообразных действий приводит к тому, что человек теряет

интерес к работе, вследствие чего снижается его концентрация внимания и производительность;

б) умственное и эмоциональное перенапряжение, в частности, из-за большого объема работы, постоянных переработок сверх нормы рабочего времени, сжатых сроков, может привести к нервному истощению и эмоциональному «перегоранию» работника;

в) эмоциональные перегрузки из-за негативных взаимоотношений в трудовом коллективе отрицательно сказываются на ментальном здоровье человека, что также может привести к стрессам и нервному истощению;

г) перенапряжение органов чувств (в рассматриваемом случае, органов зрения) может привести к постоянным головным болям и тошноте.

Постоянные психоэмоциональные стрессы наносят вред не только нервной системе человека, но и его физическому здоровью.

Физические перегрузки бывают:

- статические (например, длительное пребывание человека в неудобной позе или статическое напряжение отдельных групп мышц);
- динамические (например, частые наклоны туловища, наклоны и повороты головы и т.д);
- гиподинамические (например, перегрузки опорно-двигательного аппарата или системы дыхания).

Физические перегрузки часто приводят к перенапряжению нервной системы человека, заболеваниям конечностей, шеи, поясницы и т. д.

Для того, чтобы свести к минимуму негативные последствия указанных психофизиологических факторов, рекомендуется использовать следующие методы и средства физической культуры:

- гигиенические факторы;
- физические упражнения;

– природные ресурсы.

К гигиеническим факторам относится не только соблюдение личной и общественной гигиены, но и следование принципам здорового питания, соблюдение режима дня и сна, полноценный отдых и т. д.

Физические упражнения являются одним из основных средств физической культуры. Регулярные физические нагрузки активизируют приток кислорода в ткани мозга, позволяют снизить умственное и физическое напряжение, избавиться от неврозов, а также приносят положительные эмоции и повышают активность головного мозга. В настоящее время во многих компаниях присутствуют регулярные корпоративные игры в футбол или баскетбол после рабочего дня, а в некоторых – занятия йогой в обеденный перерыв.

К природным ресурсам относятся солнце, воздух и вода. Они используются как дополнительные факторы, которые создают благоприятные условия для физических нагрузок, а также как самостоятельные средства оздоровления организма (например, закаливание, солнечные ванны и т. д.).

Заключение

В данной выпускной квалификационной работе рассмотрена проблема мониторинга сетевой активности приложений на устройствах под управлением ОС Android.

В процессе выполнения данной выпускной квалификационной работы выполнены обзор, сравнение и выявление недостатков у существующих аналогов разрабатываемого программного обеспечения, анализ существующих способов программной реализации поставленной задачи и изучение используемых протоколов передачи данных, а также рассмотрена архитектура операционной системы Android и структура Android-приложения.

В ходе выполнения данной выпускной квалификационной работы выполнено проектирование архитектуры программного обеспечения, а также разработаны алгоритмы работы приложения.

В итоге получена программа, реализующая перехват сетевого трафика на устройствах под управлением ОС Android версии 4.0 и выше, которая позволяет проводить мониторинг сетевой активности в режиме реального времени, просматривать перехваченный трафик и детальную информацию о каждом пакете, а также выполнять фильтрацию трафика по заданным пользователем параметрам – IP-адресам отправителя и получателя, приложению, которое занимается обработкой пакета, а также по типу используемого приложения. В завершении выпускной квалификационной работы выполнена разработка руководства пользователя, а также рассмотрены вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации и средства физической культуры, снижающие их воздействие

Таким образом, задание на выпускную квалификационную работу выполнено в полном объеме.

Список используемых источников

- 1 Тайная жизнь мобильных приложений [Электронный ресурс]. – Режим доступа: <https://www.kaspersky.ru/blog/secret-life-of-apps/14313>, свободный. – Загл. с экрана (дата обращения: 09.01.2018).
- 2 Systems analysis and design. Alan Dennis, Barbara Haley Wixom, Roberta M. Roth [Электронный ресурс]. – Режим доступа: http://www.saigontech.edu.vn/faculty/huynq/SAD/Systems_Analysis_Design_UML_5th%20ed.pdf, свободный. – Загл. с экрана (дата обращения: 03.12.2017).
- 3 В. Олифер, Н. Олифер. Компьютерные сети. Изд. 4-е. М., 2010. 943 с. (дата обращения: 03.12.2017).
- 4 RFCs [Электронный ресурс]. – Режим доступа: <https://www.ietf.org/standards/rfcs>, свободный. – Загл. с экрана (дата обращения: 03.12.2017).
- 5 А. Бхаргава. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. Санкт-Петербург: изд. Питер, 2017. 288 с.
- 6 Э.Фримен, Э.Фримен. Паттерны проектирования. Санкт-Петербург: изд. Питер, 2013. 656 с.
- 7 К. Сьерра, Б. Бертс. Изучаем Java. М.: изд. Эксмо, 2017 г. 720 с.
- 8 Б. Эккель. Философия Java. Санкт-Петербург: изд. Питер, 2009 г. 640 с.
- 9 Android Developers [Электронный ресурс]. – Режим доступа: <https://developer.android.com/index.html>, свободный. – Загл. с экрана (дата обращения: 24.12.2017).
- 10 Meet Android Studio [Электронный ресурс]. – Режим доступа: <https://developer.android.com/studio/intro/index.html>, свободный. – Загл. с экрана (дата обращения: 24.12.2017).

- 11 Создание архитектуры программы или как проектировать табуретку [Электронный ресурс]. – Режим доступа: <https://habrahabr.ru/post/276593/>, свободный. – Загл. с экрана (дата обращения: 18.12.2017).
- 12 RFC 791 – Internet Protocol [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc791>, свободный. – Загл. с экрана (дата обращения: 11.12.2017).
- 13 RFC 793 – Transmission Control Protocol [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc793>, свободный. – Загл. с экрана (дата обращения: 11.12.2017).
- 14 RFC 768 – User Datagram Protocol [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc768>, свободный. – Загл. с экрана (дата обращения: 11.12.2017).
- 15 Platform Architecture [Электронный ресурс]. – Режим доступа: <https://developer.android.com/guide/platform/index.html>, свободный. – Загл. с экрана (дата обращения: 19.12.2017).
- 16 Activity [Электронный ресурс]. – Режим доступа: <https://developer.android.com/reference/android/app/Activity.html>, свободный. – Загл. с экрана (дата обращения: 19.12.2017).
- 17 Fragments [Электронный ресурс]. – Режим доступа: <https://developer.android.com/guide/components/fragments.html>, свободный. – Загл. с экрана (дата обращения: 19.12.2017).
- 18 Model-View-Presenter: Android guidelines. Francesco Cervone [Электронный ресурс]. – Режим доступа: <https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf>, свободный. – Загл. с экрана (дата обращения: 21.12.2017).

- 19 Dagger. A fast dependency injector for Android [Электронный ресурс]. – Режим доступа: <https://google.github.io/dagger/>, свободный. – Загл. с экрана (дата обращения: 21.12.2017).
- 20 Паттерн Издатель-Подписчик [Электронный ресурс]. – Режим доступа: <https://reyvart.com/tech/publisher-subscriber-pattern/>, свободный. – Загл. с экрана (дата обращения: 21.12.2017).
- 21 GitHub – greenrobot/EventBus [Электронный ресурс]. – Режим доступа: <https://github.com/greenrobot/EventBus>, свободный. – Загл. с экрана (дата обращения: 21.12.2017).
- 22 VpnService [Электронный ресурс]. – Режим доступа: <https://developer.android.com/reference/android/net/VpnService.html>, свободный. – Загл. с экрана (дата обращения: 09.01.2018).
- 23 VpnService.Builder[Электронный ресурс]. – Режим доступа: <https://developer.android.com/reference/android/net/VpnService.Builder.html>, свободный. – Загл. с экрана (дата обращения: 09.01.2018).
- 24 Разработка VPN-клиента под Android (Часть 1) [Электронный ресурс]. – Режим доступа: <https://habrahabr.ru/company/mobisoft/blog/231827/>, свободный. – Загл. с экрана (дата обращения: 17.12.2017).
- 25 RFC 790 – Assigned Numbers [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc790>, свободный. – Загл. с экрана (дата обращения: 11.12.2017).

Приложение А
(обязательное)
Код программы

MainActivity.java

```
public class MainActivity extends AppCompatActivity {  
    private FloatingActionButton fab;  
    private PacketFilter packetFilter;  
    private boolean isConnected = false;  
    @NonNull  
    private AppComponent applicationComponent;  
  
    @NonNull  
    public static MainActivity get(@NonNull Context context) {  
        return (MainActivity) context;  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        ArrayList<TCPInfo> tcpInfos = new ArrayList<>();  
        Utils.getInstalledAppsInfo(getPackageManager());  
    }  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
        setSupportActionBar(toolbar);  
    }  
}
```

```

fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (!isConnected) {
            startVPN();
        } else {
            isConnected = false;
            MintferVpnService.stop("stopped", getApplicationContext());
            fab.setImageResource(R.drawable.ic_start);
        }
    }
});
applicationComponent = prepareAppComponent().build();

if (savedInstanceState == null) {

    FragmentManager manager = getSupportFragmentManager();
    manager.beginTransaction()
        .add(R.id.container, new FragmentPackets())
        .commit();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
}

```

```
    return true;
}
```

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    int id = item.getItemId();
```

```
    switch (id) {
```

```
        case R.id.action_filters:
```

```
            FragmentManager manager = getSupportFragmentManager();
```

```
            manager.beginTransaction()
```

```
                .replace(R.id.container, new FragmentFilters())
```

```
                .addToBackStack(null)
```

```
                .commit();
```

```
            return true;
```

```
        case R.id.action_clear:
```

```
            EventBus.getDefault().post(new ClearPacketListEvent());
```

```
    }
```

```
    return super.onOptionsItemSelected(item);
```

```
}
```

```
@Override
```

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
    super.onActivityResult(requestCode, resultCode, data);
```

```
    if (requestCode == Constants.VPN_REQUEST_CODE && resultCode ==  
    RESULT_OK) {
```

```

        MintferVpnService.start("prepared", this);
        isConnected = true;
        fab.setImageResource(R.drawable.ic_stop);
    }
}

private void startVPN() {
    Intent vpnIntent = VpnService.prepare(this);
    if (vpnIntent != null)
        startActivityForResult(vpnIntent, Constants.VPN_REQUEST_CODE);
    else
        onActivityResult(Constants.VPN_REQUEST_CODE, RESULT_OK, null);
}

@NonNull
private DaggerAppComponent.Builder prepareAppComponent() {
    return DaggerAppComponent.builder()
        .appModule(new AppModule(this));
}

@NonNull
public AppComponent applicationComponent() {
    return applicationComponent;
}

public PacketFilter getPacketFilter() {
    return packetFilter;
}

```

```

    }

    public void setPacketFilter(PacketFilter packetFilter) {
        this.packetFilter = packetFilter;
    }
}

```

MintferVpnService.java

```

public class MintferVpnService extends VpnService {
    private static final String TAG = MintferVpnService.class.getSimpleName();
    private long jni_context = 0;
    private Thread tunnelThread = null;
    private MintferVpnService.Builder last_builder = null;
    private ParcelFileDescriptor vpn = null;
    private boolean temporarilyStopped = false;
    private Map<String, Boolean> mapHostsBlocked = new HashMap<>();
    private Map<Integer, Forward> mapForward = new HashMap<>();
    private ReentrantReadWriteLock lock = new ReentrantReadWriteLock(true);
    private volatile Looper commandLooper;
    private volatile Looper packetLooper;
    private volatile CommandHandler commandHandler;
    private volatile PacketHandler packetHandler;
    public static final String EXTRA_COMMAND = "Command";
    private static final String EXTRA_REASON = "Reason";
    public static final String EXTRA_INTERACTIVE = "Interactive";
    public static final String EXTRA_TEMPORARY = "Temporary";
    private static final int MSG_PACKET = 4;
}

```

```

public enum Command {run, start, reload, stop, set}

private static volatile PowerManager.WakeLock wlInstance = null;

private native long jni_init(int sdk);

private native void jni_start(long context, int loglevel);

private native void jni_run(long context, int tun, boolean fwd53, int rcode);

private native void jni_stop(long context);

private native void jni_clear(long context);

private native int jni_get_mtu();

private native int[] jni_get_stats(long context);

private static native void jni_pcap(String name, int record_size, int file_size);

    private native void jni_socks5(String addr, int port, String username, String
password);

private native void jni_done(long context);


synchronized private static PowerManager.WakeLock getLock(Context context) {
    if (wlInstance == null) {

                                PowerManager  pm    = (PowerManager)
context.getSystemService(Context.POWER_SERVICE);

        wlInstance = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
context.getString(R.string.app_name) + " wakelock");

        wlInstance.setReferenceCounted(true);
    }

    return wlInstance;
}


synchronized private static void releaseLock(Context context) {
    if (wlInstance != null) {

```

```

        while (wlInstance.isHeld())
            wlInstance.release();
        wlInstance = null;
    }
}

```

```

private final class CommandHandler extends Handler {

```

```

    public int queue = 0;

```

```

    public CommandHandler(Looper looper) {
        super(looper);
    }

```

```

    public void queue(Intent intent) {
        synchronized (this) {
            queue++;
        }
    }

```

```

        Command cmd = (Command)

```

```

        intent.getSerializableExtra(EXTRA_COMMAND);

```

```

        Message msg = commandHandler.obtainMessage();

```

```

        msg.obj = intent;

```

```

        msg.what = cmd.ordinal();

```

```

        commandHandler.sendMessage(msg);
    }

```

```

@Override

```

```

    public void handleMessage(Message msg) {

```

```

try {
    synchronized (MintferVpnService.this) {
        handleIntent((Intent) msg.obj);
    }
} catch (Throwable ex) {
    Log.e(TAG, ex.toString() + "\n" + Log.getStackTraceString(ex));
} finally {
    synchronized (this) {
        queue--;
    }
}
}
}

```

```

private void handleIntent(Intent intent) {
    final SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(MintferVpnService.this);
    Command cmd = (Command)
intent.getSerializableExtra(EXTRA_COMMAND);

```

```

try {
    switch (cmd) {
        case start:
            start();
            break;
        case stop:
            stop(temporarilyStopped);
            break;

```

```

        default:
            break;
    }
} catch (Throwable e) {
    e.printStackTrace()
}
}

```

```

private void start() {
    if (vpn == null) {
        last_builder = getBuilder();
        vpn = startVPN(last_builder);

        startNative(vpn);
    }
}

```

```

private void stop(boolean temporary) {
    if (vpn != null) {
        stopNative(vpn, true);
        stopVPN(vpn);
        vpn = null;
    }
}

```

```

private class StartFailedException extends IllegalStateException {
    public StartFailedException(String msg) {

```

```

        super(msg);
    }
}
}

```

```

@TargetApi(Build.VERSION_CODES.LOLLIPOP)

```

```

private ParcelFileDescriptor startVPN(Builder builder) throws SecurityException {
    try {
        return builder.establish();
    } catch (SecurityException ex) {
        throw ex;
    } catch (Throwable ex) {
        return null;
    }
}
}

```

```

private Builder getBuilder() {

```

```

        SharedPreferences prefs =

```

```

        PreferenceManager.getDefaultSharedPreferences(this);

```

```

// Build VPN service

```

```

Builder builder = new Builder();

```

```

builder.setSession(getString(R.string.app_name));

```

```

// VPN address

```

```

builder.addAddress(Constants.IPV4_ADDRESS, 32);

```

```

builder.addAddress(Constants.IPV6_ADDRESS, 128);

```

```

// Routers
builder.addRoute(Constants.IPV4_ROUTE, 0);
builder.addRoute(Constants.IPV6_ROUTE, 3); // unicast

// MTU
int mtu = jni_get_mtu();
builder.setMtu(mtu);

try {
    builder.addDisallowedApplication(getPackageName());
} catch (Exception e) {
    e.printStackTrace();
}

return builder;
}

private void startNative(final ParcelFileDescriptor vpn) {

    jni_socks5("", 0, "", "");

    if (tunnelThread == null) {
        jni_start(jni_context, Log.WARN);

        tunnelThread = new Thread(new Runnable() {
            @Override

```

```

    public void run() {
        jni_run(jni_context, vpn.getFd(), mapForward.containsKey(53), 3);
        tunnelThread = null;
    }
});
tunnelThread.start();
}
}

private void stopNative(ParcelFileDescriptor vpn, boolean clear) {
    if (tunnelThread != null) {
        jni_stop(jni_context);
        Thread thread = tunnelThread;
        while (thread != null)
            try {
                thread.join();
                break;
            } catch (Exception e) {
                e.printStackTrace();
            }
        tunnelThread = null;

        if (clear)
            jni_clear(jni_context);
    }
}
}

```

```
private void stopVPN(ParcelFileDescriptor pfd) {
    Log.i(TAG, "Stopping");
    try {
        pfd.close();
    } catch (IOException ex) {
        Log.e(TAG, ex.toString() + "\n" + Log.getStackTraceString(ex));
    }
}
```

```
private void logPacket(Packet packet) {
    Message msg = packetHandler.obtainMessage();
    msg.obj = packet;
    msg.what = MSG_PACKET;
    packetHandler.sendMessage(msg);
}
```

@Override

```
public void onCreate() {
    Log.i(TAG, "Create version=" + Util.getSelfVersionName(this) + "/" +
Util.getSelfVersionCode(this));
```

```
        SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(this);
```

```
jni_context = jni_init(Build.VERSION.SDK_INT);
boolean pcap = prefs.getBoolean("pcap", false);
setPcap(pcap, this);
```

```
super.onCreate();
```

```
        HandlerThread      commandThread      =      new  
HandlerThread(getString(R.string.app_name)      +      "      command",  
Process.THREAD_PRIORITY_FOREGROUND);  
        HandlerThread      packetThread      =      new  
HandlerThread(getString(R.string.app_name)      +      "      log",  
Process.THREAD_PRIORITY_BACKGROUND);
```

```
        commandThread.start();
```

```
        packetThread.start();
```

```
        commandLooper = commandThread.getLooper();
```

```
        packetLooper = packetThread.getLooper();
```

```
        commandHandler = new CommandHandler(commandLooper);
```

```
        packetHandler = new PacketHandler(packetLooper);
```

```
    }
```

```
@Override
```

```
public int onStartCommand(Intent intent, int flags, int startId) {
```

```
    getLock(this).acquire();
```

```
    SharedPreferences      prefs      =
```

```
PreferenceManager.getDefaultSharedPreferences(this);
```

```
    boolean enabled = prefs.getBoolean("enabled", false);
```

```

        Command cmd = (Command)
intent.getSerializableExtra(EXTRA_COMMAND);
        if (cmd == null)
            intent.putExtra(EXTRA_COMMAND, enabled ? Command.start :
Command.stop);
        commandHandler.queue(intent);

        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        synchronized (this) {
            Log.i(TAG, "Destroy");
            commandLooper.quit();
            packetLooper.quit();

            for (Command command : Command.values())
                commandHandler.removeMessages(command.ordinal());
            releaseLock(this);

            try {
                if (vpn != null) {
                    stopNative(vpn, true);
                    stopVPN(vpn);
                    vpn = null;
                }
            }

```

```

        } catch (Exception e) {
            e.printStackTrace();
        }

        jni_done(jni_context);
    }

    super.onDestroy();
}

private void showErrorNotification(String message) {
    Toast.makeText(getApplicationContext(), message,
Toast.LENGTH_SHORT).show();
}

private class Builder extends VpnService.Builder {
    private NetworkInfo networkInfo;
    private int mtu;
    private List<String> listAddress = new ArrayList<>();
    private List<String> listRoute = new ArrayList<>();
    private List<InetAddress> listDns = new ArrayList<>();
    private List<String> listDisallowed = new ArrayList<>();

    private Builder() {
        super();
    }
}

```

@Override

```
public VpnService.Builder setMtu(int mtu) {  
    this.mtu = mtu;  
    super.setMtu(mtu);  
    return this;  
}
```

@Override

```
public Builder addAddress(String address, int prefixLength) {  
    listAddress.add(address + "/" + prefixLength);  
    super.addAddress(address, prefixLength);  
    return this;  
}
```

@Override

```
public Builder addRoute(String address, int prefixLength) {  
    listRoute.add(address + "/" + prefixLength);  
    super.addRoute(address, prefixLength);  
    return this;  
}
```

@Override

```
public Builder addDnsServer(InetAddress address) {  
    listDns.add(address);  
    super.addDnsServer(address);  
    return this;  
}
```

@Override

```
    public Builder addDisallowedApplication(String packageName) throws  
PackageManager.NameNotFoundException {  
        listDisallowed.add(packageName);  
        super.addDisallowedApplication(packageName);  
        return this;  
    }  
}
```

```
private class IPRule {  
    private boolean block;  
    private long expires;  
  
    public IPRule(boolean block, long expires) {  
        this.block = block;  
        this.expires = expires;  
    }  
  
    public boolean isBlocked() {  
        return this.block;  
    }  
  
    public boolean isExpired() {  
        return System.currentTimeMillis() > this.expires;  
    }  
}
```

```

public void updateExpires(long expires) {
    this.expires = Math.max(this.expires, expires);
}

```

```

@Override

```

```

public boolean equals(Object obj) {
    IPRule other = (IPRule) obj;
    return (this.block == other.block && this.expires == other.expires);
}
}

```

```

public static void start(String reason, Context context) {
    Intent intent = new Intent(context, MintferVpnService.class);
    intent.putExtra(EXTRA_COMMAND, Command.start);
    intent.putExtra(EXTRA_REASON, reason);
    context.startService(intent);
}

```

```

public static void stop(String reason, Context context) {
    Intent intent = new Intent(context, MintferVpnService.class);
    intent.putExtra(EXTRA_COMMAND, Command.stop);
    intent.putExtra(EXTRA_REASON, reason);
    context.startService(intent);
}
}

```

FragmentPackets.java

```

public class FragmentPackets extends Fragment implements IPacketsView {

```

@Inject

IPacketsPresenter presenter;

private RecyclerView recyclerViewPackets;

private TextView empty;

private PacketsListAdapter adapter;

List<Packet> packetsList = new ArrayList<>();

```
public FragmentPackets() {  
}
```

@Override

```
public void onStart() {  
    super.onStart();  
    presenter.registerEvent();  
}
```

@Override

```
public void onDestroy() {  
    presenter.unbindView();  
    super.onDestroy();  
}
```

@Override

```
public void onCreate(@Nullable Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
MainActivity.get(getActivity()).applicationComponent()
    .plus(new PacketsModule()).inject(this);
}

```

@Override

```

public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {

```

```

    View view = inflater.inflate(R.layout.fragment_packets, container, false);
    empty = (TextView) view.findViewById(R.id.empty_view);

```

```

    recyclerViewPackets = (RecyclerView) view.findViewById(R.id.list_packets);
    recyclerViewPackets.setHasFixedSize(true);

```

```

        RecyclerView.LayoutManager layoutManager = new
LinearLayoutManager(getActivity());

```

```

    recyclerViewPackets.setLayoutManager(layoutManager);

```

```

    DividerItemDecoration dividerItemDecoration

```

```

        = new DividerItemDecoration(recyclerViewPackets.getContext(),
DividerItemDecoration.VERTICAL);

```

```

    recyclerViewPackets.addItemDecoration(dividerItemDecoration);

```

```

    isEmpty();

```

```

    adapter = new PacketsListAdapter(getContext(), packetsList);

```

```

    recyclerViewPackets.setAdapter(adapter);

```

```

    presenter.bindView(this);

```

```
    return view;
}
```

```
@Override
public void onDestroyView() {
    presenter.unbindView();
    super.onDestroyView();
}
```

```
@Override
public void onResume() {
    super.onResume();
    PacketFilter packetFilter = ((MainActivity) getActivity()).getPacketFilter();
    if (packetFilter != null)
        presenter.setPacketsFilter(packetFilter);
}
```

```
@Override
public void updatePacketsListView(List<Packet> packets) {
    this.packetsList = packets;
    adapter.setListPackets(packetsList);
    adapter.notifyDataSetChanged();
    isEmpty();
}
```

```
@Override
public void showPacketDetails(Packet packet) {
```

```
// todo: maybe replace
```

```
Bundle bundle = new Bundle();
```

```
bundle.putSerializable("packet", packet);
```

```
FragmentDetails fragmentDetails = new FragmentDetails();
```

```
fragmentDetails.setArguments(bundle);
```

```
FragmentManager manager = getFragmentManager();
```

```
manager.beginTransaction()
```

```
    .replace(R.id.container, fragmentDetails)
```

```
    .addToBackStack(null)
```

```
    .commit();
```

```
}
```

```
private void isEmpty() {
```

```
    if (packetsList.isEmpty()) {
```

```
        recyclerViewPackets.setVisibility(View.GONE);
```

```
        empty.setVisibility(View.VISIBLE);
```

```
    } else {
```

```
        empty.setVisibility(View.GONE);
```

```
        recyclerViewPackets.setVisibility(View.VISIBLE);
```

```
    }
```

```
}
```

```
}
```

IpacketsView.java

```

public interface IPacketsView {
    void updatePacketsListView(List<Packet> packetList);
    void showPacketDetails(Packet packet);
}

```

PacketsPresenter.java

```

public class PacketsPresenter implements IPacketsPresenter {

    private IPacketsView iPacketsView;

    private List<Packet> listPackets = new ArrayList<>();
    private PacketFilter packetsFilter;
    private Map<Integer, ApplicationInfo> installedApps = new HashMap<>();
    private ArrayList<TCPInfo> tcpInfos = new ArrayList<>();

    public PacketsPresenter() {
    }

    @Override
    public void bindView(IPacketsView iPacketsView) {
        this.iPacketsView = iPacketsView;
    }

    @Override
    public void unbindView() {
        iPacketsView = null;
        EventBus.getDefault().unregister(this);
    }
}

```

```
}
```

```
@Override
```

```
public void registerEvent() {  
    if (!EventBus.getDefault().isRegistered(this))  
        EventBus.getDefault().register(this);  
}
```

```
@Override
```

```
public void updatePacketsList(Packet packet) {  
  
    packet.setApplicationInfo(installedApps.get(packet.uid));  
    if (packet.getApplicationInfo() != null)  
        listPackets.add(packet);  
    if (packetsFilter != null && packetsFilter.isFiltering()) {  
        filterPackets();  
    } else {  
        iPacketsView.updatePacketsListView(listPackets);  
    }  
}
```

```
@Subscribe
```

```
public void onUpdatePacketsEvent(UpdatePacketsEvent event) {  
    updatePacketsList(event.packet);  
}
```

```
@Subscribe
```

```
public void onDetailsEvent(DetailsEvent event) {  
    iPacketsView.showPacketDetails(event.packet);  
}
```

@Subscribe

```
public void onAppsInfo(AppInfosEvent event) {  
    this.installedApps = event.installedApps;  
}
```

@Subscribe

```
public void onClearPacketsListEvent(ClearPacketListEvent event) {  
    listPackets.clear();  
    iPacketsView.updatePacketsListView(listPackets);  
}
```

@Override

```
public void setPacketsFilter(PacketFilter packetsFilter) {  
    this.packetsFilter = packetsFilter;  
  
    if (packetsFilter.isFiltering()) {  
        filterPackets();  
    } else {  
        iPacketsView.updatePacketsListView(listPackets);  
    }  
}
```

```

private void filterPackets() {
    List<Packet> listFilteredPackets = FluentIterable.from(listPackets)
        .filter(predicateSource)
        .filter(predicateDestination)
        .filter(predicateProtocol).toList();

    iPacketsView.updatePacketsListView(listFilteredPackets);
}

private Predicate<Packet> predicateSource = new Predicate<Packet>() {
    @Override
    public boolean apply(Packet packet) {
        String filterSourceAddress = packetsFilter.getSourceAddress();
        String sourceAddress = packet.saddr;

        return (filterSourceAddress == null || filterSourceAddress.isEmpty())
            || sourceAddress.equals(filterSourceAddress);
    }
};

private Predicate<Packet> predicateDestination = new Predicate<Packet>() {
    @Override
    public boolean apply(Packet packet) {
        String filterDestinationAddress = packetsFilter.getDestinationAddress();
        String destinationAddress = packet.daddr;

        return (filterDestinationAddress == null || filterDestinationAddress.isEmpty())

```

```

        || destinationAddress.equals(filterDestinationAddress);
    }
};

private Predicate<Packet> predicateProtocol = new Predicate<Packet>() {
    @Override
    public boolean apply(Packet packet) {
        Constants.TransportProtocol packetProtocol
            = Constants.TransportProtocol.numberToEnum(packet.protocol);
        Constants.TransportProtocol filterProtocol = packetsFilter.getProtocol();
        return (filterProtocol == null)
            || packetProtocol.equals(filterProtocol);
    }
};

```

```

private Predicate<Packet> predicateApp = new Predicate<Packet>() {
    @Override
    public boolean apply(@Nullable Packet packet) {
        ApplicationInfo app = null;

        try {
            app = packet.getApplicationInfo();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return app == null;
    }
};

```

```
    }  
};  
  
}
```

IpacketsPresenter.java

```
public interface IPacketsPresenter {  
  
    void bindView(IPacketsView iPacketsView);  
    void unbindView();  
  
    void updatePacketsList(Packet packet);  
    void registerEvent();  
    void setPacketsFilter(PacketFilter packetFilter);  
}
```

FragmentFilters.java

```
public class FragmentFilters extends Fragment  
    implements IFiltersView {  
  
    @Inject  
    IFiltersPresenter presenter;  
  
    public FragmentFilters() {  
  
    }
```

@Override

```
public void onStart() {  
    super.onStart();  
}
```

@Override

```
public void onStop() {  
    super.onStop();  
    presenter.unbindView();  
}
```

@Override

```
public void onCreate(@Nullable Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    MainActivity.get(getActivity()).applicationComponent()  
        .plus(new FiltersModule()).inject(this);  
}
```

@Nullable

@Override

```
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup  
container, @Nullable Bundle savedInstanceState) {  
    View view = inflater.inflate(R.layout.fragment_filters, container, false);  
  
    initView(view);  
  
    presenter.bindView(this);
```

```
    return view;
}
```

```
private void initView(View view) {
```

```
        EditText editTextSource = (EditText)
view.findViewById(R.id.edit_filter_source);
        EditText editTextDestination = (EditText)
view.findViewById(R.id.edit_filter_destination);
```

```
        ArrayAdapter<String> adapterApps = new ArrayAdapter<String>(getContext(),
        android.R.layout.simple_dropdown_item_1line,
getInstalledAppsInfo(getContext().getPackageManager()));
```

```
        MaterialBetterSpinner spinnerApps = (MaterialBetterSpinner)
        view.findViewById(R.id.spinner_apps);
        spinnerApps.setAdapter(adapterApps);
```

```
        ArrayAdapter<String> adapterProtocols = new
ArrayAdapter<String>(getContext(),
        android.R.layout.simple_dropdown_item_1line,
getResources().getStringArray(R.array.array_protocols));
```

```
        MaterialBetterSpinner spinnerProtocols = (MaterialBetterSpinner)
        view.findViewById(R.id.spinner_protocol);
        spinnerProtocols.setAdapter(adapterProtocols);
```

```
        Button buttonSetFilter = (Button) view.findViewById(R.id.button_set_filter);
        buttonSetFilter.setOnClickListener(new View.OnClickListener() {
```

```

@Override
public void onClick(View view) {
    presenter.setFilter(editTextSource.getText().toString(),
        editTextDestination.getText().toString(),
        getUserApplication(getContext().getPackageManager(),
spinnerApps.getText().toString()),

Constants.TransportProtocol.valueOf(spinnerProtocols.getText().toString()));
    }
});

        Button    buttonResetFilter    =    (Button)
view.findViewById(R.id.button_reset_filter);
    buttonResetFilter.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            presenter.setFilter("",
                "",
                null,
                null);
        }
    });
}

```

```

@Override
public void setPacketFilter(PacketFilter packetFilter) {
    ((MainActivity) getActivity()).setPacketFilter(packetFilter);
}

```

```
}
```

```
private ArrayList<String> getInstalledAppsInfo(PackageManager manager) {  
    ArrayList<String> listApplications = new ArrayList<>();  
    for (ApplicationInfo ainfo : manager.getInstalledApplications(0))  
        listApplications.add(ainfo.loadLabel(manager).toString());  
    return listApplications;  
}
```

```
private ApplicationInfo getUserApplication(PackageManager manager, String  
name) {  
    for (ApplicationInfo ainfo : manager.getInstalledApplications(0))  
        if (ainfo.loadLabel(manager).toString().equals(name))  
            return ainfo;  
  
    return null;  
}  
}
```

IFiltersView.java

```
public interface IFiltersView {  
    void setPacketFilter(PacketFilter packetFilter);  
}
```

FiltersPresenter.java

```
public class FiltersPresenter implements IFiltersPresenter {
```

```
private IFiltersView iFiltersView;
```

```
@Override
```

```
public void bindView(IFiltersView view) {  
    this.iFiltersView = view;  
}
```

```
@Override
```

```
public void unbindView() {  
    this.iFiltersView = null;  
}
```

```
@Override
```

```
public void setFilter(String sourceAddress,  
                      String destinationAddress,  
                      ApplicationInfo applicationInfo,  
                      Constants.TransportProtocol protocol) {  
    boolean isFiltering = !((sourceAddress == null || sourceAddress.isEmpty())  
        && (destinationAddress == null || destinationAddress.isEmpty())  
        && protocol == null  
        && applicationInfo == null);
```

```
    PacketFilter packetFilter = new PacketFilter.Builder()  
        .sourceAddress(sourceAddress)  
        .destinationAddress(destinationAddress)  
        .protocol(protocol)
```

```
.applicationInfo(applicationInfo)
.isFiltering(isFiltering)
.build();
```

```
    iFiltersView.setPacketFilter(packetFilter);
}
```

```
}
```

IfiltersPresenter.java

```
public interface IFiltersPresenter {
    void bindView(IFiltersView view);
    void unbindView();

    void setFilter(String sourceAddress,
                   String destinationAddress,
                   ApplicationInfo applicationInfo,
                   Constants.TransportProtocol protocol);
}
```

```
public class FragmentDetails extends Fragment implements IDetailsView {
```

```
    @Inject
    IDetailsPresenter presenter;
```

```
    private Packet packet;
```

```
    public FragmentDetails() {
```

```
}
```

```
@Override
```

```
public void onStart() {  
    super.onStart();  
}
```

```
@Override
```

```
public void onStop() {  
    presenter.unbindView();  
    super.onStop();  
}
```

```
@Override
```

```
public void onCreate(@Nullable Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    MainActivity.get(getActivity()).applicationComponent()  
        .plus(new DetailsModule()).inject(this);  
    Bundle bundle = getArguments();  
  
    if (bundle != null) {  
        this.packet = (Packet) bundle.getSerializable("packet");  
    }  
}
```

```
@Nullable
```

```
@Override
```

```

    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_packet_details, container, false);
        initView(view);

        presenter.bindView(this);
        return view;
    }

    private void initView(View view) {

        if (packet != null) {
            showAppInfo(view);
            showPacketView(view);
        }
    }

    private void showAppInfo(View view) {
        ApplicationInfo appInfo = packet.getApplicationInfo();

        if (appInfo != null) {
            ImageView appIcon = (ImageView)
view.findViewById(R.id.detail_app_icon);
            TextView appName = (TextView)
view.findViewById(R.id.detail_app_name);
            TextView appPackage = (TextView)
view.findViewById(R.id.detail_app_package);

```

```

appIcon.setImageDrawable(appInfo.loadIcon(getContext().getPackageManager()));
    appName.setText(appInfo.loadLabel(getContext().getPackageManager()));
    appPackage.setText(appInfo.packageName);

    setValues((View) view.findViewById(R.id.item_app_uid),
        R.string.title_uid,
        String.valueOf(appInfo.uid));
}

```

```

setValues((View) view.findViewById(R.id.item_from),
    R.string.title_from,
    packet.getSourceDomainName());

```

```

setValues((View) view.findViewById(R.id.item_to),
    R.string.title_to,
    packet.getDestinationDomainName());

}

```

```

private void showPacketView(View view) {

```

```

    String sourceAddress = String.valueOf(packet.saddr)
        + ":" + String.valueOf(packet.sport);
    String destinationAddress = String.valueOf(packet.daddr)

```

```

        + ":" + String.valueOf(packet.dport);

setValues((View) view.findViewById(R.id.item_source),
        R.string.title_source,
        sourceAddress);

setValues((View) view.findViewById(R.id.item_destination),
        R.string.title_destination,
        destinationAddress);

setValues((View) view.findViewById(R.id.item_protocol),
        R.string.title_protocol,
        Constants.TransportProtocol.numberToEnum(packet.protocol).name());

setValues((View) view.findViewById(R.id.item_version),
        R.string.title_version,
        Constants.Version.numberToEnum(packet.version).name());

setValues((View) view.findViewById(R.id.item_flags),
        R.string.title_flags,
        Constants.Flags.stringToEnum(packet.flags).name());

setValues((View) view.findViewById(R.id.item_date),
        R.string.title_data,
        String.valueOf(packet.getDate()));
}

```

```
private void setValues(View view, int title, String subtitle) {  
    TextView textTitle = (TextView) view.findViewById(R.id.title);  
    TextView textSubtitle = (TextView) view.findViewById(R.id.subtitle);  
    textTitle.setText(title);  
    textSubtitle.setText(subtitle);  
}  
}
```

Приложение Б

(обязательное)

Протокол проверки ВКР в системе «Антиплагиат»

Ректору Пензенского
государственного
университета
А. Д. Гулякову
студента 6 курса ФПИТЭ
очной формы обучения
Медведевой
Анастасии Викторовны

ЗАЯВЛЕНИЕ

Я, Медведева Анастасия Викторовна, студент 6 курса очной формы обучения направления подготовки 10.05.02 Информационная безопасность телекоммуникационных систем, прошу провести проверку с использованием системы «Антиплагиат» ВКР специалиста на тему «Программное обеспечение перехвата сетевого трафика на устройствах под управлением ОС Android», выполненной мной самостоятельно, на содержание элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим Стандартом университета «Выпускная квалификационная работа обучающихся по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры», согласно которому обнаружение плагиата является основанием для недопуска ВКР к защите и отчисления из университета.

21.01.2018



22.01.2018

ВКР представлена на проверку

Руководитель ВКР



О.В. Липилин

Протокол

проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Программное обеспечение перехвата сетевого трафика на устройствах под управлением ОС Android» студента 6 курса А.В. Медведевой специальности 10.05.02 «Информационная безопасность телекоммуникационных систем» в

Руководитель ВКР к.т.н., доцент О.В. Липилин

Отчет о проверке на заимствования

Автор: Медведева Анастасия Викторовна yemorphine@gmail.com / ID: 656
Проверяющий: Фатеев Алексей Григорьевич (ibst@pnzgu.ru) / ID: 393
Организация: Пензенский гос. Университет

Отчет предоставлен сервисом «Антиплагиат» - <http://pnzgu.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 43
Начало загрузки: 22.01.2018 23:15:30
Длительность загрузки: 00:00:04
Имя исходного файла: Медведева Анастасия Викторовна
ВКР_12ПТТ_Медведева.doc
Размер текста: 1402 кБ
Символов в тексте: 94018
Слов в тексте: 10608
Число предложений: 894

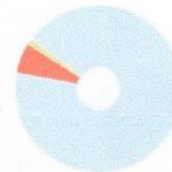
ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 22.01.2018 23:22:40
Длительность проверки: 00:00:21
Комментарий: не указано
Модули поиска: Модуль поиска ЭБС "библиоРоссика", Модуль поиска ЭБС "BOOK.ru", Коллекция РГБ, Цитирование, Модуль поиска ЭБС "Университетская библиотека онлайн", Коллекция eLIBRARY.RU, Модуль поиска ЭБС "Айбукс", Модуль поиска Интернет, Модуль поиска ЭБС "Лань", Модуль поиска "ПГУ", Кольцо вузов

ЗАИМСТВОВАНИЯ
6,05%

ЦИТИРОВАНИЯ
0,09%

ОРИГИНАЛЬНОСТЬ
93,86%



«23» января 2018 г

О.В. Липилин