

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

ОБНАРУЖЕНИЕ СЕТЕВОГО P2P-ТРАФИКА
КУРСОВАЯ РАБОТА

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Научный руководитель

д. к.ю. н., доцент

А. В. Гортинский

Заведующий кафедрой

д. ф.-м. н., доцент

М. Б. Абросимов

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Архитектура.....	5
1.1 Базовые элементы P2P-сетей	5
1.1.1 Узел P2P-сети	5
1.1.2 Группа узлов	6
1.1.3 Сетевой транспорт	6
1.2 Маршрутизация	6
1.2.1 Неструктурированные сети	6
1.2.2 Структурированные сети	7
1.2.3 Гибридные модели	7
1.3 Безопасность	8
1.3.1 Маршрутизационные атаки	8
1.3.2 Поврежденные данные и вредоносные программы	8
1.4 Отказоустойчивость и масштабируемость сети	9
1.5 Распределенное хранение и поиск	9
2 Применение P2P	10
3 Обнаружение P2P трафика без анализа полезной нагрузки.....	11
3.1 Анализ портов	11
3.2 Эвристические предположения	11
3.2.1 TCP/UDP-эвристика	12
3.2.2 IP/Port-эвристика	12
3.3 Обнаружение BitTorrent	15
3.3.1 Подключенные IP-адреса	15
3.3.2 Передача данных	16
3.3.3 Двусторонняя передача данных	16
3.3.4 Изменение отношений	16
3.3.5 Алгоритм	17
3.4 Исключения	18
3.4.1 Почта	18
3.4.2 DNS	19
3.4.3 Игры и вредоносные программы	20
3.4.4 Сканирование	22
3.4.5 Известные порты	22

4	Обнаружение P2P трафика при помощи анализа полезной нагрузки	23
4.1	Обнаружение BitTorrent	23
4.2	Обнаружение Bitcoin	24
5	Описание программы	25
5.1	Тестирование.....	27
ЗАКЛЮЧЕНИЕ		41
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		42
Приложение А Листинг main.py		43
Приложение Б Листинг sniffer.py		54

ВВЕДЕНИЕ

С развитием Интернета развивались файлообменные сети, благодаря которым появилась **P2P (peer-to-peer)** — одноранговая, децентрализованная или пиринговая сеть. Это распределённая архитектура приложения, которая разделяет задачи между узлами (peer). Узлы имеют одинаковые привилегии в приложении и образуют сеть равносильных узлов.

Узлы делают свои ресурсы, такие как вычислительная мощность, объем диска или пропускная способность, напрямую доступными остальным членам сети, без необходимости координировать действия с помощью серверов. Узлы являются одновременно поставщиками и потребителями ресурсов, в отличие от стандартной клиент-сервер модели, где поставщик и потребитель ресурсов разделены. [1]

В мае 1999 года, в Интернет с более чем миллионом пользователей, Шон Фэннинг внедрил приложение файлообменник Napster. Napster стал началом P2P-сети, такой какую мы знаем её сейчас, пользователи участвуют в создании виртуальной сети, полностью независимой от физической, без администрирования и каких-либо ограничений.

Концепция вдохновила новую философию во многих областях человеческого взаимодействия. P2P-технология позволяет пользователям интернета образовывать группы и коллаборации, формируя, тем самым, пользовательские поисковые движки, виртуальные суперкомпьютеры и файловые системы. Видение Всемирной паутины Тима Бернерса-Ли было близко к P2P-сети, в том смысле, что каждый пользователь является активным создателем и редактором контента.

В тоже время с появлением P2P появилась необходимость обнаруживать соответствующий трафик в сети. Универсального способа обнаружения работающего P2P-приложения нет. С развитием файлообменных сетей стало затруднительно идентифицировать P2P-трафик с помощью номеров портов. Появилась необходимость исследования трафика на основании поведения узлов сети. Однако даже поведение такого трафика, его сигнатура и прочие признаки также могут изменяться со временем, поэтому все существующие методы должны обновляться и совершенствоваться, чтобы поспевать за развитием P2P-приложений.

1 Архитектура

P2P-сеть строится вокруг понятия равноправных узлов — клиенты и серверы одинаково взаимодействуют с другими узлами сети. Такая модель построения сети отличается от модели клиент-сервер, где взаимодействие идет с центральным сервером. На рисунке 1 а) изображены архитектура клиент-сервера и б) архитектура P2P. Типичным примером передачи файла в модели клиент-сервер является File Transfer Protocol (FTP), в котором программы клиента и сервера разделены: клиент инициирует передачу, а сервер отвечает на запросы.



Рисунок 1 – Архитектура клиент-сервера и P2P

1.1 Базовые элементы P2P-сетей

1.1.1 Узел P2P-сети

Узел (Peer) — фундаментальный составляющий блок любой одноранговой сети. Каждый узел имеет уникальный идентификатор и принадлежит одной или нескольким группам. Он может взаимодействовать с другими узлами как в своей, так и в других группах. [2]

Виды узлов:

- **Простой узел.** Обеспечивает работу конечного пользователя, предоставляя ему сервисы других узлов и обеспечивая предоставление ресурсов пользовательского компьютера другим участникам сети.
- **Роутер.** Обеспечивает механизм взаимодействия между узлами, отделёнными от сети брандмауэрами или NAT-системами.

1.1.2 Группа узлов

Группа узлов — набор узлов, сформированный для решения общей задачи или достижения общей цели. Могут предоставлять членам своей группы такие наборы сервисов, которые недоступны узлам, входящим в другие группы.

Группы узлов могут разделяться по следующим признакам:

- приложение, ради которого они объединены в группу;
- требования безопасности;
- необходимость информации о статусе членов группы.

1.1.3 Сетевой транспорт

Конечные точки (Endpoints) — источники и приёмники любого массива данных передаваемых по сети.

Пайпы (Pipes) — однонаправленные, асинхронные виртуальные коммуникационные каналы, соединяющие две или более конечные точки.

Сообщения — контейнеры информации, которая передаётся через пайп от одной конечной точки до другой.

1.2 Маршрутизация

P2P относят к прикладному уровню сетевых протоколов, а P2P-сети обычно реализуют некоторую форму виртуальной (логической) сети, наложенной поверх физической, то есть описывающей реальное расположение и связи между узлами, такой сети, где узлы образуют подмножество узлов в физической сети. Данные по-прежнему обмениваются непосредственно над базовой TCP/IP сетью, а на прикладном уровне узлы имеют возможность взаимодействовать друг с другом напрямую, с помощью логических связей. Наложение используется для индексации и обнаружения узлов, что позволяет системе P2P быть независимой от физической сети. На основании того, как узлы соединены друг с другом внутри сети, и как ресурсы индексированы и расположены, сети классифицируются на **неструктурированные** и **структурированные** (или как их **гибрид**).

1.2.1 Неструктурированные сети

Неструктурированная P2P сеть не формирует определенную структуру сети, а случайным образом соединяет узлы друг с другом. Неструктурированные сети легко организуются и доступны для локальных оптимизаций, так как не существует глобальной структуры формирования сети. Кроме того, поскольку

роль всех узлов в сети одинакова, неструктурированные сети являются весьма надежными в условиях, когда большое количество узлов часто подключаются к сети или отключаются от неё.

Однако из-за отсутствия структуры возникают некоторые ограничения. В частности, когда узел хочет найти нужный фрагмент данных в сети, поисковый запрос должен быть направлен через сеть, чтобы найти как можно больше узлов, которые обмениваются данными. Такой запрос вызывает очень высокое количество сигнального трафика в сети, требует высокой производительности и не гарантирует, что поисковые запросы всегда будут решены.

1.2.2 Структурированные сети

В структурированных P2P-сетях наложение организуется в определенную топологию, и протокол гарантирует, что любой узел может эффективно участвовать в поиске файла или ресурса, даже если ресурс использовался крайне редко.

Наиболее распространенный тип структурированных сетей P2P реализуется распределенными хэш-таблицами (DHT), в котором последовательное хеширование используется для привязки каждого файла к конкретному узлу. Это позволяет узлам искать ресурсы в сети, используя хэш-таблицы, хранящие пару ключ-значение, и любой участвующий узел может эффективно извлекать значение, связанное с заданным ключом.

Тем не менее, для эффективной маршрутизации трафика через сеть, узлы структурированной сети должны обладать списком соседей, которые удовлетворяют определенным критериям. Это делает их менее надежными в сетях с высоким уровнем оттока абонентов (т.е. с большим количеством узлов, часто подключающихся к сети или отключающихся от нее).

1.2.3 Гибридные модели

Гибридные модели представляют собой сочетание P2P-сети и модели клиент-сервер. Гибридная модель должна иметь центральный сервер, который помогает узлам находить друг друга. Есть целый ряд гибридных моделей, которые находят компромисс между функциональностью, обеспечиваемой структурированной сетью модели клиент-сервер, и равенством узлов, обеспечиваемым чистыми одноранговыми неструктурированными сетями. В настоящее время гибридные модели имеют более высокую производительность, чем чисто неструк-

турированные или чисто структурированные сети.

1.3 Безопасность

Как и любая другая форма программного обеспечения, P2P-приложения могут содержать уязвимости. Особенно опасным для P2P программного обеспечения, является то, что P2P-приложения действуют и в качестве серверов, и в качестве клиентов, а это означает, что они могут быть более уязвимы для удаленных эксплоитов.

1.3.1 Маршрутизационные атаки

Поскольку каждый узел играет роль в маршрутизации трафика через сеть, злоумышленники могут выполнять различные «маршрутизационные атаки» или атаки отказа в обслуживании. Примеры распространенных атак маршрутизации включают в себя «неправильную маршрутизацию поиска», когда вредоносные узлы преднамеренно пересылают запросы неправильно или возвращают ложные результаты, «неправильную маршрутизацию обновления», когда вредоносные узлы изменяют таблицы маршрутизации соседних узлов, посылая им ложную информацию, и «неправильную маршрутизацию разделения сети», когда новые узлы подключаются через вредоносный узел, который помещает новичков в разделе сети, заполненной другими вредоносными узлами.

1.3.2 Поврежденные данные и вредоносные программы

Распространенность вредоносных программ варьируется между различными протоколами одноранговых сетей. Исследования, анализирующие распространение вредоносных программ по сети P2P, обнаружили, например, что 63% запросов на загрузку по сети Limewire содержали некоторую форму вредоносных программ, в то время как на OpenFT только 3% запросов содержали вредоносное программное обеспечение. Другое исследование анализа трафика в сети Kazaa показало, что 15% от 500 000 отобранных файлов были инфицированы одним или несколькими из 365 различных компьютерных вирусов.

Поврежденные данные также могут быть распределены по P2P-сети путем изменения файлов, которые уже были в сети. Например, в сети FastTrack, RIAA удалось внедрить фальшивые данные в текущий список загрузок и в уже загруженные файлы (в основном файлы MP3). Файлы, инфицированные вирусом RIAA, были непригодны впоследствии и содержали вредоносный код.

Следовательно, P2P-сети сегодня внедрили огромное количество механизмов безопасности и проверки файлов. Современное хеширование, проверка данных и различные методы шифрования сделали большинство сетей устойчивыми к практически любому типу атак, даже когда основные части соответствующей сети были заменены фальшивыми или нефункциональными узлами.

1.4 Отказоустойчивость и масштабируемость сети

Децентрализованность P2P-сетей повышает их надежность, так как этот метод взаимодействия устраняет ошибку единой точки разрыва, присущую клиент-серверным моделям. С ростом числа узлов объем трафика внутри системы увеличивается, масштаб сети так же увеличивается, что приводит к уменьшению вероятности отказа. Если один узел перестанет функционировать должным образом, то система в целом все равно продолжит работу. В модели клиент-сервер с ростом количества пользователей уменьшается количество ресурсов выделяемых на одного пользователя, что приводит к риску возникновения ошибок.

1.5 Распределенное хранение и поиск

Возможность резервного копирования данных, восстановление и доступность приводят как к преимуществам, так и к недостаткам P2P-сетей. В централизованной сети только системный администратор контролирует доступность файлов. Если администраторы решили больше не распространять файл, его достаточно удалить с серверов, и файл перестанет быть доступным для пользователей. Другими словами, клиент-серверные модели имеют возможность управлять доступностью файлов. В P2P-сети доступность контента определяется степенью его популярности, так как поиск идет по всем узлам, через которые файл проходил. То есть, в P2P-сетях нет централизованного управления как системного администратора в клиент-серверном варианте, а сами пользователи определяют уровень доступности файла.

2 Применение P2P

В P2P сетях, пользователи передают и используют контент сети. Это означает, что, в отличие от клиент-серверных сетей, скорость доступа к данным возрастает с увеличением числа пользователей, использующих этот контент. На этой идее построен протокол BitTorrent — пользователи, скачавшие файл, становятся узлами и помогают другим пользователям скачать файл быстрее. Эта особенность является главным преимуществом P2P сетей.

Множество файлообменных систем, таких как Gnutella, G2 и eDonkey популяризовали P2P технологии:

- Пиринговые системы распространения контента.
- Пиринговые системы обслуживания, например, повышение производительности, в частности, Correli Caches.
- Пиринговые платёжные системы.
- Публикация и распространение программного обеспечения (Linux, видео-игры).
- Аудио- и видео-звонки.

В связи децентрализованностью доступа к данным в P2P сетях возникает проблема нарушения авторских прав. Компании, занимающиеся разработкой P2P приложений часто принимают участие в судебных конфликтах. Самые известные судебные дела это Grokster против RIAA и MGM Studios, Inc. против Grokster Ltd., где в обоих случаях технологии файлообменных систем признавались законными.

3 Обнаружение P2P трафика без анализа полезной нагрузки

3.1 Анализ портов

Многие P2P-приложения работают на определённых портах. Некоторые из таких указаны в таблице 1 [3].

Таблица 1 – Список наиболее известных портов, используемых P2P-протоколами

Протоколы	Номера TCP/UDP портов
BitTorrent	6881-6999
Direct Connect	411, 412, 1025-32000
eDonkey	2323, 3306, 4242, 4500, 4501, 4661-4674, 4677, 4678, 4711, 4712
FastTrack	1214, 1215, 1331, 1337, 1683, 4329
Yahoo	5000-50010, 5050, 5100
Napster	5555, 6257, 6666, 6677, 6688, 6699-6701
MSN	1863, 6891-6901
MP2P	10240-20480, 22321, 41170
Kazaa	1214
Gnutella	6346, 6347
ARES Galaxy	32285
AIM	1024-5000, 5190
Skype	3478-3481
Steam (голосовой чат)	27015-27030

Для реализации данного метода достаточно обнаружить в сетевом трафике соединения, использующие такие порты. Очевидно, что данный способ легко реализовать, однако он имеет недостатки. Во-первых, многие приложения могут использовать случайные порты, или же пользователь может сам выбрать номер порта. Во-вторых, такие порты могут использоваться не P2P-приложениями и наоборот, P2P-приложения могут использовать номера портов известных приложений, например, 80 или 443 порты — HTTP и HTTPS. Так, в работе [4] приведены результаты, которые показывают, что зачастую на основе данного метода можно определить лишь 30% P2P-трафика.

3.2 Эвристические предположения

Две основные эвристики были получены в ходе статистического анализа объёма сетевого трафика, проходящего через интернет-провайдеров в течение

определённого времени. В работах [5] и [6] приводится информация о трассах, на которых проводились исследования трафика.

3.2.1 TCP/UDP-эвристика

Часть протоколов P2P используют одновременно TCP и UDP в качестве транспортных протоколов. Как правило, управляющий трафик, запросы и ответы на запросы используют UDP, а фактическая передача данных — TCP. Тогда для идентификации узлов P2P можно искать пары источник-назначение, которые используют оба транспортных протокола.

Хотя одновременное использование TCP и UDP типично для множества P2P-протоколов, оно также используется и в других протоколах. Например, это DNS, NetBIOS, IRC, игры и потоковое вещание, которые обычно используют небольшой набор стандартных портов, таких как 135, 137, 139, 445, 53 и так далее. Таким образом, если пара адресов источник-назначение одновременно использует TCP и UDP в качестве транспортных протоколов, и порты источника или назначения не входят в набор исключений, то потоки между этой парой будут считаться как P2P.

3.2.2 IP/Port-эвристика

Вторая эвристика основана на отслеживании шаблонов соединений пар {IP, Port}. В распределённых сетях, например, BitTorrent, клиент поддерживает некоторый стартовый кэш других хостов. В зависимости от сети, этот кэш может содержать IP-адреса других пиров, серверов или **суперпиров**. Суперпиры — узлы P2P-сети, которые выполняют дополнительные функции, такие как маршрутизация и распространение запросов. Набор адресов, которые они содержат, обеспечивает первоначальное подключение нового пира к уже существующей P2P-сети.

При установлении соединения с одним из IP-адресов в кэше, который будет являться суперпиром, новый хост А сообщит этому суперпиру свой IP-адрес и номер порта (и другую информацию, зависящую от конкретной сети), на котором он будет принимать соединения от остальных пиров. Если раньше в P2P-сетях прослушиваемый порт был чётко задан для каждой сети, что упрощало классификацию P2P-трафика, то сейчас более новые версии позволяют либо настроить свой, произвольный номер порта, либо использовать случайный. Суперпир же должен распространить полученную информацию, в основном имен-

но IP-адрес и порт нового хоста А остальным участникам сети. Рисунки 2 и 3 демонстрируют этот процесс.

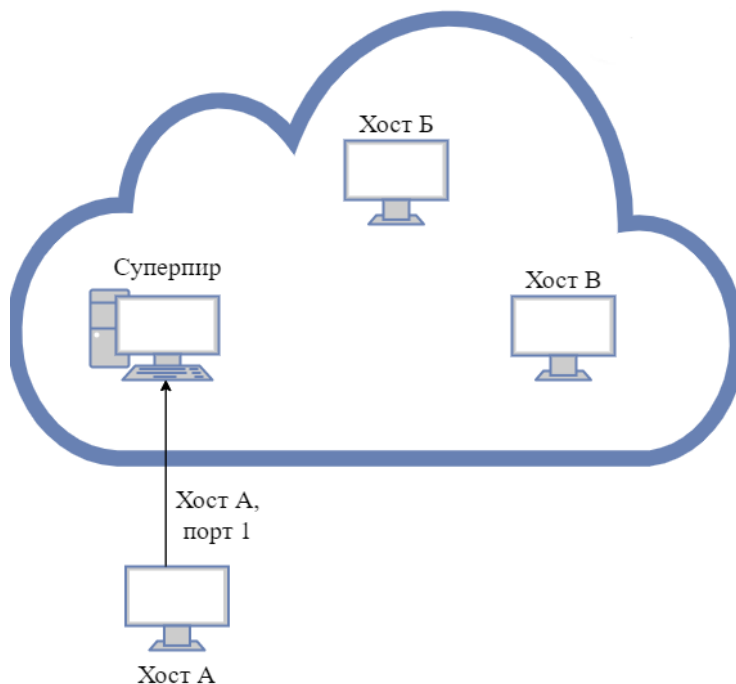


Рисунок 2 – Отправка информации хоста А о себе суперпир

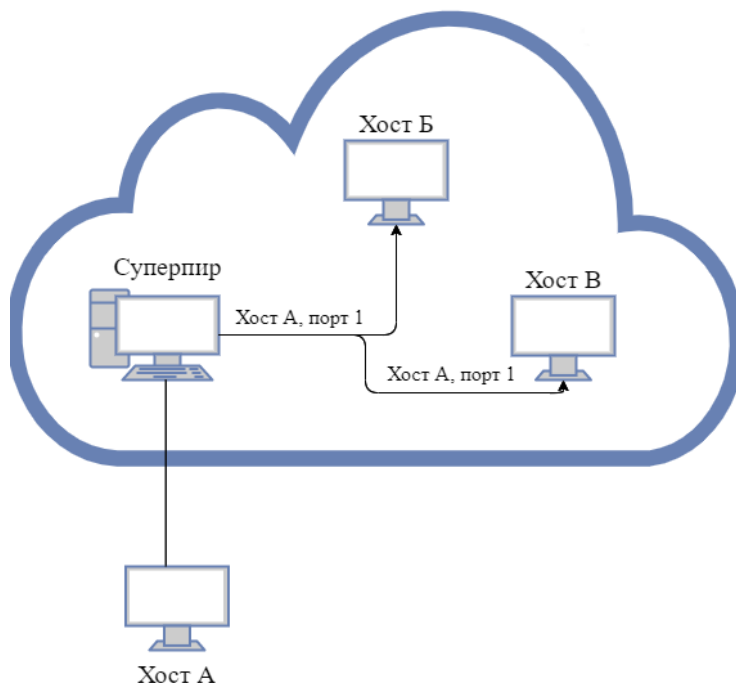


Рисунок 3 – Суперпир распространяет информацию о хосте А остальной части сети

По существу, пара IP-адрес и порт — идентификатор нового хоста, который другие пиры должны использовать для подключения к нему. Когда P2P-хост инициирует TCP или UDP соединение с хостом А, порт назначения будет

портом, который прослушивает хост А, а порт источника будет случайным, выбранным клиентом.

Обычно пиры поддерживают не более одного TCP соединения с каждым другим пиром, но, как описано ранее, можно быть ещё один UDP поток. Итак, множественные соединения между пирами это редкое явление. Рассмотрим случай, если, например, 20 пиров подключатся к хосту А. Каждый из них выберет временный порт источника и подключится к объявленному порту, который прослушивает хост А. Таким образом, объявленная пара IP-адреса и порта хоста А будет связана с 20 различными IP-адресами и 20 различными портами. Таким образом, для пары хоста А количество различных IP-адресов и различных портов, используемых для подключения к нему, будет равно. Рисунок 4 иллюстрирует данный случай.

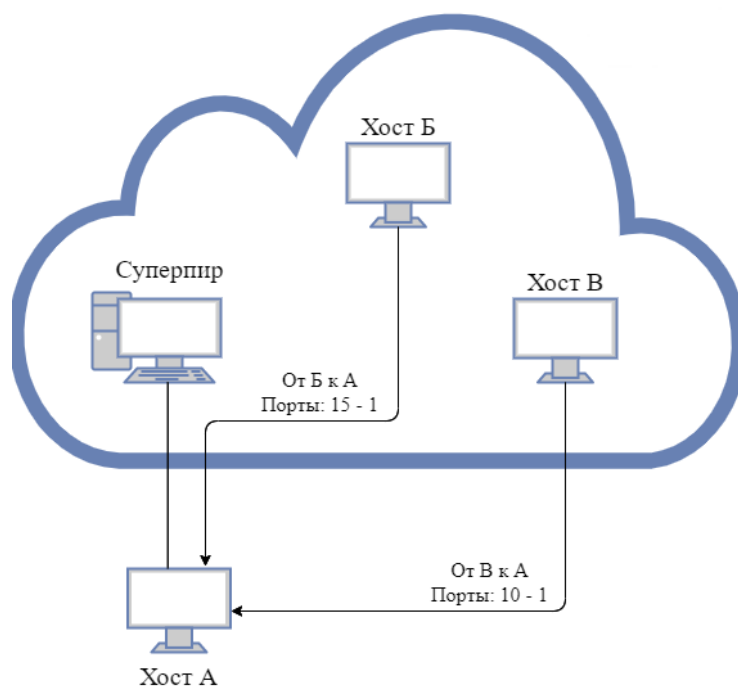


Рисунок 4 – К хосту А подключены hosts Б и В с 2 разными IP и 2 разными портами

С другой стороны, рассмотрим случай, когда используется сеть с архитектурой клиент-сервер, пусть это будет веб-сервер. Как и в случае с P2P, каждый хост подключается к заранее определённой паре, например, IP-адрес веб-сервера и 80 порт. Однако хост, подключающийся к веб-серверу обычно иницирует несколько одновременных соединений, например, для параллельной загрузки. Тогда веб-трафик будет иметь более высокое, по сравнению с P2P-трафиком, соотношение числа отдельных портов к числу отдельных IP-адресов.

В работе [5] приводятся графики (рисунок 5) зависимости между количеством IP-адресов назначения и портов назначения для веб- и P2P-приложений. В веб-случае большинство точек концентрируется выше диагонали, представляя параллельные соединения в основном одновременных загрузок веб-объектов. Напротив, в P2P-случае большинство точек группируется ближе к диагонали, либо немного ниже (что характерно для случаев, когда номер порта постоянен).

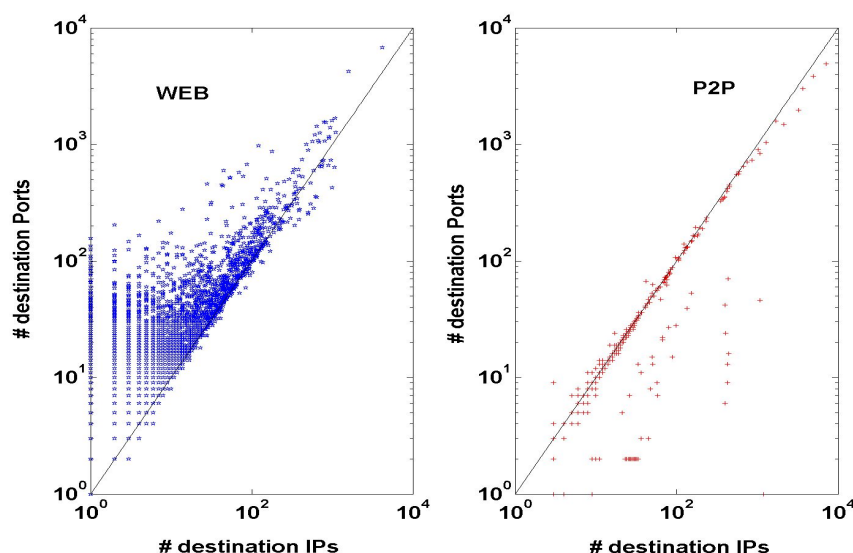


Рисунок 5

Таким образом, если для пары {IP, Port} множество адресов источников содержит больше двух элементов и разница между длинами этого множества и множества портов источников меньше 2 (или меньше 10, если один из портов относится к известным P2P-портам), то пара заносится в список адресов данной эвристики. Если же разница больше 10, то пара заносится в список адресов-исключений.

3.3 Обнаружение BitTorrent

В работе [7] предложен алгоритм, который основывается на четырёх критериях.

3.3.1 Подключенные IP-адреса

Первый критерий основан на IP/Port-эвристике. Хосты BitTorrent всегда подключены ко многим IP-адресам. Под подключенными IP-адресами понимается, что они передали друг другу хотя бы по одному TCP-пакету. В BitTorrent это может быть необходимо для подключения к раздаче и передачи особенных

сообщений (*choke*, *have*, *keepalive*). Причём каждый пир (участник) пытается поддерживать не менее 20 пиров, следовательно, каждый пир периодически отправляет несколько TCP-пакетов на один и тот же набор IP-адресов.

3.3.2 Передача данных

BitTorrent разбивает исходные файлы на небольшие части, поэтому пользователи могут скачивать разные файлы от разных пользователей. Это можно определить по значимому соотношению активных передач. Под активной передачей подразумевается хотя бы 5 больших TCP-пакетов, т.е. размер пакета должен быть примерно равен *MTU* (максимальная единица передачи). В Ethernet это около 1500 байт. Передача пакетов максимального размера необходима для того, чтобы их количество было минимальным для передачи файла.

Однако пиры BitTorrent не всегда одновременно обмениваются данными между собой. Это связано с *алгоритмом дросселирования* (*choke*). Этот алгоритм выбирает соседей, которым будут раздаваться или с которых будут скачиваться файлы. В любой момент времени пир загружает данные не более, чем с 4 пиров, которые обеспечивают самую высокую скорость загрузки.

3.3.3 Двусторонняя передача данных

Процесс выбора в алгоритме дросселирования приводит к двусторонней передаче данных. В отличие от BitTorrent, другие интернет-приложения обычно работают по схеме клиент-сервер, поэтому данные передаются только в одном направлении в определённый промежуток времени. Кроме того, в других протоколах P2P-обмена между элементами нет взаимного обмена, который заложен в алгоритме дросселирования. Пирам в этих протоколах не нужно загружать свои фрагменты другим пирам, с которых они скачивают данные.

3.3.4 Изменение отношений

В алгоритме дросселирования все пиры в наборе сортируются каждые 10 секунд в порядке убывания скорости загрузки данных. После сортировки локальный пир будет раздавать данные только первым четырём пирам в отсортированном списке. Учитывая, что скорость передачи довольно динамична, выбранные пиры будут часто меняться. Таким образом, пара пиров может активно передавать данные друг другу, но потом внезапно может стать неактивной. В

результате хост BitTorrent может быть идентифицирован по значимому соотношению изменений IP-отношений к активным передачам.

3.3.5 Алгоритм

На основании четырёх критериев создаются специальные метрики, которые рассчитываются каждые 30 секунд и сравниваются с пороговым значением, чтобы определить, является ли хост пиром BitTorrent.

1. **Подключения.** Подсчитывается число C — количество пиров, которые общались с хостом. Если это количество будет больше или равно порогу $C_{\text{порог.}}$, то хост будет идентифицирован как BitTorrent-хост.

$$C \geq C_{\text{порог.}}$$

2. **Коэффициент активной передачи.** Коэффициент активной передачи хоста R_{AT} — отношение числа активных подключений AT к общему числу подключений C . Если этот коэффициент больше или равен пороговому $R_{AT\text{порог.}}$, то хост будет идентифицирован как BitTorrent-хост.

$$R_{AT} \geq R_{AT\text{порог.}},$$

где $R_{AT} = \frac{AT}{C}$.

3. **Двусторонние передачи данных.** Измеряется количество подключений $BiAT$, по которым одновременно принимаются и отправляются данные. Если это число больше или равно пороговому $BiAT_{\text{порог.}}$, то хост будет идентифицирован как BitTorrent-хост.

$$BiAT \geq BiAT_{\text{порог.}}$$

4. **Коэффициент изменений отношений.** Коэффициент изменений отношений R_{RC} — отношение числа изменений отношений RC к числу активных передач AT . Если этот коэффициент больше или равен пороговому $R_{RC\text{порог.}}$, то хост будет идентифицирован как BitTorrent-хост.

$$R_{RC} \geq R_{RC\text{порог.}},$$

где $R_{RC} = \frac{RC}{AT}$.

В качестве пороговых в программе используются следующие значения:

- $C_{\text{порог.}} = 20$
- $R_{AT\text{порог.}} = 0.35$
- $BiAT_{\text{порог.}} = 5$
- $R_{RC\text{порог.}} = 0.5$

Данные значения являются наиболее оптимальными. Метрики «коэффициент активной передачи» и «коэффициент изменений отношений» сравниваются с пороговыми при условии, что C (число подключений) больше или равно половине своего порогового значения $C_{\text{порог.}}$, то есть 10.

3.4 Исключения

Чтобы снизить количество ложных срабатываний, необходимо учитывать протоколы, поведение которых может быть схожим с поведением P2P-протоколов. Стандартные сетевые протоколы обычно имеют стандартные номера портов, что очень удобно для фильтрации трафика. В то же время, для некоторых приложений всё же необходимо использовать иные подходы.

3.4.1 Почта

Поведение почтовых протоколов, таких как SMTP и POP, может вызвать ложное срабатывание, поскольку оно похоже на IP/Port-эвристику. Почтовые серверы возможно идентифицировать на основе использования ими портов 25 для SMTP, 110 для POP или 113 для сервиса аутентификации, который обычно используется почтовыми серверами, а также на основе наличия различных потоков в течение некоторого временного интервала, которые используют порт 25 как для порта источника, так и для порта назначения.

Таблица 2 иллюстрирует характерное поведение почтовых серверов:

Таблица 2 – Пример почтового ТСП трафика

IP-адрес источника	IP-адрес назначения	Порт источника	Порт назначения
238.30.35.43	115.78.57.213	25	3267
238.30.35.43	238.45.242.104	25	25
238.30.35.43	0.32.132.109	22092	50827
238.30.35.43	71.199.74.68	25	25
238.30.35.43	4.87.3.29	21961	25
238.30.35.43	4.87.3.29	22016	25
238.30.35.43	4.170.125.67	25	3301
238.30.35.43	5.173.60.126	22066	25
238.30.35.43	5.173.60.126	22067	25
238.30.35.43	227.186.155.214	22265	25
238.30.35.43	227.186.155.214	22266	25
238.30.35.43	5.170.237.207	25	3872

В этом примере показаны потоки для IP-адреса 238.30.35.43, где порт 25 является портом источника в одних потоках и назначения в других. Такое поведение характерно для почтовых серверов, которые инициируют подключения к другим почтовым серверам для распространения сообщений электронной почты. Для выявления такой модели отслеживается набор номеров портов назначения для каждого IP-адреса, для которого существует пара-источник {IP, 25}. Если этот набор номеров портов назначения также содержит порт 25, то этот IP считается за почтовый сервер, и все его потоки классифицируются как не P2P. Аналогично для набора портов источника IP, для которого существует пара-назначение {IP, 25}. В приведённом выше примере для пары {238.30.35.43, 25} набор портов назначения: 3267, 25, 50827, 3301, 3872. Так как в этом наборе есть порт 25, то из этого следует вывод, что данный IP-адрес относится к почтовому серверу и все его потоки будут считать не P2P.

3.4.2 DNS

Протокол DNS, как и почтовые протоколы, может быть ложно принят за P2P из-за IP/Port-эвристики, хотя DNS легче идентифицировать, поскольку обычно порты источника и назначения равны 53.

Таким образом, если найдётся пара {IP, 53}, которая будет либо источником, либо назначением, то все потоки, содержащие данный IP-адрес, будут

считаться как не P2P. Заметим, что при этом потоки, содержащие обращения к DNS-службе со стороны участников P2P-обмена, также считаются не P2P. Однако P2P-клиенты имеют небольшое количество обращений к DNS-службе, так как получают нужную информацию друг от друга.

3.4.3 Игры и вредоносные программы

Игры и вредоносные программы (malware) характеризуются однотипными потоками, имеющими одну и ту же длину или небольшой разброс средних размеров пакетов в потоке. Для исключения такого взаимодействия сохраняется соответствующая информация и проводится проверка. Однако такая проверка трудно реализуема, поскольку размеры пакетов будут зависеть от каждой конкретной игры или вредоносной программы. В работе [8] выдвигается предположение, что множество длин не будет превышать, например, трёх. Хотя на практике множество длин обычно намного больше, чем три.

Например, на рисунках 6 и 7 изображены гистограммы, построенные на основе перехваченного сетевого трафика двух многопользовательских игр: *War Thunder* и *Dota 2* в течение одной игровой сессии. Трафик обеих игр был определён реализованной в данной работе программой как P2P по IP/Port-эвристике.

Здесь каждому столбцу по горизонтали соответствует диапазон размеров пакетов в байтах и по вертикали среднее их количество по диапазону. Так, в *War Thunder* большая часть пакетов имеет размер 18 байт, в то время как в *Dota 2* — около 150 байт.

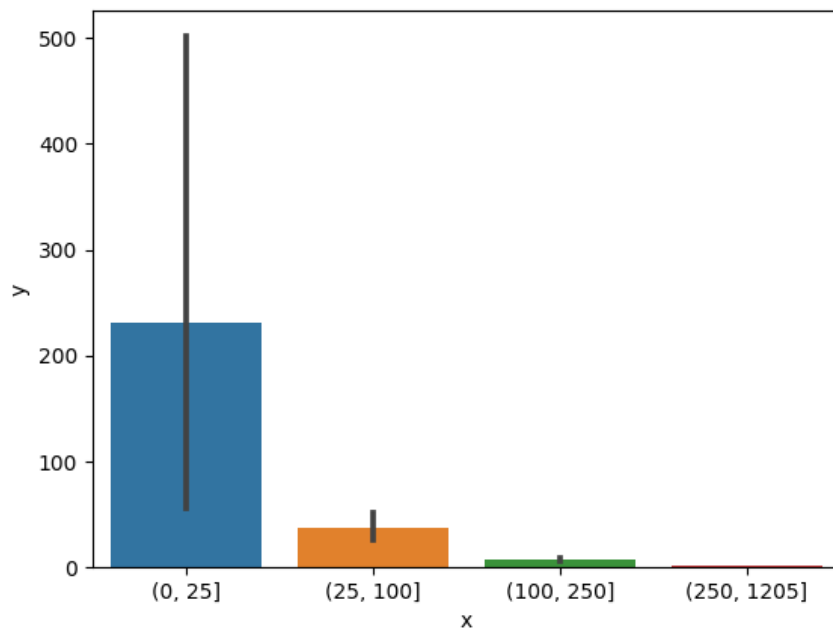


Рисунок 6 – Графическое представление среднего количества пакетов различных диапазонов их размеров в игре War Thunder

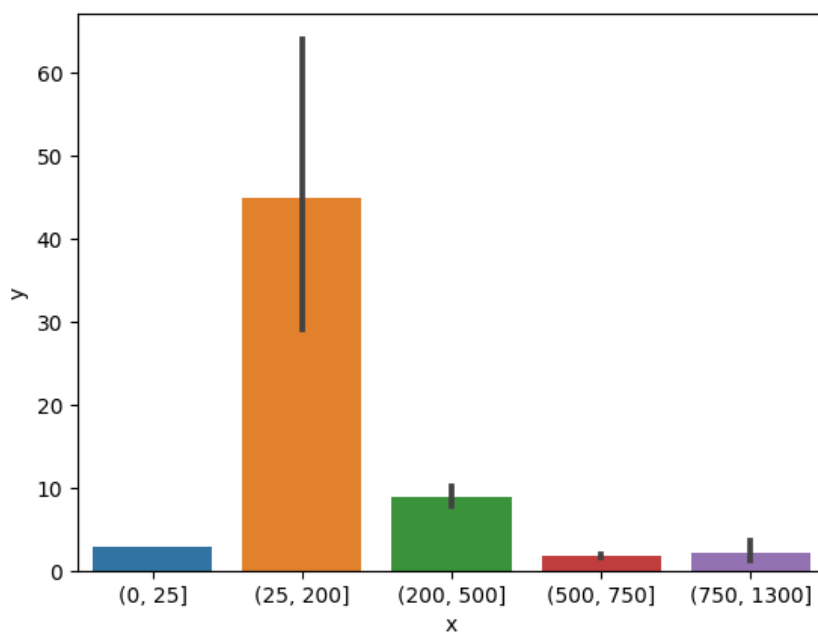


Рисунок 7 – Графическое представление среднего количества пакетов различных диапазонов их размеров в игре Dota 2

Хорошо видно, что игровой трафик обычно характеризуется отправкой небольших пакетов. Но могут встречаться и большие пакеты, например, при передаче больших объемов данных или при обмене файлами внутри игрового

приложения. Тем не менее, только лишь по данному признаку нельзя точно определить игровой трафик, так как и другие сетевые приложения могут иметь данный признак. Для большей точности, предположительно, стоит дополнительно использовать определение по номерам портов и анализ полезной нагрузки пакетов.

3.4.4 Сканирование

Если адрес назначения {IP, Port} подвергается распределенному сканированию или атаке со стороны множества адресов, то обычно ответы от пары {IP, Port} отсутствуют или их крайне мало. В таком случае, если данная пара не была определена ранее как P2P, то она считается как не P2P, несмотря на верность IP/Port-эвристики. В таком случае говорят, что верна эвристика сканирования.

3.4.5 Известные порты

Наконец, если в потоке порт источника и порт назначения совпадают, и оба меньше или равны 500, то такой поток считается не P2P. Подобное поведение нехарактерно для P2P, но характерно для ряда легальных взаимодействий, например, для сервисов NTP (порт 123) или DNS (порт 53).

4 Обнаружение P2P трафика при помощи анализа полезной нагрузки

Анализ полезной нагрузки пакетов может оказаться достаточно трудоёмким или вовсе не реализуемым в конкретный временной промежуток процессом, поскольку существует множество факторов, ограничивающих исследование передаваемых данных. Во-первых, всё большее число приложений и протоколов используют шифрование и TLS (transport layer security) при передаче пакетов по сети. По этой причине сопоставить некоторые шаблонные строки с информацией, обнаруженной внутри перехваченного пакета, становится невозможно. Во-вторых, сигнатуры каждого конкретного приложения могут меняться, поэтому их базу придётся регулярно обновлять. В-третьих, некоторые протоколы, в особенности проприетарные, например, протокол Skype, используют обфускацию данных в пакете, что дополнительно усложняет их анализ [9].

Тем не менее, некоторые современные протоколы могут передавать часть информации в открытом, незашифрованном виде. Если обнаружить момент передачи такой информации и идентифицировать протокол, с помощью которого эти данные были переданы, то далее в определённый временной промежуток можно считать пару адресов, участвующих в этой передаче, за участников или пользователей некой сети (в данной работе интерес представляют именно P2P-сети).

4.1 Обнаружение BitTorrent

Первым сообщением, которое обязан передать клиент перед началом соединения, является handshake (рукопожатие). Формат рукопожатия следующий [10]:

- **pstrlen**: длина имени протокола;
- **pstr**: имя протокола;
- **reserved**: 8 резервных байт;
- **info_hash**: 20-байтовый SHA1 хэш информационного ключа файла Metainfo;
- **peer_id**: 20-байтовая строка, представляющая собой уникальный номер клиента.

Именно handshake пакеты представляют интерес при обнаружении BitTorrent, поскольку первые два заголовка передаются в открытом виде. На

основе этих заголовков и формируются условия, при которых пакет относится к BitTorrent:

1. Минимальная длина полезной нагрузки пакета 20 байт.
2. Байт со значением 19.
3. Следующая за ним строка «BitTorrent protocol».

В шестнадцатиричном формате заголовки *pstrlen* и *pstr* будут выглядеть как «13 42 69 74 54 6f 72 72 65 6e 74 20 70 72 6f 74 6f 63 6f 6c».

При выполнении всех перечисленных условий считается, что пара адресов (вместе с номерами портов) взаимодействует при помощи BitTorrent, поэтому они отмечаются как P2P. В дальнейшем, все проходящие пакеты между этой парой адресов считаются как пакеты BitTorrent.

4.2 Обнаружение Bitcoin

Сеть Bitcoin использует специальный порт для обмена данными между узлами — 8333 для протокола TCP и 8334 для протокола UDP. При этом, обмен данными в сети Bitcoin шифруется, что затрудняет идентификацию трафика.

Однако Bitcoin использует специфичные сообщения (назовём их словами Bitcoin)[11]: `version`, `verack`, `addr`, `inv`, `getdata`, `notfound`, `getblocks`, `getheaders`, `tx`, `block`, `headers`, `getaddr`, `mempool`, `checkorder`, `submitorder`, `reply`, `ping`, `pong`, `reject`, `filterload`, `filteradd`, `filterclear`, `merkleblock`, `alert`, `sendheaders`, `feefilter`, `sendcmpct`, `cmpctlblock`, `getblocktxn`, `blocktxn`, `Satoshi`.

Исходя из данных особенностей, пара адресов (вместе с номерами портов) считается участниками Bitcoin-сети, если выполняются следующие условия:

1. Минимальная длина полезной нагрузки пакета 20 байт.
2. Порт источника или назначения равен 8333 или 8334.
3. В пакете содержится любое из слов Bitcoin.

Выполнение одновременно 2 и 3 условий необходимо для того, чтобы, насколько это возможно, исключить те случаи, когда иные приложения используют порты Bitcoin или те же самые слова. Например, слово «version» может встречаться в HTTP, FTP, Git, SVN и так далее.

5 Описание программы

В данной работе был разработан **сниффер** — анализатор сетевого трафика. Используемый язык программирования — Python. Программа выводит на экран информацию о перехваченных пакетах таких сетевых протоколов как *IPv4*, *TCP* и *UDP* и анализирует перехваченный трафик на его принадлежность к P2P-сетям, а также определяет часть P2P-протоколов и приложений, например, BitTorrent, Bitcoin и Skype. Сканирование производится каждые 75 мс. Дополнительно последний вывод программы сохраняется в текстовые файлы.

При запуске необходимо выбрать прослушиваемый сетевой интерфейс из предложенного списка:

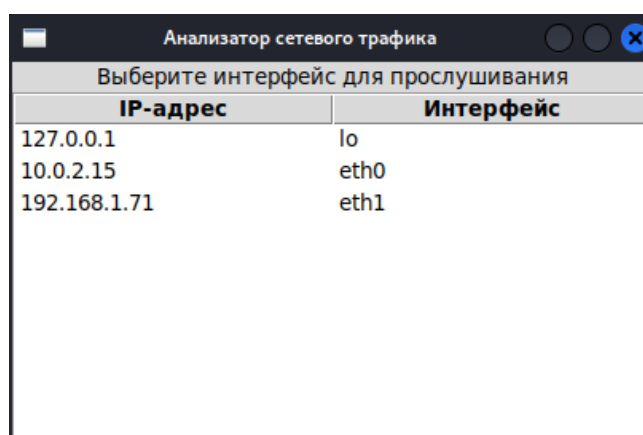


Рисунок 8 – Выбор сетевого интерфейса при старте программы

Для указанного сетевого интерфейса программа включает неразборчивый режим.

Далее происходит перехват TCP/UDP трафика и вывод информации о нём в левой части окна. В правой части расположены списки адресов тех узлов, которые программа определила как участников P2P-сети. Каждому списку соответствует конкретный метод обнаружения. В правом нижнем списке с подписью «Пересечение методов» выводятся адреса, которые были обнаружены хотя бы двумя различными методами одновременно.

Анализатор сетевого трафика									
Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов		IP/Port эвристика
13:07:25	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б		15.235.40.193:6881		192.168.1.132:55069
13:07:25	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent	46.173.42.166:6881		192.168.1.132:6881
13:07:25	192.168.1.132	192.168.1.255	137 -> 137	UDP	50 Б		81.28.188.57:6881		192.168.1.132:52050
13:07:25	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent	46.229.188.217:6883		
13:07:25	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent	89.109.199.212:6881		
13:07:25	192.168.1.132	5.165.50.102	55069 -> 41741	UDP	20 Б	P2P BitTorrent	5.165.240.56:6881		
13:07:25	20.135.20.1	192.168.1.132	443 -> 50108	TCP	1452 Б		5.136.193.67:6881		
13:07:25	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б		109.167.170.28:6881		
13:07:25	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б		88.86.76.18:6881		
13:07:26	20.135.20.1	192.168.1.132	443 -> 50108	TCP	2904 Б		95.29.97.200:6881		
13:07:26	95.73.67.8	192.168.1.132	55316 -> 49433	TCP	0 Б		95.190.58.55:6881		
13:07:26	192.168.1.132	95.73.67.8	49433 -> 55316	TCP	6 Б		176.215.151.39:6881		
13:07:26	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent	TCP/UDP эвристика		По полезной нагрузке
13:07:26	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent	31.163.71.193		31.134.181.131:24737
13:07:26	192.168.1.132	5.165.50.102	55069 -> 41741	UDP	20 Б	P2P BitTorrent	192.168.1.132		192.168.1.132:64474
13:07:26	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent			192.168.1.132:55069
13:07:26	5.165.50.102	192.168.1.132	41741 -> 55069	UDP	1427 Б	P2P BitTorrent			212.93.112.149:28457
13:07:26	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б				93.23.157.130:59590
13:07:26	20.135.20.1	192.168.1.132	443 -> 50108	TCP	1452 Б				192.168.1.132:64496
13:07:26	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent			81.198.235.120:21467
13:07:26	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent			
13:07:27	20.135.20.1	192.168.1.132	443 -> 50108	TCP	1452 Б				
13:07:27	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б				
13:07:27	20.135.20.1	192.168.1.132	443 -> 50108	TCP	2904 Б				
13:07:27	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent	По метрикам BT		Пересечение методов
13:07:27	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent	192.168.1.132:55069		192.168.1.132:55069
13:07:27	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent			192.168.1.132:6881
13:07:27	192.168.1.132	5.137.229.92	55069 -> 62030	UDP	20 Б	P2P BitTorrent			
13:07:27	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent			
13:07:27	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent			
13:07:27	192.168.1.132	5.137.229.92	55069 -> 62030	UDP	20 Б	P2P BitTorrent			
13:07:27	5.137.229.92	192.168.1.132	62030 -> 55069	UDP	1427 Б	P2P BitTorrent			
13:07:28	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б				
13:07:28	20.135.20.1	192.168.1.132	443 -> 50108	TCP	7260 Б				
13:07:28	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б				
13:07:28	20.135.20.1	192.168.1.132	443 -> 50108	TCP	2904 Б				
13:07:28	192.168.1.132	20.135.20.1	50108 -> 443	TCP	6 Б				

Рисунок 9 – Основное окно программы

Для обработки исключений используется следующий алгоритм: для каждой пары адресов {src_ip, src_port} → {dest_ip, dest_port} проверяется, если один из портов находится в списке портов-исключений или порт источника src_port равен порту назначения dest_port, и они оба меньше или равны 500, то обе пары адресов заносятся в список адресов-исключений. Далее при вызове любого из методов обнаружения P2P-трафика сначала проверяется, не является ли пара исключением. Если нет, то применяется сам метод.

Метод анализа портов и полезной нагрузки вызываются для каждого нового пакета. Проверка TCP/UDP- и IP/Port-эвристики вызывается каждые 15 секунд. Сравнение метрик BitTorrent проводится каждые 30 секунд.

Определение P2P-протоколов выполняется при помощи анализа портов, полезной нагрузки и метода сравнения метрик BitTorrent. Увидеть результат определения можно в левой части окна программы в столбце «Инфо».

5.1 Тестирование

Тестирование проводилось при следующих условиях: программа запущена на виртуальной машине под операционной системой Linux, дистрибутив Kali в программе VirtualBox. Хостовая машина под Windows 10 подключена к виртуальной через сетевой мост с включенным неразборчивым режимом. Все приложения запускаются на хостовой машине, а программа, запущенная на виртуальной машине, перехватывает сетевой трафик этих приложений.

IP-адрес хостовой машины: 192.168.1.132, виртуальной — 192.168.1.71.

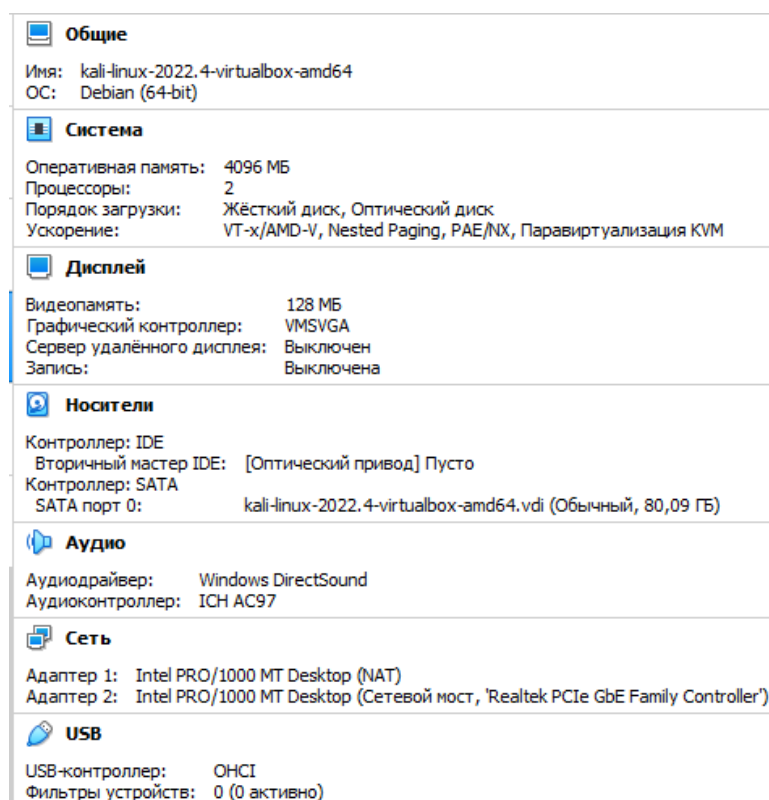


Рисунок 10 – Настройки виртуальной машины

С помощью монитора ресурсов «resmon» на Windows 10 можно просмотреть список TCP-подключений и прослушиваемые TCP/UDP порты, чтобы проверить принадлежность адреса или порта к какому-либо сетевому приложению.

Работа программы при активном веб-трафике (видео, музыка, социальные сети и загрузка файла с облака):

Анализатор сетевого трафика								
Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов	IP/Port эвристика
14:40:58	87.240.129.131	192.168.1.132	443 -> 54765	TCP	6 Б			
14:40:58	87.240.129.131	192.168.1.132	443 -> 54765	TCP	35 Б			
14:40:58	192.168.1.132	87.240.129.131	54765 -> 443	TCP	6 Б			
14:40:58	192.168.1.118	224.0.0.7	8001 -> 8001	UDP	200 Б			
14:40:58	192.168.1.132	213.59.237.157	58408 -> 443	UDP	1077 Б			
14:40:58	213.59.237.157	192.168.1.132	443 -> 58408	UDP	34 Б			
14:40:58	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:40:58	95.108.245.54	192.168.1.132	443 -> 56226	TCP	0 Б			
14:40:59	192.168.1.132	213.59.237.157	58408 -> 443	UDP	35 Б			
14:40:59	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:40:59	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:40:59	192.168.1.132	213.59.237.157	58408 -> 443	UDP	35 Б			
14:40:59	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:40:59	192.168.1.132	26.255.255.255	137 -> 137	UDP	50 Б			
14:40:59	192.168.1.132	216.58.207.238	56238 -> 443	TCP	0 Б			
14:40:59	216.58.207.238	192.168.1.132	443 -> 56238	TCP	0 Б			
14:40:59	192.168.1.132	216.58.207.238	56238 -> 443	TCP	6 Б			
14:40:59	192.168.1.132	216.58.207.238	56238 -> 443	TCP	517 Б			
14:41:00	192.168.1.132	216.58.207.238	56238 -> 443	TCP	517 Б			
14:41:00	192.168.1.118	224.0.0.7	8001 -> 8001	UDP	200 Б			
14:41:00	192.168.1.132	87.240.190.90	55746 -> 443	TCP	1595 Б			
14:41:00	192.168.1.132	216.58.207.238	56238 -> 443	TCP	517 Б			
14:41:00	87.240.190.90	192.168.1.132	443 -> 55746	TCP	6 Б			
14:41:00	87.240.190.90	192.168.1.132	443 -> 55746	TCP	185 Б			
14:41:00	192.168.1.132	213.59.237.157	58408 -> 443	UDP	1077 Б			
14:41:00	213.59.237.157	192.168.1.132	443 -> 58408	UDP	34 Б			
14:41:00	213.59.237.157	192.168.1.132	443 -> 58408	UDP	147 Б			
14:41:00	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1245 Б			
14:41:00	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:41:00	192.168.1.132	213.59.237.157	58408 -> 443	UDP	35 Б			
14:41:01	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:41:01	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:41:01	213.59.237.157	192.168.1.132	443 -> 58408	UDP	1250 Б			
14:41:01	192.168.1.132	213.59.237.157	58408 -> 443	UDP	49 Б			
14:41:01	192.168.1.132	216.58.207.238	56238 -> 443	TCP	517 Б			
14:41:01	149.154.167.41	192.168.1.132	443 -> 57383	TCP	6 Б			
14:41:01	192.168.1.132	149.154.167.41	57383 -> 443	TCP	6 Б			
Стоп								

Рисунок 11 – Тестирование при активном веб-трафике

Программа определила два IP-адреса как P2P с помощью одного метода — TCP/UDP-эвристики. Первый адрес является адресом хостовой машины. На рисунке 12 можно увидеть, что второй адрес относится к браузеру, следовательно, срабатывание было ложным.

TCP-подключения							
Образ	ИД п...	Локальный ад...	Локальный порт	Удаленный адрес	Удал...	Поте...	Заде...
browser.exe	4676	192.168.1.132	56161	213.59.237.157	443	-	-
browser.exe	4676	192.168.1.132	56160	213.59.237.157	443	-	-

Рисунок 12 – Информация об адресе в resmon

В каждом последующем тестовом запуске будет присутствовать активный веб-трафик.

Следующий запуск проводился при загрузке файла через µTorrent, клиент BitTorrent:

Анализатор сетевого трафика									
Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов		IP/Port эвристика
15:23:50	192.168.1.132	37.98.165.140	61547 -> 19520	TCP	6 Б		94.243.211.140:6881	192.168.1.132:55555	
15:23:50	193.233.235.144	192.168.1.132	12383 -> 55555	UDP	896 Б	P2P BitTorrent	94.243.211.176:6881		
15:23:50	193.233.235.144	192.168.1.132	12383 -> 55555	UDP	896 Б	P2P BitTorrent	23.137.251.45:6969		
15:23:50	77.232.12.31	192.168.1.132	6881 -> 61554	TCP	1452 Б	P2P BitTorrent	35.167.186.212:6881		
15:23:50	192.168.1.132	46.43.206.83	61616 -> 6881	TCP	6 Б	P2P BitTorrent	83.221.167.36:6881		
15:23:50	192.168.1.132	217.107.199.166	55555 -> 28634	UDP	181 Б	P2P BitTorrent	13.58.27.33:6881		
15:23:50	192.168.1.132	109.229.119.186	55555 -> 46663	UDP	1103 Б	P2P BitTorrent	208.83.20.20:6969		
15:23:50	193.233.235.144	192.168.1.132	12383 -> 55555	UDP	896 Б	P2P BitTorrent	94.31.98.221:6881		
15:23:50	37.98.165.140	192.168.1.132	19520 -> 61547	TCP	1452 Б		46.43.206.83:6881		
15:23:50	37.98.165.140	192.168.1.132	19520 -> 61547	TCP	1452 Б		213.136.79.7:6881		
15:23:51	192.168.1.132	94.243.211.176	61622 -> 6881	TCP	6 Б	P2P BitTorrent	65.108.201.176:6881		
15:23:51	192.168.1.132	94.243.211.176	61622 -> 6881	TCP	6 Б	P2P BitTorrent	94.140.231.95:6881		
15:23:51	94.243.211.176	192.168.1.132	6881 -> 61622	TCP	7260 Б	P2P BitTorrent	TCP/UDP эвристика		
15:23:51	193.233.235.144	192.168.1.132	12383 -> 55555	UDP	896 Б	P2P BitTorrent	158.255.212.172	По полезной нагрузке	
15:23:51	31.162.156.79	192.168.1.132	62403 -> 55555	UDP	1428 Б	P2P BitTorrent	176.116.164.77		
15:23:51	192.168.1.1	239.255.255.250	57497 -> 1900	UDP	396 Б		178.178.93.183		
15:23:51	192.168.1.1	239.255.255.250	57497 -> 1900	UDP	405 Б		94.243.211.176		
15:23:51	192.168.1.132	77.232.12.31	61554 -> 6881	TCP	6 Б	P2P BitTorrent	46.43.206.83		
15:23:51	192.168.1.1	239.255.255.250	57497 -> 1900	UDP	396 Б		94.230.252.188		
15:23:51	192.168.1.1	239.255.255.250	57497 -> 1900	UDP	468 Б		94.243.211.140		
15:23:51	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б		192.168.1.132		
15:23:51	192.168.1.132	213.59.237.157	62279 -> 443	UDP	44 Б		94.140.231.95		
15:23:52	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б		82.112.28.120		
15:23:52	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б		128.204.69.22		
15:23:52	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б		178.64.119.94		
15:23:52	192.168.1.132	213.59.237.157	62279 -> 443	UDP	48 Б		По метрикам BT		
15:23:52	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б		192.168.1.132:55555	Пересечение методов	
15:23:52	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б				
15:23:52	46.43.206.83	192.168.1.132	6881 -> 61616	TCP	6 Б	P2P BitTorrent			
15:23:52	192.168.1.132	77.232.12.31	61554 -> 6881	TCP	0 Б	P2P BitTorrent			
15:23:52	192.168.1.132	46.43.206.83	61616 -> 6881	TCP	6 Б	P2P BitTorrent			
15:23:52	192.168.1.132	94.243.211.176	61622 -> 6881	TCP	6 Б	P2P BitTorrent			
15:23:52	77.232.12.31	192.168.1.132	6881 -> 61554	TCP	1452 Б	P2P BitTorrent			
15:23:52	77.232.12.31	192.168.1.132	6881 -> 61554	TCP	1452 Б	P2P BitTorrent			
15:23:53	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б				
15:23:53	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б				
15:23:53	213.59.237.157	192.168.1.132	443 -> 62279	UDP	1250 Б				
15:23:53	192.168.1.132	213.59.237.157	62279 -> 443	UDP	35 Б				
Стоп									

Рисунок 13 – Тестирование при загрузке файла через µTorrent

Видно, что было обнаружено множество адресов с помощью всех реализованных методов. В частности, адрес 192.168.1.132:55555 является адресом входящих соединений µTorrent. На рисунке 16 можно увидеть данную настройку. Этот адрес был помечен сразу несколькими методами: по IP/Port-эвристике, полезной нагрузке и метрикам BitTorrent. Не все пиры были обнаружены программой, однако, если сравнить результат работы программы и информацию на рисунках 14 и 15, то можно увидеть, что многие адреса всё же были определены.

Также в выводе информации о трафике появляются пометки в потоках, принадлежащих BitTorrent.

TCP-подключения					
Отфильтровано по: uTorrent.exe					
Образ	ИД п...	Локальный ад...	Локальный порт	Удаленный адрес	Удал...
uTorrent.exe	21276	192.168.1.132	61869	88.147.152.228	3593
uTorrent.exe	21276	192.168.1.132	61854	217.107.199.166	28634
uTorrent.exe	21276	192.168.1.132	61845	178.218.117.97	16180
uTorrent.exe	21276	192.168.1.132	61844	213.136.79.7	6881
uTorrent.exe	21276	192.168.1.132	61839	212.75.103.72	21126
uTorrent.exe	21276	192.168.1.132	61838	62.118.86.104	24962
uTorrent.exe	21276	192.168.1.132	61837	5.141.195.117	58063
uTorrent.exe	21276	192.168.1.132	61836	85.15.85.113	32392
uTorrent.exe	21276	192.168.1.132	61835	85.249.20.192	43739
uTorrent.exe	21276	192.168.1.132	61834	5.18.225.180	1
uTorrent.exe	21276	192.168.1.132	61833	178.140.151.210	33351
uTorrent.exe	21276	192.168.1.132	61832	213.136.79.238	6881
uTorrent.exe	21276	192.168.1.132	61831	176.116.164.77	44459
uTorrent.exe	21276	192.168.1.132	61830	37.76.34.79	27787
uTorrent.exe	21276	192.168.1.132	61829	138.199.6.206	54728
uTorrent.exe	21276	192.168.1.132	61826	178.68.130.57	52933
uTorrent.exe	21276	192.168.1.132	61825	92.248.185.121	60232
uTorrent.exe	21276	192.168.1.132	61824	178.69.144.38	15311
uTorrent.exe	21276	192.168.1.132	61823	84.22.146.55	10861
uTorrent.exe	21276	192.168.1.132	61822	176.212.104.35	1
uTorrent.exe	21276	192.168.1.132	61821	128.204.69.22	52831
uTorrent.exe	21276	192.168.1.132	61820	188.37.234.47	28861
uTorrent.exe	21276	192.168.1.132	61819	91.207.170.127	28071
uTorrent.exe	21276	192.168.1.132	61818	79.139.195.101	17998
uTorrent.exe	21276	192.168.1.132	61817	185.195.233.245	51413
uTorrent.exe	21276	192.168.1.132	61816	178.64.34.65	14441
uTorrent.exe	21276	192.168.1.132	61814	85.140.6.210	29179
uTorrent.exe	21276	192.168.1.132	61813	95.210.97.11	42071
uTorrent.exe	21276	192.168.1.132	61811	78.107.207.248	49882
uTorrent.exe	21276	192.168.1.132	61622	94.243.211.176	6881
uTorrent.exe	21276	192.168.1.132	61621	104.21.31.24	443
uTorrent.exe	21276	192.168.1.132	61616	46.43.206.83	6881
uTorrent.exe	21276	192.168.1.132	61554	77.232.12.31	6881
uTorrent.exe	21276	192.168.1.132	61547	37.98.165.140	19520
uTorrent.exe	21276	Петлевой адр...	3395	Петлевой адрес в IPv4	61644

Рисунок 14 – Информация о TCP-подключениях клиента µTorrent в resmon

Файлы		Информация	Пир	Трекер
IP				Порт
31.162.156.79 [uTP]				62403
254C224F.nat.pool.telekom.hu [uTP]				27787
ip140.net2-165.ivn.ttksever.ru				19520
46-43-206-83.achinsk.net				6881
77.232.12.31				6881
85.175.216.224 [uTP]				27367
94.140.231.95 [uTP]				53944
94.243.211.176				6881
186-119-229-109.broadband.telenettv.ru [uTP]				46663
193.233.235.144 [uTP]				12383
dynamic-nat4.lipetsk.zelenaya.net [uTP]				28634

Рисунок 15 – Активные пиры клиента µTorrent

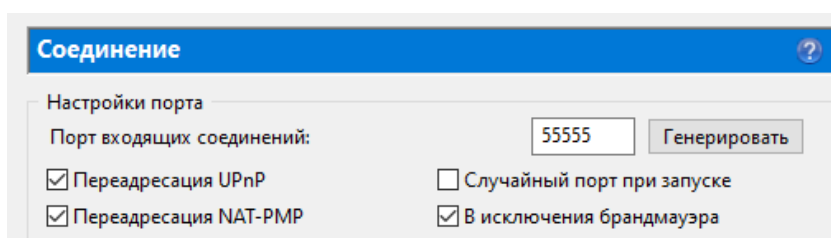


Рисунок 16 – Настройка порта входящих соединений клиента µTorrent

Аналогичные результаты были получены при загрузке файла с помощью браузерного BitTorrent клиента µTorrent Web:

Анализатор сетевого трафика									
Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов		IP/Port эвристика
17:18:50	83.69.10.162	192.168.1.132	51413 -> 56834	TCP	6 Б		81.161.213.149:6881		192.168.1.132:6881
17:18:50	94.140.137.178	192.168.1.132	16953 -> 64484	UDP	1068 Б	P2P BitTorrent	109.94.21.0:6881		192.168.1.132:64484
17:18:50	91.108.27.7	192.168.1.132	51413 -> 56660	TCP	2904 Б		88.86.76.18:6881		
17:18:50	31.31.26.87	192.168.1.132	51413 -> 64484	UDP	1402 Б	P2P BitTorrent	187.46.183.62:6881		
17:18:50	109.94.21.0	192.168.1.132	6881 -> 64484	UDP	147 Б	P2P BitTorrent	45.150.67.207:6969		
17:18:50	91.108.27.7	192.168.1.132	51413 -> 56660	TCP	2904 Б		188.18.216.172:6881		
17:18:50	185.13.112.25	192.168.1.132	42977 -> 64484	UDP	1359 Б	P2P BitTorrent	188.235.168.171:6882		
17:18:51	83.69.10.162	192.168.1.132	51413 -> 56834	TCP	1436 Б		37.192.73.107:6881		
17:18:51	192.168.1.132	83.69.10.162	56834 -> 51413	TCP	6 Б		92.94.157.97:6881		
17:18:51	83.69.10.162	192.168.1.132	51413 -> 56834	TCP	1436 Б		37.19.38.96:6881		
17:18:51	192.168.1.132	83.69.10.162	56834 -> 51413	TCP	6 Б		78.30.235.17:6881		
17:18:51	192.168.1.132	194.37.1.217	56708 -> 51413	TCP	6 Б		46.180.243.226:6881		
17:18:51	194.37.1.217	192.168.1.132	51413 -> 56708	TCP	1072 Б		TCP/UDP эвристика		По полезной нагрузке
17:18:51	31.31.26.87	192.168.1.132	51413 -> 64484	UDP	1402 Б	P2P BitTorrent	91.108.27.7		192.168.1.132:64484
17:18:51	94.45.199.247	192.168.1.132	6881 -> 64484	UDP	938 Б	P2P BitTorrent	178.140.35.127		195.164.225.138:62203
17:18:51	31.31.26.87	192.168.1.132	51413 -> 64484	UDP	1402 Б	P2P BitTorrent	142.250.74.174		54.164.44.3:80
17:18:51	185.13.112.25	192.168.1.132	42977 -> 64484	UDP	20 Б	P2P BitTorrent	192.168.1.132		192.168.1.132:56659
17:18:51	136.169.120.70	192.168.1.132	6881 -> 64484	UDP	26 Б	P2P BitTorrent	89.179.78.234		92.94.157.97:6881
17:18:51	192.168.1.132	136.169.120.70	64484 -> 6881	UDP	20 Б	P2P BitTorrent			192.168.1.132:6881
17:18:51	94.140.137.178	192.168.1.132	16953 -> 64484	UDP	1068 Б	P2P BitTorrent			192.168.1.132:58262
17:18:52	94.45.199.247	192.168.1.132	6881 -> 64484	UDP	655 Б	P2P BitTorrent	По метрикам ВТ		Пересечение методов
17:18:52	192.168.1.132	94.45.199.247	64484 -> 6881	UDP	20 Б	P2P BitTorrent	192.168.1.132:6881		54.164.44.3:80
17:18:52	78.30.235.17	192.168.1.132	6881 -> 64484	UDP	20 Б	P2P BitTorrent	192.168.1.132:64484		92.94.157.97:6881
17:18:52	78.30.235.17	192.168.1.132	6881 -> 64484	UDP	461 Б	P2P BitTorrent	54.164.44.3:80		192.168.1.132:64484
17:18:52	185.13.112.25	192.168.1.132	42977 -> 64484	UDP	1359 Б	P2P BitTorrent			192.168.1.132:6881
17:18:52	192.168.1.132	185.13.112.25	64484 -> 42977	UDP	37 Б	P2P BitTorrent	По метрикам ВТ		Пересечение методов
17:18:52	178.166.157.68	192.168.1.132	27538 -> 64484	UDP	20 Б	P2P BitTorrent	192.168.1.132:6881		54.164.44.3:80
17:18:52	45.140.24.84	192.168.1.132	6881 -> 64484	UDP	20 Б	P2P BitTorrent	192.168.1.132:64484		92.94.157.97:6881
17:18:52	192.168.1.132	91.108.27.7	56660 -> 51413	TCP	6 Б		54.164.44.3:80		192.168.1.132:64484
17:18:52	192.168.1.132	91.108.27.7	56660 -> 51413	TCP	6 Б				192.168.1.132:6881
17:18:52	178.214.245.126	192.168.1.132	30044 -> 64484	UDP	1464 Б	P2P BitTorrent			
17:18:53	37.204.64.76	192.168.1.132	6882 -> 64484	UDP	1464 Б	P2P BitTorrent			
17:18:53	37.204.64.76	192.168.1.132	6882 -> 64484	UDP	1464 Б	P2P BitTorrent			
17:18:53	178.214.245.126	192.168.1.132	30044 -> 64484	UDP	1464 Б	P2P BitTorrent			
17:18:53	192.168.1.132	185.244.46.69	64484 -> 6881	UDP	20 Б	P2P BitTorrent			
17:18:53	185.244.46.69	192.168.1.132	6881 -> 64484	UDP	1464 Б	P2P BitTorrent			
17:18:53	188.35.4.12	192.168.1.132	30683 -> 64484	UDP	20 Б	P2P BitTorrent			
17:18:53	188.35.4.12	192.168.1.132	30683 -> 64484	UDP	707 Б	P2P BitTorrent			

Рисунок 17 – Тестирование при загрузке файла через µTorrent Web

TCP-подключения					
Отфильтровано по: utweb.exe					
Образ	ИД п...	Локальный ад...	Локальный порт	Удаленный адрес	Удал...
utweb.exe	14856	192.168.1.132	56680	178.66.231.209	6262
utweb.exe	14856	192.168.1.132	57538	46.48.190.132	7920
utweb.exe	14856	192.168.1.132	56913	31.132.230.114	46613
utweb.exe	14856	192.168.1.132	57386	89.39.19.60	6881
utweb.exe	14856	192.168.1.132	57113	5.138.210.206	51017
utweb.exe	14856	192.168.1.132	56821	146.70.179.36	446
utweb.exe	14856	192.168.1.132	57702	146.70.161.196	446
utweb.exe	14856	192.168.1.132	57366	149.102.244.22	446
utweb.exe	14856	192.168.1.132	56748	146.70.161.170	446
utweb.exe	14856	192.168.1.132	57482	77.220.34.82	8685
utweb.exe	14856	192.168.1.132	56708	194.37.1.217	51413
utweb.exe	14856	192.168.1.132	56762	146.70.179.29	446
utweb.exe	14856	192.168.1.132	56794	154.47.24.197	446
utweb.exe	14856	192.168.1.132	56796	95.78.76.55	37884
utweb.exe	14856	192.168.1.132	56763	146.70.179.21	446
utweb.exe	14856	192.168.1.132	56761	146.70.179.55	446
utweb.exe	14856	192.168.1.132	57200	95.210.97.11	42071
utweb.exe	14856	192.168.1.132	56538	89.179.78.234	59753
utweb.exe	14856	192.168.1.132	58268	185.107.44.95	446
utweb.exe	14856	192.168.1.132	56857	154.47.24.195	446
utweb.exe	14856	192.168.1.132	57561	109.174.125.208	60527
utweb.exe	14856	192.168.1.132	56840	185.107.80.107	446
utweb.exe	14856	192.168.1.132	56695	89.36.76.137	446
utweb.exe	14856	192.168.1.132	56717	85.206.163.146	446
utweb.exe	14856	192.168.1.132	56706	185.107.44.150	446
utweb.exe	14856	192.168.1.132	56798	185.107.56.87	446
utweb.exe	14856	192.168.1.132	56754	193.29.107.249	446
utweb.exe	14856	192.168.1.132	56750	87.250.15.50	51417
utweb.exe	14856	192.168.1.132	58488	92.255.182.137	51413
utweb.exe	14856	192.168.1.132	56807	95.181.112.83	6881
utweb.exe	14856	192.168.1.132	56834	83.69.10.162	51413
utweb.exe	14856	192.168.1.132	57616	128.68.65.166	51413
utweb.exe	14856	192.168.1.132	56805	91.122.45.192	9091
utweb.exe	14856	192.168.1.132	57073	93.100.112.71	16881
utweb.exe	14856	192.168.1.132	56660	91.108.27.7	51413
utweb.exe	14856	192.168.1.132	57582	178.140.35.127	15500
utweb.exe	14856	192.168.1.132	58438	176.15.240.53	51413
utweb.exe	14856	192.168.1.132	58431	176.107.14.7	51413
utweb.exe	14856	192.168.1.132	57475	78.107.232.234	16881

Рисунок 18 – Информация о TCP-подключениях клиента µTorrent Web в resmon

utweb.exe	14856	192.168.1.132	57475	78.107.232.234	16881	0	23
utweb.exe	14856	192.168.1.132	57582	178.140.35.127	15500	0	21
utweb.exe	14856	192.168.1.132	58524	31.204.101.12	37360	0	18
utweb.exe	14856	192.168.1.132	59021	88.147.152.228	3681	-	-
utweb.exe	14856	192.168.1.132	58930	91.116.3.252	17921	-	-
utweb.exe	14856	192.168.1.132	58966	188.186.32.58	4878	-	-
utweb.exe	14856	192.168.1.132	58961	185.237.216.107	6881	-	-
utweb.exe	14856	192.168.1.132	58694	109.254.254.18	62561	-	-
utweb.exe	14856	192.168.1.132	58766	94.19.198.44	51413	-	-
utweb.exe	14856	192.168.1.132	58826	46.188.25.62	51413	-	-
utweb.exe	14856	192.168.1.132	58805	185.82.244.114	51413	-	-
utweb.exe	14856	192.168.1.132	58707	178.208.76.6	49675	-	-
utweb.exe	14856	192.168.1.132	58702	188.243.182.95	40218	-	-
utweb.exe	14856	192.168.1.132	58765	109.95.73.241	38639	-	-
utweb.exe	14856	192.168.1.132	58764	93.178.86.96	25399	-	-
utweb.exe	14856	192.168.1.132	56792	90.188.239.170	24777	0	-
utweb.exe	14856	192.168.1.132	57424	213.171.50.78	23264	0	-
utweb.exe	14856	192.168.1.132	58666	89.109.48.131	21315	-	-
utweb.exe	14856	192.168.1.132	58665	109.126.211.132	21305	-	-
utweb.exe	14856	192.168.1.132	58647	37.144.33.129	19751	-	-
utweb.exe	14856	192.168.1.132	58701	178.66.156.195	15044	-	-
utweb.exe	14856	192.168.1.132	58693	111.44.237.161	15000	-	-
utweb.exe	14856	192.168.1.132	58726	196.196.53.110	6881	-	-

Рисунок 19 – Информация о TCP-подключениях клиента µTorrent Web в resmon

Прослушиваемые порты					
Отфильтровано по: utweb.exe					
Образ	ИД п...	Адрес	Порт	Протокол	Состояни...
utweb.exe	14856	IPv6 не задан	64487	UDP	Разреше...
utweb.exe	14856	IPv4 не задан	64486	UDP	Разреше...
utweb.exe	14856	IPv6 не задан	64485	UDP	Разреше...
utweb.exe	14856	IPv4 не задан	64484	UDP	Разреше...
utweb.exe	14856	26.163.144.113	64483	UDP	Разреше...
utweb.exe	14856	fe80::8a7:dfb9:a7...	64482	UDP	Разреше...
utweb.exe	14856	2001:0:284a:364:8...	64481	UDP	Разреше...
utweb.exe	14856	fe80::470c:91a1:3...	64479	UDP	Разреше...
utweb.exe	14856	fe80::df1a:3f4b:7...	64478	UDP	Разреше...
utweb.exe	14856	fe80::31b1:bb1b:...	64477	UDP	Разреше...
utweb.exe	14856	fe80::ec76:1edc:e...	64476	UDP	Разреше...
utweb.exe	14856	fe80::caf:7319:550...	64475	UDP	Разреше...
utweb.exe	14856	fe80::ed39:ddfe:d...	64474	UDP	Разреше...
utweb.exe	14856	fdfd::1aa3:9071	64473	UDP	Разреше...
utweb.exe	14856	Петлевой адрес ...	64472	UDP	Разреше...
utweb.exe	14856	192.168.1.132	64471	UDP	Разреше...
utweb.exe	14856	192.168.56.1	64470	UDP	Разреше...
utweb.exe	14856	26.163.144.113	64469	UDP	Разреше...
utweb.exe	14856	IPv4 не задан	19577	TCP	Разреше...
utweb.exe	14856	IPv4 не задан	19576	TCP	Разреше...
utweb.exe	14856	IPv4 не задан	19575	TCP	Разреше...
utweb.exe	14856	fe80::ed39:ddfe:d...	6881	TCP	Разреше...
utweb.exe	14856	fe80::ec76:1edc:e...	6881	TCP	Разреше...
utweb.exe	14856	fe80::df1a:3f4b:7...	6881	TCP	Разреше...
utweb.exe	14856	fe80::8a7:dfb9:a7...	6881	TCP	Разреше...
utweb.exe	14856	fdfd::1aa3:9071	6881	TCP	Разреше...
utweb.exe	14856	2001:0:284a:364:8...	6881	TCP	Разреше...
utweb.exe	14856	IPv4 не задан	6881	TCP	Разреше...
utweb.exe	14856	fe80::ed39:ddfe:d...	6881	UDP	Разреше...
utweb.exe	14856	fe80::ec76:1edc:e...	6881	UDP	Разреше...
utweb.exe	14856	fe80::df1a:3f4b:7...	6881	UDP	Разреше...
utweb.exe	14856	fe80::8a7:dfb9:a7...	6881	UDP	Разреше...

Рисунок 20 – Информация о прослушиваемых портах клиента µTorrent Web в resmon

Программа успешно обнаруживает работу клиента Bitcoin Core:

Анализатор сетевого трафика								
Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов	IP/Port эвристика
18:06:53	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:53	192.168.1.132	93.186.225.198	56971 -> 443	TCP	6 Б			
18:06:53	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:53	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:53	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	0 Б	P2P Bitcoin		
18:06:53	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:53	192.168.1.132	8.212.50.206	57709 -> 8333	TCP	0 Б			
18:06:53	5.9.5.171	192.168.1.132	8333 -> 57634	TCP	6 Б	P2P Bitcoin		
18:06:54	192.168.1.132	5.9.5.171	57634 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:54	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	4356 Б	P2P Bitcoin		
18:06:54	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:54	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:54	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	2904 Б	P2P Bitcoin	TCP/UDP эвристика	По полезной нагрузке
18:06:54	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:54	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:54	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	2904 Б	P2P Bitcoin		
18:06:54	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	2904 Б	P2P Bitcoin		
18:06:54	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:55	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:55	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:55	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:55	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:55	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin	По метрикам ВТ	Пересечение методов
18:06:55	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	2904 Б	P2P Bitcoin		
18:06:55	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:55	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:55	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	6 Б	P2P Bitcoin		
18:06:55	185.31.136.246	192.168.1.132	8333 -> 57633	TCP	24 Б	P2P Bitcoin		
18:06:55	185.31.136.246	192.168.1.132	8333 -> 57633	TCP	8712 Б	P2P Bitcoin		
18:06:56	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	0 Б	P2P Bitcoin		
18:06:56	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	135 Б	P2P Bitcoin		
18:06:56	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	0 Б	P2P Bitcoin		
18:06:56	160.3.194.234	192.168.1.132	8333 -> 57701	TCP	1452 Б	P2P Bitcoin		
18:06:56	192.168.1.132	160.3.194.234	57701 -> 8333	TCP	0 Б	P2P Bitcoin		
18:06:56	192.168.1.132	162.254.198.104	56552 -> 27035	TCP	54 Б			

Рисунок 21 – Тестирование при работе Bitcoin Core

TCP-подключения					
Отфильтровано по: bitcoin-qt.exe					
Образ	ИД п...	Локальный ад...	Локальный порт	Удаленный адрес	Удал...
bitcoin-qt.exe	15712	192.168.1.132	57692	133.18.241.193	8333
bitcoin-qt.exe	15712	192.168.1.132	57702	3.37.69.205	8333
bitcoin-qt.exe	15712	192.168.1.132	57701	160.3.194.234	8333
bitcoin-qt.exe	15712	192.168.1.132	57707	45.79.192.236	8333
bitcoin-qt.exe	15712	192.168.1.132	57686	157.245.140.69	8333
bitcoin-qt.exe	15712	192.168.1.132	57664	188.83.134.205	8333
bitcoin-qt.exe	15712	192.168.1.132	57693	212.14.60.54	8333
bitcoin-qt.exe	15712	192.168.1.132	57634	5.9.5.171	8333
bitcoin-qt.exe	15712	192.168.1.132	57694	62.210.6.33	8333
bitcoin-qt.exe	15712	192.168.1.132	57708	38.242.147.45	8333
bitcoin-qt.exe	15712	192.168.1.132	57633	185.31.136.246	8333
bitcoin-qt.exe	15712	192.168.1.132	57713	171.104.220.219	8333
bitcoin-qt.exe	15712	192.168.1.132	57709	8.212.50.206	8333
bitcoin-qt.exe	15712	192.168.1.132	57705	73.188.229.220	8333
bitcoin-qt.exe	15712	192.168.1.132	57699	146.70.42.148	8333
bitcoin-qt.exe	15712	192.168.1.132	57703	97.126.119.222	39388
bitcoin-qt.exe	15712	192.168.1.132	57697	167.172.26.145	39388
bitcoin-qt.exe	15712	Петлевой адр...	57629	Петлевой адрес в IPv4	57628
bitcoin-qt.exe	15712	Петлевой адр...	57628	Петлевой адрес в IPv4	57629
bitcoin-qt.exe	15712	Петлевой адр...	57631	Петлевой адрес в IPv4	57630
bitcoin-qt.exe	15712	Петлевой адр...	57630	Петлевой адрес в IPv4	57631

Рисунок 22 – Информация о TCP-подключениях клиента Bitcoin Core

Skype успешно детектируется во время звонка при помощи анализа портов и TCP/UDP-эвристики:

Анализатор сетевого трафика								
Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов	IP/Port эвристика
18:19:19	192.168.1.132	185.32.251.54	56986 -> 443	TCP	6 Б		20.101.67.88:3478	
18:19:19	185.32.251.54	192.168.1.132	443 -> 56986	TCP	6 Б		20.202.144.11:3478	
18:19:19	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	104 Б	P2P Skype	20.202.1.45:3478	
18:19:20	192.168.1.118	224.0.0.7	8001 -> 8001	UDP	200 Б		20.202.147.83:3478	
18:19:20	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	116 Б	P2P Skype	155.133.252.39:27025	
18:19:20	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	72 Б	P2P Skype	20.91.206.115:3480	
18:19:20	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	56 Б	P2P Skype		
18:19:20	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	1232 Б	P2P Skype		
18:19:20	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	1232 Б	P2P Skype		
18:19:20	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	1232 Б	P2P Skype		
18:19:20	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	1232 Б	P2P Skype		
18:19:20	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	96 Б	P2P Skype		
18:19:21	192.168.1.132	213.180.204.179	65143 -> 443	TCP	6 Б			
18:19:21	213.180.204.179	192.168.1.132	443 -> 65143	TCP	0 Б			
18:19:21	192.168.1.132	192.168.1.1	62771 -> 53	UDP	44 Б			
18:19:21	192.168.1.1	192.168.1.132	53 -> 62771	UDP	341 Б			
18:19:21	192.168.1.132	146.158.48.5	56930 -> 443	TCP	81 Б			
18:19:21	192.168.1.132	146.158.48.5	56930 -> 443	TCP	6 Б			
18:19:21	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	76 Б	P2P Skype		
18:19:21	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	172 Б	P2P Skype		
18:19:21	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	116 Б	P2P Skype		
18:19:21	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	44 Б	P2P Skype		
18:19:21	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	1220 Б	P2P Skype		
18:19:22	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	1220 Б	P2P Skype		
18:19:22	192.168.1.132	20.101.67.88	53518 -> 3478	UDP	1032 Б	P2P Skype		
18:19:22	20.101.67.88	192.168.1.132	3478 -> 53518	UDP	18 Б	P2P Skype		
18:19:22	20.101.67.88	192.168.1.132	3478 -> 53518	UDP	37 Б	P2P Skype		
18:19:22	20.101.67.88	192.168.1.132	3478 -> 53518	UDP	360 Б	P2P Skype		
18:19:22	192.168.1.132	20.101.67.88	53518 -> 3478	UDP	18 Б	P2P Skype		
18:19:22	20.101.67.88	192.168.1.132	3478 -> 53518	UDP	18 Б	P2P Skype		
18:19:22	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	104 Б	P2P Skype		
18:19:22	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	116 Б	P2P Skype		
18:19:22	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	72 Б	P2P Skype		
18:19:23	20.91.206.115	192.168.1.132	3480 -> 11534	UDP	88 Б	P2P Skype		
18:19:23	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	104 Б	P2P Skype		
18:19:23	192.168.1.132	87.240.190.90	57012 -> 443	TCP	6 Б			
18:19:23	87.240.190.90	192.168.1.132	443 -> 57012	TCP	6 Б			
18:19:23	192.168.1.132	20.91.206.115	11534 -> 3480	UDP	56 Б	P2P Skype		
							TCP/UDP эвристика	По полезной нагрузке
							192.168.1.132	
							96.16.49.209	
							20.101.67.88	
							По метрикам ВТ	Пересечение методов

Рисунок 23 – Тестирование при работе Skype

Сетевая активность					
■ 1 кбит/с - сетевой ввод-вывод					
Отфильтровано по: Skype.exe, Skype.exe					
Образ	ИД п...	Адрес	Отправле...	Получен...	Всего (ба...
Skype.exe	10772	192.168.1.132	0	975	975
Skype.exe	10772	20.189.173.9	114	242	356
Skype.exe	8300	13.83.65.43	665	174	839
Skype.exe	8300	20.189.173.14	227	120	346
Skype.exe	10772	137.135.225.146	11	62	73
Skype.exe	10772	20.101.67.88	56	20	76
Skype.exe	8300	13.107.226.45	49	14	63
Skype.exe	10772	20.91.206.115	1 938	0	1 938
Skype.exe	10772	20.202.144.11	10	0	10
Skype.exe	10772	20.202.147.83	3	0	3
Skype.exe	10772	20.202.147.239	3	0	3
TCP-подключения					
Отфильтровано по: Skype.exe, Skype.exe					
Образ	ИД п...	Локальный ад...	Локальный порт	Удаленный адрес	Удал...
Skype.exe	8300	192.168.1.132	57803	13.107.4.52	80
Skype.exe	8300	192.168.1.132	57801	13.83.65.43	443
Skype.exe	10772	192.168.1.132	57808	137.135.225.146	443
Skype.exe	10772	192.168.1.132	57846	20.189.173.9	443
Skype.exe	8300	192.168.1.132	57833	13.83.65.43	443
Skype.exe	8300	192.168.1.132	57832	13.83.65.43	443
Skype.exe	10772	192.168.1.132	57810	13.107.3.128	443

Рисунок 24 – Информация о TCP-подключениях и прослушиваемых портах Skype

Для следующего теста была создана общая папка на хостовой машине. Доступ к ней был открыт в локальной сети.

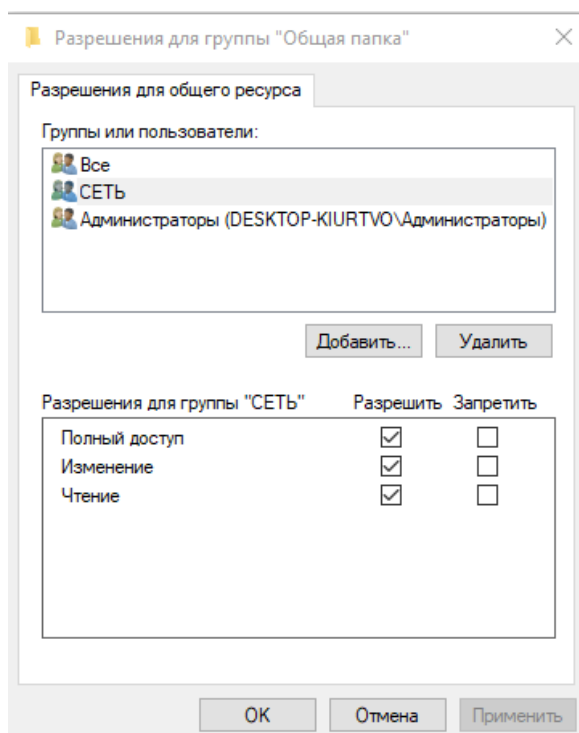


Рисунок 25 – Настройки доступа к папке

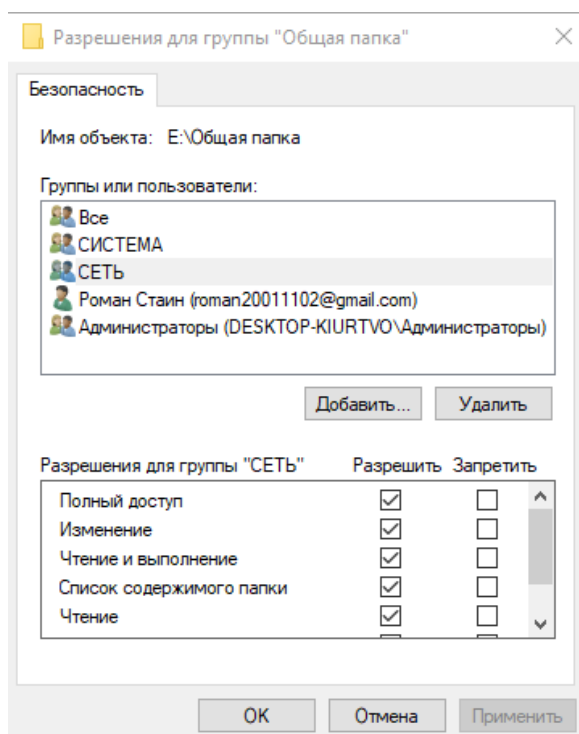


Рисунок 26 – Настройки безопасности папки

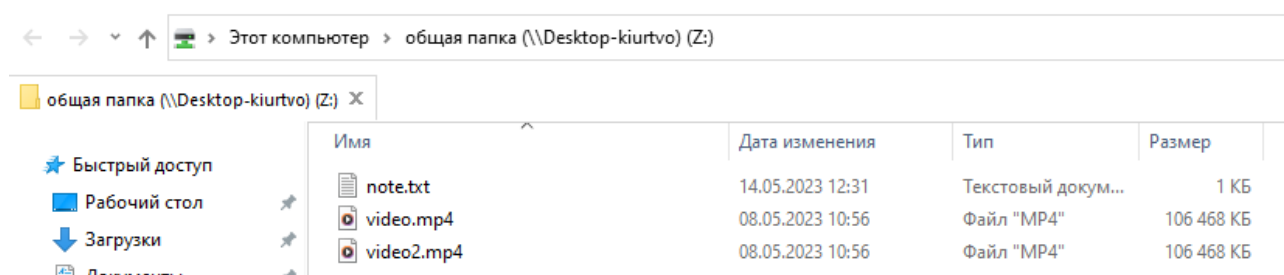


Рисунок 27 – Просмотр содержимого общей папки на хостовой машине

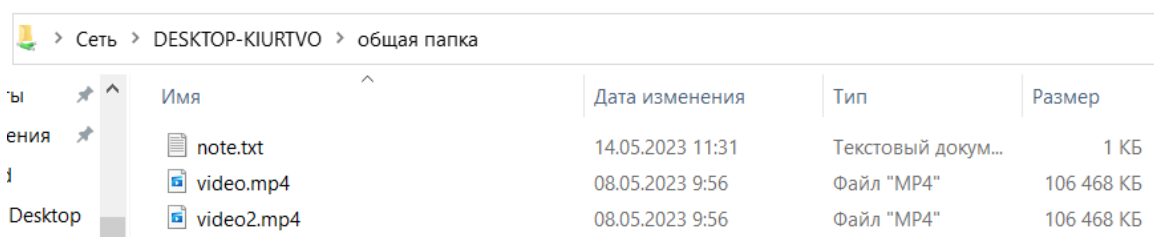


Рисунок 28 – Просмотр содержимого общей папки на другой машине в локальной сети

С одной машины на другую передавался видеофайл размером 103 МБ. На рисунке 29 можно увидеть, что передаются пакеты между двумя локальными адресами: 192.168.1.132 и 192.168.1.142. В итоге программа не определила данную передачу как P2P.

Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов	IP/Port эвристика
12:33:20	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:20	192.168.1.142	192.168.1.132	54880 -> 445	TCP	8760 Б			
12:33:21	192.168.1.142	192.168.1.132	54880 -> 445	TCP	7300 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:21	192.168.1.142	192.168.1.132	54880 -> 445	TCP	2920 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:21	192.168.1.142	192.168.1.132	54880 -> 445	TCP	8760 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:21	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:21	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б		TCP/UDP эвристика	По полезной нагрузке
12:33:21	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:21	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:22	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:22	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	7300 Б			
12:33:22	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	5840 Б			
12:33:22	192.168.1.132	192.168.1.142	445 -> 54880	TCP	504 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	0 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:22	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б		По метрикам BT	Пересечение методов
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:23	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	0 Б			
12:33:23	192.168.1.142	192.168.1.132	54880 -> 445	TCP	5840 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	0 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:23	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:23	192.168.1.142	192.168.1.132	54880 -> 445	TCP	1460 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			
12:33:23	192.168.1.142	192.168.1.132	54880 -> 445	TCP	6 Б			
12:33:23	192.168.1.132	192.168.1.142	445 -> 54880	TCP	6 Б			

Рисунок 29 – Результат работы программы

Последнее тестирование проводилось не на хостовой машине, а на виртуальной Kali Linux. Выполнение команды `apt-get update`, которая обновляет базу данных доступных пакетов, программа ложно определила как P2P с помощью метрик BitTorrent:

Время	Источник	Назначение	Порты	Протокол	Длина	Инфо	Анализ портов	IP/Port эвристика
12:13:06	108.157.214.49	192.168.1.132	443 -> 64282	TCP	193 Б	P2P BitTorrent	155.133.226.78:27018	
12:13:06	108.157.214.49	192.168.1.132	443 -> 64282	TCP	6 Б	P2P BitTorrent		
12:13:06	108.157.214.49	192.168.1.132	443 -> 64282	TCP	6 Б	P2P BitTorrent		
12:13:06	192.168.1.132	108.157.214.49	64283 -> 443	TCP	721 Б	P2P BitTorrent		
12:13:06	108.157.214.49	192.168.1.132	443 -> 64283	TCP	6 Б	P2P BitTorrent		
12:13:06	108.157.214.49	192.168.1.132	443 -> 64283	TCP	645 Б	P2P BitTorrent		
12:13:06	108.157.214.49	192.168.1.132	443 -> 64283	TCP	6 Б	P2P BitTorrent		
12:13:06	192.168.1.132	87.240.185.163	58661 -> 443	TCP	6 Б			
12:13:07	192.168.1.132	87.240.185.137	58620 -> 443	TCP	6 Б			
12:13:07	108.157.214.49	192.168.1.132	443 -> 64284	TCP	633 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent	TCP/UDP эвристика	По полезной нагрузке
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent	По метрикам BT	Пересечение методов
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:07	108.157.214.49	192.168.1.132	443 -> 64285	TCP	6 Б	P2P BitTorrent		
12:13:08	87.240.185.132	192.168.1.132	443 -> 58637	TCP	0 Б		108.157.214.49:443 108.157.214.63:443	
12:13:08	192.168.1.132	93.186.227.153	58667 -> 443	TCP	6 Б			
12:13:08	87.240.185.132	192.168.1.132	443 -> 58631	TCP	0 Б			
12:13:08	108.157.214.49	192.168.1.132	443 -> 64288	TCP	6 Б	P2P BitTorrent		
12:13:08	108.157.214.49	192.168.1.132	443 -> 64288	TCP	6 Б	P2P BitTorrent		
12:13:08	192.168.1.132	108.157.214.49	64289 -> 443	TCP	0 Б	P2P BitTorrent		
12:13:08	192.168.1.118	224.0.0.7	8001 -> 8001	UDP	200 Б			
12:13:08	192.168.1.132	87.250.251.20	61688 -> 443	TCP	6 Б			
12:13:08	108.157.214.49	192.168.1.132	443 -> 64289	TCP	6 Б	P2P BitTorrent		
12:13:08	192.168.1.132	108.157.214.49	64289 -> 443	TCP	6 Б	P2P BitTorrent		
12:13:08	108.157.214.49	192.168.1.132	443 -> 64289	TCP	6 Б	P2P BitTorrent	108.157.214.49:443 108.157.214.63:443	
12:13:09	108.157.214.49	192.168.1.132	443 -> 64289	TCP	6 Б	P2P BitTorrent		
12:13:09	192.168.1.132	108.157.214.49	64289 -> 443	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		
12:13:09	108.157.214.49	192.168.1.132	443 -> 64290	TCP	6 Б	P2P BitTorrent		

Рисунок 30 – Результат работы программы

Данный метод сработал, поскольку для двух адресов, которые можно увидеть на рисунке 30, максимальное значение метрики C (количество подключений) было 51 и метрики $BiAT$ (двусторонние передачи) 38, что больше пороговых значений.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены теоретические сведения о технологии P2P: особенности её архитектуры, применение и способы обнаружения, которые, так или иначе, имеют некоторую степень погрешности. Вместе с тем, были приведены характерные черты такого протокола как BitTorrent, который является одним из самых распространённых среди P2P-сетей. Поэтому обнаружение BitTorrent можно считать наиболее востребованным.

В практической части была реализована программа — сниффер или анализатор сетевого трафика на языке Python, которая позволяет перехватывать TCP и UDP трафик и анализировать его на присутствие P2P-активности, а также определять некоторые протоколы и приложения. Были реализованы методы анализа портов, обнаружения TCP/UDP- и IP/Port-эвристики, анализ полезной нагрузки для BitTorrent и Bitcoin и пороговый метод сравнения характерных метрик BitTorrent.

Таким образом, изучение P2P-сетей несомненно является актуальным, поскольку они активно используются пользователями Интернета, в следствие чего не останавливается и их развитие. Однако иногда необходимо фильтровать и блокировать P2P-трафик, поэтому необходимо также быстро развивать методы его обнаружения, которые могут устаревать со временем. P2P-протоколы меняют своё поведение, могут использовать случайные номера портов, изменять сигнатуры. Кроме того, многие другие сетевые протоколы могут иметь схожее поведение, поэтому крайне важно различать их между собой, обновлять способы исключения таких протоколов. Из-за множества подобных факторов не существует универсального способа обнаружения P2P-трафика. Тем не менее, есть необходимое количество узконаправленных методов, которые в совокупности с достаточной точностью могут определить P2P-активность в сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 P2P (Peer-to-Peer) [Электронный ресурс]. — URL: [https://web.archive.org/web/20171205204322/http://ru.bmstu.wiki/P2P_\(Peer-to-Peer\)](https://web.archive.org/web/20171205204322/http://ru.bmstu.wiki/P2P_(Peer-to-Peer)) (Дата обращения 19.05.2023). Загл. с экр. Яз. рус.
- 2 P2P [Электронный ресурс]. — URL: https://glebradchenko.susu.ru/courses/bachelor/odp/2013/SUSU_Distr_11_P2P.pdf (Дата обращения 14.05.2023). Загл. с экр. Яз. рус.
- 3 Bhatia, M. Multi-level p2p traffic classification using heuristic and statistical-based techniques: A hybrid approach / M. Bhatia, V. Sharma, P. Singh, M. Masud // Symmetry. — 12 2020. — Vol. 12. — P. 2117.
- 4 Karagiannis, T. A longitudinal study of p2p traffic classification / T. Karagiannis, K. Papagiannaki, M. Faloutsos // Proc. of 14th IEEE International Symposium on Modeling, Analysis, and Simulation. — 2005.
- 5 Karagiannis, T. BLINC: multilevel traffic classification in the dark / T. Karagiannis, K. Papagiannaki, M. Faloutsos. — 2005. — Pp. 229–240.
- 6 Karagiannis, T. Transport layer identification of P2P traffic / T. Karagiannis, A. Broido, M. Faloutsos, K. C. Claffy. — 2004. — Pp. 121–134.
- 7 Ngiwlay, W. Bittorrent peer identification based on behaviors of a choke algorithm / W. Ngiwlay, C. Intanagonwiwat, Y. Teng-amnuay // Association for Computing Machinery. — 2008.
- 8 Бредихин, С. В. Диагностика p2p-АКТИВНОСТИ на основе анализа потоков netflow / С. В. Бредихин, Н. Г. Щербакова // Проблемы информатики. — 2012. — № 1. — С. 40–51.
- 9 Desclaux, F. Vanilla skype part 1 / F. Desclaux, K. Kortchinsky // Proc of RECON2006. — 2006.
- 10 BitTorrentSpecification - TheoryOrg. [Электронный ресурс]. — URL: <https://wiki.theory.org/BitTorrentSpecification> (Дата обращения 19.05.2023). Загл. с экр. Яз. англ.
- 11 Protocol documentation - Bitcoin Wiki. [Электронный ресурс]. — URL: https://en.bitcoin.it/wiki/Protocol_documentation (Дата обращения 19.05.2023). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А

Листинг main.py

```
1  #!/usr/bin/env python3
2  import tkinter as tk
3  from tkinter import ttk
4  import socket
5  import sniffer
6  from datetime import datetime
7  import os
8  import sys
9  import netifaces as ni
10 import select
11
12 TAB_2 = ' \t * '
13 SCAN_RATE_S = 0.075
14 SCAN_RATE_MS = int(SCAN_RATE_S * 1000)
15
16
17 class Menu(tk.Frame):
18     def __init__(self, master):
19         super().__init__(master)
20         self.master = master
21         self.grid(row=0, column=0, sticky=tk.NSEW)
22         self.last_time = ''
23         self.output_list = []
24         self.conn = None
25         self.osflag = None
26
27         self.frame_choose_interface = ttk.Frame(self, width=150, height=75)
28         self.frame_choose_interface.grid(row=0, column=0)
29
30         self.label_choose_interface = ttk.Label(self.frame_choose_interface,
31                                                  text='Выберите интерфейс для
32                                                  ↪ прослушивания')
33
34         self.label_choose_interface.grid(row=0, column=0)
35
36         self.loi_columns = ['1', '2']
37         self.list_of_interfaces = ttk.Treeview(self.frame_choose_interface,
38                                                show='headings',
39                                                ↪ columns=self.loi_columns,
40                                                ↪ height=10)
```

```

37     self.list_of_interfaces.heading('1', text='IP-адрес')
38     self.list_of_interfaces.heading('2', text='Интерфейс')
39     self.list_of_interfaces.grid(row=1, column=0)
40
41     for inter in inters_ips:
42         self.list_of_interfaces.insert(parent='', index='end',
43             ↪ values=[inter, inters_ips[inter]])
44
45     self.list_of_interfaces.bind('<Double-1>', self.start)
46
47     self.frame_main = ttk.Frame(self)
48
49     self.columns = ['1', '2', '3', '4', '5', '6', '7']
50     self.output = ttk.Treeview(self.frame_main, show='headings',
51         ↪ columns=self.columns, height=38)
52     self.output.heading('1', text='Время')
53     self.output.heading('2', text='Источник')
54     self.output.heading('3', text='Назначение')
55     self.output.heading('4', text='Порты')
56     self.output.heading('5', text='Протокол')
57     self.output.heading('6', text='Длина')
58     self.output.heading('7', text='Инфо')
59
60     self.output.column('1', minwidth=0, width=65)
61     self.output.column('2', minwidth=0, width=120)
62     self.output.column('3', minwidth=0, width=120)
63     self.output.column('4', minwidth=0, width=125)
64     self.output.column('5', minwidth=0, width=77)
65     self.output.column('6', minwidth=0, width=60)
66     self.output.column('7', minwidth=0, width=180)
67
68     self.output.tag_configure("highlight", background="#FCA89F")
69
70     # Таблицы P2P адресов
71     self.frame = ttk.Frame(self.frame_main)
72     self.p2p_table_1 = ttk.Treeview(self.frame, show='headings',
73         ↪ columns=['1'], height=12)
74     self.p2p_table_2 = ttk.Treeview(self.frame, show='headings',
75         ↪ columns=['2'], height=12)
76     self.p2p_table_3 = ttk.Treeview(self.frame, show='headings',
77         ↪ columns=['3'], height=12)

```

```

73     self.p2p_table_4 = ttk.Treeview(self.frame, show='headings',
    ↪     columns=['4'], height=12)
74     self.p2p_table_5 = ttk.Treeview(self.frame, show='headings',
    ↪     columns=['5'], height=12)
75     self.p2p_table_6 = ttk.Treeview(self.frame, show='headings',
    ↪     columns=['6'], height=12)
76
77     self.p2p_table_1.heading('1', text='Анализ портов')
78     self.p2p_table_2.heading('2', text='IP/Port эвристика')
79     self.p2p_table_3.heading('3', text='TCP/UDP эвристика')
80     self.p2p_table_4.heading('4', text='По полезной нагрузке')
81     self.p2p_table_5.heading('5', text='По метрикам BT')
82     self.p2p_table_6.heading('6', text='Пересечение методов')
83
84     self.p2p_table_1.column('1', minwidth=0, width=175)
85     self.p2p_table_2.column('2', minwidth=0, width=175)
86     self.p2p_table_3.column('3', minwidth=0, width=175)
87     self.p2p_table_4.column('4', minwidth=0, width=175)
88     self.p2p_table_5.column('5', minwidth=0, width=175)
89     self.p2p_table_6.column('6', minwidth=0, width=175)
90
91     self.scroll_out = ttk.Scrollbar(self.frame_main,
    ↪     command=self.output.yview)
92     self.output.config(yscrollcommand=self.scroll_out.set)
93
94     self.stop_btn = ttk.Button(self.frame_main, text='Cmon',
    ↪     command=self.stop)
95
96     def start(self, _):
97         select = self.list_of_interfaces.selection()[0]
98         item = self.list_of_interfaces.item(select)
99         interface = item['values'][1]
100        self.conn, self.osflag = create_socket(interface)
101        self.frame_choose_interface.forget()
102
103        self.frame_main.grid(row=0, column=0)
104        self.output.grid(row=0, column=0, padx=(5, 0), sticky=tk.NW)
105
106        self.frame.grid(row=0, column=1)
107        self.p2p_table_1.grid(row=0, column=0, padx=(5, 0), sticky=tk.NE)
108        self.p2p_table_2.grid(row=0, column=1, padx=(0, 5), sticky=tk.NE)

```

```

109         self.p2p_table_3.grid(row=1, column=0, padx=(5, 0), sticky=tk.NE)
110         self.p2p_table_4.grid(row=1, column=1, padx=(0, 5), sticky=tk.NE)
111         self.p2p_table_5.grid(row=2, column=0, padx=(5, 0), sticky=tk.NE)
112         self.p2p_table_6.grid(row=2, column=1, padx=(0, 5), sticky=tk.NE)
113
114         self.stop_btn.grid(row=1, column=0, pady=(10, 10))
115
116         self.call_sniff()
117         self.call_find_p2p()
118         self.call_bt_stats()
119
120         # Авто пролистывание до последней строки при прокручивании колеса мыши
121         ↪ вниз
122         def auto_down_scroll(self):
123             last_row = self.output.get_children()[-1]
124             last_row_bbox = self.output.bbox(last_row)
125
126             if len(last_row_bbox) > 0:
127                 self.output.see(last_row)
128
129         def call_sniff(self):
130             ready = select.select([self.conn], [], [], SCAN_RATE_S)
131             if ready[0]:
132                 out = sniffer.sniff(self.conn, self.osflag)
133                 if out:
134                     time = str(datetime.now().strftime('%H:%M:%S'))
135                     out.insert(0, time)
136                     self.output_list.append(out)
137                     self.output.insert(parent='', index='end', values=out)
138                     # Подсветка
139                     # if out[-1][0:3] == "P2P":
140                     #     self.output.insert(parent='', index='end', values=out,
141                     ↪ tags=("highlight",))
142                     # else:
143                     #     self.output.insert(parent='', index='end', values=out)
144                     self.auto_down_scroll()
145
146                     # Вывод информации о пакете
147                     for s in out:
148                         file.write(s + ' ')
149                     file.write('\n')

```

```

148
149     root.after(SCAN_RATE_MS, self.call_sniff) # сканирование каждые 0.1
        ↪ сек
150
151     def call_find_p2p(self):
152         sniffer.find_p2p()
153
154         for item_id in self.p2p_table_1.get_children():
155             self.p2p_table_1.delete(item_id)
156         for addr in sniffer.p2p_pairs_p:
157             self.p2p_table_1.insert(parent='', index='end', values=[addr[0] +
        ↪ ":" + str(addr[1])])
158
159         for item_id in self.p2p_table_2.get_children():
160             self.p2p_table_2.delete(item_id)
161         for addr in sniffer.p2p_pairs_ipp:
162             self.p2p_table_2.insert(parent='', index='end', values=[addr[0] +
        ↪ ":" + str(addr[1])])
163
164         for item_id in self.p2p_table_3.get_children():
165             self.p2p_table_3.delete(item_id)
166         for addr in sniffer.p2p_addrs_tu:
167             self.p2p_table_3.insert(parent='', index='end', values=[addr])
168
169         for item_id in self.p2p_table_4.get_children():
170             self.p2p_table_4.delete(item_id)
171         for addr in sniffer.bittorrent_addrs:
172             self.p2p_table_4.insert(parent='', index='end', values=[addr[0] +
        ↪ ":" + str(addr[1])])
173         for addr in sniffer.bitcoin_addrs:
174             self.p2p_table_4.insert(parent='', index='end', values=[addr[0] +
        ↪ ":" + str(addr[1])])
175
176         intersection = set()
177         intersection = intersection | (sniffer.p2p_pairs_p
178                                     & (sniffer.p2p_pairs_ipp |
        ↪ sniffer.bittorrent_addrs
179                                     | sniffer.bitcoin_addrs |
        ↪ sniffer.bittorrent_addrs2))
180
181         intersection = intersection | (sniffer.p2p_pairs_ipp

```

```

182                                     & (sniffer.p2p_pairs_p |
                                     ↪ sniffer.bittorrent_addrs
183                                     | sniffer.bitcoin_addrs |
                                     ↪ sniffer.bittorrent_addrs2))

184
185 intersection = intersection | ((sniffer.bittorrent_addrs |
186 ↪ sniffer.bitcoin_addrs)
187                                     & (sniffer.p2p_pairs_p |
                                     ↪ sniffer.p2p_pairs_ipp |
                                     ↪ sniffer.bittorrent_addrs2))

188 intersection = intersection | (sniffer.bittorrent_addrs2
189 ↪ (sniffer.p2p_pairs_p |
190 ↪ sniffer.bittorrent_addrs
191 ↪ sniffer.bitcoin_addrs |
192 ↪ sniffer.p2p_pairs_ipp))

193 for item_id in self.p2p_table_6.get_children():
194     self.p2p_table_6.delete(item_id)
195 for addr in intersection:
196     self.p2p_table_6.insert(parent='', index='end', values=[addr[0] +
197 ↪ ":" + str(addr[1])])

198
199 root.after(15000, self.call_find_p2p)
200
201 def call_bt_stats(self):
202     for ipp in sniffer.dict_ippport:
203         sniffer.dict_ippport[ipp].bt_stats()
204
205     for item_id in self.p2p_table_5.get_children():
206         self.p2p_table_5.delete(item_id)
207     for addr in sniffer.bittorrent_addrs2:
208         self.p2p_table_5.insert(parent='', index='end', values=[addr[0] +
209 ↪ ":" + str(addr[1])])

210
211 root.after(30000, self.call_bt_stats)
212
213 def stop(self):
214     file2.write('Список IP-адресов, взаимодействующих через P2P: \n ')
215     file2.write('Анализ портов: \n ')
216     for row in self.p2p_table_1.get_children():

```



```

214         addr = self.p2p_table_1.item(row) ['values'] [0]
215         file2.write(' * ' + addr + '\n')
216
217     file2.write('IP/Port-эвристика: \n')
218     for row in self.p2p_table_2.get_children():
219         addr = self.p2p_table_2.item(row) ['values'] [0]
220         file2.write(' * ' + addr + '\n')
221
222     file2.write('TCP/UDP-эвристика: \n')
223     for row in self.p2p_table_3.get_children():
224         addr = self.p2p_table_3.item(row) ['values'] [0]
225         file2.write(' * ' + addr + '\n')
226
227     file2.write('По полезной нагрузке: \n')
228     for row in self.p2p_table_4.get_children():
229         addr = self.p2p_table_4.item(row) ['values'] [0]
230         file2.write(' * ' + addr + '\n')
231
232     file2.write('По метрикам Bittorrent: \n')
233     for row in self.p2p_table_5.get_children():
234         addr = self.p2p_table_5.item(row) ['values'] [0]
235         file2.write(' * ' + addr + '\n')
236
237     file2.write('Пересечение методов: \n')
238     for row in self.p2p_table_6.get_children():
239         addr = self.p2p_table_6.item(row) ['values'] [0]
240         file2.write(' * ' + addr + '\n')
241
242     file2.write('Конец списка. \n')
243
244     file2.write('\nСписок исключений P2P-адресов: \n')
245     for pair in sniffer.rejected:
246         file2.write(' * ' + pair[0] + ':' + str(pair[1]) + '\n')
247
248     self.conn.close()
249     file2.close()
250     file.close()
251     root.destroy()
252
253
254     def create_socket(interface):

```

```

255     try:
256         # Windows needs IP ?
257         if os.name == 'nt':
258             osflag = False
259             conn = socket.socket(socket.AF_INET, socket.SOCK_RAW,
260                                  ↪ socket.IPPROTO_IP)
261             conn.bind((interface, 0))
262             conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
263             conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
264             conn.setblocking(False)
265             # conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
266
267         # Linux needs interface's name
268         else:
269             osflag = True
270
271             if len(sys.argv) > 1:
272                 interface = sys.argv[1]
273                 os.system("ip link set {} promisc on".format(interface)) # ret =
274                 conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
275                                       ↪ socket.ntohs(3))
276                 conn.bind((interface, 0))
277                 conn.setblocking(False)
278             return conn, osflag
279         except socket.error as msg:
280             print('Сокет не может быть создан. Код ошибки : ' + str(msg[0]) + '
281                   ↪ Сообщение ' + msg[1])
282             sys.exit()
283
284     # Расшифровка названия интерфейса на Windows
285     def get_connection_name_from_guid(iface_guids):
286         iface_names = ['(unknown)' for i in range(len(iface_guids))]
287         reg = wr.ConnectRegistry(None, wr.HKEY_LOCAL_MACHINE)
288         reg_key = wr.OpenKey(reg,
289                              ↪ r'SYSTEM\CurrentControlSet\Control\Network\{4d36e972-e325-11ce-bfc1-08002be10
290         for i in range(len(iface_guids)):
291             try:
292                 reg_subkey = wr.OpenKey(reg_key, iface_guids[i] + r'\Connection')
293                 iface_names[i] = wr.QueryValueEx(reg_subkey, 'Name')[0]
294             except FileNotFoundError:

```

```

292         pass
293     return iface_names
294
295
296 # For Linux
297 def get_local_interfaces():
298     import array
299     import struct
300     import fcntl
301     """ Returns a dictionary of name:ip key value pairs. """
302     MAX_BYTES = 4096
303     FILL_CHAR = b '\0 '
304     SIOCGIFCONF = 0x8912
305     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
306     names = array.array('B', MAX_BYTES * FILL_CHAR)
307     names_address, names_length = names.buffer_info()
308     mutable_byte_buffer = struct.pack('iL', MAX_BYTES, names_address)
309     mutated_byte_buffer = fcntl.ioctl(sock.fileno(), SIOCGIFCONF,
        ↪ mutable_byte_buffer)
310     max_bytes_out, names_address_out = struct.unpack('iL',
        ↪ mutated_byte_buffer)
311     namestr = names.tobytes()
312     namestr[:max_bytes_out]
313     bytes_out = namestr[:max_bytes_out]
314     ip_dict = {}
315     for i in range(0, max_bytes_out, 40):
316         name = namestr[i: i + 16].split(FILL_CHAR, 1)[0]
317         name = name.decode('utf-8')
318         ip_bytes = namestr[i+20:i+24]
319         full_addr = []
320         for netaddr in ip_bytes:
321             if isinstance(netaddr, int):
322                 full_addr.append(str(netaddr))
323             elif isinstance(netaddr, str):
324                 full_addr.append(str(ord(netaddr)))
325         # ip_dict[name] = '.'.join(full_addr)
326         ip_dict['.'.join(full_addr)] = name # я сделал наоборот, потому что
        ↪ для линукса у меня нужно имя, а не айпи
327
328     return ip_dict
329

```

```

330
331 if __name__ == "__main__":
332     # Получение списка интерфейсов и их IP
333     if os.name == 'nt':
334         osflag = False
335         import winreg as wr
336
337         interfaces = []
338         ips = []
339
340         x = ni.interfaces()
341         for interface in x:
342             addr = ni.ifaddresses(interface)
343             try:
344                 ip = addr[ni.AF_INET][0]['addr']
345                 interfaces.append(interface)
346                 ips.append(ip)
347             except:
348                 pass
349         interfaces = get_connection_name_from_guid(interfaces)
350         inters_ips = dict(zip(interfaces, ips))
351
352     else:
353         osflag = True
354         inters_ips = get_local_interfaces()
355
356         # interfaces = ['enp6s0']
357         # ips = ['192.168.1.132']
358
359
360     #
361     ↪ print(ni.ifaddresses(_get_default_iface_linux()).setdefault(ni.AF_INET)[0]['a
362     # print(ni.interfaces())
363
364     # В файл сохраняется последний вывод программы
365     file = open('out.txt', 'w+')
366     # Список IP-адресов, взаимодействующих через P2P
367     file2 = open('ip_list.txt', 'w+')
368
369     root = tk.Tk()
370     root.title("Анализатор сетевого трафика")

```

```
370     menu = Menu(root)
371     root.mainloop()
```

ПРИЛОЖЕНИЕ Б

Листинг sniffer.py

```
1 import socket
2 import struct
3
4 # Отступы для вывода информации
5 TAB_1 = '\t - '
6
7 # Список пар порт-приложение
8 LIST_P2P = {6881: 'BitTorrent', 6882: 'BitTorrent', 6883: 'BitTorrent',
9             6884: 'BitTorrent', 6885: 'BitTorrent', 6886: 'BitTorrent',
10            6887: 'BitTorrent', 6888: 'BitTorrent', 6889: 'BitTorrent',
11            6969: 'BitTorrent', 411: 'Direct Connect', 412: 'Direct Connect',
12            # 2323: 'eDonkey', 3306: 'eDonkey', 4242: 'eDonkey',
13            # 4500: 'eDonkey', 4501: 'eDonkey', 4677: 'eDonkey',
14            # 4678: 'eDonkey', 4711: 'eDonkey', 4712: 'eDonkey',
15            # 7778: 'eDonkey', 1214: 'FastTrack', 1215: 'FastTrack',
16            # 1331: 'FastTrack', 1337: 'FastTrack', 1683: 'FastTrack',
17            # 4329: 'FastTrack', 5000: 'Yahoo', 5001: 'Yahoo',
18            # 5002: 'Yahoo', 5003: 'Yahoo', 5004: 'Yahoo', 5005: 'Yahoo',
19            # 5006: 'Yahoo', 5007: 'Yahoo', 5008: 'Yahoo', 5009: 'Yahoo',
20            # 5010: 'Yahoo', 5050: 'Yahoo', 5100: 'Yahoo', 5555: 'Napster',
21            # 6257: 'Napster', 6666: 'Napster', 6677: 'Napster',
22            # 6688: 'Napster', 6699: 'Napster', 6700: 'Napster',
23            # 6701: 'Napster', 6346: 'Gnutella', 6347: 'Gnutella', 5190:
24            ↪ 'AIM',
25            3478: 'Skype', 3479: 'Skype', 3480: 'Skype', 3481: 'Skype',
26            4379: 'Steam', 4380: 'Steam (voice chat)', 27014: 'Steam',
27            27015: 'Steam', 27016: 'Steam', 27017: 'Steam', 27018: 'Steam',
28            27019: 'Steam', 27020: 'Steam', 27021: 'Steam', 27022: 'Steam',
29            27023: 'Steam', 27024: 'Steam', 27025: 'Steam', 27026: 'Steam',
30            27027: 'Steam', 27028: 'Steam', 27029: 'Steam', 27030: 'Steam',
31            899: 'Radmin VPN', 12975: 'Hamachi', 32976: 'Hamachi'}
32
33 # Список портов исключений
34 EXCEPTIONS = {137, 138, 139, 445, 53, 123, 500, 554, 1900, 7070,
35              6970, 1755, 5000, 5001, 6112, 6868, 6899, 6667, 7000, 7514,
36              20, 21, 3396, 66, 1521, 1526, 1524, 22, 23, 25, 513, 543}
37
38 C_threshold = 20
39 RAT_threshold = 0.35
```

```

39 BIAT_threshold = 5
40 RRC_threshold = 0.5
41
42 TCP_addrs = set()
43 UDP_addrs = set()
44 p2p_addrs_tu = set() # адреса, взаимодействующие одновременно по TCP и UDP
45 p2p_pairs_p = set() # адреса, порт которых входит в список P2P-портов
46 p2p_pairs_ipp = set() # адреса, подходящие к IPPort эвристике
47 rejected = set() # адреса, не относящиеся к P2P (исключения)
48 dict_ipport = dict() # словарь вида (ip+port -> объект класса IPPort)
49
50 bittorrent_addrs = set() # адреса, относящиеся к BitTorrent
51 bittorrent_addrs2 = set() # адреса, относящиеся к BitTorrent, обнаруженные по
    ↪ метрикам
52 bitcoin_addrs = set() # адреса, относящиеся к Bitcoin
53 bitcoin_phrases = ['version', 'verack', 'addr', 'inv', 'getdata',
    ↪ 'notfound', 'getblocks',
54                     'getheaders', 'tx', 'block', 'headers', 'getaddr',
    ↪ 'mempool', 'checkorder',
55                     'submitorder', 'reply', 'ping', 'pong', 'reject',
    ↪ 'filterload', 'filteradd',
56                     'filterclear', 'merkleblock', 'alert', 'sendheaders',
    ↪ 'feefilter',
57                     'sendcmpct', 'cmpctlblock', 'getblocktxn', 'blocktxn',
    ↪ 'Satoshi']
58
59
60 class IPPort:
61     def __init__(self, dst_ip, dst_port):
62         self.dst_ip = dst_ip
63         self.dst_port = dst_port
64         self.IPSet = set() # IP-адреса источников
65         self.PortSet = set() # Порты источников
66         self.srcs_addrs = set()
67         self.in_packets = dict()
68         self.dest_addrs = set()
69         self.old_bi = set()
70         self.rc = 0
71
72     def add_sources(self, ip, port):
73         self.IPSet.add(ip)

```

```

74         self.PortSet.add(port)
75         self.srcs_addrs.add((ip, port))
76
77     def add_packets(self, src_addr, size):
78         if src_addr in self.in_packets.keys():
79             self.in_packets[src_addr].append(size)
80         else:
81             self.in_packets[src_addr] = [size]
82
83     def add_out_addrs(self, dest_addr):
84         self.dest_addrs.add(dest_addr)
85
86     # Добавление адресов, которые взаимодействовали с адресами из p2p_addrs_tu
87     def add_to_p2p_addrs1(self):
88         for ip in self.IPSet:
89             if ip not in [iport[0] for iport in rejected]:
90                 p2p_addrs_tu.add('(' + ip)
91
92     def bt_stats(self):
93         # 1
94         c = len(self.srcs_addrs)
95         self.srcs_addrs = set()
96
97         # 2
98         at = 0
99         for addr in self.in_packets:
100             packets = self.in_packets[addr]
101             pack_size = len(packets)
102             if pack_size > 4:
103                 average_size = 0
104                 for p in packets:
105                     average_size += p
106                 average_size /= pack_size
107                 if average_size > 1375:
108                     at += 1
109
110         # 3
111         bi = self.in_packets.keys() & self.dest_addrs
112
113         # 4
114         # self.rc = 0 # ?

```



```

115         if len(bi) > len(self.old_bi):
116             self.rc += len(bi - self.old_bi)
117         else:
118             self.rc += len(self.old_bi - bi)
119
120     self.old_bi = bi
121     # Проверка граничных значений
122     if (self.dst_ip, self.dst_port) not in rejected:
123         if c >= C_threshold:
124             print('C', c, self.dst_ip, self.dst_port)
125             bittorrent_addrs2.add((self.dst_ip, self.dst_port))
126         elif len(bi) >= BIAT_threshold:
127             print('bi', len(bi), self.dst_ip, self.dst_port)
128             bittorrent_addrs2.add((self.dst_ip, self.dst_port))
129         # Следующие метрики связываются с метрикой C для большей точности
130         elif c >= C_threshold / 2:
131             if at / c >= RAT_threshold:
132                 print('at / c', at / c, self.dst_ip, self.dst_port)
133                 bittorrent_addrs2.add((self.dst_ip, self.dst_port))
134             elif at > 0:
135                 if self.rc / at >= RRC_threshold:
136                     print('rc / at', self.rc / at, self.dst_ip,
137                           ↪ self.dst_port)
138                     bittorrent_addrs2.add((self.dst_ip, self.dst_port))
139
140 def sniff(conn, os):
141     output = ''
142     data, addr = conn.recvfrom(65536)
143     if os:
144         dest_mac, src_mac, eth_proto, data = ethernet_frame(data)
145     else:
146         eth_proto = 8
147
148     # IPv4
149     if eth_proto == 8:
150         version, header_length, ttl, proto, src, dest, data =
151             ↪ ipv4_packet(data)
152
153     if proto == 6 or proto == 17:

```

```

154         # TCP
155         if proto == 6:
156             src_port, dest_port, data = tcp_segment(data)
157
158             check_exceptions(src, dest, src_port, dest_port)
159             if (src, src_port) not in rejected and (dest, dest_port) not
160                 ↪ in rejected:
161                 TCP_addrs.add((src, dest))
162
163             addition_info = add_info(src, dest, src_port, dest_port)
164             output = [src, dest, str(src_port) + ' -> ' + str(dest_port),
165                 ↪ 'TCP', str(len(data)) + ' Б',
166                     addition_info]
167
168         # UDP
169         else:
170             src_port, dest_port, length, data = udp_segment(data)
171
172             check_exceptions(src, dest, src_port, dest_port)
173             if (src, src_port) not in rejected and (dest, dest_port) not
174                 ↪ in rejected:
175                 UDP_addrs.add((src, dest))
176
177             addition_info = add_info(src, dest, src_port, dest_port)
178             output = [src, dest, str(src_port) + ' -> ' + str(dest_port),
179                 ↪ 'UDP', str(len(data)) + ' Б',
180                     addition_info]
181
182         add_ipport(dest, dest_port, src, src_port, len(data))
183         payload_analysis(src, dest, src_port, dest_port, data)
184
185     return output
186
187     # после проверки портов функция
188     # добавляет к строке вывода информацию для столбца info,
189     # если адрес p2p и добавляет протокол по возможности
190     def add_info(src, dest, src_port, dest_port):
191         addition_info = ''
192         if LIST_P2P.get(src_port, False):
193             p2p_pairs_p.add((src, src_port))

```

```

191         addition_info = 'P2P ' + LIST_P2P[src_port]
192     elif LIST_P2P.get(dest_port, False):
193         p2p_pairs_p.add((dest, dest_port))
194         addition_info = 'P2P ' + LIST_P2P[dest_port]
195     elif (src, src_port) in bittorrent_addrs | bittorrent_addrs2:
196         addition_info = 'P2P BitTorrent'
197     elif (dest, dest_port) in bittorrent_addrs | bittorrent_addrs2:
198         addition_info = 'P2P BitTorrent'
199     elif (src, src_port) in bitcoin_addrs:
200         addition_info = 'P2P Bitcoin'
201     elif (dest, dest_port) in bitcoin_addrs:
202         addition_info = 'P2P Bitcoin'
203     return addition_info
204
205
206 def add_ipport(dest, dest_port, src, src_port, size):
207     ipport = dest + ':' + str(dest_port)
208     if ipport not in dict_ipport:
209         x = IPPort(dest, dest_port)
210         x.add_sources(src, src_port)
211         dict_ipport[ipport] = x
212         x.add_packets(src + ':' + str(src_port), size)
213     else:
214         dict_ipport[ipport].add_sources(src, src_port)
215         dict_ipport[ipport].add_packets(src + ':' + str(src_port), size)
216
217     ipport_src = src + ':' + str(src_port)
218     if ipport_src in dict_ipport:
219         dict_ipport[ipport_src].add_out_addrs(ipport)
220
221
222 # Добавление адресов с портами в список исключений
223 def check_exceptions(src, dest, src_port, dest_port):
224     if src_port in EXCEPTIONS \
225         or dest_port in EXCEPTIONS \
226         or (src_port == dest_port and src_port < 500):
227         rejected.add((src, src_port))
228         rejected.add((dest, dest_port))
229
230
231 # Анализ полезной нагрузки пакетов,

```

```

232 def payload_analysis(src, dest, src_port, dest_port, data):
233     if (src, src_port) not in rejected and (dest, dest_port) not in rejected:
234         sdata = str(data)
235         if len(data) >= 20:
236             if 'BitTorrent protocol' in sdata:
237                 bittorrent_addrs.add((src, src_port))
238                 bittorrent_addrs.add((dest, dest_port))
239             elif src_port == 8333 or dest_port == 8333 or src_port == 8334 or
↳ dest_port == 8334:
240                 for word in bitcoin_phrases:
241                     if word in sdata:
242                         bitcoin_addrs.add((src, src_port))
243                         bitcoin_addrs.add((dest, dest_port))
244                         break
245
246
247 def find_p2p():
248     # 1 Заполнение p2p_addrs адресами, взаимодействующими одновременно по TCP
↳ и UDP
249     inter = TCP_addrs & UDP_addrs
250     for addrs in inter:
251         p2p_addrs_tu.add(addrs[0])
252         p2p_addrs_tu.add(addrs[1])
253
254     # 2 Заполнение p2p_pairs_ipp адресами, выбранными исходя из check_p2p
255     for ippport in dict_ippport:
256         ipp = dict_ippport[ippport]
257
258         ip = ipp.dst_ip
259         port = ipp.dst_port
260
261         # Добавление адресов, взаимодействующие с адресами из TCP/UDP пар
262         # if ip in p2p_addrs_tu:
263         #     ipp.add_to_p2p_addrs1()
264
265         compare_dif = 2
266
267         # Если порт из известных p2p портов, то разница должна быть увеличена
↳ до 10
268         if ippport in p2p_pairs_p:
269             compare_dif = 10

```

```

270
271     cur_dif = len(ipp.IPSet) - len(ipp.PortSet)
272     if len(ipp.IPSet) > 2 and (cur_dif < compare_dif):
273         if (ip, port) not in rejected:
274             p2p_pairs_ipp.add((ip, port))
275
276     # Если разница больше 10, то, скорее всего, это не р2р и можно
277     → добавить в исключения.
278     elif cur_dif > 10:
279         rejected.add((ip, port))
280
281 # Распаковка ethernet кадра
282 def ethernet_frame(data):
283     dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
284     return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto),
285     → data[14:]
286
287 # Форматирование MAC-адреса
288 def get_mac_addr(bytes_addr):
289     bytes_str = map('{:02x}'.format, bytes_addr)
290     return ':'.join(bytes_str).upper()
291
292
293 # Распаковка IPv4 пакета
294 def ipv4_packet(data):
295     version_header_length = data[0]
296     version = version_header_length >> 4
297     header_length = (version_header_length & 15) * 4
298     ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
299     return version, header_length, ttl, proto, ipv4(src), ipv4(target),
300     → data[header_length:]
301
302 # Форматирование IP-адреса
303 def ipv4(addr):
304     return '.'.join(map(str, addr))
305
306
307 # Распаковка TCP сегмента

```

```

308 def tcp_segment(data):
309     (src_port, dest_port, _, _, offset_reserved_flags) = struct.unpack('! H H
    ↪  L L H', data[:14])
310     offset = (offset_reserved_flags >> 12) * 4
311     # flag_urg = (offset_reserved_flags & 32) >> 5
312     # flag_ack = (offset_reserved_flags & 16) >> 5
313     # flag_psh = (offset_reserved_flags & 8) >> 5
314     # flag_rst = (offset_reserved_flags & 4) >> 5
315     # flag_syn = (offset_reserved_flags & 2) >> 5
316     # flag_fin = offset_reserved_flags & 1
317     return src_port, dest_port, data[offset:]
318
319
320 # Распаковка UDP сегмента
321 def udp_segment(data):
322     src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
323     return src_port, dest_port, size, data[8:]

```