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 12
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 16
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
6 Тестирование 27
ЗАКЛЮЧЕНИЕ 39
Приложение А Код `main.py` 40
Приложение Б Код `sniffer.py` 49

ВВЕДЕНИЕ

С развитием Интернета развивались файлообменные сети, благодаря которым появилась **P2P** (**p**eer-**to**-**p**eer) — 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. [**?**]

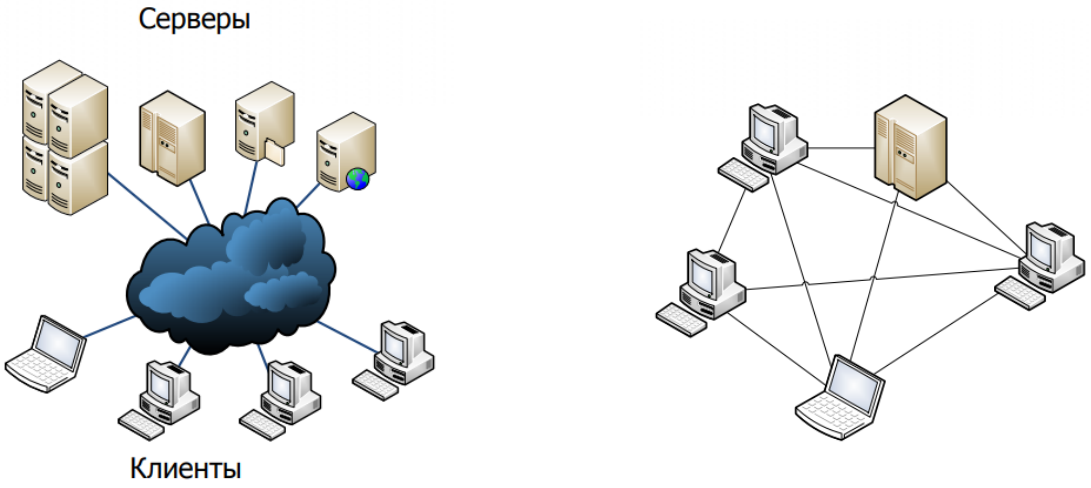
В мае 1999 года, в Интернет с более чем миллионом пользователей, Шон Фэннинг внедрил приложение файлообменник Napster. Napster стал началом P2P-сети, такой какую мы знаем её сейчас, пользователи участвуют в создании виртуальной сети, полностью независимой от физической, без администрирования и каких-либо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



а) б)

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. [**?**]

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.
- Публикация и распространение программного обеспечения (Linux, видеоигры).

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 [**?**].

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

Протоколов	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, 7778
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 [**?**] приведены результаты, которые показывают, что зачастую на основе данного метода можно определить лишь 30% P2P трафика.

Особенности данного метода:

— Необходимо постоянное обновление базы сигнатур.

— Трафик зачастую зашифрован, что сильно затрудняет анализ.

— Поиск сигнатур на прикладном сетевом уровне очень ресурсоёмкий.

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 **ССЫЛКИ НА НИХ** 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 A 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 именно IP-адрес и порт нового хоста А остальным участникам сети. Рисунки 2 и 3 демонстрируют этот процесс.

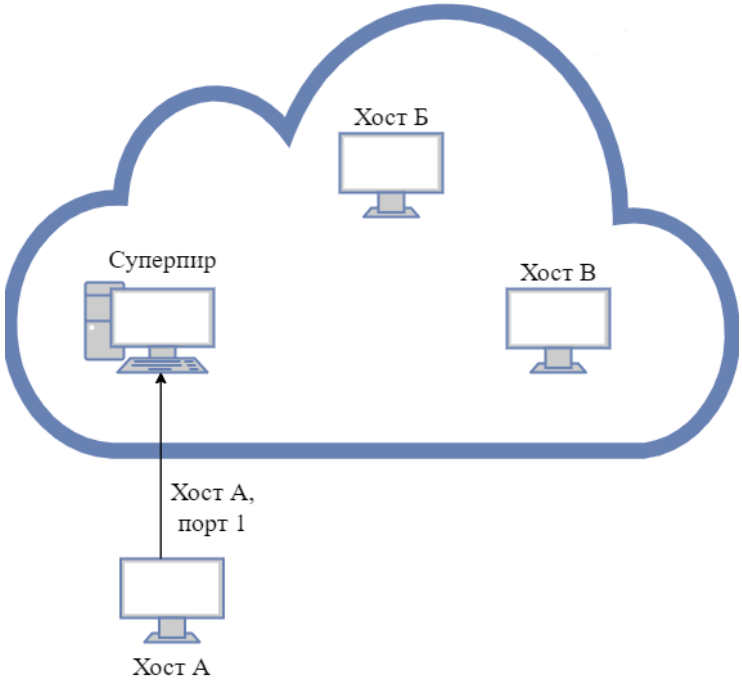


Рисунок 2 – Отправка информации хоста А о себе суперпиру

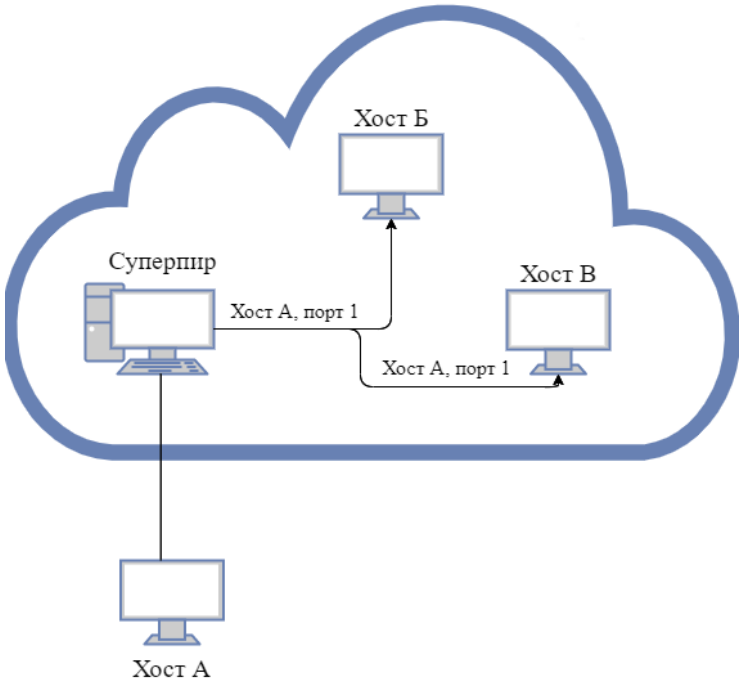


Рисунок 3 – Суперпир распространяет информацию о хосте А остальной части сети

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 A, порт назначения будет портом, который прослушивает хост A, 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 A. Каждый из них выберет временный порт источника и подключится к объявленному порту, который прослушивает хост A. Таким образом, объявленная пара IP-адреса и порта хоста A будет связана с 20 различным IP-адресами и 20 различным портами. Таким образом, для пары хоста A количество различных IP-адресов и различных портов, используемых для подключения к нему, будет равно. Рисунок 4 иллюстрирует данный случай.

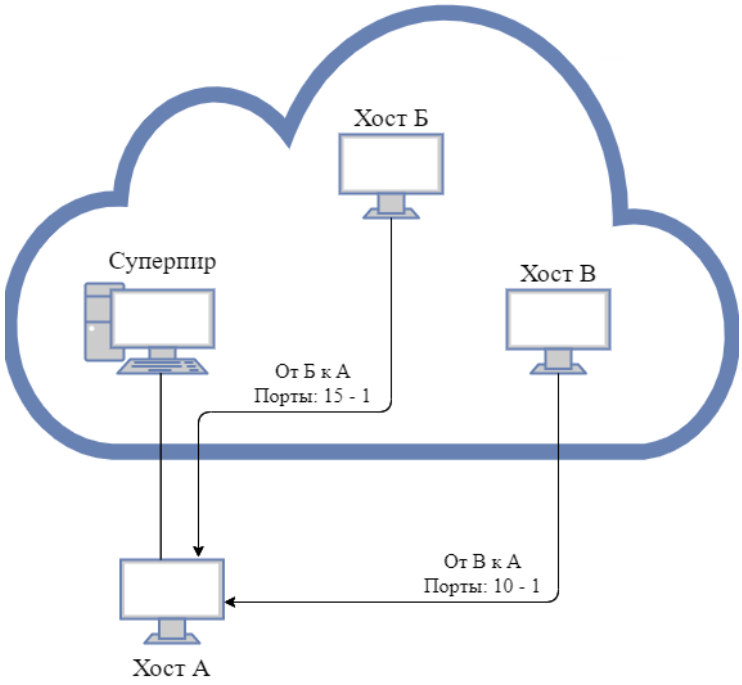


Рисунок 4 – К хосту A подключены хосты Б и В с 2 разными IP и 2 разными портами

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 инициирует несколько одновременных соединений, например, для параллель-

ной загрузки. Тогда веб-трафик будет иметь более высокое, по сравнению с P2P трафиком, соотношение числа отдельных портов к числу отдельных IP-адресов.

В работе **ССЫЛКА НА BLINC** 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).

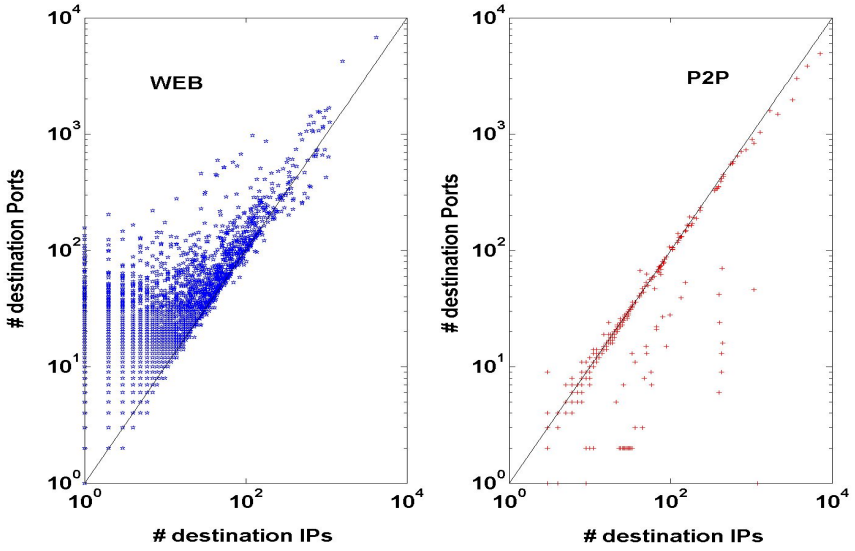


Рисунок 5

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 [**?**] 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 пороговой при условии, что C (число подключений) больше или равно половины 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 TCP 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 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 **ССЫЛКА НА НЕТФЛОУ** 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 обоих 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.

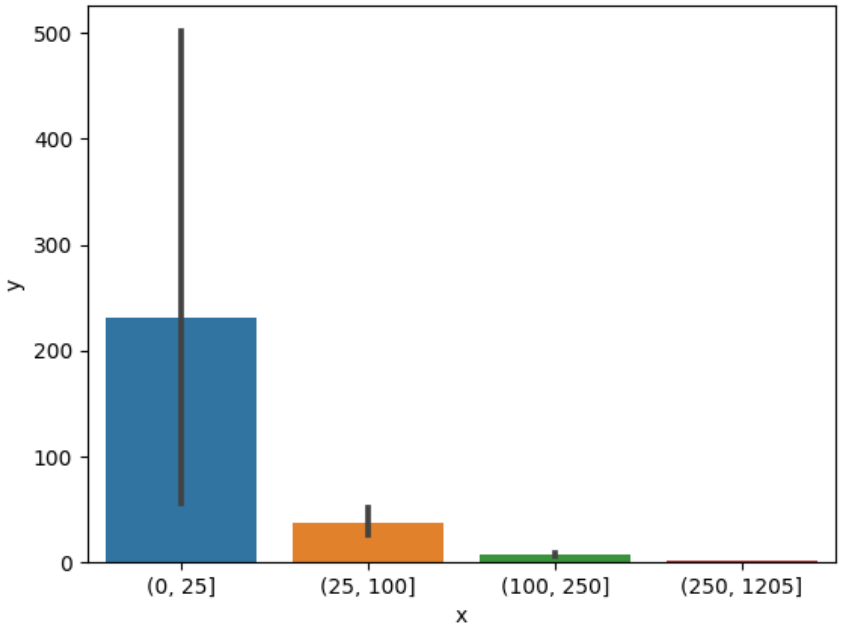


Рисунок 6 – Графическое представление среднего количества пакетов различных диапазонов их размеров в игре War Thunder

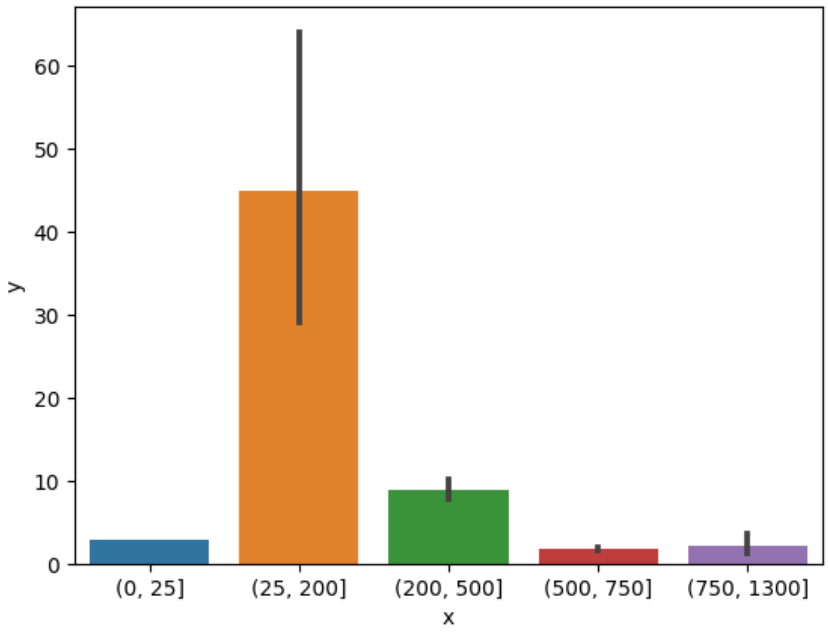


Рисунок 7 – Графическое представление среднего количества пакетов различных диапазонов их размеров в игре Dota 2

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 анализу 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.

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 рукопожатие (handshake). Формат рукопожатия следующий:

- **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 пакеты с рукопожатием представляю интерес при обнаружении BitTorrent, поскольку первые два заголовка передаются в открытом виде. На основе этих заголовков и формируется алгоритм:

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 команды и сообщения (назовём их словами Bitcoin): `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, чтоб, насколько это возможно, исключить те случаи, когда иные приложения используют порты Bitcoin или те же самые слова.

5 Описание программы

В данной работе был разработан **сниффер** — анализатор сетевого трафика. 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 так же 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:

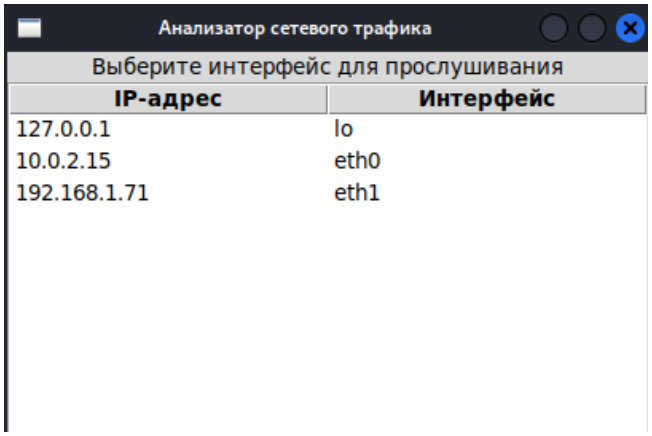


Рисунок 8 – Выбор сетевого интерфейса при старте программы

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.

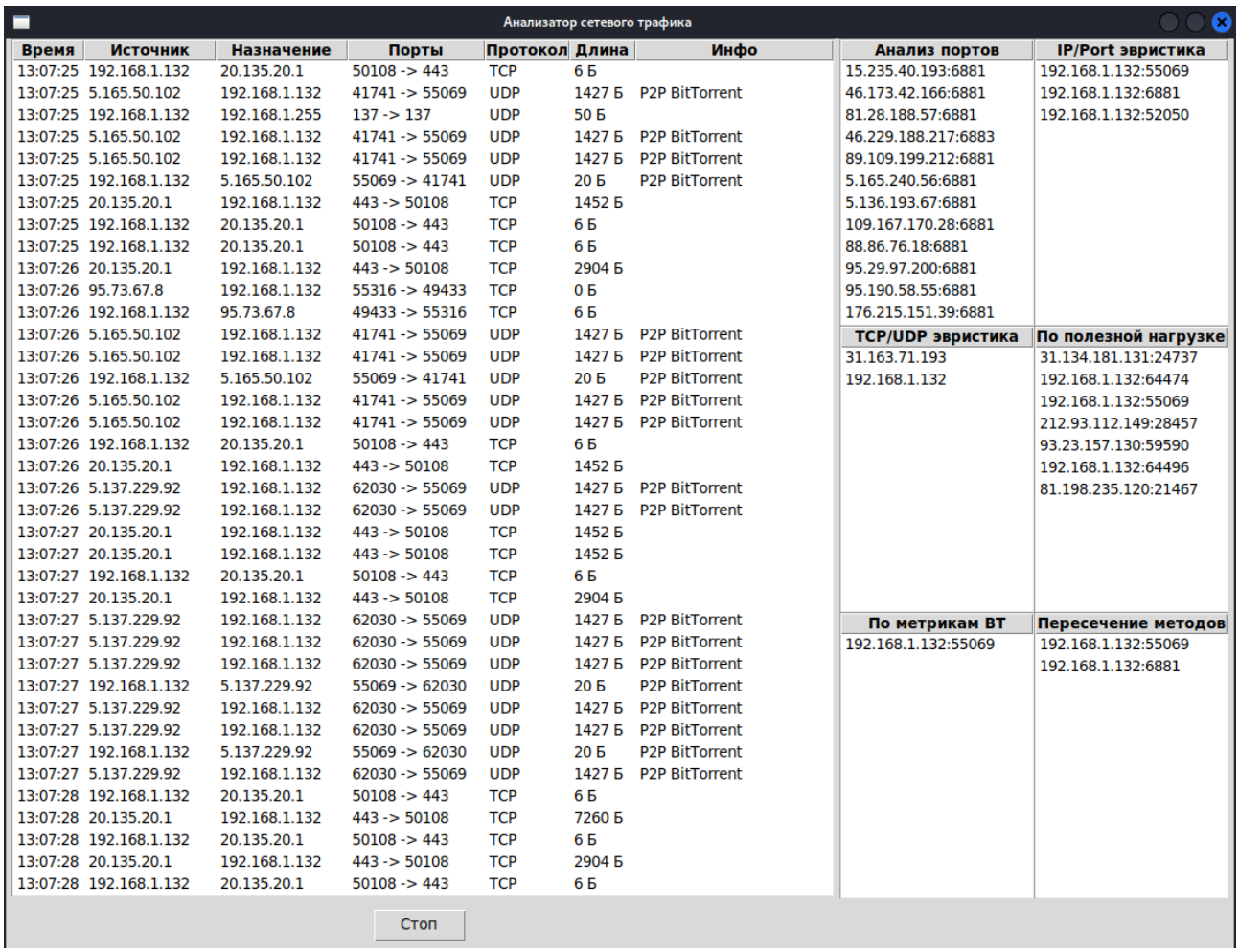


Рисунок 9 – Основное окно программы

6 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.

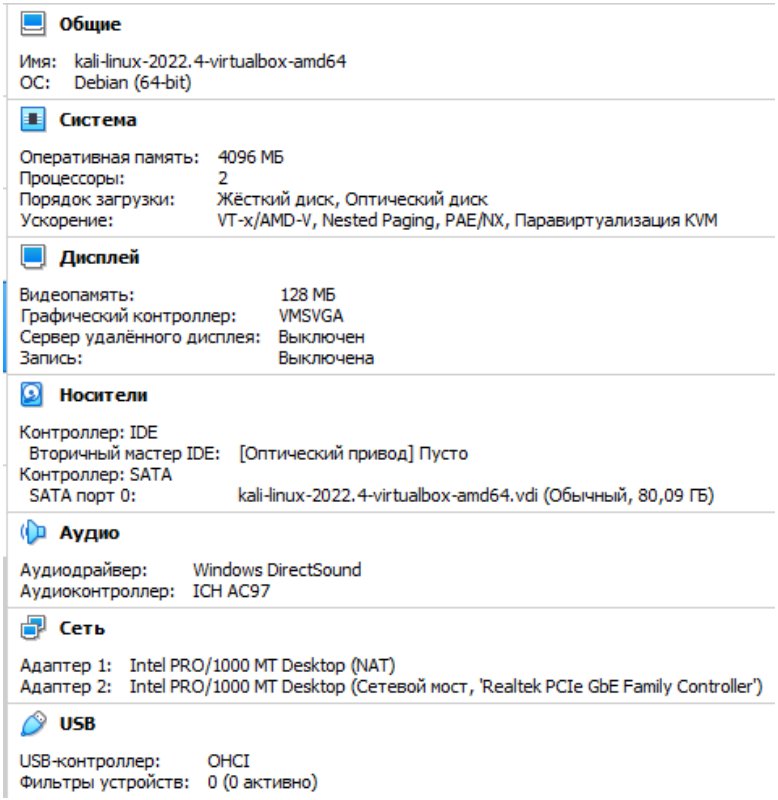


Рисунок 10 – Настройки виртуальной машины

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):

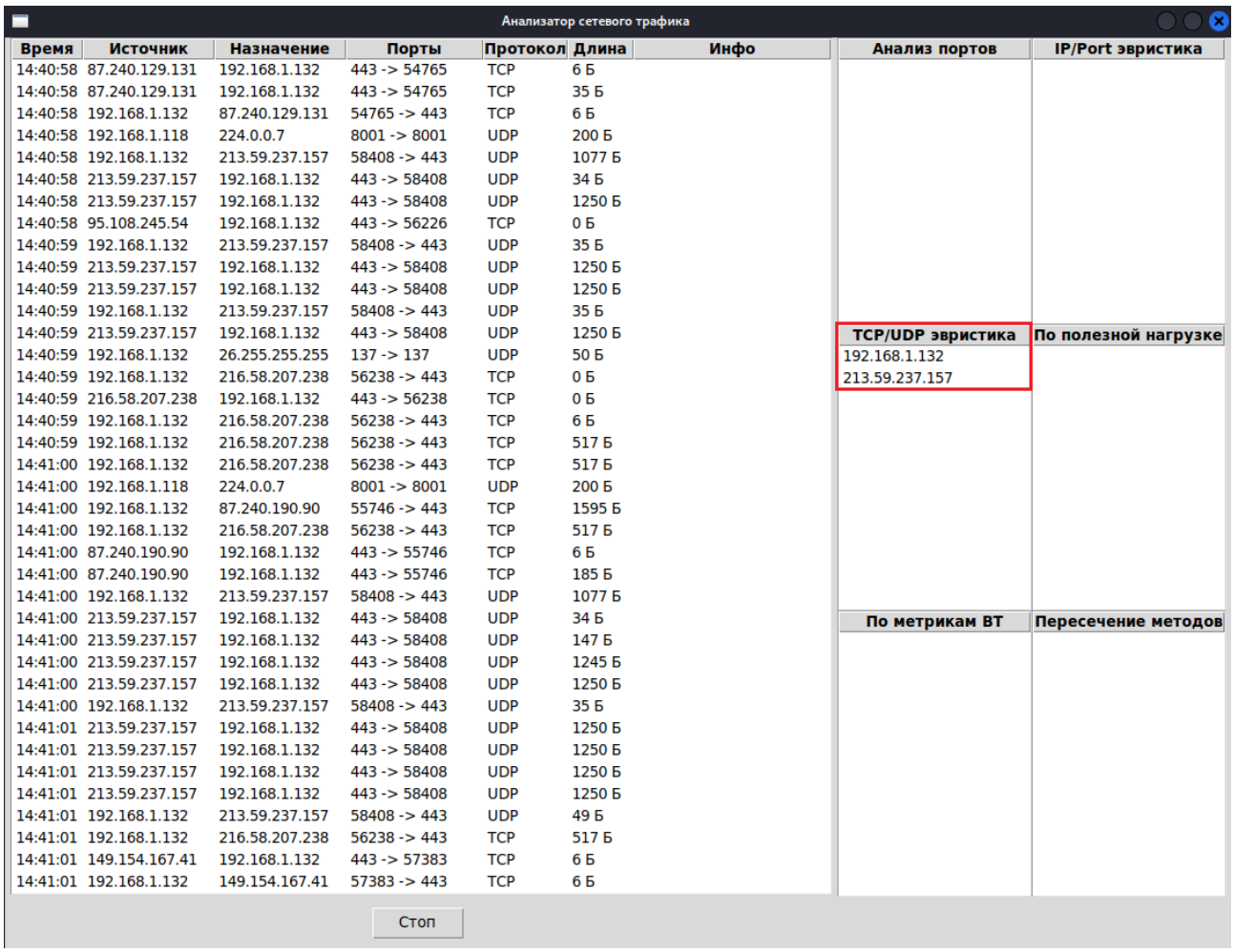


Рисунок 11 – Тестирование при активном веб-трафике

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.

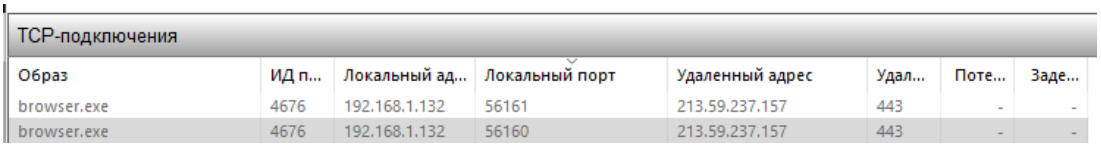


Рисунок 12 – Информация об адресе в resmon

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

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

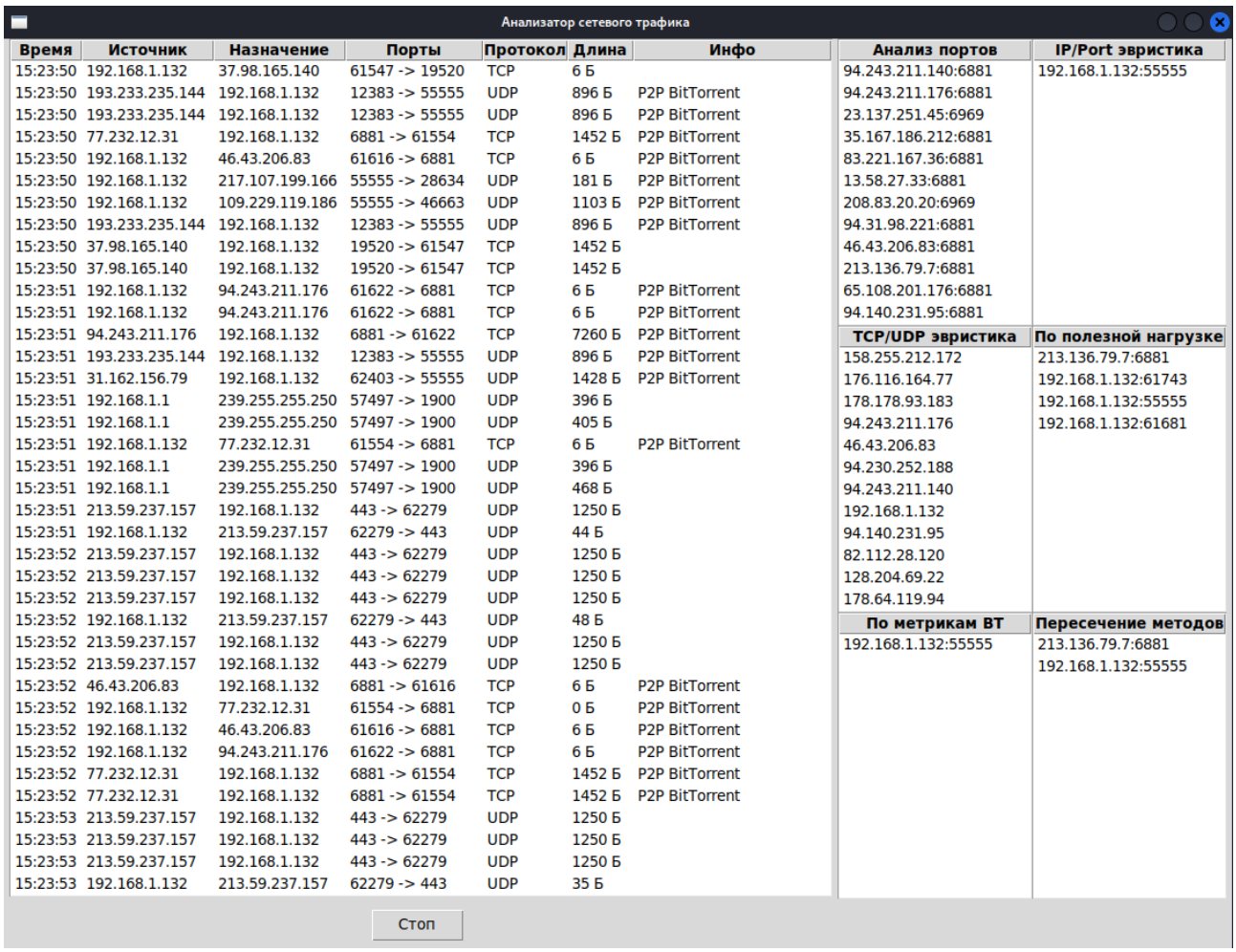


Рисунок 13 – Тестирование при загрузке файла через µTorrent

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

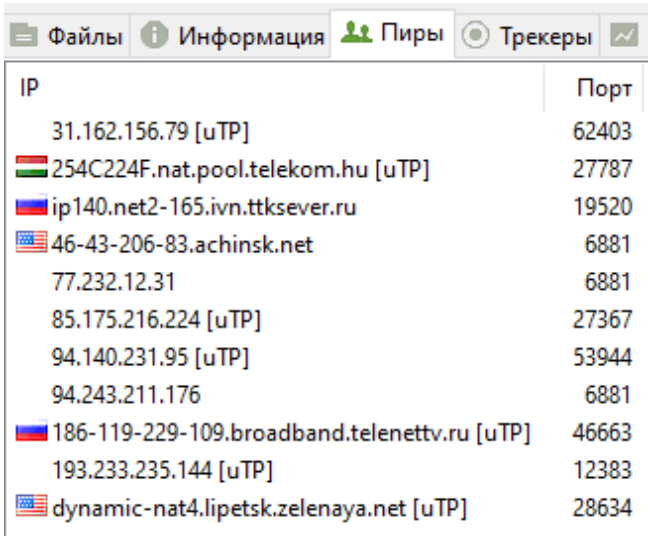


Рисунок 15 – Активные пиры клиента µTorrent

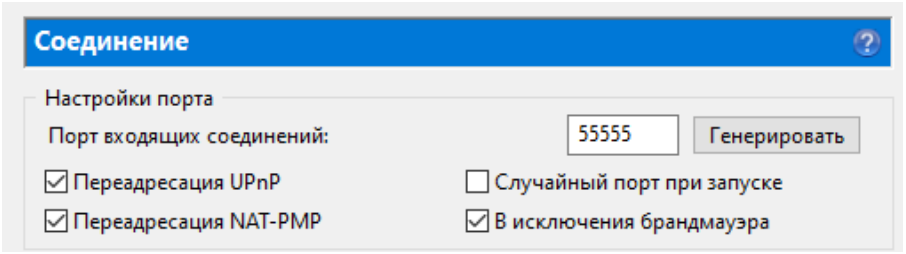


Рисунок 16 – Настройка порта входящих соединений клиента µTorrent

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

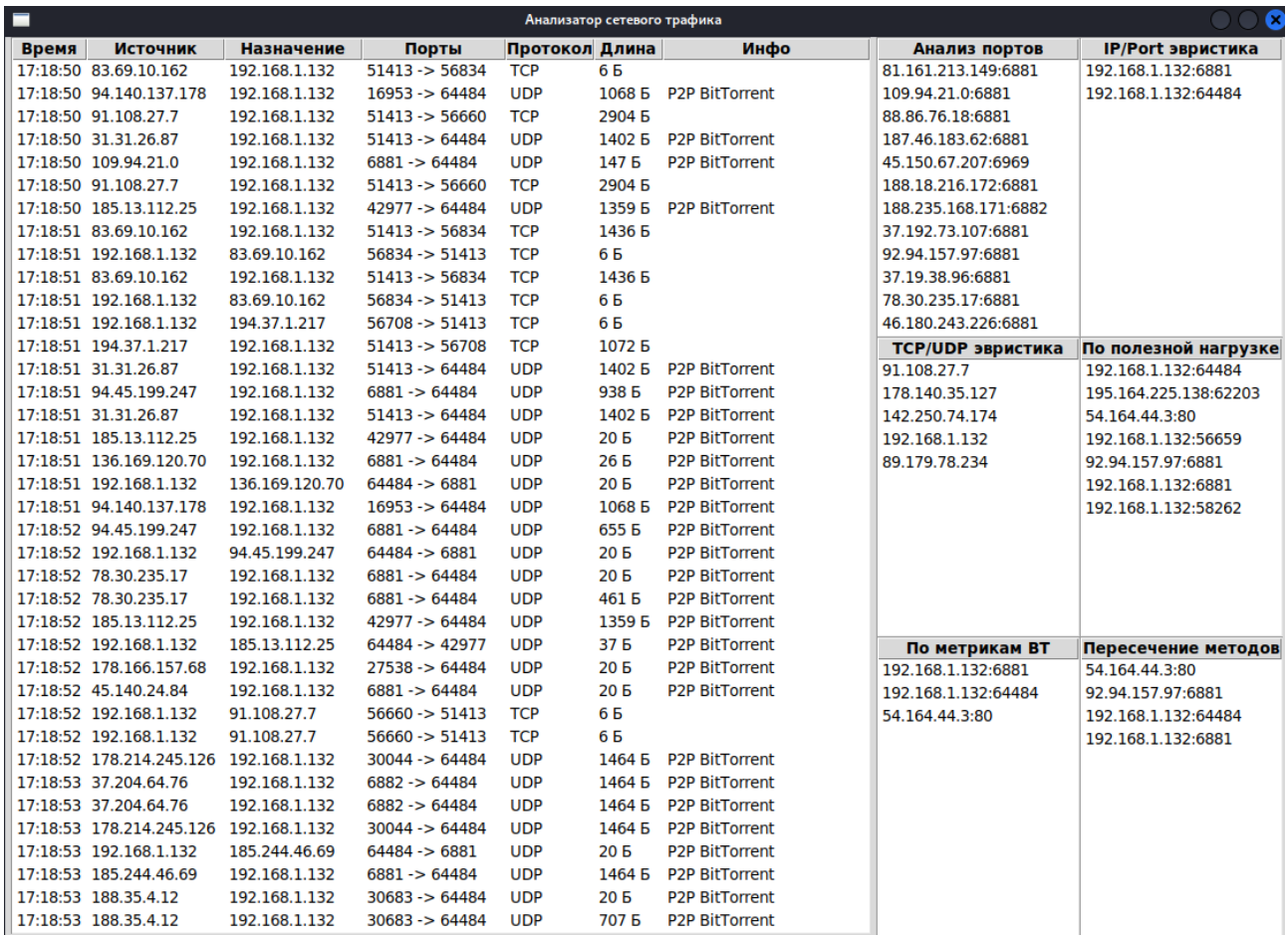


Рисунок 17 – Тестирование при загрузке файла через µTorrent Web

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:

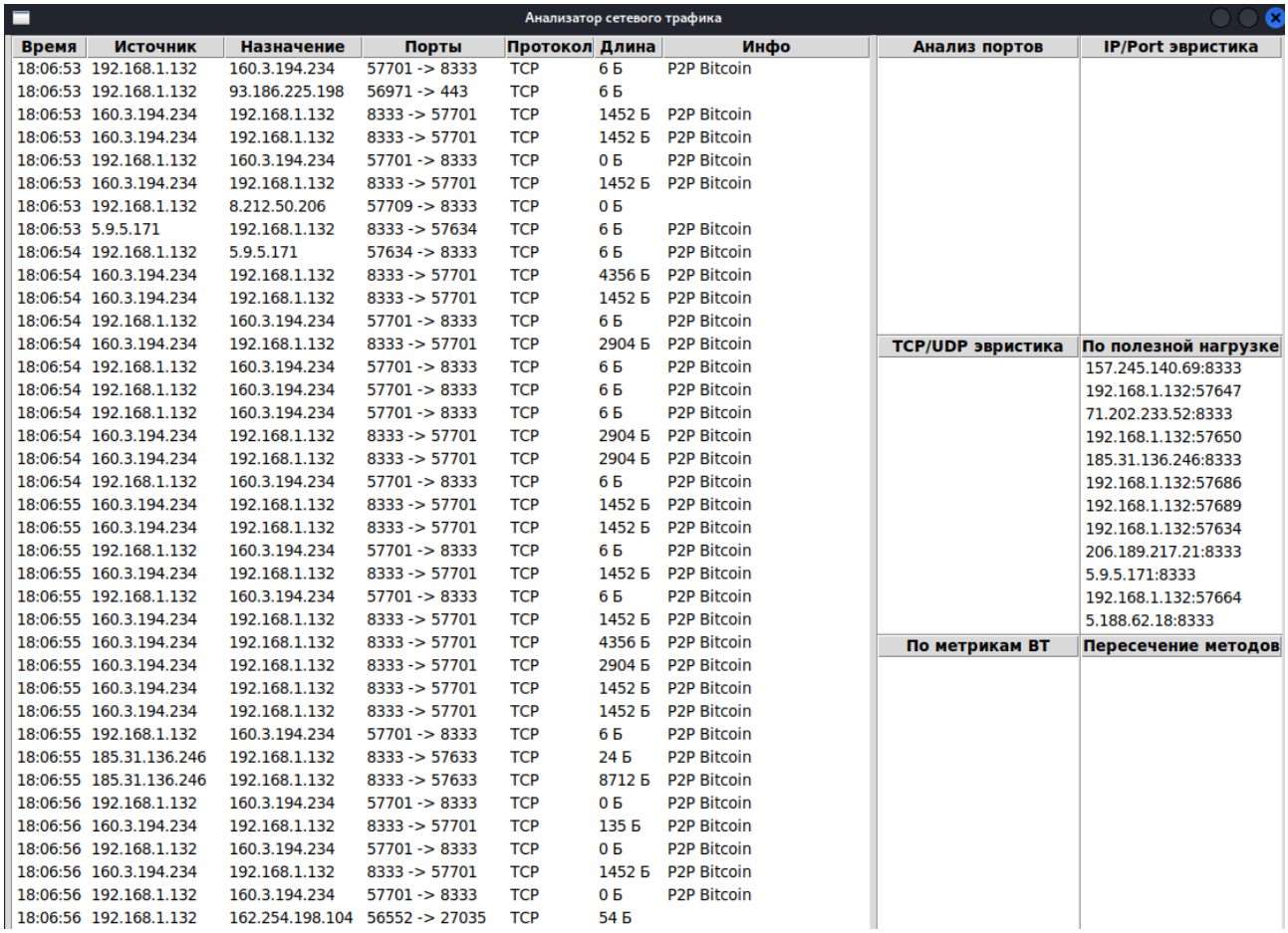


Рисунок 21 – Тестирование при работе Bitcoin Core

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:

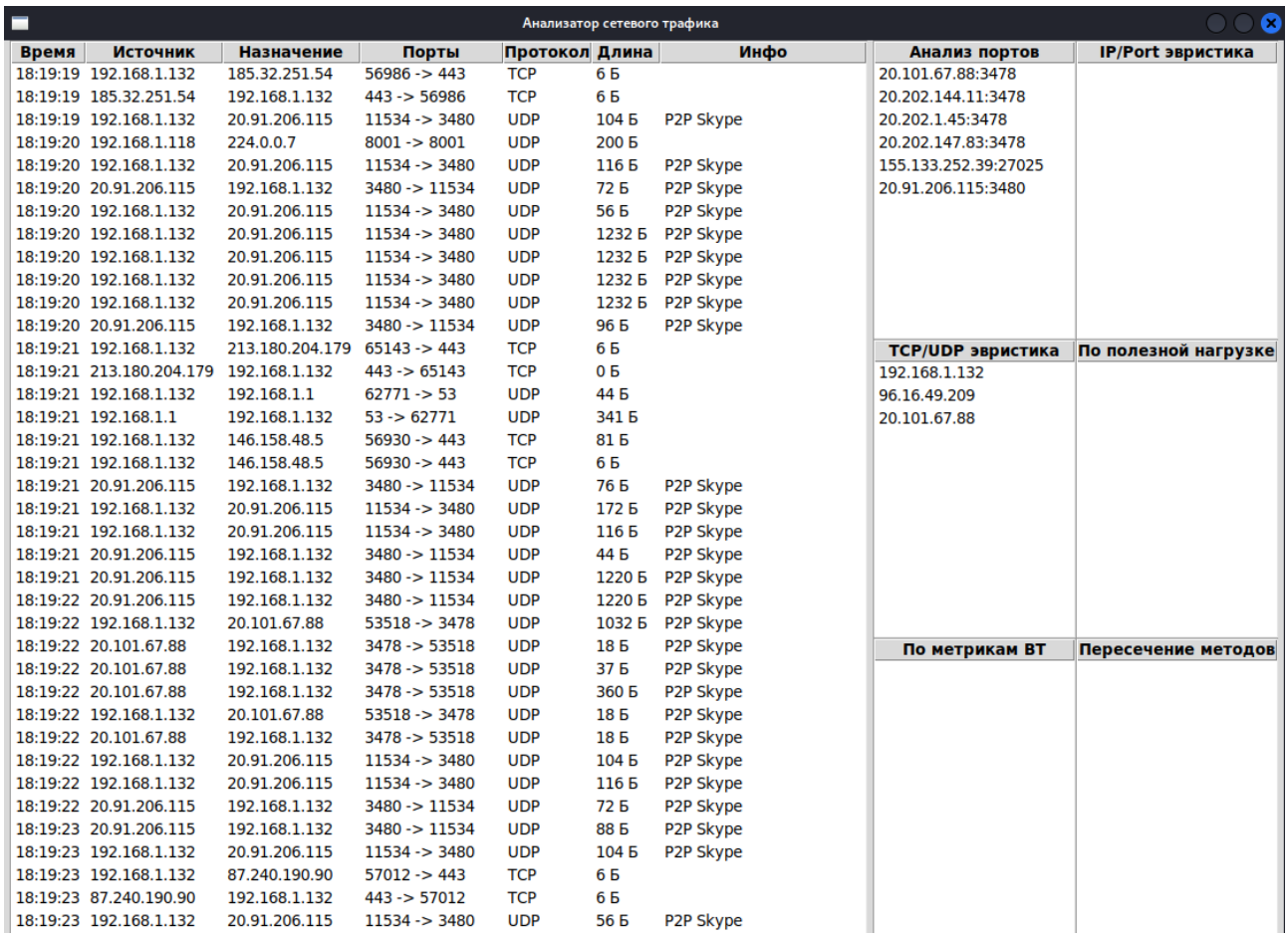


Рисунок 23 – Тестирование при работе Skype

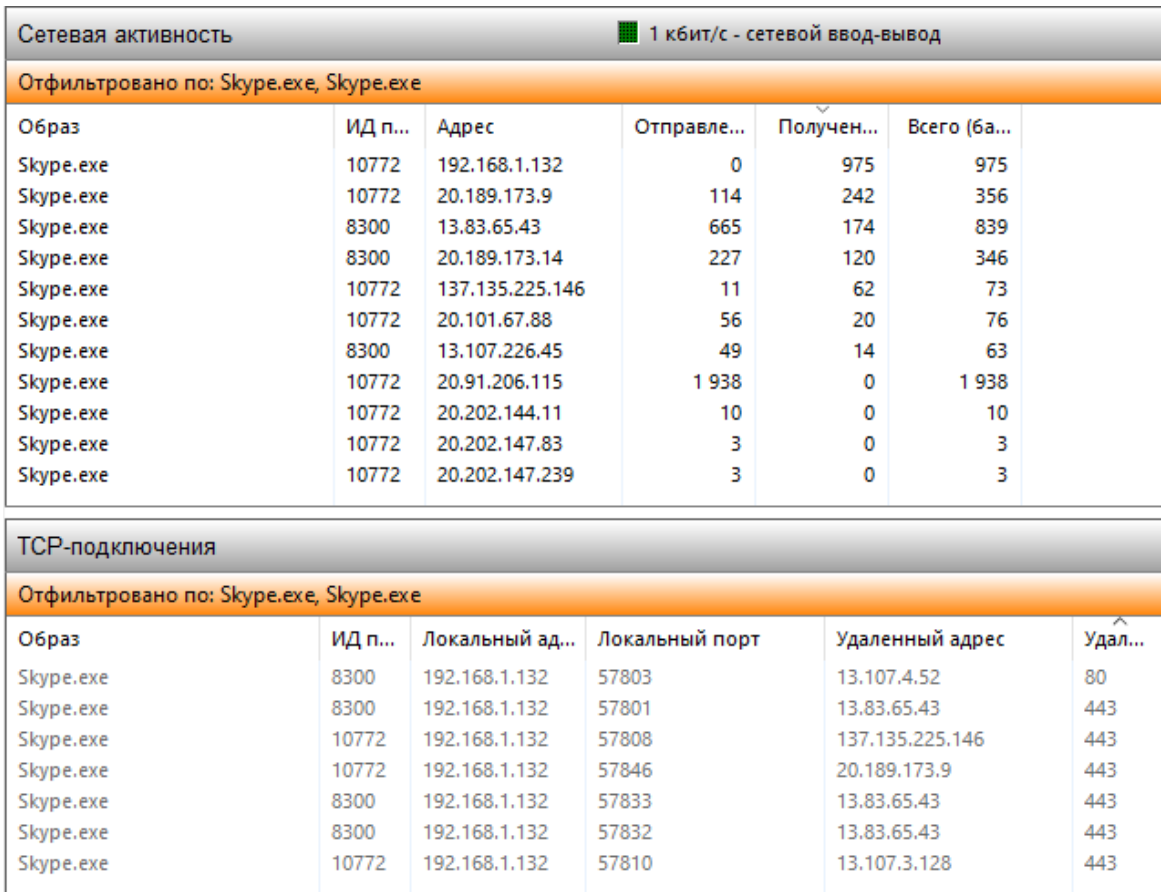


Рисунок 24 – Информация о TCP-подключениях и прослушиваемых портах Skype

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, которая позволяет перехватывать *TCP* и *UDP* трафик и анализировать его на присутствие P2P-активности. Были реализованы методы анализирования портов и обнаружения TCP/UDP- и IP/Port-эвристики.

Таким образом, изучение P2P-сетей несомненно является актуальным, поскольку они активно используются пользователи 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 таковых 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.

ПРИЛОЖЕНИЕ А

Код 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
                                                 ↪  прослушивания')
32         self.label_choose_interface.grid(row=0, column=0)
33
34         self.loi_columns = ['1', '2']
35         self.list_of_interfaces = ttk.Treeview(self.frame_choose_interface,
36                                                show='headings', columns=self.loi_columns,
                                                 ↪  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', values=[inter,
                ↪  inters_ips[inter]])
```

```
43
44          self.list_of_interfaces.bind('<Double-1>', self.start)
45
46          self.frame_main = ttk.Frame(self)
47
48          self.columns = ['1', '2', '3', '4', '5', '6', '7']
49          self.output = ttk.Treeview(self.frame_main, show='headings', columns=self.columns,
            ↪  height=38)
50          self.output.heading('1', text='Время')
51          self.output.heading('2', text='Источник')
52          self.output.heading('3', text='Назначение')
53          self.output.heading('4', text='Порты')
54          self.output.heading('5', text='Протокол')
55          self.output.heading('6', text='Длина')
56          self.output.heading('7', text='Инфо')
57
58          self.output.column('1', minwidth=0, width=65)
59          self.output.column('2', minwidth=0, width=120)
60          self.output.column('3', minwidth=0, width=120)
61          self.output.column('4', minwidth=0, width=125)
62          self.output.column('5', minwidth=0, width=77)
63          self.output.column('6', minwidth=0, width=60)
64          self.output.column('7', minwidth=0, width=180)
65
66          self.output.tag_configure("highlight", background="#FCA89F")
67
68          # Таблицы P2P адресов
69          self.frame = ttk.Frame(self.frame_main)
70          self.p2p_table_1 = ttk.Treeview(self.frame, show='headings', columns=['1'],
            ↪  height=12)
71          self.p2p_table_2 = ttk.Treeview(self.frame, show='headings', columns=['2'],
            ↪  height=12)
72          self.p2p_table_3 = ttk.Treeview(self.frame, show='headings', 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='Стоп', 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 вниз
121     def auto_down_scroll(self):
122         last_row = self.output.get_children()[-1]
123         last_row_bbox = self.output.bbox(last_row)
124
125         if len(last_row_bbox) > 0:
126             self.output.see(last_row)
127
128     def call_sniff(self):
129         ready = select.select([self.conn], [], [], SCAN_RATE_S)
130         if ready[0]:
131             out = sniffer.sniff(self.conn, self.osflag)
```

```
132             if out:
133                 time = str(datetime.now().strftime('%H:%M:%S'))
134                 out.insert(0, time)
135                 self.output_list.append(out)
136                 self.output.insert(parent='', index='end', values=out)
137                 # Подсветка
138                 # if out[-1][0:3] == "P2P":
139                 #     self.output.insert(parent='', index='end', values=out,
                    ↪  tags=("highlight",))
140                 # else:
141                 #     self.output.insert(parent='', index='end', values=out)
142                 self.auto_down_scroll()
143 
144                 # Вывод информации о пакете
145                 for s in out:
146                     file.write(s + ' ')
147                 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 | sniffer.bitcoin_addrs)
186                                        & (sniffer.p2p_pairs_p | sniffer.p2p_pairs_ipp |
                                           ↪ sniffer.bittorrent_addrs2))
187
188         intersection = intersection | (sniffer.bittorrent_addrs2
189                                         & (sniffer.p2p_pairs_p  | sniffer.bittorrent_addrs
190                                            | sniffer.bitcoin_addrs | sniffer.p2p_pairs_ipp))
191
192         for item_id in self.p2p_table_6.get_children():
193             self.p2p_table_6.delete(item_id)
194         for addr in intersection:
195             self.p2p_table_6.insert(parent='', index='end', values=[addr[0] + ":" +
                ↪ str(addr[1])])
196
197         root.after(15000, self.call_find_p2p)
198
199     def call_bt_stats(self):
200         for ipp in sniffer.dict_ipport:
201             sniffer.dict_ipport[ipp].bt_stats()
202
203         for item_id in self.p2p_table_5.get_children():
204             self.p2p_table_5.delete(item_id)
205         for addr in sniffer.bittorrent_addrs2:
206             self.p2p_table_5.insert(parent='', index='end', values=[addr[0] + ":" +
                ↪ str(addr[1])])
207
208         root.after(30000, self.call_bt_stats)
209
210     def stop(self):
211         file2.write('Список IP-адресов, взаимодействующих через P2P: \n')
212         file2.write('Анализ портов: \n')
213         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, socket.IPPROTO_IP)
260             conn.bind((interface, 0))
261             conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
262             conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
263             conn.setblocking(False)
264             # conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
265 
```

```
266         # Linux needs interface's name
267         else:
268             osflag = True
269 
270             if len(sys.argv) > 1:
271                 interface = sys.argv[1]
272             os.system("ip link set {} promisc on".format(interface))  # ret =
273             conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
274             conn.bind((interface, 0))
275             conn.setblocking(False)
276         return conn, osflag
277     except socket.error as msg:
278         print('Сокет не может быть создан. Код ошибки : ' + str(msg[0]) + ' Сообщение ' +
        ↪ msg[1])
279         sys.exit()
280 
281 
282 # Расшифровка названия интерфейса на Windows
283 def get_connection_name_from_guid(iface_guids):
284     iface_names = ['(unknown)' for i in range(len(iface_guids))]
285     reg = wr.ConnectRegistry(None, wr.HKEY_LOCAL_MACHINE)
286     reg_key = wr.OpenKey(reg,
    ↪ r'SYSTEM\CurrentControlSet\Control\Network\{4d36e972-e325-11ce-bfc1-08002be10318}')
287     for i in range(len(iface_guids)):
288         try:
289             reg_subkey = wr.OpenKey(reg_key, iface_guids[i] + r'\Connection')
290             iface_names[i] = wr.QueryValueEx(reg_subkey, 'Name')[0]
291         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     # print(ni.ifaddresses(_get_default_iface_linux()).setdefault(ni.AF_INET)[0]['addr'])
361     # print(ni.interfaces())
362
363     # В файл сохраняется последний вывод программы
364     file = open('out.txt', 'w+')
365     # Список IP-адресов, взаимодействующих через P2P
366     file2 = open('ip_list.txt', 'w+')
367
368     root = tk.Tk()
369     root.title("Анализатор сетевого трафика")
370     menu = Menu(root)
371     root.mainloop()
```

ПРИЛОЖЕНИЕ Б

Код sniffer.py

```
import socket
import struct

# Отступы для вывода информации
TAB_1 = '\t - '

# Список пар порт-приложение
LIST_P2P = {6881: 'BitTorrent', 6882: 'BitTorrent', 6883: 'BitTorrent',
            6884: 'BitTorrent', 6885: 'BitTorrent', 6886: 'BitTorrent',
            6887: 'BitTorrent', 6888: 'BitTorrent', 6889: 'BitTorrent',
            6969: 'BitTorrent', 411: 'Direct Connect', 412: 'Direct Connect',
            # 2323: 'eDonkey', 3306: 'eDonkey', 4242: 'eDonkey',
            # 4500: 'eDonkey', 4501: 'eDonkey', 4677: 'eDonkey',
            # 4678: 'eDonkey', 4711: 'eDonkey', 4712: 'eDonkey',
            # 7778: 'eDonkey', 1214: 'FastTrack', 1215: 'FastTrack',
            # 1331: 'FastTrack', 1337: 'FastTrack', 1683: 'FastTrack',
            # 4329: 'FastTrack', 5000: 'Yahoo', 5001: 'Yahoo',
            # 5002: 'Yahoo', 5003: 'Yahoo', 5004: 'Yahoo', 5005: 'Yahoo',
            # 5006: 'Yahoo', 5007: 'Yahoo', 5008: 'Yahoo', 5009: 'Yahoo',
            # 5010: 'Yahoo', 5050: 'Yahoo', 5100: 'Yahoo', 5555: 'Napster',
            # 6257: 'Napster', 6666: 'Napster', 6677: 'Napster',
            # 6688: 'Napster', 6699: 'Napster', 6700: 'Napster',
            # 6701: 'Napster', 6346: 'Gnutella', 6347: 'Gnutella', 5190: 'AIM',
            3478: 'Skype', 3479: 'Skype', 3480: 'Skype', 3481: 'Skype',
            4379: 'Steam', 4380: 'Steam (voice chat)', 27014: 'Steam',
            27015: 'Steam', 27016: 'Steam', 27017: 'Steam', 27018: 'Steam',
            27019: 'Steam', 27020: 'Steam', 27021: 'Steam', 27022: 'Steam',
            27023: 'Steam', 27024: 'Steam', 27025: 'Steam', 27026: 'Steam',
            27027: 'Steam', 27028: 'Steam', 27029: 'Steam', 27030: 'Steam',
            899: 'Radmin VPN', 12975: 'Hamachi', 32976: 'Hamachi'}

# Список портов исключений
EXCEPTIONS = {137, 138, 139, 445, 53, 123, 500, 554, 1900, 7070,
              6970, 1755, 5000, 5001, 6112, 6868, 6899, 6667, 7000, 7514,
              20, 21, 3396, 66, 1521, 1526, 1524, 22, 23, 25, 513, 543}

C_threshold = 20
RAT_threshold = 0.35
BIAT_threshold = 5
RRC_threshold = 0.5

TCP_addrs = set()
UDP_addrs = set()
p2p_addrs_tu = set()  # адреса, взаимодействующие одновременно по TCP и UDP
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 = set()
67         self.in_packets = dict()
68         self.dest_addrs = set()
69         self.old_bi = set()
70         self.rc = 0
71         self.p2p = False  # НЕ ИСПОЛЬЗУЕТСЯ
72
73     def add_sources(self, ip, port):
74         self.IPSet.add(ip)
75         self.PortSet.add(port)
76         self.srcs.add((ip, port))
77
78     def add_packets(self, src_addr, size):
79         if src_addr in self.in_packets.keys():
80             self.in_packets[src_addr].append(size)
81         else:
82             self.in_packets[src_addr] = [size]
83
84     def add_out_addrs(self, dest_addr):
85         self.dest_addrs.add(dest_addr)
86
87     # Добавление адресов, которые взаимодействовали с адресами из p2p_addrs_tu
88     def add_to_p2p_addrs1(self):
89         for ip in self.IPSet:
90             if ip not in [ipport[0] for ipport in rejected]:
```

```
91                  p2p_addrs_tu.add('(*) ' + ip)
92
93      def bt_stats(self):
94          # 1
95          c = len(self.srcs)
96          self.srcs = set()
97
98          # 2
99          at = 0
100         for addr in self.in_packets:
101             packets = self.in_packets[addr]
102             pack_size = len(packets)
103             if pack_size > 4:
104                 average_size = 0
105                 for p in packets:
106                     average_size += p
107                 average_size /= pack_size
108                 # 1250 или больше поставить?
109                 if average_size > 1250:
110                     at += 1
111
112         # 3
113         bi = self.in_packets.keys() & self.dest_addrs
114
115         # 4
116         # self.rc = 0  # ?
117         if len(bi) > len(self.old_bi):
118             self.rc += len(bi - self.old_bi)
119         else:
120             self.rc += len(self.old_bi - bi)
121
122         self.old_bi = bi
123         # Проверка граничных значений
124         if c > C_threshold:
125             bittorrent_addrs.add((self.dst_ip, self.dst_port))
126             bittorrent_addrs2.add((self.dst_ip, self.dst_port))
127         elif len(bi) > BIAT_threshold:
128             bittorrent_addrs.add((self.dst_ip, self.dst_port))
129             bittorrent_addrs2.add((self.dst_ip, self.dst_port))
130         # Следующие метрики связываются с метрикой C для большей точности
131         elif c > C_threshold / 2:
132             if at / c > RAT_threshold:
133                 bittorrent_addrs.add((self.dst_ip, self.dst_port))
134                 bittorrent_addrs2.add((self.dst_ip, self.dst_port))
135             elif at > 0:
136                 if self.rc / at > RRC_threshold:
137                     bittorrent_addrs.add((self.dst_ip, self.dst_port))
138                     bittorrent_addrs2.add((self.dst_ip, self.dst_port))
```

```
139
140
141 def sniff(conn, os):
142     output = ''
143     data, addr = conn.recvfrom(65536)
144     if os:
145         dest_mac, src_mac, eth_proto, data = ethernet_frame(data)
146     else:
147         eth_proto = 8
148
149     # IPv4
150     if eth_proto == 8:
151         version, header_length, ttl, proto, src, dest, data = ipv4_packet(data)
152
153         if proto == 6 or proto == 17:
154
155             # TCP
156             if proto == 6:
157                 src_port, dest_port, data = tcp_segment(data)
158
159                 check_exceptions(src, dest, src_port, dest_port)
160                 if (src, src_port) not in rejected and (dest, dest_port) not 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), 'TCP',
                    ↪  str(len(data)) + ' B',
165                           addition_info]
166
167             # UDP
168             else:
169                 src_port, dest_port, length, data = udp_segment(data)
170
171                 check_exceptions(src, dest, src_port, dest_port)
172                 if (src, src_port) not in rejected and (dest, dest_port) not in rejected:
173                     UDP_addrs.add((src, dest))
174
175                 addition_info = add_info(src, dest, src_port, dest_port)
176                 output = [src, dest, str(src_port) + ' -> ' + str(dest_port), 'UDP',
                    ↪  str(len(data)) + ' B',
177                           addition_info]
178
179             add_ipport(dest, dest_port, src, src_port, len(data))
180             payload_analysis(src, dest, src_port, dest_port, data)
181
182         return output
183
184
```

```
185 # после проверки портов функция
186 # добавляет к строке вывода информацию для столбца info,
187 # если адрес p2p и добавляет протокол по возможности
188 def add_info(src, dest, src_port, dest_port):
189     addition_info = ''
190     if LIST_P2P.get(src_port, False):
191         p2p_pairs_p.add((src, src_port))
192         addition_info = 'P2P ' + LIST_P2P[src_port]
193     elif LIST_P2P.get(dest_port, False):
194         p2p_pairs_p.add((dest, dest_port))
195         addition_info = 'P2P ' + LIST_P2P[dest_port]
196     elif (src, src_port) in bittorrent_addrs:
197         addition_info = 'P2P BitTorrent'
198     elif (dest, dest_port) in bittorrent_addrs:
199         addition_info = 'P2P BitTorrent'
200     elif (src, src_port) in bitcoin_addrs:
201         addition_info = 'P2P Bitcoin'
202     elif (dest, dest_port) in bitcoin_addrs:
203         addition_info = 'P2P Bitcoin'
204     return addition_info
205
206
207 def add_ipport(dest, dest_port, src, src_port, size):
208     ipport = dest + ':' + str(dest_port)
209     if ipport not in dict_ipport:
210         x = IPPort(dest, dest_port)
211         x.add_sources(src, src_port)
212         dict_ipport[ipport] = x
213         x.add_packets(src + ':' + str(src_port), size)
214     else:
215         dict_ipport[ipport].add_sources(src, src_port)
216         dict_ipport[ipport].add_packets(src + ':' + str(src_port), size)
217
218     ipport_src = src + ':' + str(src_port)
219     if ipport_src in dict_ipport:
220         dict_ipport[ipport_src].add_out_addrs(ipport)
221
222
223 # Добавление адресов с портами в список исключений
224 def check_exceptions(src, dest, src_port, dest_port):
225     if src_port in EXCEPTIONS \
226             or dest_port in EXCEPTIONS \
227             or (src_port == dest_port and src_port < 500):
228         rejected.add((src, src_port))
229         rejected.add((dest, dest_port))
230
231
232 # Анализ полезной нагрузки пакетов,
```

```
233 def payload_analysis(src, dest, src_port, dest_port, data):
234     # Для BitTorrent
235     sdata = str(data)
236     if len(data) >= 20:
237         if 'BitTorrent protocol' in sdata:
238             bittorrent_addrs.add((src, src_port))
239             bittorrent_addrs.add((dest, dest_port))
240         elif src_port == 8333 or dest_port == 8333 or src_port == 8334 or dest_port == 8334:
241             # print(sdata)
242             for word in bitcoin_phrases:
243                 if word in sdata:
244                     bitcoin_addrs.add((src, src_port))
245                     bitcoin_addrs.add((dest, dest_port))
246                     break
247
248
249 def find_p2p():
250     # 1 Заполнение p2p_addrs адресами, взаимодействующими одновременно по TCP и UDP
251     inter = TCP_addrs & UDP_addrs
252     for addrs in inter:
253         p2p_addrs_tu.add(addrs[0])
254         p2p_addrs_tu.add(addrs[1])
255
256     # 2 Заполнение p2p_pairs_ipp адресами, выбранными исходя из check_p2p
257     for ipport in dict_ipport:
258         ipp = dict_ipport[ipport]
259
260         ip = ipp.dst_ip
261         port = ipp.dst_port
262
263         # Добавление адресов, взаимодействующие с адресами из TCP/UDP пар
264         # if ip in p2p_addrs_tu:
265         #     ipp.add_to_p2p_addrs1()
266
267         compare_dif = 2
268
269         # Если порт из известных p2p портов, то разница должна быть увеличена до 10
270         if ipport in p2p_pairs_p:
271             compare_dif = 10
272
273         cur_dif = len(ipp.IPSet) - len(ipp.PortSet)
274         if len(ipp.IPSet) > 2 and (cur_dif < compare_dif):
275             if (ip, port) not in rejected:
276                 p2p_pairs_ipp.add((ip, port))
277
278         # Если разница больше 10, то, скорее всего, это не p2p и можно добавить в
         ↪ исключения.
279         elif cur_dif > 10:
```

```
280             rejected.add((ip, port))
281
282
283 # Распаковка ethernet кадра
284 def ethernet_frame(data):
285     dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
286     return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto), data[14:]
287
288
289 # Форматирование MAC-адреса
290 def get_mac_addr(bytes_addr):
291     bytes_str = map('{:02x}'.format, bytes_addr)
292     return ':'.join(bytes_str).upper()
293
294
295 # Распаковка IPv4 пакета
296 def ipv4_packet(data):
297     version_header_length = data[0]
298     version = version_header_length >> 4
299     header_length = (version_header_length & 15) * 4
300     ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
301     return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]
302
303
304 # Форматирование IP-адреса
305 def ipv4(addr):
306     return '.'.join(map(str, addr))
307
308
309 # Распаковка TCP сегмента
310 def tcp_segment(data):
311     (src_port, dest_port, _, _, offset_reserved_flags) = struct.unpack('! H H L L H',
        ↪ data[:14])
312     offset = (offset_reserved_flags >> 12) * 4
313     # flag_urg = (offset_reserved_flags & 32) >> 5
314     # flag_ack = (offset_reserved_flags & 16) >> 5
315     # flag_psh = (offset_reserved_flags & 8) >> 5
316     # flag_rst = (offset_reserved_flags & 4) >> 5
317     # flag_syn = (offset_reserved_flags & 2) >> 5
318     # flag_fin = offset_reserved_flags & 1
319     return src_port, dest_port, data[offset:]
320
321
322 # Распаковка UDP сегмента
323 def udp_segment(data):
324     src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
325     return src_port, dest_port, size, data[8:]
```