

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 Исключения	15
3.3.1 Почта	15
3.3.2 DNS	17
3.3.3 Игры и вредоносные программы	17
4 Обнаружение P2P трафика при помощи анализа полезной нагрузки	19
4.1 Обнаружение BitTorrent	19
4.2 Обнаружение Bitcoin	20
5 Описание программы	21
5.1 Функция call_sniff	21
5.2 Функция sniff	22
5.3 Определение P2P трафика	23
5.3.1 Метод анализирования портов	23

5.3.2	Метод анализов потока	24
5.4	Тестирование программы	25
5.4.1	Запуск программы при отсутствии P2P-активности	25
5.4.2	Запуск программы при запущенном клиенте Bittorrent	26
5.4.3	Запуск программы при запущенном аудио звонке Skype	28
ЗАКЛЮЧЕНИЕ		29
Приложение А Код main.py		30
Приложение Б Код sniffer.py		38

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. [?]

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 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.



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, видео-игры).

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

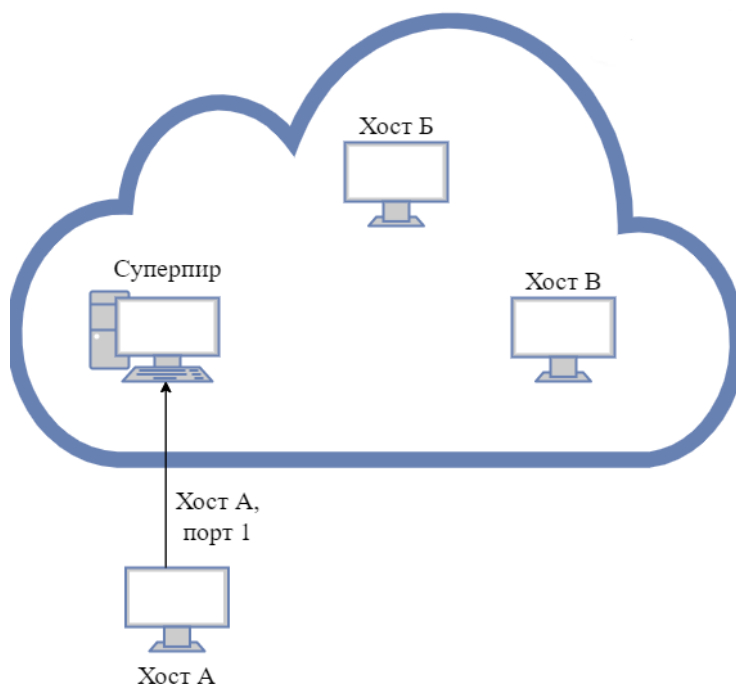


Рисунок 2 – Отправка информации хоста А о себе суперпиру

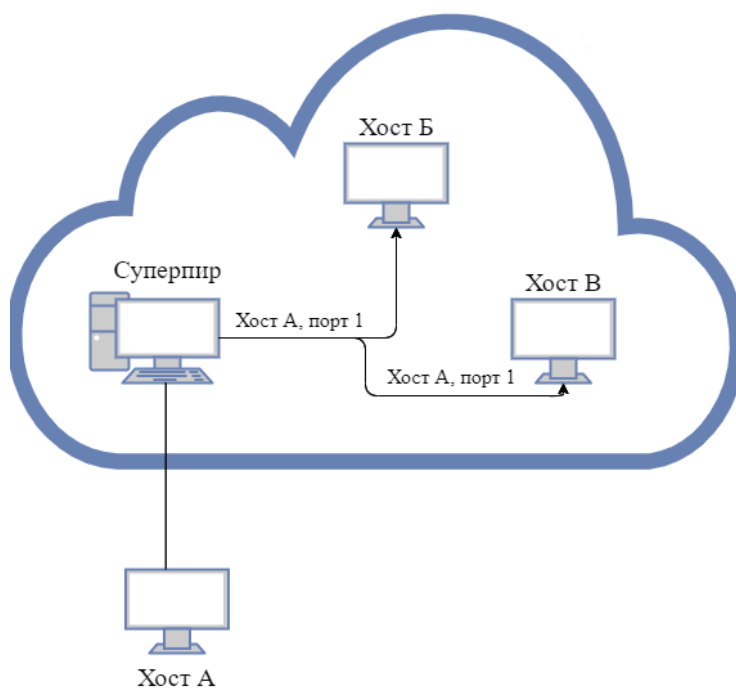


Рисунок 3 – Суперпир распространяет информацию о хосте А остальной части сети

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 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.

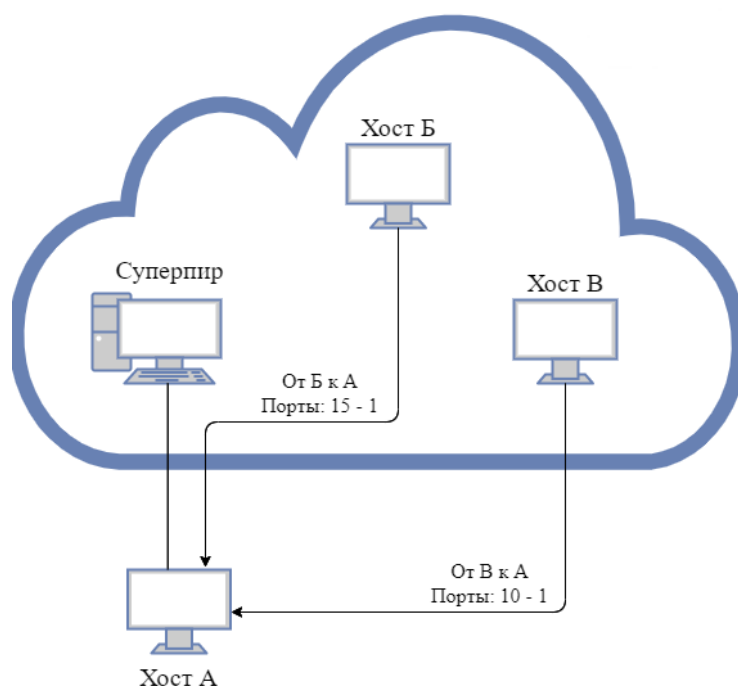


Рисунок 4 – К хосту А подключены хосты Б и В с 2 разными IP и 2 разными портами

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).

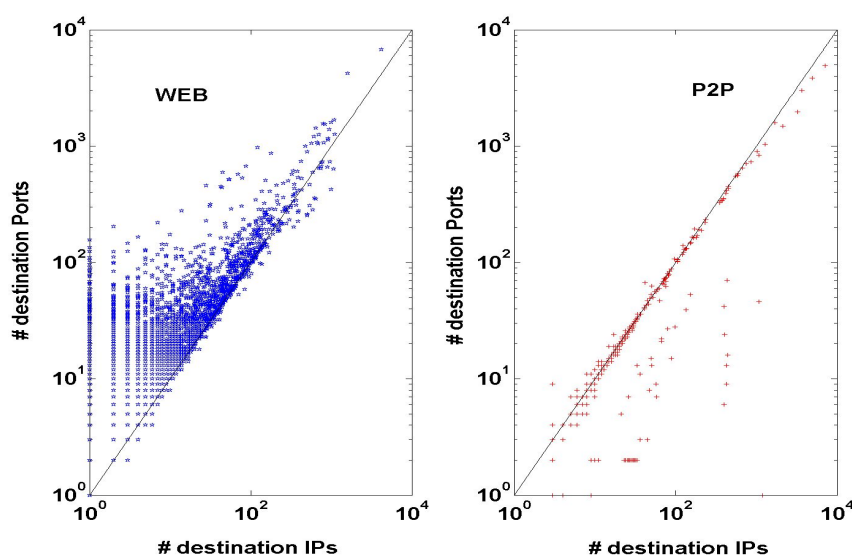


Рисунок 5

3.3 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.

3.3.1 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 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.

3.3.2 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.

3.3.3 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 (например, множество длин не превышает трех). Для исключения такого взаимодействия сохраняется соответствующая информация и проводится проверка. Однако такая проверка трудно реализуема, поскольку размеры пакетов будут зависеть от каждой конкретной игры. Вдобавок в этом, на практике множество длин обычно намного больше, чем три.

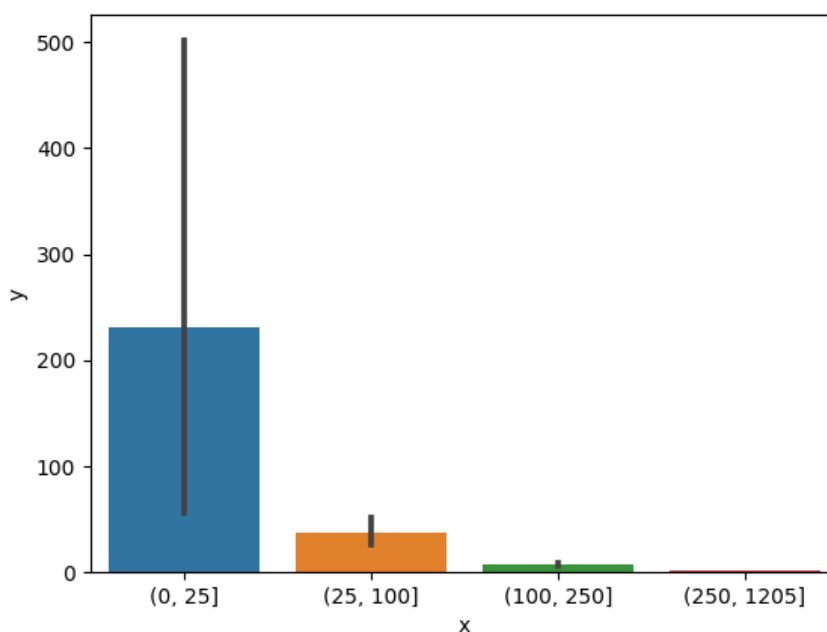


Рисунок 6 – Вар тандер

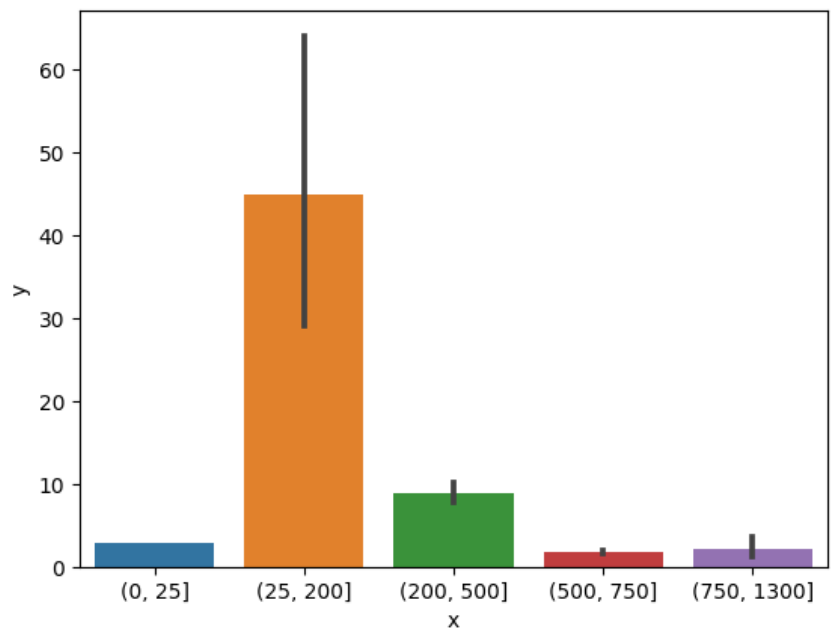


Рисунок 7 – Дота 2

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, которые можно 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 представляют 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, чтобы, 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*. Дополнительно последний вывод программы сохраняется в текстовые файлы. Программа обрабатывает трафик, проходящий через сетевой интерфейс, указанный как аргумент при её запуске. По умолчанию в программе указан интерфейс *enp0s3*.

```
1 sudo ./window.py enp0s3
```

Для указанного сетевого интерфейса программа включает неразборчивый режим.

При запуске `window.py` создаётся **сокет** — программный интерфейс для обеспечения обмена данными между процессами. Через него проходит весь сетевой трафик на той виртуальной машине, на которой он находится.

```
1 ret = os.system("ip link set {} promisc on".format(interface))
2
3 conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
4 conn.bind((interface, 0))
```

5.1 Функция `call_sniff`

Функция `call_sniff` вызывает функцию `sniff` из `sniffer.py`, которая передаёт информацию о пакете для вывода на экран. Эта информация сохраняется в список `out`. Затем информация о пакетах выводится на экран и сохраняется в текстовый файл `out.txt`, а списки обнаруженных адресов, взаимодействующих через P2P, сохраняются в `ip_list.txt`.

```
1 def call_sniff(self):
2     out = sniffer.sniff(conn)
3     if out:
4         # Вывод времени
5         time = str(datetime.now().strftime('%H:%M:%S')) + "\n"
6         if time != self.last_time:
7             self.output.insert('end', time)
8             file.write(time)
9             self.last_time = time
10
11     # Вывод информации о пакете
12     for s in out:
13         file.write(s)
```

```

14         self.output.insert('end', s)
15         file.write('\n')
16         self.output.insert('end', '\n')
17
18         root.after(100, self.call_sniff) # сканирование каждые 0.1 сек

```

Функция `call_sniff` вызывается каждые 0.1 секунды, то есть сканирование происходит раз в 0.1 секунды. Эмпирически было установлено, что для анализа трафика пары приложений, одно из которых взаимодействует через P2P достаточно сканировать раз в 0.2-0.3 секунды. Однако при более активном трафике сканирование следует проводить чаще, чтобы не было пакетов, которые не оказались бы перехваченными.

5.2 Функция `sniff`

Функция `sniff` обрабатывает информацию о пакете и сохраняет некоторые данные с помощью функции `save` в глобальные переменные. В множества `TCP_addrs` и `UDP_addrs` сохраняются пары IP-адресов вида (IP-отправителя, IP-получателя), взаимодействующих по соответствующим протоколам. В множество `rejected` добавляются IP-адреса с портами, если порт является одним из перечисленных в списке портов-исключений `EXCEPTIONS`. Это необходимо, чтобы отсеять их при дальнейшем анализе, поскольку их активность схожа с P2P-активностью или порт является известным. Например, это могут быть почтовые, игровые или *DNS* сервисы.

В словарь `dict_ipport` сохраняются пары вида (`dest_ip + dest_port` → объект класса `IPPort`). В таких объектах сохраняется информация о различных адресах источника и различных портах источника для каждой пары адреса назначения (`dest_ip + dest_port`).

```

1 def sniff(conn):
2     output = ''
3     raw_data, addr = conn.recvfrom(65536)
4     dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)
5
6     # IPv4
7     if eth_proto == 8:
8         (version, header_length, ttl, proto, src, dest, data) = ipv4_packet(data)
9
10    # TCP
11    if proto == 6:
12        src_port, dest_port, sequence, ack, flag_urg, flag_ack, \

```

```

13         flag_psh, flag_rst, flag_syn, flag_fin, data = tcp_segment(data)
14
15         output = [TAB_1, 'TCP: ', src, ':', str(src_port), ' -> ', dest, ':',
16                     str(dest_port), ', ', str(len(data)), ' B']
17
18         save(True, src, dest, src_port, dest_port)
19         check_ports(src, dest, src_port, dest_port)
20
21     # UDP
22     elif proto == 17:
23         src_port, dest_port, length, data = udp_segment(data)
24
25         output = [TAB_1, 'UDP: ', src, ':', str(src_port), ' -> ', dest, ':',
26                     str(dest_port), ', ', str(len(data)), ' B']
27
28         check_ports(src, dest, src_port, dest_port)
29         save(False, src, dest, src_port, dest_port)
30
31     return output

```

5.3 Определение P2P трафика

5.3.1 Метод анализирования портов

С помощью функции `check_ports`, которая вызывается при каждом запуске `sniff`, проводится анализ портов. Если был обнаружен порт, который присутствует в списке `LIST_P2P`, то IP-адрес вместе с портом заносится в `p2p_pairs_p`.

```

1 def check_ports(src, dest, src_port, dest_port):
2     if LIST_P2P.get(src_port, False):
3         p2p_pairs_p.add((src, src_port))
4     elif LIST_P2P.get(dest_port, False):
5         p2p_pairs_p.add((dest, dest_port))

```

В списке пар порт-приложение `LIST_P2P` находится информация об используемых портах некоторых P2P-приложений, а именно:

- Bittorrent;
- Direct Connect;
- eDonkey;
- FastTrack;
- Yahoo;
- Napster;
- Gnutella;

- AIM;
- Skype;
- Steam;
- Hamachi;
- Radmin VPN;

5.3.2 Метод анализирования потоков

Данный метод реализуется в функции `find_p2p`:

```

1 def find_p2p():
2     # 1 Заполнение p2p_addrs адресами, взаимодействующими одновременно по TCP и UDP с учётом
    ↪ исключений
3     inter = TCP_addrs & UDP_addrs
4     for pair_addrs in inter:
5         for ipport in rejected:
6             if pair_addrs[0] != ipport[0] and pair_addrs[1] != ipport[0]:
7                 p2p_addrs.add(pair_addrs)
8
9     # 2 Заполнение p2p_addrs адресами, выбранными исходя из check_p2p с учётом исключений
10    for ipport in dict_ipport:
11        ipp = dict_ipport[ipport]
12        ipp.add_to_p2p_addrs1() # Заполнение массива p2p_addrs1
13        ip = ipp.dst_ip
14        port = ipp.dst_port
15        if ipp.check_p2p() and (ip, port) not in rejected:
16            p2p_pairs_ipp.add((ip, port))

```

Данная функция работает по следующему алгоритму [?]:

Шаг 1 (проверка TCP/UDP-эвристики). Рассматриваются пары адресов, взаимодействующих одновременно по протоколам *TCP* и *UDP*. Если при этом IP-адреса не входят в список исключений *rejected*, то эта пара адресов заносится в массив `p2p_addrs`.

Шаг 2 (проверка IP/Port-эвристики). Для каждой пары адресов из `dict_ipport` проверяется, что IP-адрес с портом не находится в списке исключений, и проверяется условие — если массив различных адресов источника `IPSet` содержит более двух адресов, а разница между количеством элементов этого массива и массива различных портов `PortSet` источника меньше двух (или меньше 10, если порт является исключением, т.е. находится в списке `EXCEPTIONS`), то считается, что пара принимает участие в P2P-деятельности и добавляется в `p2p_pairs_ipp`.

Эти действия проводятся каждые 15 секунд с момента запуска программы.

5.4 Тестирование программы

Тестирование программы проводилось по следующей схеме: на хостовой машине с *Windows 10* была запущена виртуальная машина с помощью *Virtual Box*. На виртуальной машине установлена *Manjaro Linux*. Между виртуальной и хостовой машиной сетевой мост с неразборчивым режимом. Программа запущена на виртуальной машине, а проверяемые P2P-приложения на хостовой (рисунок 8).

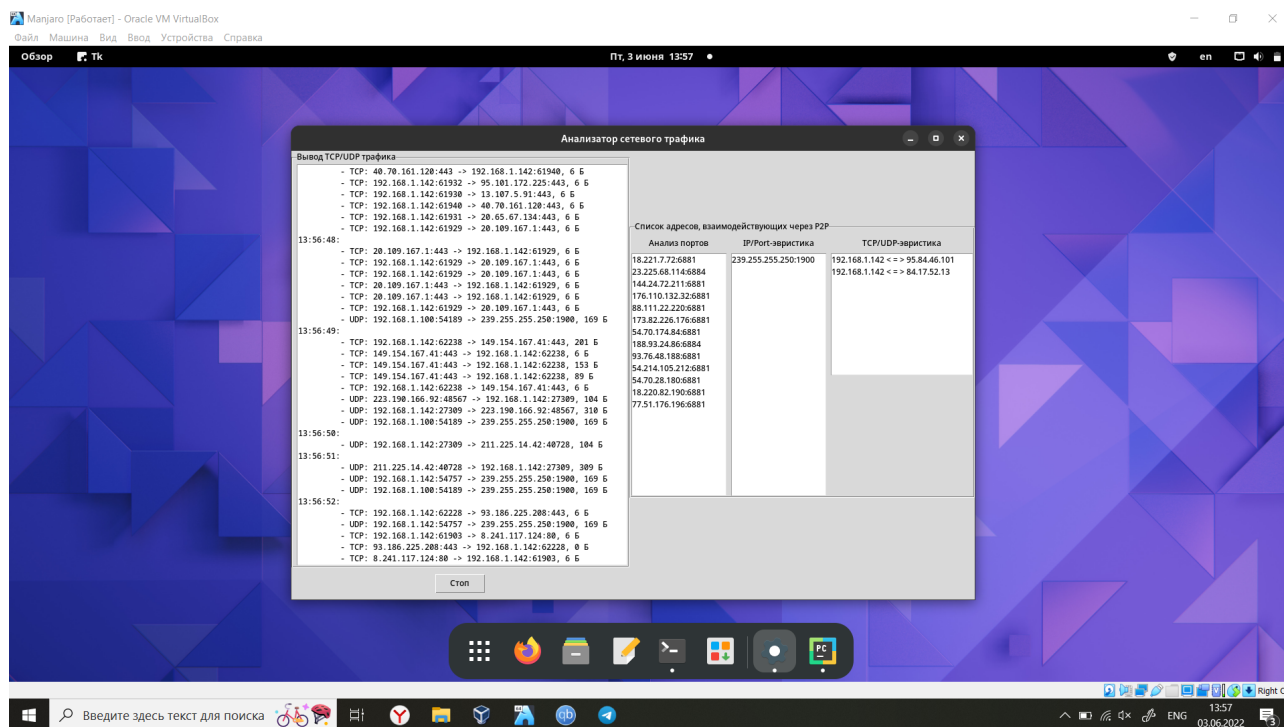


Рисунок 8 – Демонстрация схемы тестирования

5.4.1 Запуск программы при отсутствии P2P-активности

Ни на одной из машин не запущено ни одно P2P-приложение. На виртуальной машине запущено видео на *Youtube*.

На рисунке 9 видно, что присутствуют ложные срабатывания обнаружения P2P-активности. Анализ портов и TCP/UDP-эвристика дали ложный положительный результат.

Адрес 239.255.255.250:1900 обнаружен ложно, поскольку порт 1900 применяется протоколами SSDP и UPnP для обнаружения новых устройств в локальной сети. В качестве одного из методов обнаружения поддерживается M-SEARCH, подразумевающий отправку multicast-запросов по адресу 239.255.255.250. [?]

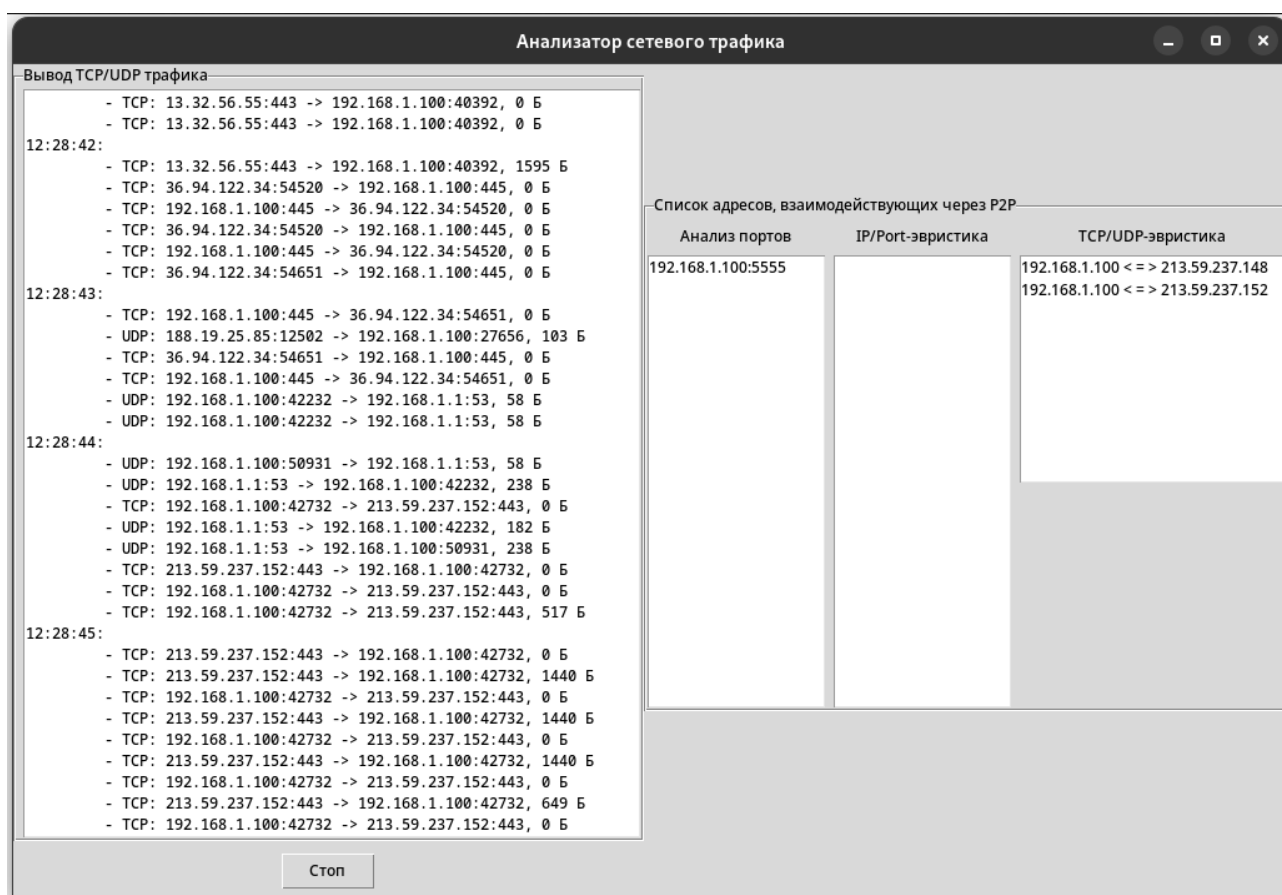


Рисунок 9 – Тестирование программы при запущенном видео на *Youtube*

5.4.2 Запуск программы при запущенном клиенте Bittorrent

На хостовой машине запущен *qBittorrent*.

На рисунке 10 видно, что методом анализирования портов была обнаружена P2P-активность. Порты 6881-6889 относятся к Bittorrent. С помощью IP/Port-эвристики был обнаружен адрес 192.168.1.142:27309. IP данного адреса принадлежит хостовой машине, а порт является портом для входящих соединений в клиенте qBittorrent (рисунок 11). Данный порт является случайным, поэтому методом анализирования портов его не удалось бы обнаружить. Также TCP/UDP-эвристика показывает достаточно большое количество пар адресов, между которыми была обнаружена P2P-активность.

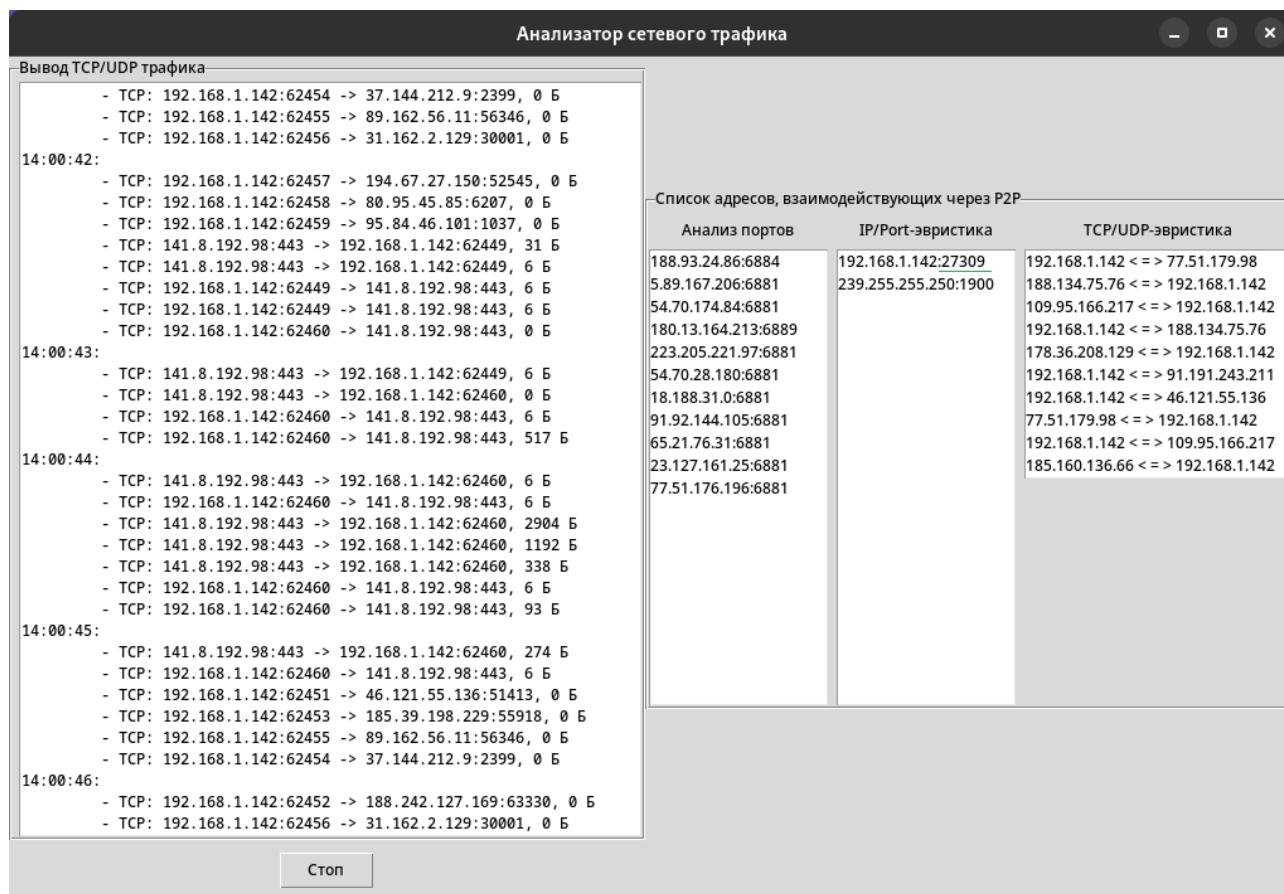


Рисунок 10 – Тестирование программы при запущенном клиенте Bittorrent

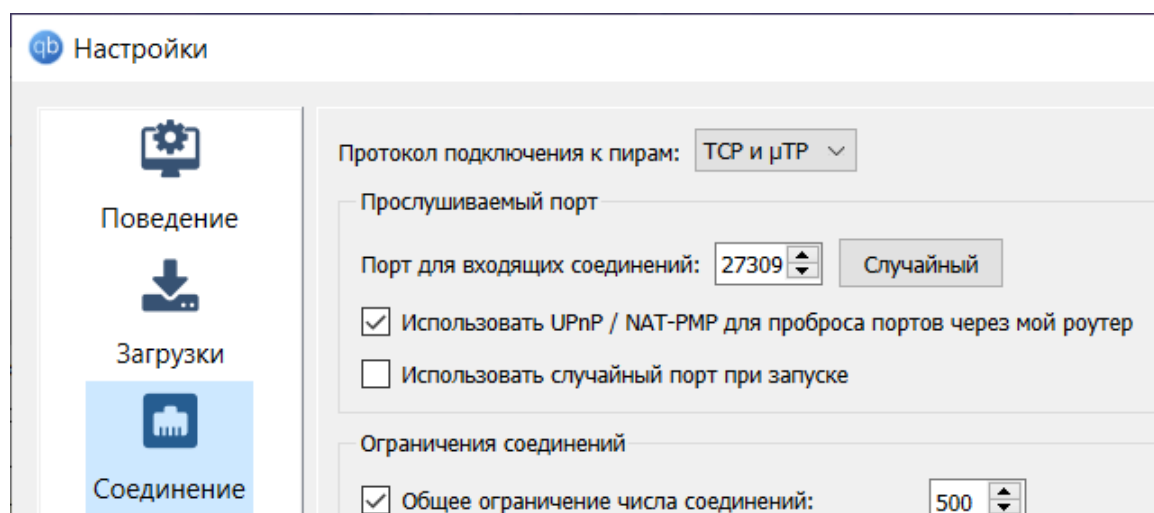


Рисунок 11 – Порт входящих соединений qBittorrent

5.4.3 Запуск программы при запущенном аудио звонке Skype

На хостовой машине запущен аудио звонок в Skype.

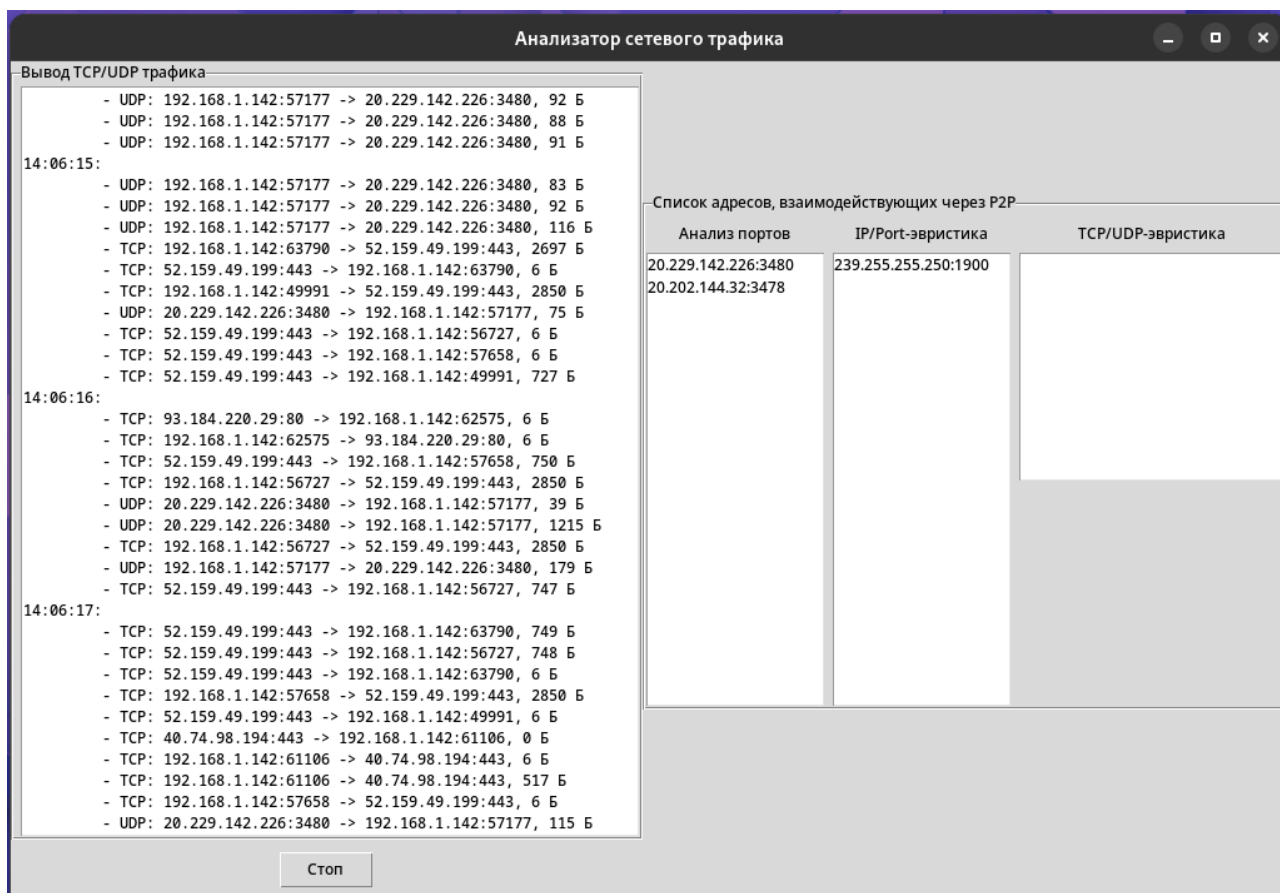


Рисунок 12 – Тестирование программы при запущенном аудио звонке в Skype

P2P-активность аудио звонка Skype легко обнаруживается с помощью анализа портов (порты 3478 и 3480), поскольку его порты не меняются.

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 анализа 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 = 0.1
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         self.label_choose_interface.grid(row=0, column=0)
34
35         self.loi_columns = ['1', '2']
36         self.list_of_interfaces = ttk.Treeview(self.frame_choose_interface,
37                                                show='headings', columns=self.loi_columns,
38                                                ↪ height=10)
39         self.list_of_interfaces.heading('1', text='IP-адрес')
40         self.list_of_interfaces.heading('2', text='Интерфейс')
41         self.list_of_interfaces.grid(row=1, column=0)
42
43         for inter in inters_ips:
44             self.list_of_interfaces.insert(parent='', index='end', values=[inter,
45                                     ↪ 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=25)
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
75     self.p2p_table_1.heading('1', text='Анализ портов')
76     self.p2p_table_2.heading('2', text='IP/Port эвристика')
77     self.p2p_table_3.heading('3', text='TCP/UDP эвристика')
78     self.p2p_table_4.heading('4', text='По полезной нагрузке')
79
80     self.p2p_table_1.column('1', minwidth=0, width=175)
81     self.p2p_table_2.column('2', minwidth=0, width=175)
82     self.p2p_table_3.column('3', minwidth=0, width=175)
83     self.p2p_table_4.column('4', minwidth=0, width=175)
84
85     self.scroll_out = ttk.Scrollbar(self.frame_main, command=self.output.yview)

```

```

86         self.output.config(yscrollcommand=self.scroll_out.set)
87
88         self.stop_btn = ttk.Button(self.frame_main, text='Cmon', command=self.stop)
89
90     def start(self, _):
91         select = self.list_of_interfaces.selection()[0]
92         item = self.list_of_interfaces.item(select)
93         interface = item['values'][1]
94         self.conn, self.osflag = create_socket(interface)
95         self.frame_choose_interface.forget()
96
97         self.frame_main.grid(row=0, column=0)
98         self.output.grid(row=0, column=0, padx=(5, 0), sticky=tk.NW)
99
100        self.frame.grid(row=0, column=1)
101        self.p2p_table_1.grid(row=0, column=0, padx=(5, 0), sticky=tk.NE)
102        self.p2p_table_2.grid(row=0, column=1, padx=(0, 5), sticky=tk.NE)
103        self.p2p_table_3.grid(row=1, column=0, padx=(5, 0), sticky=tk.NE)
104        self.p2p_table_4.grid(row=1, column=1, padx=(0, 5), sticky=tk.NE)
105
106        self.stop_btn.grid(row=1, column=0, pady=(10, 10))
107
108        self.call_sniff()
109        self.call_find_p2p()
110
111        # Авто пролистывание до последней строки при прокручивании колеса мыши вниз
112    def auto_down_scroll(self):
113        last_row = self.output.get_children()[-1]
114        last_row_bbox = self.output.bbox(last_row)
115
116        if len(last_row_bbox) > 0:
117            self.output.see(last_row)
118
119    def call_sniff(self):
120        ready = select.select([self.conn], [], [], SCAN_RATE_S)
121        if ready[0]:
122            out = sniffer.sniff(self.conn, self.osflag)
123            if out:
124                time = str(datetime.now().strftime('%H:%M:%S'))
125                out.insert(0, time)
126                self.output_list.append(out)
127                self.output.insert(parent='', index='end', values=out)
128                # Подсъемка
129                # if out[-1][0:3] == "P2P":
130                #     self.output.insert(parent='', index='end', values=out,
131                ↪     tags=("highlight",))
132                # else:
133                #     self.output.insert(parent='', index='end', values=out)

```



```

133         self.auto_down_scroll()
134
135         # Вывод информации о пакете
136         for s in out:
137             file.write(s + ' ')
138         file.write('\n')
139
140     root.after(SCAN_RATE_MS, self.call_sniff) # сканирование каждые 0.1 сек
141
142     def call_find_p2p(self):
143         sniffer.find_p2p()
144
145         for item_id in self.p2p_table_1.get_children():
146             self.p2p_table_1.delete(item_id)
147         for addr in sniffer.p2p_pairs_p:
148             self.p2p_table_1.insert(parent='', index='end', values=[addr[0] + ":" +
↪ str(addr[1])])
149
150         for item_id in self.p2p_table_2.get_children():
151             self.p2p_table_2.delete(item_id)
152         for addr in sniffer.p2p_pairs_ipp:
153             self.p2p_table_2.insert(parent='', index='end', values=[addr[0] + ":" +
↪ str(addr[1])])
154
155         for item_id in self.p2p_table_3.get_children():
156             self.p2p_table_3.delete(item_id)
157         for addr in sniffer.p2p_addrs_tu:
158             self.p2p_table_3.insert(parent='', index='end', values=[addr])
159
160         for item_id in self.p2p_table_4.get_children():
161             self.p2p_table_4.delete(item_id)
162         for addr in sniffer.bittorrent_addrs:
163             self.p2p_table_4.insert(parent='', index='end', values=[addr[0] + ":" +
↪ str(addr[1])])
164         for addr in sniffer.bitcoin_addrs:
165             self.p2p_table_4.insert(parent='', index='end', values=[addr[0] + ":" +
↪ str(addr[1])])
166
167     root.after(15000, self.call_find_p2p)
168
169     def stop(self):
170         file2.write('Список IP-адресов, взаимодействующих через P2P: \n ')
171         file2.write('Анализ портов: \n ')
172         for row in self.p2p_table_1.get_children():
173             addr = self.p2p_table_1.item(row)['values'][0]
174             file2.write(' * ' + addr + '\n ')
175
176         file2.write('IP/Port-эвентурика: \n ')

```

```

177         for row in self.p2p_table_2.get_children():
178             addr = self.p2p_table_2.item(row)['values'][0]
179             file2.write(' * ' + addr + '\n')
180
181         file2.write('TCP/UDP-эвристика: \n')
182         for row in self.p2p_table_3.get_children():
183             addr = self.p2p_table_3.item(row)['values'][0]
184             file2.write(' * ' + addr + '\n')
185
186         file2.write('По полезной нагрузке: \n')
187         for row in self.p2p_table_4.get_children():
188             addr = self.p2p_table_4.item(row)['values'][0]
189             file2.write(' * ' + addr + '\n')
190
191         file2.write('Конец списка. \n')
192
193         file2.write('\nСписок исключений P2P-адресов: \n')
194         for pair in sniffer.rejected:
195             file2.write(' * ' + pair[0] + ':' + str(pair[1]) + '\n')
196
197         self.conn.close()
198         file2.close()
199         file.close()
200         root.destroy()
201
202
203 def create_socket(interface):
204     try:
205         # Windows needs IP ?
206         if os.name == 'nt':
207             osflag = False
208             conn = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
209             conn.bind((interface, 0))
210             conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
211             conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
212             conn.setblocking(False)
213             # conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
214
215         # Linux needs interface's name
216         else:
217             osflag = True
218
219             if len(sys.argv) > 1:
220                 interface = sys.argv[1]
221                 os.system("ip link set {} promisc on".format(interface)) # ret =
222                 conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
223                 conn.bind((interface, 0))
224                 conn.setblocking(False)

```

```

225         return conn, osflag
226     except socket.error as msg:
227         print('Сокет не может быть создан. Код ошибки : ' + str(msg[0]) + ' Сообщение ' +
                ↪ msg[1])
228         sys.exit()
229
230
231 # Расшифровка названия интерфейса на Windows
232 def get_connection_name_from_guid(iface_guids):
233     iface_names = ['(unknown)' for i in range(len(iface_guids))]
234     reg = wr.ConnectRegistry(None, wr.HKEY_LOCAL_MACHINE)
235     reg_key = wr.OpenKey(reg,
                ↪ r'SYSTEM\CurrentControlSet\Control\Network\{4d36e972-e325-11ce-bfc1-08002be10318}')
236     for i in range(len(iface_guids)):
237         try:
238             reg_subkey = wr.OpenKey(reg_key, iface_guids[i] + r'\Connection')
239             iface_names[i] = wr.QueryValueEx(reg_subkey, 'Name')[0]
240         except FileNotFoundError:
241             pass
242     return iface_names
243
244
245 # For Linux
246 def get_local_interfaces():
247     import array
248     import struct
249     import fcntl
250     """ Returns a dictionary of name:ip key value pairs. """
251     MAX_BYTES = 4096
252     FILL_CHAR = b'\0'
253     SIOCGIFCONF = 0x8912
254     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
255     names = array.array('B', MAX_BYTES * FILL_CHAR)
256     names_address, names_length = names.buffer_info()
257     mutable_byte_buffer = struct.pack('iL', MAX_BYTES, names_address)
258     mutated_byte_buffer = fcntl.ioctl(sock.fileno(), SIOCGIFCONF, mutable_byte_buffer)
259     max_bytes_out, names_address_out = struct.unpack('iL', mutated_byte_buffer)
260     namestr = names.tobytes()
261     namestr[:max_bytes_out]
262     bytes_out = namestr[:max_bytes_out]
263     ip_dict = {}
264     for i in range(0, max_bytes_out, 40):
265         name = namestr[i: i + 16].split(FILL_CHAR, 1)[0]
266         name = name.decode('utf-8')
267         ip_bytes = namestr[i+20:i+24]
268         full_addr = []
269         for netaddr in ip_bytes:
270             if isinstance(netaddr, int):

```

```

271         full_addr.append(str(netaddr))
272     elif isinstance(netaddr, str):
273         full_addr.append(str(ord(netaddr)))
274     # ip_dict[name] = '.'.join(full_addr)
275     ip_dict['.'.join(full_addr)] = name # я сделал наоборот, потому что для линукса у
    ↪ меня нужно имя, а не айпи
276
277     return ip_dict
278
279
280 if __name__ == "__main__":
281     # Получение списка интерфейсов и их IP
282     if os.name == 'nt':
283         osflag = False
284         import winreg as wr
285
286         interfaces = []
287         ips = []
288
289         x = ni.interfaces()
290         for interface in x:
291             addr = ni.ifaddresses(interface)
292             try:
293                 ip = addr[ni.AF_INET][0]['addr']
294                 interfaces.append(interface)
295                 ips.append(ip)
296             except:
297                 pass
298         interfaces = get_connection_name_from_guid(interfaces)
299         inters_ips = dict(zip(interfaces, ips))
300
301     else:
302         osflag = True
303         inters_ips = get_local_interfaces()
304
305         # interfaces = ['enp6s0']
306         # ips = ['192.168.1.132']
307
308
309     # print(ni.ifaddresses(_get_default_iface_linux()).setdefault(ni.AF_INET)[0]['addr'])
310     # print(ni.interfaces())
311
312     # В файл сохраняется последний вывод программы
313     file = open('out.txt', 'w+')
314     # Список IP-адресов, взаимодействующих через P2P
315     file2 = open('ip_list.txt', 'w+')
316
317     root = tk.Tk()

```

```
318     root.title("Анализатор сетевого трафика")
319     menu = Menu(root)
320     root.mainloop()
```

ПРИЛОЖЕНИЕ Б

Код 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: 'AIM',
24             3478: 'Skype', 3479: 'Skype', 3480: 'Skype', 3481: 'Skype',
25             4379: 'Steam', 4380: 'Steam (voice chat)', 27014: 'Steam',
26             27015: 'Steam', 27016: 'Steam', 27017: 'Steam', 27018: 'Steam',
27             27019: 'Steam', 27020: 'Steam', 27021: 'Steam', 27022: 'Steam',
28             27023: 'Steam', 27024: 'Steam', 27025: 'Steam', 27026: 'Steam',
29             27027: 'Steam', 27028: 'Steam', 27029: 'Steam', 27030: 'Steam',
30             899: 'Radmin VPN', 12975: 'Hamachi', 32976: 'Hamachi'}
31
32 # Список портов исключений
33 EXCEPTIONS = {137, 138, 139, 445, 53, 123, 500, 554, 1900, 7070,
34               6970, 1755, 5000, 5001, 6112, 6868, 6899, 6667, 7000, 7514,
35               20, 21, 3396, 66, 1521, 1526, 1524, 22, 23, 25, 513, 543}
36
37 TCP_addrs = set()
38 UDP_addrs = set()
39 p2p_addrs_tu = set() # адреса, взаимодействующие одновременно по TCP и UDP
40 p2p_addrs1 = set() # НЕ ИСПОЛЬЗУЕТСЯ! адреса, которые взаимодействовали с адресами из
    ↪ p2p_addrs1
41 p2p_pairs_p = set() # адреса, порт которых входит в список P2P-портов
42 p2p_pairs_ipp = set() # адреса, подходящие к IPPort эвристике
43 rejected = set() # адреса, не относящиеся к P2P (исключения)
44 dict_ipport = dict() # словарь вида (ip+port -> объект класса IPPort)
```

```

45
46 bittorrent_addrs = set() # адреса, относящиеся к BitTorrent
47 bitcoin_addrs = set() # адреса, относящиеся к Bitcoin
48 bitcoin_phrases = ['version', 'verack', 'addr', 'inv', 'getdata', 'notfound',
↳ 'getblocks',
49                     'getheaders', 'tx', 'block', 'headers', 'getaddr', 'mempool',
↳ 'checkorder',
50                     'submitorder', 'reply', 'ping', 'pong', 'reject', 'filterload',
↳ 'filteradd',
51                     'filterclear', 'merkleblock', 'alert', 'sendheaders', 'feefilter',
52                     'sendcmpct', 'cmpctlblock', 'getblocktxn', 'blocktxn', 'Satoshi']
53
54
55 class IPPort:
56     def __init__(self, dst_ip, dst_port):
57         self.dst_ip = dst_ip
58         self.dst_port = dst_port
59         self.IPSet = set() # IP-адреса источников
60         self.PortSet = set() # Порты источников
61         self.p2p = False # НЕ ИСПОЛЬЗУЕТСЯ
62
63     def add_sources(self, ip, port):
64         self.IPSet.add(ip)
65         self.PortSet.add(port)
66
67     # Добавление в p2p_addrs1 адресов, которые взаимодействовали с адресами из p2p_addrs_tu
68     def add_to_p2p_addrs1(self):
69         for ip in self.IPSet:
70             if ip not in [ipport[0] for ipport in rejected]:
71                 # добавляю в p2p_addrs_tu, чтобы относилось к одной эвристике, хотя по сути
↳ это p2p_addrs1
72                 p2p_addrs_tu.add('(' + ip)
73
74
75     def sniff(conn, os):
76         output = ''
77         data, addr = conn.recvfrom(65536)
78         if os:
79             dest_mac, src_mac, eth_proto, data = ethernet_frame(data)
80         else:
81             eth_proto = 8
82
83     # IPv4
84     if eth_proto == 8:
85         version, header_length, ttl, proto, src, dest, data = ipv4_packet(data)
86
87         if proto == 6 or proto == 17:
88

```

```

89         # TCP
90         if proto == 6:
91             src_port, dest_port, data = tcp_segment(data)
92
93             check_exceptions(src, dest, src_port, dest_port)
94             if (src, src_port) not in rejected and (dest, dest_port) not in rejected:
95                 TCP_addrs.add((src, dest))
96
97             add_ipport(dest, dest_port, src, src_port)
98             addition_info = add_info(src, dest, src_port, dest_port)
99             output = [src, dest, str(src_port) + ' -> ' + str(dest_port), 'TCP',
100                      ↪ str(len(data)) + ' Б',
101                      addition_info]
102
103         # UDP
104         else:
105             src_port, dest_port, length, data = udp_segment(data)
106
107             check_exceptions(src, dest, src_port, dest_port)
108             if (src, src_port) not in rejected and (dest, dest_port) not in rejected:
109                 UDP_addrs.add((src, dest))
110
111             add_ipport(dest, dest_port, src, src_port)
112             addition_info = add_info(src, dest, src_port, dest_port)
113             output = [src, dest, str(src_port) + ' -> ' + str(dest_port), 'UDP',
114                      ↪ str(len(data)) + ' Б',
115                      addition_info]
116
117         payload_analysis(src, dest, src_port, dest_port, data)
118
119     return output
120
121     # Раньше была функция check_ports, теперь после проверки портов функция
122     # добавляет к строке вывода информацию для столбца info, если адрес p2p и добавляет протокол
123     ↪ по возможности
124     def add_info(src, dest, src_port, dest_port):
125         addition_info = ''
126         if LIST_P2P.get(src_port, False):
127             p2p_pairs_p.add((src, src_port))
128             addition_info = 'P2P ' + LIST_P2P[src_port]
129         elif LIST_P2P.get(dest_port, False):
130             p2p_pairs_p.add((dest, dest_port))
131             addition_info = 'P2P ' + LIST_P2P[dest_port]
132         elif (src, src_port) in bittorrent_addrs:
133             addition_info = 'P2P BitTorrent'
134         elif (dest, dest_port) in bittorrent_addrs:
135             addition_info = 'P2P BitTorrent'

```



```

134     elif (src, src_port) in bitcoin_addrs:
135         addition_info = 'P2P Bitcoin'
136     elif (dest, dest_port) in bitcoin_addrs:
137         addition_info = 'P2P Bitcoin'
138     return addition_info
139
140
141 def add_ipport(dest, dest_port, src, src_port):
142     ipport = dest + ':' + str(dest_port)
143     if ipport not in dict_ipport:
144         x = IPPort(dest, dest_port)
145         x.add_sources(src, src_port)
146         dict_ipport[ipport] = x
147     else:
148         dict_ipport[ipport].add_sources(src, src_port)
149
150
151 # Добавление адресов с портами в список исключений
152 def check_exceptions(src, dest, src_port, dest_port):
153     if src_port in EXCEPTIONS \
154         or dest_port in EXCEPTIONS \
155         or (src_port == dest_port and src_port < 500):
156         rejected.add((src, src_port))
157         rejected.add((dest, dest_port))
158
159
160 # Анализ полезной нагрузки пакетов,
161 def payload_analysis(src, dest, src_port, dest_port, data):
162     # Для BitTorrent
163     sdata = str(data)
164     if len(data) >= 20
165         if 'BitTorrent protocol' in sdata:
166             bittorrent_addrs.add((src, src_port))
167             bittorrent_addrs.add((dest, dest_port))
168         elif src_port == 8333 or dest_port == 8333 or src_port == 8334 or dest_port == 8334:
169             # print(sdata)
170             for word in bitcoin_phrases:
171                 if word in sdata:
172                     bitcoin_addrs.add((src, src_port))
173                     bitcoin_addrs.add((dest, dest_port))
174                 break
175
176
177 def find_p2p():
178     # 1 Заполнение p2p_addrs адресами, взаимодействующими одновременно по TCP и UDP
179     inter = TCP_addrs & UDP_addrs
180     for addrs in inter:
181         p2p_addrs_tu.add(addrs[0])

```

```

182         p2p_addrs_tu.add(addr[1])
183
184     # 2 Заполнение p2p_pairs_ipp адресами, выбранными исходя из check_p2p
185     for ipp in dict_ipp:
186         ipp = dict_ipp[ipp]
187
188         ip = ipp.dst_ip
189         port = ipp.dst_port
190
191         # Добавление адресов, взаимодействующие с адресами из TCP/UDP пар
192         if ip in p2p_addrs_tu:
193             ipp.add_to_p2p_addrs1()
194
195         compare_dif = 2
196
197         # Если порт из известных p2p портов, то разница между IPSet и PortSet должна быть
198         ↪ увеличена до 10
199         if ipp in p2p_pairs_p:
200             compare_dif = 10
201
202         cur_dif = len(ipp.IPSet) - len(ipp.PortSet)
203         if len(ipp.IPSet) > 2 and (cur_dif < compare_dif):
204             if (ip, port) not in rejected:
205                 p2p_pairs_ipp.add((ip, port))
206
207         # Если разница больше 10, то, скорее всего, это не p2p и можно добавить в
208         ↪ исключения.
209         elif cur_dif > 10:
210             rejected.add((ip, port))
211
212     # Распаковка ethernet кадра
213     def ethernet_frame(data):
214         dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
215         return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto), data[14:]
216
217     # Форматирование MAC-адреса
218     def get_mac_addr(bytes_addr):
219         bytes_str = map('{:02x}'.format, bytes_addr)
220         return ':'.join(bytes_str).upper()
221
222     # Распаковка IPv4 пакета
223     def ipv4_packet(data):
224         version_header_length = data[0]
225         version = version_header_length >> 4
226         header_length = (version_header_length & 15) * 4

```

```

228     ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
229     return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]
230
231
232     # Форматирование IP-адреса
233     def ipv4(addr):
234         return '.'.join(map(str, addr))
235
236
237     # Распаковка TCP сегмента
238     def tcp_segment(data):
239         (src_port, dest_port, sequence, ack, offset_reserved_flags) = struct.unpack('! H H L L
        ↪  H', data[:14])
240         offset = (offset_reserved_flags >> 12) * 4
241         flag_urg = (offset_reserved_flags & 32) >> 5
242         flag_ack = (offset_reserved_flags & 16) >> 5
243         flag_psh = (offset_reserved_flags & 8) >> 5
244         flag_rst = (offset_reserved_flags & 4) >> 5
245         flag_syn = (offset_reserved_flags & 2) >> 5
246         flag_fin = offset_reserved_flags & 1
247         return src_port, dest_port, data[offset:]
248
249
250     # Распаковка UDP сегмента
251     def udp_segment(data):
252         src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
253         return src_port, dest_port, size, data[8:]

```