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

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. [**?**]

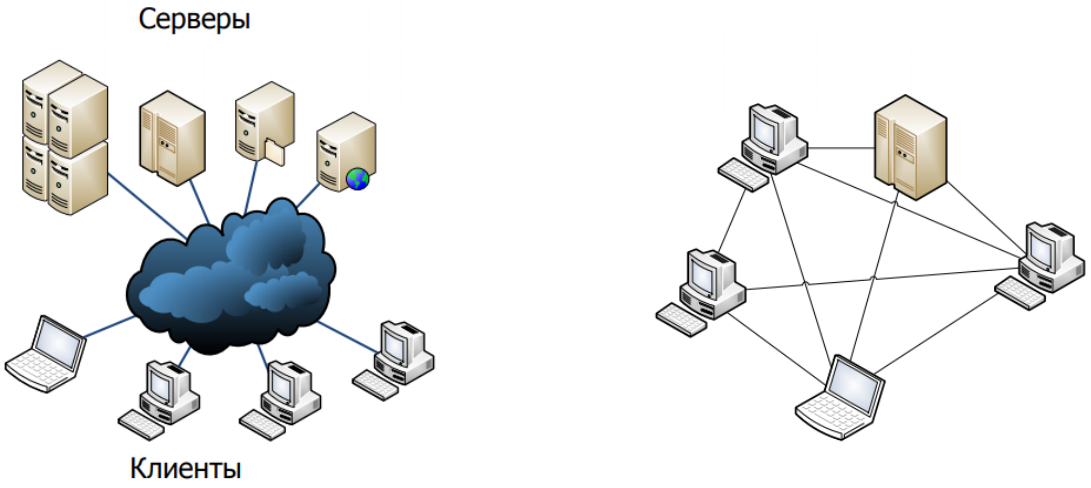
В мае 1999 года, в Интернет с более чем миллионом пользователей, Шон Фэннинг внедрил приложение файлообменник Napster. Napster стал началом P2P-сети, такой какую мы знаем её сейчас, пользователи участвуют в создании виртуальной сети, полностью независимой от физической, без администрирования и каких-либо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

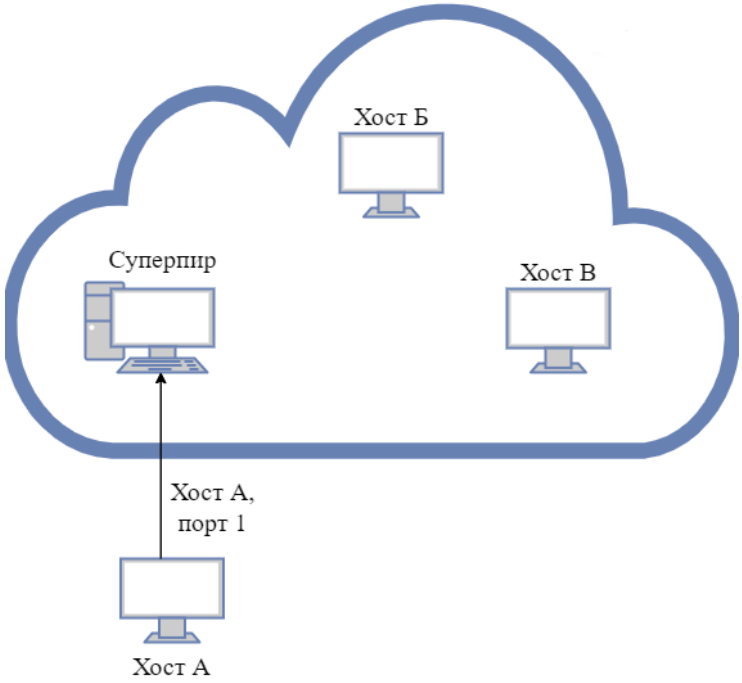


Рисунок 2 – Отправка информации хоста А о себе суперпиру

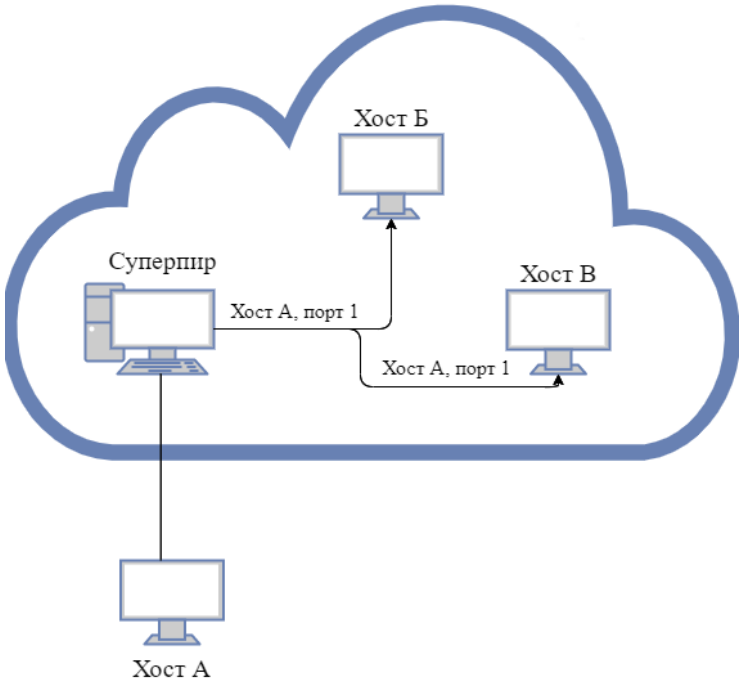


Рисунок 3 – Суперпир распространяет информацию о хосте А остальной части сети

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.

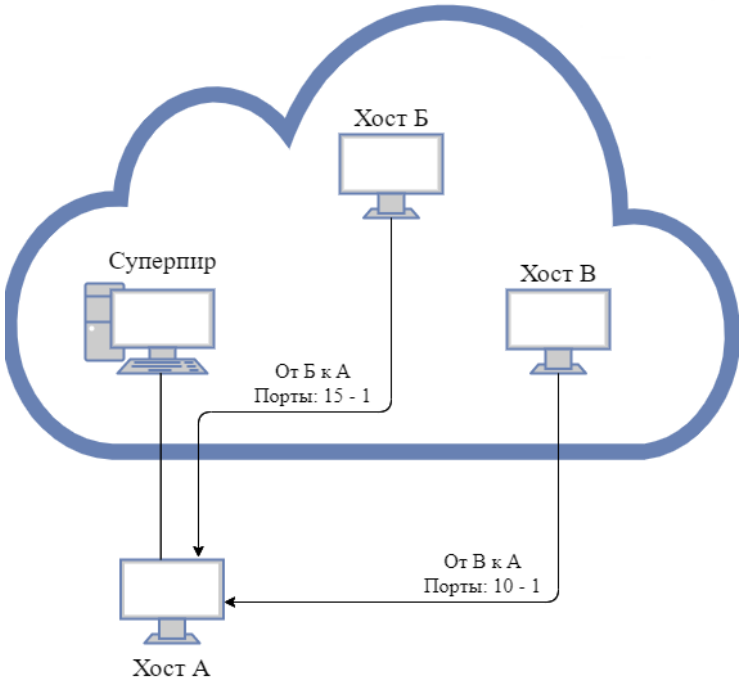


Рисунок 4 – К хосту A подключены хосты Б и В с 2 разными IP и 2 разными портами

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).

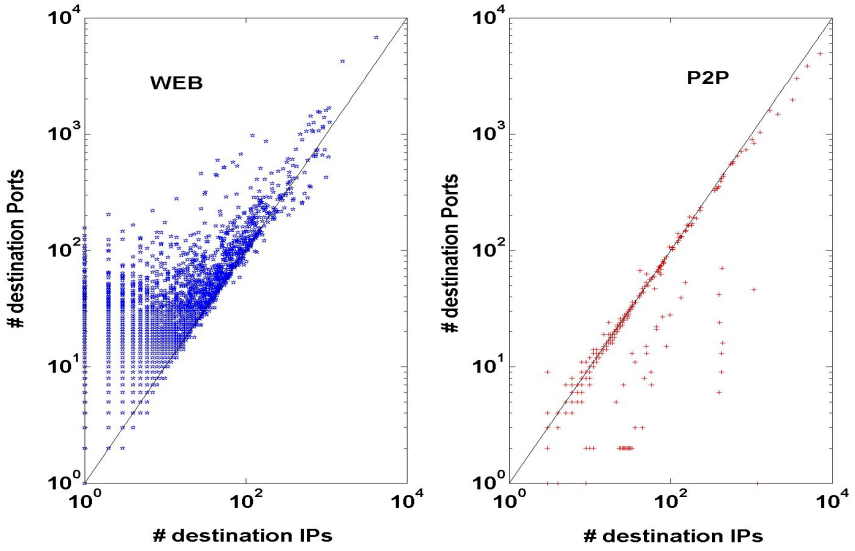


Рисунок 5

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 $R_{RC} = \frac{RC}{AT}$.

Точность данного алгоритма зависит от выбранных пороговых значений. Они могут быть получены эмпирическим путём.

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.

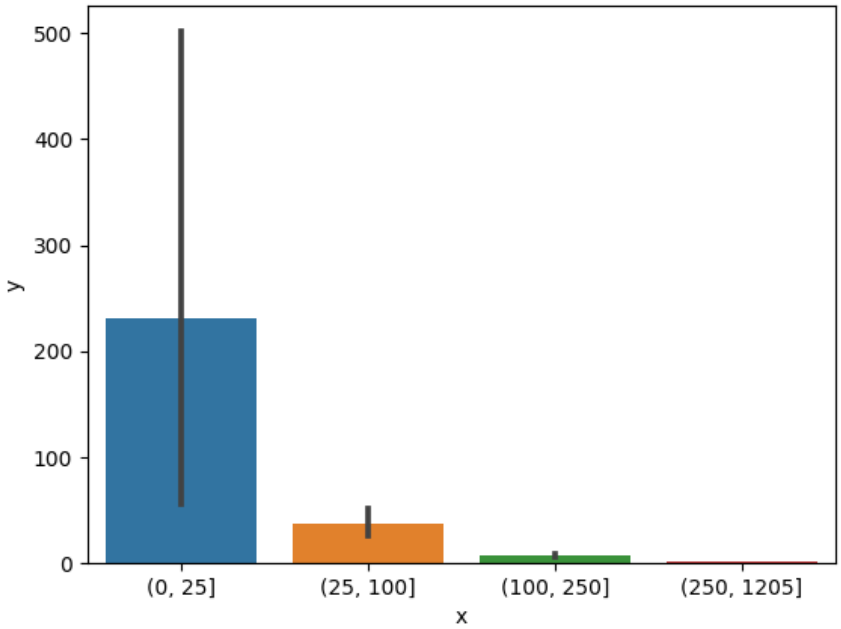


Рисунок 6 – Графическое представление среднего количества пакетов различных диапазонов их размеров в игре War Thunder

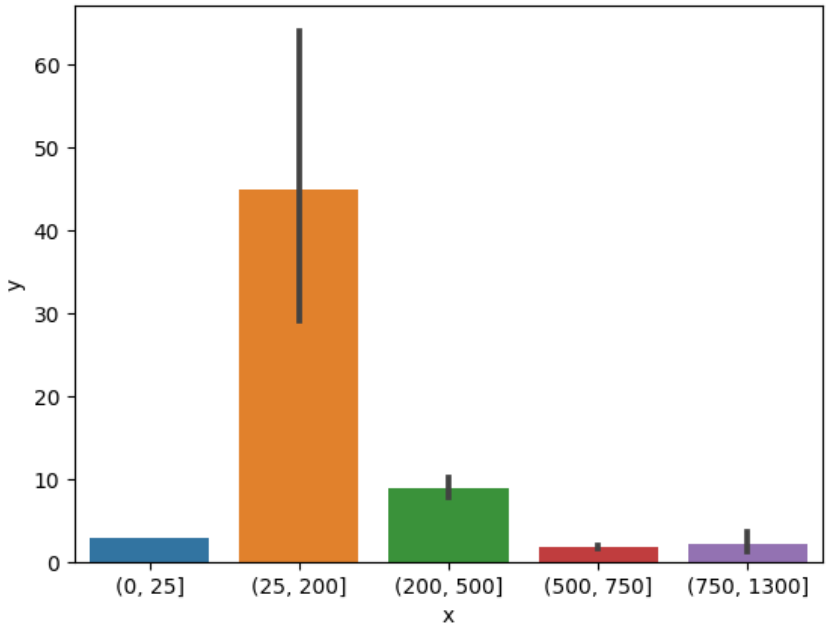


Рисунок 7 – Графическое представление среднего количества пакетов различных диапазонов их размеров в игре Dota 2

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 поведе-

ние 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

Описание программы.

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=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         self.call_bt_stats()
111
112     # Авто пролистывание до последней строки при прокручивании колеса мыши вниз
113     def auto_down_scroll(self):
114         last_row = self.output.get_children()[-1]
115         last_row_bbox = self.output.bbox(last_row)
116
117         if len(last_row_bbox) > 0:
118             self.output.see(last_row)
119
120     def call_sniff(self):
121         ready = select.select([self.conn], [], [], SCAN_RATE_S)
122         if ready[0]:
123             out = sniffer.sniff(self.conn, self.osflag)
124             if out:
125                 time = str(datetime.now().strftime('%H:%M:%S'))
126                 out.insert(0, time)
127                 self.output_list.append(out)
128                 self.output.insert(parent='', index='end', values=out)
129                 # Подсветка
130                 # if out[-1][0:3] == "P2P":
131                 #     self.output.insert(parent='', index='end', values=out,
                    ↪  tags=("highlight",))
132                 # else:
```

```
133                 #     self.output.insert(parent='', index='end', values=out)
134                 self.auto_down_scroll()
135
136                 # Вывод информации о пакете
137                 for s in out:
138                     file.write(s + ' ')
139                 file.write('\n')
140
141         root.after(SCAN_RATE_MS, self.call_sniff)  # сканирование каждые 0.1 сек
142
143     def call_find_p2p(self):
144         sniffer.find_p2p()
145
146         for item_id in self.p2p_table_1.get_children():
147             self.p2p_table_1.delete(item_id)
148         for addr in sniffer.p2p_pairs_p:
149             self.p2p_table_1.insert(parent='', index='end', values=[addr[0] + ":" +
                ↪ str(addr[1])])
150
151         for item_id in self.p2p_table_2.get_children():
152             self.p2p_table_2.delete(item_id)
153         for addr in sniffer.p2p_pairs_ipp:
154             self.p2p_table_2.insert(parent='', index='end', values=[addr[0] + ":" +
                ↪ str(addr[1])])
155
156         for item_id in self.p2p_table_3.get_children():
157             self.p2p_table_3.delete(item_id)
158         for addr in sniffer.p2p_addrs_tu:
159             self.p2p_table_3.insert(parent='', index='end', values=[addr])
160
161         for item_id in self.p2p_table_4.get_children():
162             self.p2p_table_4.delete(item_id)
163         for addr in sniffer.bittorrent_addrs:
164             self.p2p_table_4.insert(parent='', index='end', values=[addr[0] + ":" +
                ↪ str(addr[1])])
165         for addr in sniffer.bitcoin_addrs:
166             self.p2p_table_4.insert(parent='', index='end', values=[addr[0] + ":" +
                ↪ str(addr[1])])
167
168         root.after(15000, self.call_find_p2p)
169
170     def call_bt_stats(self):
171         for ipp in sniffer.dict_ipport:
172             c, at, bi, rc = sniffer.dict_ipport[ipp].bt_stats()
173             if c > 0 and at > 0:
174                 print(ipp)
175                 print(c, at, bi, rc)
176
```

```
177         root.after(30000, self.call_bt_stats)
178 
179     def stop(self):
180         file2.write('Список IP-адресов, взаимодействующих через P2P: \n')
181         file2.write('Анализ портов: \n')
182         for row in self.p2p_table_1.get_children():
183             addr = self.p2p_table_1.item(row)['values'][0]
184             file2.write(' * ' + addr + '\n')
185 
186         file2.write('IP/Port-эвристика: \n')
187         for row in self.p2p_table_2.get_children():
188             addr = self.p2p_table_2.item(row)['values'][0]
189             file2.write(' * ' + addr + '\n')
190 
191         file2.write('TCP/UDP-эвристика: \n')
192         for row in self.p2p_table_3.get_children():
193             addr = self.p2p_table_3.item(row)['values'][0]
194             file2.write(' * ' + addr + '\n')
195 
196         file2.write('По полезной нагрузке: \n')
197         for row in self.p2p_table_4.get_children():
198             addr = self.p2p_table_4.item(row)['values'][0]
199             file2.write(' * ' + addr + '\n')
200 
201         file2.write('Конец списка. \n')
202 
203         file2.write('\nСписок исключений P2P-адресов: \n')
204         for pair in sniffer.rejected:
205             file2.write(' * ' + pair[0] + ':' + str(pair[1]) + '\n')
206 
207         self.conn.close()
208         file2.close()
209         file.close()
210         root.destroy()
211 
212 
213 def create_socket(interface):
214     try:
215         # Windows needs IP ?
216         if os.name == 'nt':
217             osflag = False
218             conn = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
219             conn.bind((interface, 0))
220             conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
221             conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
222             conn.setblocking(False)
223             # conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
224 
```

```
225         # Linux needs interface's name
226         else:
227             osflag = True
228 
229             if len(sys.argv) > 1:
230                 interface = sys.argv[1]
231             os.system("ip link set {} promisc on".format(interface))  # ret =
232             conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
233             conn.bind((interface, 0))
234             conn.setblocking(False)
235         return conn, osflag
236     except socket.error as msg:
237         print('Сокет не может быть создан. Код ошибки : ' + str(msg[0]) + ' Сообщение ' +
        ↪ msg[1])
238         sys.exit()
239 
240 
241 # Расшифровка названия интерфейса на Windows
242 def get_connection_name_from_guid(iface_guids):
243     iface_names = ['(unknown)' for i in range(len(iface_guids))]
244     reg = wr.ConnectRegistry(None, wr.HKEY_LOCAL_MACHINE)
245     reg_key = wr.OpenKey(reg,
    ↪ r'SYSTEM\CurrentControlSet\Control\Network\{4d36e972-e325-11ce-bfc1-08002be10318}')
246     for i in range(len(iface_guids)):
247         try:
248             reg_subkey = wr.OpenKey(reg_key, iface_guids[i] + r'\Connection')
249             iface_names[i] = wr.QueryValueEx(reg_subkey, 'Name')[0]
250         except FileNotFoundError:
251             pass
252     return iface_names
253 
254 
255 #  For Linux
256 def get_local_interfaces():
257     import array
258     import struct
259     import fcntl
260     """ Returns a dictionary of name:ip key value pairs. """
261     MAX_BYTES = 4096
262     FILL_CHAR = b'\0'
263     SIOCGIFCONF = 0x8912
264     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
265     names = array.array('B', MAX_BYTES * FILL_CHAR)
266     names_address, names_length = names.buffer_info()
267     mutable_byte_buffer = struct.pack('iL', MAX_BYTES, names_address)
268     mutated_byte_buffer = fcntl.ioctl(sock.fileno(), SIOCGIFCONF, mutable_byte_buffer)
269     max_bytes_out, names_address_out = struct.unpack('iL', mutated_byte_buffer)
270     namestr = names.tobytes()
```

```
271     namestr[:max_bytes_out]
272     bytes_out = namestr[:max_bytes_out]
273     ip_dict = {}
274     for i in range(0, max_bytes_out, 40):
275         name = namestr[i: i + 16].split(FILL_CHAR, 1)[0]
276         name = name.decode('utf-8')
277         ip_bytes = namestr[i+20:i+24]
278         full_addr = []
279         for netaddr in ip_bytes:
280             if isinstance(netaddr, int):
281                 full_addr.append(str(netaddr))
282             elif isinstance(netaddr, str):
283                 full_addr.append(str(ord(netaddr)))
284         # ip_dict[name] = '.'.join(full_addr)
285         ip_dict['.'.join(full_addr)] = name # я сделал наоборот, потому что для линукса у
        ↪  меня нужно имя, а не айпи
286
287     return ip_dict
288
289
290 if __name__ == "__main__":
291     # Получение списка интерфейсов и их IP
292     if os.name == 'nt':
293         osflag = False
294         import winreg as wr
295
296         interfaces = []
297         ips = []
298
299         x = ni.interfaces()
300         for interface in x:
301             addr = ni.ifaddresses(interface)
302             try:
303                 ip = addr[ni.AF_INET][0]['addr']
304                 interfaces.append(interface)
305                 ips.append(ip)
306             except:
307                 pass
308         interfaces = get_connection_name_from_guid(interfaces)
309         inters_ips = dict(zip(interfaces, ips))
310
311     else:
312         osflag = True
313         inters_ips = get_local_interfaces()
314
315         # interfaces = ['enp6s0']
316         # ips = ['192.168.1.132']
317
```

```
318
319     # print(ni.ifaddresses(_get_default_iface_linux()).setdefault(ni.AF_INET)[0]['addr'])
320     # print(ni.interfaces())
321
322     # В файл сохраняется последний вывод программы
323     file = open('out.txt', 'w+')
324     # Список IP-адресов, взаимодействующих через P2P
325     file2 = open('ip_list.txt', 'w+')
326
327     root = tk.Tk()
328     root.title("Анализатор сетевого трафика")
329     menu = Menu(root)
330     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}

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
p2p_pairs_ipp = set()  # адреса, подходящие к IPPort эвристике
rejected = set()  # адреса, не относящиеся к P2P (исключения)
dict_ipport = dict()  # словарь вида (ip+port -> объект класса IPPort)

bittorrent_addrs = set()  # адреса, относящиеся к BitTorrent
```

```
46  bitcoin_addrs = set()  # адреса, относящиеся к Bitcoin
47  bitcoin_phrases = ['version', 'verack', 'addr', 'inv', 'getdata', 'notfound',
    ↪  'getblocks',
48                     'getheaders', 'tx', 'block', 'headers', 'getaddr', 'mempool',
                       ↪  'checkorder',
49                     'submitorder', 'reply', 'ping', 'pong', 'reject', 'filterload',
                       ↪  'filteradd',
50                     'filterclear', 'merkleblock', 'alert', 'sendheaders', 'feefilter',
51                     'sendcmpct', 'cmpctlblock', 'getblocktxn', 'blocktxn', 'Satoshi']
52
53
54  class IPPort:
55      def __init__(self, dst_ip, dst_port):
56          self.dst_ip = dst_ip
57          self.dst_port = dst_port
58          self.IPSet = set()  # IP-адреса источников
59          self.PortSet = set()  # Порты источников
60          self.srcs = set()
61          self.in_packets = dict()
62          self.dest_addrs = set()
63          self.old_bi = set()
64          self.p2p = False  # НЕ ИСПОЛЬЗУЕТСЯ
65
66      def add_sources(self, ip, port):
67          self.IPSet.add(ip)
68          self.PortSet.add(port)
69          self.srcs.add((ip, port))
70
71      def add_packets(self, src_addr, size):
72          if src_addr in self.in_packets.keys():
73              self.in_packets[src_addr].append(size)
74          else:
75              self.in_packets[src_addr] = [size]
76
77      def add_out_addrs(self, dest_addr):
78          self.dest_addrs.add(dest_addr)
79
80      # Добавление в p2p_addrs1 адресов, которые взаимодействовали с адресами из p2p_addrs_tu
81      def add_to_p2p_addrs1(self):
82          for ip in self.IPSet:
83              if ip not in [ipport[0] for ipport in rejected]:
84                  # добавляю в p2p_addrs_tu, чтобы относилось к одной эвристике, хотя по сути
                    ↪  это p2p_addrs1
85                  p2p_addrs_tu.add('(*) ' + ip)
86
87      def bt_stats(self):
88          # 1
89          c = len(self.srcs)
```

```
90          self.srcs = set()
91
92          # 2
93          at = 0
94          for addr in self.in_packets:
95              packets = self.in_packets[addr]
96              pack_size = len(packets)
97              if pack_size > 4:
98                  average_size = 0
99                  for p in packets:
100                     average_size += p
101                 average_size /= pack_size
102                 # 1250 или больше поставить?
103                 if average_size > 1250:
104                     at += 1
105
106         # 3
107         bi = self.in_packets.keys() & self.dest_addrs
108
109         # 4
110         if len(bi) > len(self.old_bi):
111             rc = len(bi - self.old_bi)
112         else:
113             rc = len(self.old_bi - bi)
114
115         self.old_bi = bi
116
117         return c, at, len(bi), rc
118
119 def sniff(conn, os):
120     output = ''
121     data, addr = conn.recvfrom(65536)
122     if os:
123         dest_mac, src_mac, eth_proto, data = ethernet_frame(data)
124     else:
125         eth_proto = 8
126
127     # IPv4
128     if eth_proto == 8:
129         version, header_length, ttl, proto, src, dest, data = ipv4_packet(data)
130
131         if proto == 6 or proto == 17:
132
133             # TCP
134             if proto == 6:
135                 src_port, dest_port, data = tcp_segment(data)
136
137                 check_exceptions(src, dest, src_port, dest_port)
```

```
138                 if (src, src_port) not in rejected and (dest, dest_port) not in rejected:
139                     TCP_addrs.add((src, dest))
140 
141                 addition_info = add_info(src, dest, src_port, dest_port)
142                 output = [src, dest, str(src_port) + ' -> ' + str(dest_port), 'TCP',
                    ↪  str(len(data)) + ' Б',
143                           addition_info]
144 
145             # UDP
146             else:
147                 src_port, dest_port, length, data = udp_segment(data)
148 
149                 check_exceptions(src, dest, src_port, dest_port)
150                 if (src, src_port) not in rejected and (dest, dest_port) not in rejected:
151                     UDP_addrs.add((src, dest))
152 
153                 addition_info = add_info(src, dest, src_port, dest_port)
154                 output = [src, dest, str(src_port) + ' -> ' + str(dest_port), 'UDP',
                    ↪  str(len(data)) + ' Б',
155                           addition_info]
156 
157             add_ipport(dest, dest_port, src, src_port, len(data))
158             payload_analysis(src, dest, src_port, dest_port, data)
159 
160         return output
161 
162 
163 # после проверки портов функция
164 # добавляет к строке вывода информацию для столбца info,
165 # если адрес p2p и добавляет протокол по возможности
166 def add_info(src, dest, src_port, dest_port):
167     addition_info = ''
168     if LIST_P2P.get(src_port, False):
169         p2p_pairs_p.add((src, src_port))
170         addition_info = 'P2P ' + LIST_P2P[src_port]
171     elif LIST_P2P.get(dest_port, False):
172         p2p_pairs_p.add((dest, dest_port))
173         addition_info = 'P2P ' + LIST_P2P[dest_port]
174     elif (src, src_port) in bittorrent_addrs:
175         addition_info = 'P2P BitTorrent'
176     elif (dest, dest_port) in bittorrent_addrs:
177         addition_info = 'P2P BitTorrent'
178     elif (src, src_port) in bitcoin_addrs:
179         addition_info = 'P2P Bitcoin'
180     elif (dest, dest_port) in bitcoin_addrs:
181         addition_info = 'P2P Bitcoin'
182     return addition_info
183 
```

```
184
185 def add_ipport(dest, dest_port, src, src_port, size):
186     ipport = dest + ':' + str(dest_port)
187     if ipport not in dict_ipport:
188         x = IPPort(dest, dest_port)
189         x.add_sources(src, src_port)
190         dict_ipport[ipport] = x
191         x.add_packets(src + ':' + str(src_port), size)
192     else:
193         dict_ipport[ipport].add_sources(src, src_port)
194         dict_ipport[ipport].add_packets(src + ':' + str(src_port), size)
195
196     ipport_src = src + ':' + str(src_port)
197     if ipport_src in dict_ipport:
198         dict_ipport[ipport_src].add_out_addrs(ipport)
199
200
201 # Добавление адресов с портами в список исключений
202 def check_exceptions(src, dest, src_port, dest_port):
203     if src_port in EXCEPTIONS \
204             or dest_port in EXCEPTIONS \
205             or (src_port == dest_port and src_port < 500):
206         rejected.add((src, src_port))
207         rejected.add((dest, dest_port))
208
209
210 # Анализ полезной нагрузки пакетов,
211 def payload_analysis(src, dest, src_port, dest_port, data):
212     # Для BitTorrent
213     sdata = str(data)
214     if len(data) >= 20:
215         if 'BitTorrent protocol' in sdata:
216             bittorrent_addrs.add((src, src_port))
217             bittorrent_addrs.add((dest, dest_port))
218         elif src_port == 8333 or dest_port == 8333 or src_port == 8334 or dest_port == 8334:
219             # print(sdata)
220             for word in bitcoin_phrases:
221                 if word in sdata:
222                     bitcoin_addrs.add((src, src_port))
223                     bitcoin_addrs.add((dest, dest_port))
224                     break
225
226
227 def find_p2p():
228     # 1 Заполнение p2p_addrs адресами, взаимодействующими одновременно по TCP и UDP
229     inter = TCP_addrs & UDP_addrs
230     for addrs in inter:
231         p2p_addrs_tu.add(addrs[0])
```

```
232         p2p_addrs_tu.add(addrs[1])
233 
234     # 2 Заполнение p2p_pairs_ipp адресами, выбранными исходя из check_p2p
235     for ipport in dict_ipport:
236         ipp = dict_ipport[ipport]
237 
238         ip = ipp.dst_ip
239         port = ipp.dst_port
240 
241         # Добавление адресов, взаимодействующие с адресами из TCP/UDP пар
242         if ip in p2p_addrs_tu:
243             ipp.add_to_p2p_addrs1()
244 
245         compare_dif = 2
246 
247         # Если порт из известных p2p портов, то разница должна быть увеличена до 10
248         if ipport in p2p_pairs_p:
249             compare_dif = 10
250 
251         cur_dif = len(ipp.IPSet) - len(ipp.PortSet)
252         if len(ipp.IPSet) > 2 and (cur_dif < compare_dif):
253             if (ip, port) not in rejected:
254                 p2p_pairs_ipp.add((ip, port))
255 
256         # Если разница больше 10, то, скорее всего, это не p2p и можно добавить в
        ↪  исключения.
257         elif cur_dif > 10:
258             rejected.add((ip, port))
259 
260 
261 # Распаковка ethernet кадра
262 def ethernet_frame(data):
263     dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
264     return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto), data[14:]
265 
266 
267 # Форматирование MAC-адреса
268 def get_mac_addr(bytes_addr):
269     bytes_str = map('{:02x}'.format, bytes_addr)
270     return ':'.join(bytes_str).upper()
271 
272 
273 # Распаковка IPv4 пакета
274 def ipv4_packet(data):
275     version_header_length = data[0]
276     version = version_header_length >> 4
277     header_length = (version_header_length & 15) * 4
278     ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
```

```
279     return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]
280
281
282 # Форматирование IP-адреса
283 def ipv4(addr):
284     return '.'.join(map(str, addr))
285
286
287 # Распаковка TCP сегмента
288 def tcp_segment(data):
289     (src_port, dest_port, _, _, offset_reserved_flags) = struct.unpack('! H H L L H',
        ↪ data[:14])
290     offset = (offset_reserved_flags >> 12) * 4
291     # flag_urg = (offset_reserved_flags & 32) >> 5
292     # flag_ack = (offset_reserved_flags & 16) >> 5
293     # flag_psh = (offset_reserved_flags & 8) >> 5
294     # flag_rst = (offset_reserved_flags & 4) >> 5
295     # flag_syn = (offset_reserved_flags & 2) >> 5
296     # flag_fin = offset_reserved_flags & 1
297     return src_port, dest_port, data[offset:]
298
299
300 # Распаковка UDP сегмента
301 def udp_segment(data):
302     src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
303     return src_port, dest_port, size, data[8:]
```