

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 3 курса 331 группы  
направления 10.05.01 — Компьютерная безопасность  
факультета КНИИТ  
Стаина Романа Игоревича

Научный руководитель  
доцент

\_\_\_\_\_

А. В. Гортинский

Заведующий кафедрой  
д. ф.-м. н., доцент

\_\_\_\_\_

М. Б. Абросимов

Саратов 2022

## 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 Способы обнаружения P2P трафика .....                 | 11 |
| 3.1 Анализ портов .....                                 | 11 |
| 3.2 Анализ сигнатур .....                               | 11 |
| 3.3 Эвристические предположения .....                   | 12 |
| 3.3.1 TCP/UDP-эвристика .....                           | 12 |
| 3.3.2 IP/Port-эвристика .....                           | 12 |
| 4 Идентификация Bittorrent.....                         | 13 |
| 4.1 Подключенные IP-адреса .....                        | 13 |
| 4.2 Передача данных .....                               | 13 |
| 4.3 Двусторонняя передача данных .....                  | 13 |
| 4.4 Изменение отношений .....                           | 14 |
| 4.5 Алгоритм .....                                      | 14 |
| 4.5.1 Подключения .....                                 | 14 |
| 4.5.2 Коэффициент активной передачи .....               | 14 |
| 4.5.3 Двусторонние передачи данных .....                | 14 |
| 4.5.4 Коэффициент изменений отношений .....             | 15 |
| 4.5.5 Точность алгоритма .....                          | 15 |

|       |   |    |
|-------|---|----|
| 5     | Описание программы .....                                | 16 |
| 5.1   | Функция call_sniff .....                                | 16 |
| 5.2   | Функция sniff .....                                     | 17 |
| 5.3   | Определение P2P трафика .....                           | 18 |
| 5.3.1 | Метод анализования портов.....                          | 18 |
| 5.3.2 | Метод анализования потоков .....                        | 19 |
| 5.4   | Тестирование программы.....                             | 20 |
| 5.4.1 | Запуск программы при отсутствии P2P-активности .....    | 20 |
| 5.4.2 | Запуск программы при запущенном клиенте Bittorrent..... | 21 |
| 5.4.3 | Запуск программы при запущенном аудио звонке Skype .... | 23 |
|       | ЗАКЛЮЧЕНИЕ .....  | 24 |
|       | СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....                  | 25 |
|       | Приложение А Код window.py .....                        | 26 |
|       | Приложение Б Код sniffer.py .....                       | 29 |

## ВВЕДЕНИЕ

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

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

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

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

В тоже время с появлением P2P появилась необходимость обнаруживать соответствующий трафик в сети. Универсального способа обнаружения работающего P2P-приложения нет. С развитием файлообменных сетей стало затруднительно идентифицировать P2P-трафик с помощью номеров портов. Появилась необходимость исследования трафика на основании поведения узлов сети. Однако даже поведение такого трафика, его сигнатура и прочие признаки также могут изменяться со временем, поэтому все существующие методы должны обновляться и 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. [2]

Виды узлов:

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

### 1.1.2 Группа узлов

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

Группы узлов могут разделяться по следующим признакам:

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

### 1.1.3 Сетевой транспорт

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

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

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

## 1.2 Маршрутизация

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

### 1.2.1 Неструктурированные сети

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

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

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

### 1.2.2 Структурированные сети

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

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

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

### 1.2.3 Гибридные модели

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

турированные или чисто структурированные сети.

### **1.3 Безопасность**

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

#### **1.3.1 Маршрутизационные атаки**

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

#### **1.3.2 Поврежденные данные и вредоносные программы**

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

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



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

#### **1.4 Отказоустойчивость и масштабируемость сети**

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

#### **1.5 Распределенное хранение и поиск**

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

## 2 Применение P2P

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

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

- Пиринговые системы распространения контента.
- Пиринговые системы обслуживания, например, повышение производительности, в частности, Correli Caches.
- 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 Способы обнаружения P2P трафика

#### 3.1 Анализ портов

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

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

| 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   |

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

#### 3.2 Анализ сигнатур

Суть этого способа заключается в мониторинге трафика, проходящего через сеть, на предмет обнаружения определенных **сигнатур**, специфичных для P2P-приложений, в полезной нагрузке пакетов [5]. Сетевая сигнатура — набор данных, которые необходимо найти в трафике. Это могут быть IP-адреса, порты,

флаги (например, протокола TCP) и так далее. Многие современные коммерческие и свободно распространяемые решения для обнаружения P2P-трафика основаны на этом методе. Например, система предотвращения вторжений Snort предоставляет возможность создавать набор правил, включающих информацию о характеристиках транспортного уровня и содержанием полезной нагрузки пакетов для различных приложений.

Особенности данного метода:

- Необходимо постоянное обновление базы сигнатур.
- Трафик зачастую зашифрован, что сильно затрудняет анализ.
- Поиск сигнатур на прикладном сетевом уровне очень ресурсоёмкий.

### **3.3 Эвристические предположения**

#### **3.3.1 TCP/UDP-эвристика**

Если в течение интервала времени обнаружено, что пара адресов взаимодействует и по TCP-, и по UDP-протоколу, то они предположительно участвуют в P2P-обмене.

#### **3.3.2 IP/Port-эвристика**

Ещё одной особенностью P2P является тот факт, что при обращении к паре (IP-назначения, порт-назначения) количество адресов источников практически совпадает с количеством портов источников. Подобное поведение характерно, например, для сигнального взаимодействия с раздающим данные Bittorrent-клиентом. Однако стоит отметить, что такое поведение характерно не только для P2P.

## 4 Идентификация Bittorrent

В работе [6] предложен алгоритм, который основывается на четырёх критериях.

### 4.1 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.

### 4.2 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.

### 4.3 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.

## 4.4 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.

## 4.5 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. В данном алгоритме обрабатываются только TCP-пакеты.

### 4.5.1 Подключения

Подсчитывается число  $C$  — количество пиров, которые общались с хостом. Если это количество будет больше или равно порогу  $C_{threshold}$ , то хост будет идентифицирован как Bittorrent-хост.

$$C \geq C_{threshold}$$

### 4.5.2 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  $R_{ATthreshold}$ , то хост будет идентифицирован как Bittorrent-хост.

$$R_{AT} \geq R_{ATthreshold},$$

где  $R_{AT} = \frac{AT}{C}$ .

### 4.5.3 Двусторонние передачи данных

Измеряется количество подключений  $BiAT$ , по которым одновременно принимаются и отправляются данные. Если это число больше или равно поро-

говому  $BiAT_{threshold}$ , то хост будет идентифицирован как Bittorrent-хост.

$$BiAT \geq BiAT_{threshold}$$

#### 4.5.4 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  $R_{RCthreshold}$ , то хост будет идентифицирован как Bittorrent-хост.

$$R_{RC} \geq R_{RCthreshold},$$

где  $R_{RC} = \frac{RC}{AT}$ .

#### 4.5.5 Точность алгоритма

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

## 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 [7]:

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 2).

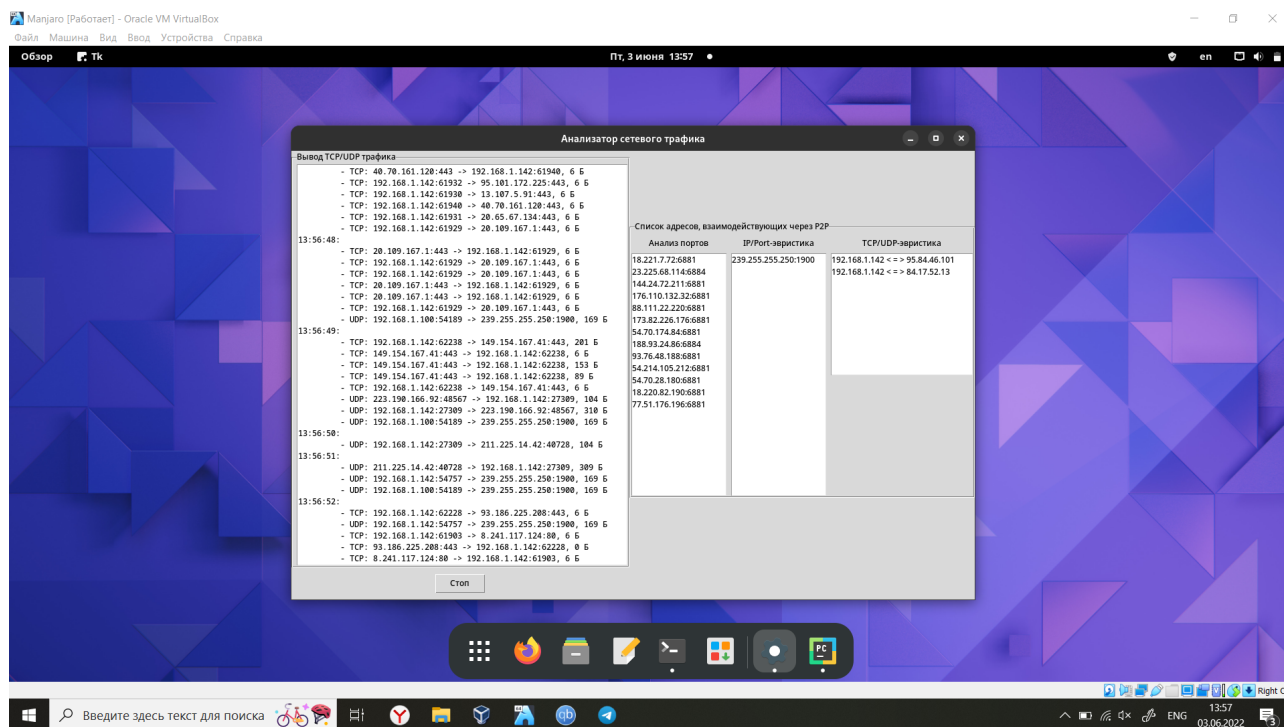


Рисунок 2 – Демонстрация схемы тестирования

### 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 3 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. [8]

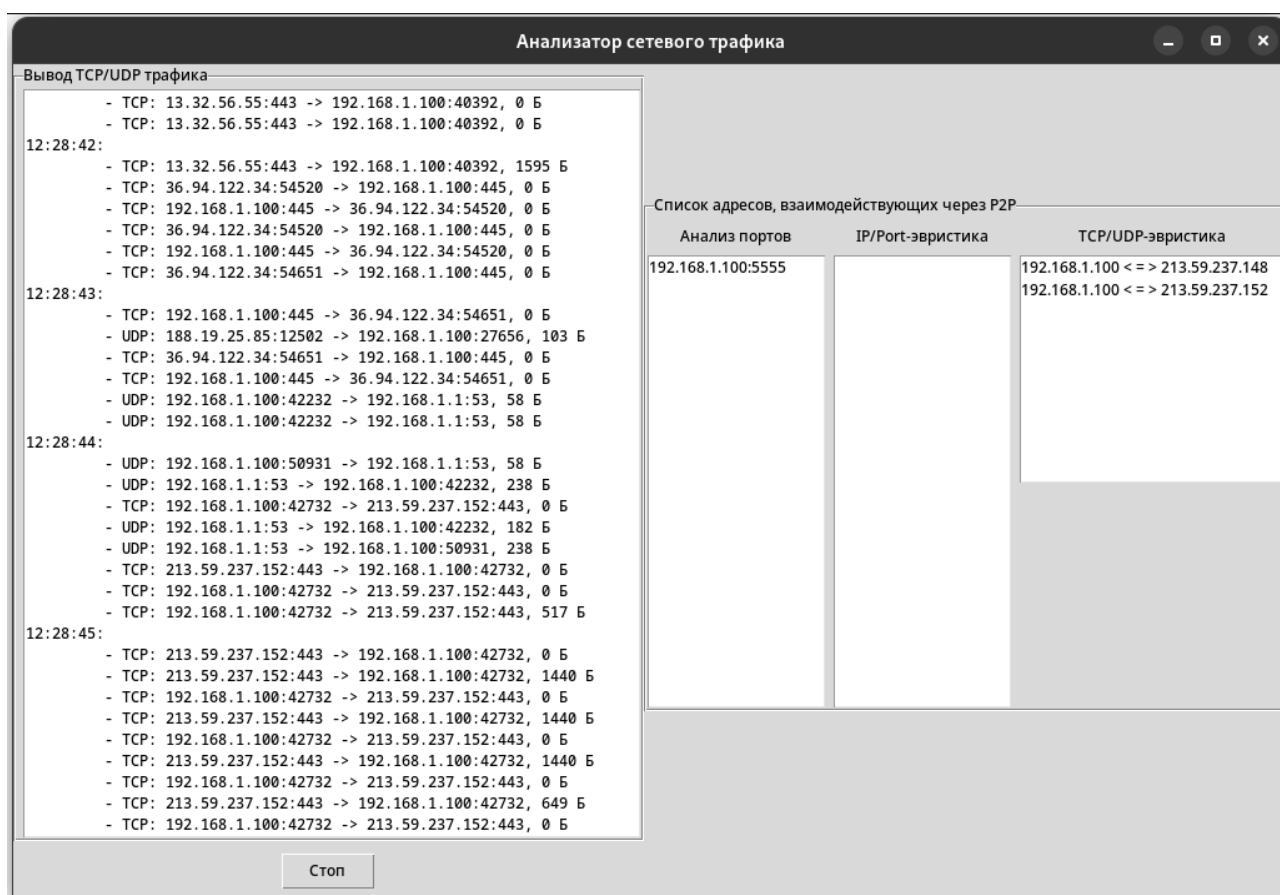


Рисунок 3 – Тестирование программы при запущенном видео на *Youtube*

#### 5.4.2 Запуск программы при запущенном клиенте Bittorrent

На хостовой машине запущен *qBittorrent*.

На рисунке 4 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 5). 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.

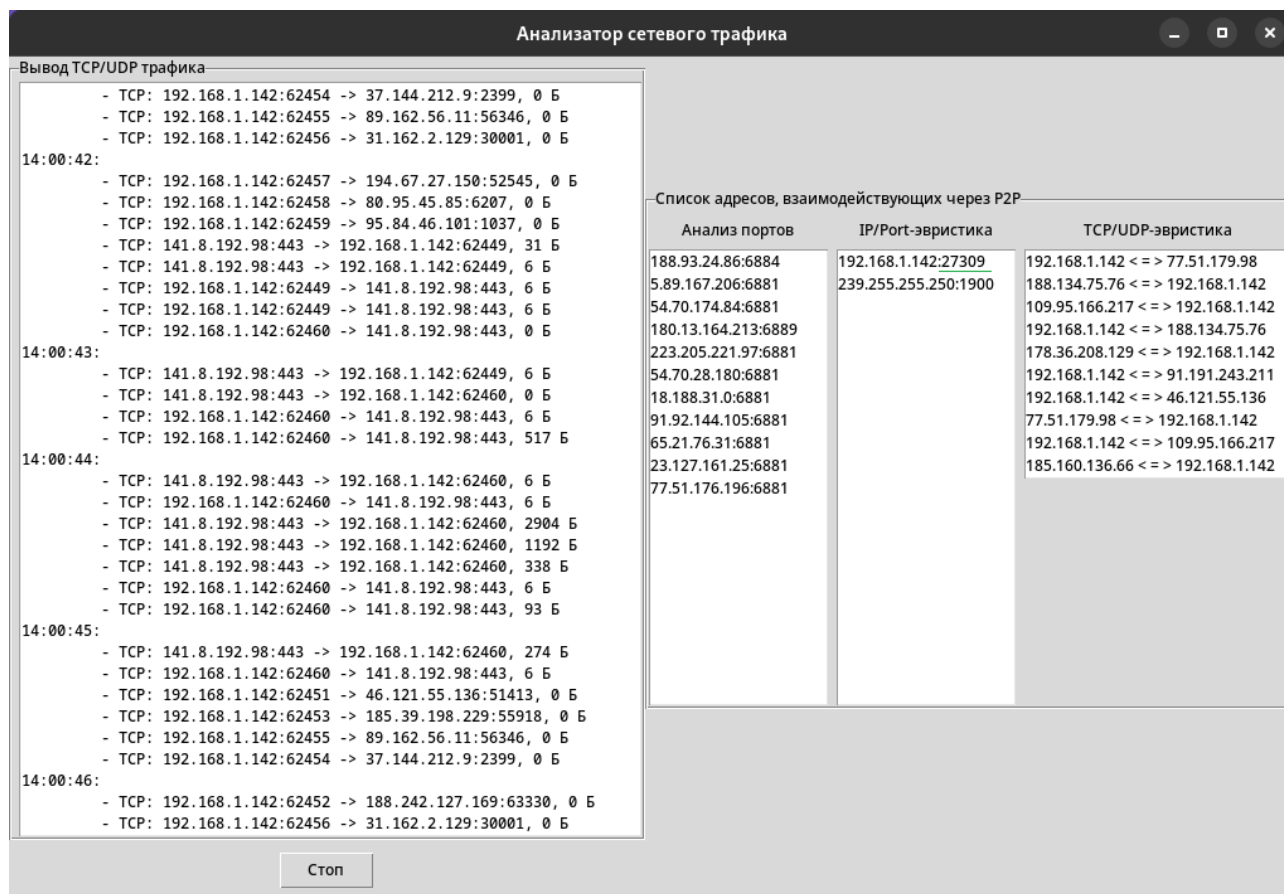


Рисунок 4 – Тестирование программы при запущенном клиенте Bittorrent

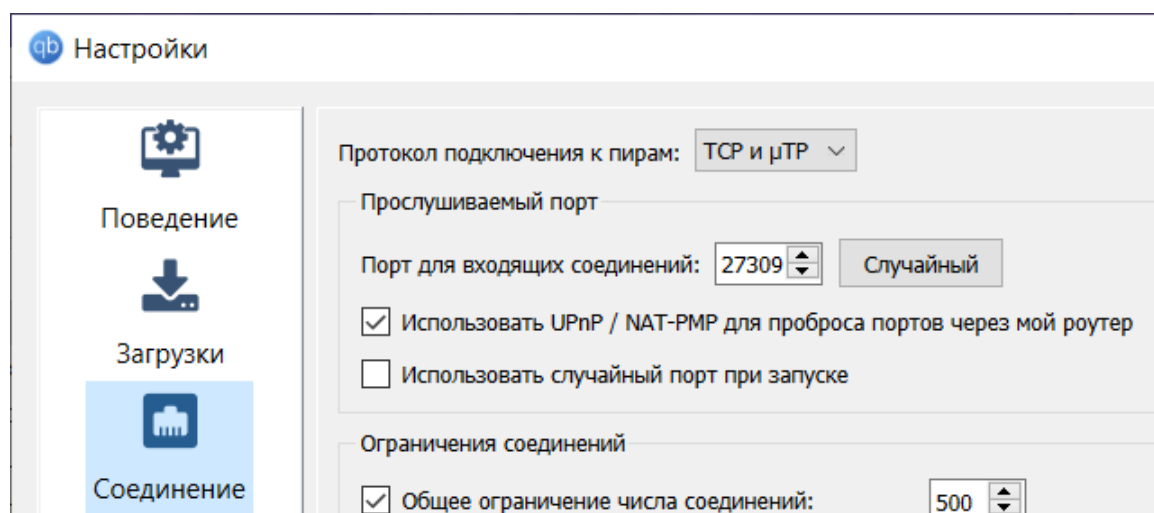


Рисунок 5 – Порт входящих соединений qBittorrent

### 5.4.3 Запуск программы при запущенном аудио звонке Skype

На хостовой машине запущен аудио звонок в Skype.

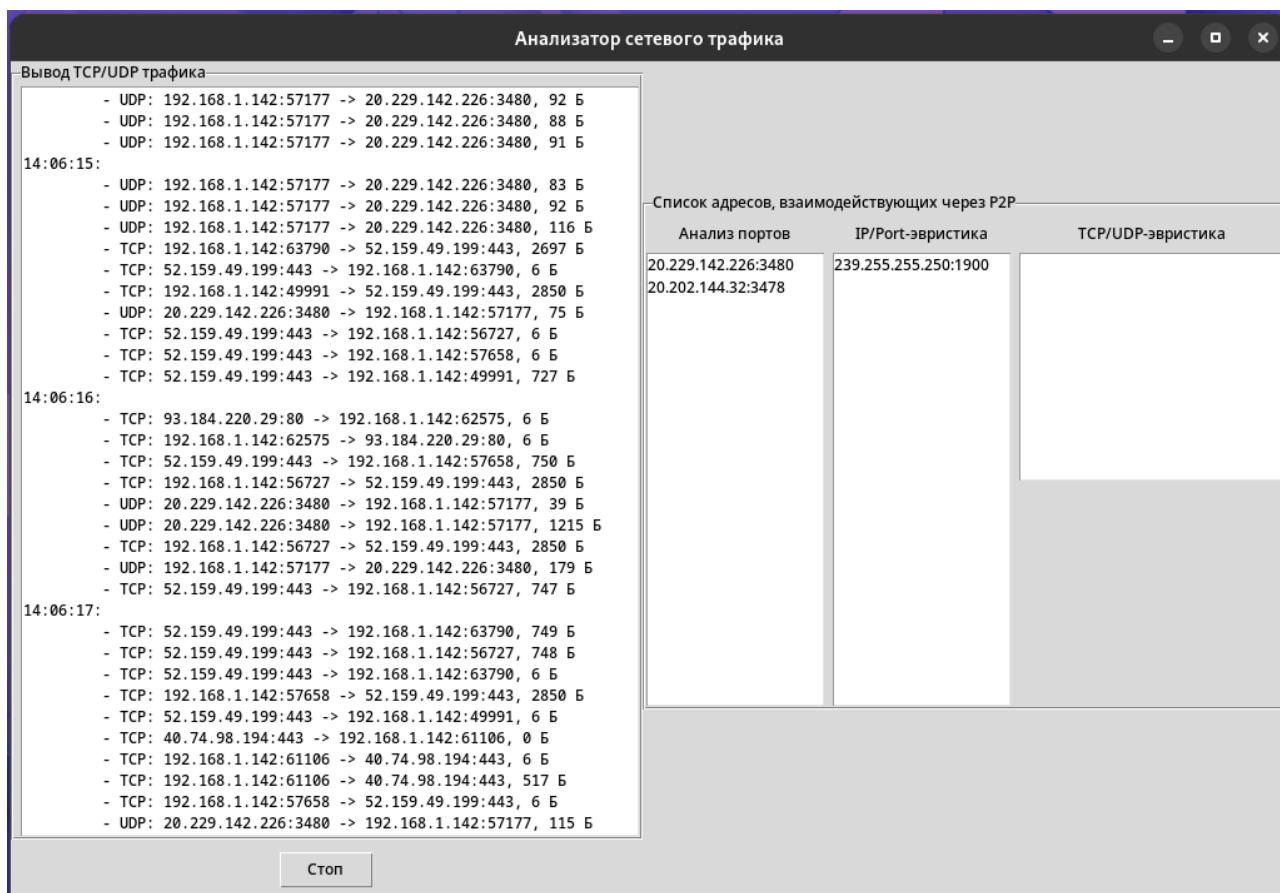


Рисунок 6 – Тестирование программы при запущенном аудио звонке в Skype

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.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 P2P (Peer-to-Peer) [Электронный ресурс]. — URL: [https://ru.bmstu.wiki/P2P\\_\(Peer-to-Peer\)](https://ru.bmstu.wiki/P2P_(Peer-to-Peer)) (Дата обращения 24.05.2022). Загл. с экр. Яз. рус.
- 2 P2P [Электронный ресурс]. — URL: [https://glebradchenko.susu.ru/courses/bachelor/odp/2013/SUSU\\_Distr\\_11\\_P2P.pdf](https://glebradchenko.susu.ru/courses/bachelor/odp/2013/SUSU_Distr_11_P2P.pdf) (Дата обращения 24.05.2022). Загл. с экр. Яз. рус.
- 3 List of well-known ports used by various peer-to-peer (P2P) protocols. [Электронный ресурс]. — URL: [https://www.researchgate.net/figure/List-of-well-known-ports-used-by-various-peer-to-peer-P2P-protocols\\_tbl1\\_347789431](https://www.researchgate.net/figure/List-of-well-known-ports-used-by-various-peer-to-peer-P2P-protocols_tbl1_347789431) (Дата обращения 25.05.2022). Загл. с экр. Яз. англ.
- 4 *Madhukar, A. A longitudinal study of p2p traffic classification / A. Madhukar, C. Williamson // Proc. of 14th IEEE International Symposium on Modeling, Analysis, and Simulation. — 2012.*
- 5 Обнаружение P2P трафика [Электронный ресурс]. — URL: <https://www.securitylab.ru/analytics/240496.php> (Дата обращения 24.05.2022). Загл. с экр. Яз. рус.
- 6 *Ngiwlay, W. Bittorrent peer identification based on behaviors of a choke algorithm / W. Ngiwlay, C. Intanagonwiwat, Y. Teng-amnuay // Association for Computing Machinery. — 2008.*
- 7 *Всеволоодович, Б. С. Диагностика р2р-активности на основе анализа потоков netflow / Б. С. Всеволоодович, Щ. Н. Григорьевна // Проблемы информатики. — 2006.*
- 8 Предупреждение о задействовании UPnP/SSDP в качестве усилителя DDoS-атак. [Электронный ресурс]. — URL: <https://www.opennet.ru/opennews/art.shtml?num=46780> (Дата обращения 03.06.2022). Загл. с экр. Яз. рус.

## ПРИЛОЖЕНИЕ А

### Код window.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
10 TAB_2 = ' \t * '
11
12
13 class Menu(tk.Frame):
14     def __init__(self, master):
15         super().__init__(master)
16         self.master = master
17         self.grid(row=0, column=0, sticky=tk.NSEW)
18         self.last_time = ''
19
20         self.frame_out = ttk.LabelFrame(self, text='Вывод TCP/UDP трафика')
21         self.frame_out.grid(row=0, column=0)
22
23         # self.label1 = ttk.Label(self, text='Вывод TCP/UDP трафика')
24         # self.label1.grid(row=0, column=0, pady=5)
25
26         self.output = tk.Text(self.frame_out, width=70, height=35)
27         self.output.grid(row=0, column=0, padx=(5, 0), sticky=tk.NW)
28
29         self.scroll_out = ttk.Scrollbar(self.frame_out, command=self.output.yview)
30         # self.scroll_out.grid(row=0, column=1, padx=(0, 15))
31         self.output.config(yscrollcommand=self.scroll_out.set)
32
33         self.frame = ttk.LabelFrame(self, text='Список адресов, взаимодействующих через  
↪ P2P')
34         self.frame.grid(row=0, column=1)
35
36         self.label2 = ttk.Label(self.frame, text='Анализ портов')
37         self.label2.grid(row=0, column=0, pady=5, sticky=tk.N)
38
39         self.p2p_lb = tk.Listbox(self.frame, height=20)
40         self.p2p_lb.grid(row=1, column=0, sticky=tk.N)
41
42         self.scroll_p2p_lb = ttk.Scrollbar(self.frame, command=self.output.yview)
43         self.p2p_lb.config(yscrollcommand=self.scroll_p2p_lb.set)
44
```

```

45     self.label3 = ttk.Label(self.frame, text='IP/Port-адреса')
46     self.label3.grid(row=0, column=1, pady=5, sticky=tk.N)
47
48     self.p2p_lb2 = tk.Listbox(self.frame, height=20)
49     self.p2p_lb2.grid(row=1, column=1, sticky=tk.N, padx=5)
50
51     self.scroll_p2p_lb2 = ttk.Scrollbar(self.frame, command=self.output.yview)
52     self.p2p_lb2.config(yscrollcommand=self.scroll_p2p_lb2.set)
53
54     self.label4 = ttk.Label(self.frame, text='TCP/UDP-адреса')
55     self.label4.grid(row=0, column=2, pady=5, sticky=tk.N)
56
57     self.p2p_lb3 = tk.Listbox(self.frame, width=30)
58     self.p2p_lb3.grid(row=1, column=2, sticky=tk.N)
59
60     self.scroll_p2p_lb3 = ttk.Scrollbar(self.frame, command=self.output.yview)
61     self.p2p_lb3.config(yscrollcommand=self.scroll_p2p_lb3.set)
62
63     self.stop_btn = ttk.Button(self, text='Cmon', command=self.stop)
64     self.stop_btn.grid(row=1, column=0, pady=(10, 10))
65
66     self.call_sniff()
67     self.call_find_p2p()
68
69     def call_sniff(self):
70         out = sniffer.sniff(conn)
71         if out:
72             # Вывод времени
73             time = str(datetime.now().strftime('%H:%M:%S')) + ":\n"
74             if time != self.last_time:
75                 self.output.insert('end', time)
76                 file.write(time)
77             self.last_time = time
78
79             # Вывод информации о пакете
80             for s in out:
81                 file.write(s)
82                 self.output.insert('end', s)
83             file.write('\n')
84             self.output.insert('end', '\n')
85
86         root.after(100, self.call_sniff) # сканирование каждые 0.1 сек
87
88     def call_find_p2p(self):
89         sniffer.find_p2p()
90         self.p2p_lb.delete(0, 'end')
91         self.p2p_lb2.delete(0, 'end')
92         self.p2p_lb3.delete(0, 'end')

```

```

93         for addr in sniffer.p2p_pairs_p:
94             self.p2p_lb.insert('end', addr[0] + ":" + str(addr[1]))
95         for addr in sniffer.p2p_pairs_ipp:
96             self.p2p_lb2.insert('end', addr[0] + ":" + str(addr[1]))
97         for addrs in sniffer.p2p_addrs:
98             self.p2p_lb3.insert('end', addrs[0] + " < = > " + addrs[1])
99         root.after(15000, self.call_find_p2p) # обнаружение p2p методом анализа
        ↪ потоков запускается каждые 15 секунд
100
101     def stop(self):
102         file2.write('Список IP-адресов, взаимодействующих через P2P: \n ')
103
104         file2.write('Анализ портов: \n ')
105         for row in self.p2p_lb.get(0, 'end'):
106             file2.write(' * ' + row + '\n ')
107
108         file2.write('IP/Port-эвентурика: \n ')
109         for row in self.p2p_lb2.get(0, 'end'):
110             file2.write(' * ' + row + '\n ')
111         file2.write('TCP/UDP-эвентурика: \n ')
112         for row in self.p2p_lb3.get(0, 'end'):
113             file2.write(' * ' + row + '\n ')
114         file2.write('Конец списка. \n ')
115
116         root.destroy()
117
118     interface = 'enp0s3'
119     if len(sys.argv) > 1:
120         interface = sys.argv[1]
121
122     ret = os.system("ip link set {} promisc on".format(interface))
123
124     conn = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
125     conn.bind((interface, 0))
126
127     # В файл сохраняется последний вывод программы
128     file = open('out.txt', 'w+')
129     # Список IP-адресов, взаимодействующих через P2P
130     file2 = open('ip_list.txt', 'w+')
131
132     root = tk.Tk()
133     root.title("Анализатор сетевого трафика")
134     menu = Menu(root)
135     root.mainloop()
136     file2.close()
137     file.close()
138     conn.close()

```

## ПРИЛОЖЕНИЕ Б

### Код sniffer.py

```
1  import socket
2  import struct
3  import textwrap
4
5  # Отступы для вывода информации
6  TAB_1 = '\t - '
7
8  # Список пар порт-приложение
9  LIST_P2P = {6881: 'BitTorrent', 6882: 'BitTorrent', 6883: 'BitTorrent',
10             6884: 'BitTorrent', 6885: 'BitTorrent', 6886: 'BitTorrent',
11             6887: 'BitTorrent', 6888: 'BitTorrent', 6889: 'BitTorrent',
12             6969: 'BitTorrent', 411: 'Direct Connect', 412: 'Direct Connect',
13             2323: 'eDonkey', 3306: 'eDonkey', 4242: 'eDonkey',
14             4500: 'eDonkey', 4501: 'eDonkey', 4677: 'eDonkey',
15             4678: 'eDonkey', 4711: 'eDonkey', 4712: 'eDonkey',
16             7778: 'eDonkey', 1214: 'FastTrack', 1215: 'FastTrack',
17             1331: 'FastTrack', 1337: 'FastTrack', 1683: 'FastTrack',
18             4329: 'FastTrack', 5000: 'Yahoo', 5001: 'Yahoo',
19             5002: 'Yahoo', 5003: 'Yahoo', 5004: 'Yahoo', 5005: 'Yahoo',
20             5006: 'Yahoo', 5007: 'Yahoo', 5008: 'Yahoo', 5009: 'Yahoo',
21             5010: 'Yahoo', 5050: 'Yahoo', 5100: 'Yahoo', 5555: 'Napster',
22             6257: 'Napster', 6666: 'Napster', 6677: 'Napster',
23             6688: 'Napster', 6699: 'Napster', 6700: 'Napster',
24             6701: 'Napster', 6346: 'Gnutella', 6347: 'Gnutella', 5190: 'AIM',
25             3478: 'Skype / Steam (voice chat)', 4379: 'Steam (voice chat)',
26             4380: 'Steam (voice chat)', 4899: 'Radmin VPN', 12975: 'Hamachi',
27             32976: 'Hamachi', 3479: 'Skype', 3480: 'Skype', 3481: 'Skype'}
28
29 # Список портов исключений
30 EXCEPTIONS = {137, 138, 139, 445, 53, 123, 500, 554, 7070, 6970, 1755, 5000, 5001, 6112,
31               ↪ 6868, 6899, 6667, 7000, 7514}
32
33 TCP_addrs = set()
34 UDP_addrs = set()
35 p2p_addrs = set()
36 p2p_addrs1 = set()
37 p2p_pairs = set()
38 p2p_pairs_p = set()
39 p2p_pairs_ipp = set()
40 rejected = set() # адреса, не относящиеся к P2P
41 dict_ipport = dict() # словарь вида (ip+port -> объект класса IPPort)
42
43 class IPPort:
44     def __init__(self, dst_ip, dst_port):
```

```

45         self.dst_ip = dst_ip
46         self.dst_port = dst_port
47         self.IPSet = set()
48         self.PortSet = set()
49         self.p2p = False
50
51     def add(self, ip, port):
52         self.IPSet.add(ip)
53         self.PortSet.add(port)
54
55     # Добавление в p2p_addrs1 адресов, которые взаимодействовали с адресами из p2p_addrs
56     def add_to_p2p_addrs1(self):
57         for addr in p2p_addrs:
58             if addr in self.IPSet and addr not in rejected:
59                 p2p_addrs1.add(addr)
60
61     # Проверка IP/Port-эвентуки
62     def check_p2p(self):
63         dif = 2
64         # Если порт из списка исключений, то разница между IPSet и PortSet должна быть
        ↪ увеличена до 10
65         if self.dst_port in EXCEPTIONS:
66             dif = 10
67         if (self.dst_ip, self.dst_port) not in rejected:
68             self.p2p = len(self.IPSet) > 2 and (len(self.IPSet) - len(self.PortSet) < dif)
69         else:
70             self.p2p = False
71         return self.p2p
72
73
74     def sniff(conn):
75         output = ''
76         raw_data, addr = conn.recvfrom(65536)
77         dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)
78
79         # IPv4
80         if eth_proto == 8:
81             (version, header_length, ttl, proto, src, dest, data) = ipv4_packet(data)
82
83         # TCP
84         if proto == 6:
85             src_port, dest_port, sequence, ack, flag_urg, flag_ack, \
86             flag_psh, flag_rst, flag_syn, flag_fin, data = tcp_segment(data)
87
88             output = [TAB_1, 'TCP: ', src, ':', str(src_port), ' -> ', dest, ':',
89                       str(dest_port), ', ', str(len(data)), ' B']
90
91             save(True, src, dest, src_port, dest_port)

```

```

92         check_ports(src, dest, src_port, dest_port)
93
94     # UDP
95     elif proto == 17:
96         src_port, dest_port, length, data = udp_segment(data)
97
98         output = [TAB_1, 'UDP: ', src, ':', str(src_port), ' -> ', dest, ':',
99                     str(dest_port), ', ', str(len(data)), ' B']
100
101         check_ports(src, dest, src_port, dest_port)
102         save(False, src, dest, src_port, dest_port)
103
104     return output
105
106
107 def save(tcp, src, dest, src_port, dest_port):
108     if tcp:
109         TCP_addrs.add((src, dest))
110     else:
111         UDP_addrs.add((src, dest))
112     check_exceptions(src, dest, src_port, dest_port)
113     add_ipport(dest, dest_port, src, src_port)
114
115
116 def check_ports(src, dest, src_port, dest_port):
117     if LIST_P2P.get(src_port, False):
118         p2p_pairs_p.add((src, src_port))
119     elif LIST_P2P.get(dest_port, False):
120         p2p_pairs_p.add((dest, dest_port))
121
122
123 def add_ipport(dest, dest_port, src, src_port):
124     ipport = dest + ':' + str(dest_port)
125     if ipport not in dict_ipport:
126         x = IPPort(dest, dest_port)
127         x.add(src, src_port)
128         dict_ipport[ipport] = x
129     else:
130         dict_ipport[ipport].add(src, src_port)
131
132
133 # Добавление адресов с портами в список исключений
134 def check_exceptions(src, dest, src_port, dest_port):
135     if src_port in EXCEPTIONS \
136         or dest_port in EXCEPTIONS \
137         or (src_port == dest_port and src_port < 500):
138         rejected.add((src, src_port))
139         rejected.add((dest, dest_port))

```

```

140
141
142 def find_p2p():
143     # 1 Заполнение p2p_addrs адресами, взаимодействующими одновременно по TCP и UDP
144     inter = TCP_addrs & UDP_addrs
145     for pair_addrs in inter:
146         for ipport in rejected:
147             if pair_addrs[0] != ipport[0] and pair_addrs[1] != ipport[0]:
148                 p2p_addrs.add(pair_addrs)
149
150     # 2 Заполнение p2p_pairs_ipp адресами, выбранными исходя из check_p2p
151     for ipport in dict_ipport:
152         ipp = dict_ipport[ipport]
153         ipp.add_to_p2p_addrs1() # Заполнение массива p2p_addrs1
154         ip = ipp.dst_ip
155         port = ipp.dst_port
156         if ipp.check_p2p() and (ip, port) not in rejected:
157             p2p_pairs_ipp.add((ip, port))
158
159
160 # Распаковка ethernet кадра
161 def ethernet_frame(data):
162     dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
163     return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto), data[14:]
164
165
166 # Форматирование MAC-адреса
167 def get_mac_addr(bytes_addr):
168     bytes_str = map('{:02x}'.format, bytes_addr)
169     return ':'.join(bytes_str).upper()
170
171
172 # Распаковка IPv4 пакета
173 def ipv4_packet(data):
174     version_header_length = data[0]
175     version = version_header_length >> 4
176     header_length = (version_header_length & 15) * 4
177     ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
178     return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]
179
180
181 # Форматирование IP-адреса
182 def ipv4(addr):
183     return '.'.join(map(str, addr))
184
185
186 # Распаковка TCP сегмента
187 def tcp_segment(data):

```



```

188     (src_port, dest_port, sequence, ack, offset_reserved_flags) = struct.unpack('! H H L L
    ↪ H', data[:14])
189     offset = (offset_reserved_flags >> 12) * 4
190     flag_urg = (offset_reserved_flags & 32) >> 5
191     flag_ack = (offset_reserved_flags & 16) >> 5
192     flag_psh = (offset_reserved_flags & 8) >> 5
193     flag_rst = (offset_reserved_flags & 4) >> 5
194     flag_syn = (offset_reserved_flags & 2) >> 5
195     flag_fin = offset_reserved_flags & 1
196     return src_port, dest_port, sequence, ack, \
197           flag_urg, flag_ack, flag_psh, flag_rst, flag_syn, flag_fin, data[offset:]
198
199
200 # Распаковка UDP сегмента
201 def udp_segment(data):
202     src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
203     return src_port, dest_port, size, data[8:]

```