

Содержание

Введение.....	6
1. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ	8
1.1 История развития средств анализа сетевого трафика	8
1.2 Направления развития технологий анализа сетевого трафика	12
1.2.1 Глубина анализа сетевых пакетов	12
1.2.2 Поверхностный анализ пакетов (SPI)	13
1.2.3 Средний анализ пакетов (MPI)	13
1.2.4 Глубокий анализ пакетов (DPI)	14
1.3 Учёт состояния потока при анализе сетевого трафика.	16
1.3.1 Анализ сетевых пакетов с учётом состояния потоков	18
1.3.2 Анализ содержимого сетевых протоколов прикладного уровня ..	20
1.4 Общая схема инфраструктурных алгоритмов анализа сетевого трафика.	22
1.4.1 Захват сетевых пакетов.....	25
1.4.2 Группировка сетевых пакетов в потоки	27
1.4.3 Классификация сетевого трафика.	29
1.4.4 Подходы на основе вывода.	33
1.4.5 Методы на основе сигнатур	34
1.4.6 Сигнатуры на основе регулярных выражений.....	35
1.4.7 Анализ данных в разных представлениях	38
1.4.8 Классификация угроз.....	40
1.5 Требования, предъявляемые к современным средствам анализа содержимого сетевого трафика.....	40

1.6 Эволюция методов классификации трафика.....	42
1.6.1 Классификация по номерам портов	42
1.6.2 Глубокий анализ пакетов	43
1.6.3 Стохастический анализ пакетов	44
1.6.4 Использование машинного обучения для классификации трафика	44
1.7 Типы классификации	45
2. СПЕЦИАЛЬНЫЙ РАЗДЕЛ.....	48
2.1 Методы машинного обучения, используемые для классификации трафика	48
2.1.1 Наивный байесовский классификатор	48
2.1.2 Метод опорных векторов	49
2.1.3 Метод k-ближайших соседей.....	49
2.1.4 Дерево принятия решений	49
2.1.5 Бэггинг.....	50
2.1.6 Бустинг	51
2.1.7 Нейронные сети.....	51
2.2 Признаки сетевого трафика, используемые для его классификации	53
2.3 Выбор и подготовка набора данных, их сравнение.....	54
2.3.1 Основные аспекты выбора набора данных	55
2.3.2 Получение правильной разметки данных	55
2.3.3 Доступность данных	57
2.3.4 Место получения данных	57
2.3.5 Репрезентативность набора данных.....	58
2.4 Используемые общедоступные наборы данных	58
3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ	60

3.1 Python	60
3.2 Scikit-learn	62
3.3 Pandas	62
3.4 NymPy.....	64
3.5 Keras	65
3.6 TensorFlow	66
3.8 Метод опорных векторов	67
3.9 Рекуррентная нейронная сеть	75
Вывод.....	85
Список литературы	86

Введение

Классификация трафика представляет собой важную задачу, так как полученные результаты находят применения в самых разных приложениях, важных для администрирования сети и для конечного пользователя.

Например, провайдер интернета может классифицировать трафик сети для:

- контроля сети и трафика в ней (например, для блокировки протоколов, таких как BitTorrent),
- изменения цен на услуги,
- планирования размещения и использования ресурсов,
- оптимизации предоставляемых сервисов и алгоритмов маршрутизации (например, для регулирования приоритетов передачи различных типов данных в случае высокой загруженности сети).

Из-за того, что нужды пользователей касательно использования Сети постоянно меняются, необходимо понимать их и изменять Сеть в соответствии с ними. Например, на сегодняшний день видна тенденция отказа от господствующего ранее принципа асимметрии устройства сети в том смысле, что клиенты загружают куда больше информации, чем отправляют. Появление P2P-приложений, VoIP, видеозвонков, потоковой передачи мультимедиа и прочих вещей должно вызвать у интернет-провайдеров необходимые меры по переустройству сети под новую реальность. Помимо этого, в настоящее время растёт число «умных устройств», которые составляют Интернет вещей: он тоже ставит перед интернет-провайдерами новые задачи для обеспечения максимальной эффективности своей работы, так как при некорректных действиях качество предоставляемых услуг может значительно снизиться.

Также следует отметить мобильные приложения, чья доля в интернет-трафике продолжает расти. Использование смартфонов и мобильных приложений можно считать более персонализированным по сравнению с «десктопным» трафиком, так как классификация и анализ такого трафика даёт

возможность составить сетевой портрет пользователя, при том лучшую его версию. Выяснение интересов, предпочтений и вкусов пользователей используется для маркетинга, позволяя проводить лучшие таргетированные рекламные кампании, и, как следствие, зарабатывать больше денег.

С точки зрения обеспечения информационной безопасности, классификация сетевого трафика может использоваться как важный признак для выявления неправомерных или необычных действий пользователя, аномалий в работе Сети, кибер-атак и прочих нарушений, так как множество вредоносных программ оставляют характерный след, который заметно отличается от обычного трафика всех пользователей или трафика конкретного пользователя.

Применяемые для классификации трафика методы меняются вместе с глобальными изменениями в устройстве трафика. К таким глобальным изменениям, влияющим на решение задачи классификации трафика можно отнести:

- прекращение использования утверждённого списка портов в зависимости от протокола/приложения;
- обfuscация протоколов с целью замаскировать те из них, которые блокируются или подавляются провайдером, что приводит также к необходимости анализировать и классифицировать зашифрованный трафик;
- широкое распространение шифрования трафика, которое не позволяет использовать для классификации содержимое полезной нагрузки пакета обычными средствами;
- постоянное появление новых протоколов, приложений и т.д.

По указанным выше причинам, задачу классификации сетевого трафика на данный момент нельзя считать решённой, поэтому продолжаются исследования, цель которых предложить новые решения, позволяющие показывать более эффективные результаты в условиях беспрестанно меняющейся ситуации.

1. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

1.1 История развития средств анализа сетевого трафика

Технологии анализа сетевого трафика начали появляться в 1990-х годах. Их потребность возникла примерно в одно и то же время во многих сферах.

Из-за сложности устройства сети и широкого спектра сетевого оборудования усложнялась и настройка, и поддержание сети в надлежащем состоянии – нужен был инструмент, который с одной стороны решал бы проблемы, а с другой обеспечивал максимально комплексное информацию насколько это возможно.

Одними из инструментов, изначально разработанных для решения этой задачи, был снiffeр [1] Wireschark [2], созданный инженером Джеральдом Комбом в 1997 году, который давно уже является стандартом при анализе трафика сети, и активно используется в том числе людьми специалистами, занятыми в сфере информационной безопасности.

Примерно в это же время начала активно использоваться технология трансляции интернет-адресов (NAT) [3], предназначенная для защиты адресов интеллектуальной собственности, чтобы внешний зритель не мог видеть ресурсы и устройства внутренней локальной сети, а также в качестве защиты от злоумышленника и для экономии IP-адресов, в силу ограниченности пространства протокола IPv4. Для реализации этой технологии требовалось устройство-транслятор аппаратных или программных адресов. В результате эта функция стала неотъемлемой частью большинства маршрутизаторов, однако существуют также и программные приложения в составе некоторых серверных операционных систем, а также отдельные программы, которые легко установить и удалить [4].

К этому же периоду относятся вирусы и первые упоминания о DoS и DDoS-атаках [5]. Для защиты от этих угроз нужен был инструмент, способный анализировать и фильтровать пакеты до достижения основного сервера.

Начальным средством защиты стали брандмауэры. Первое их поколение в основном использовало некий фильтр пакетов, который обрабатывал пакеты один за другим (без учета фона) и анализировал только на уровне L1-L3 модели OSI (см. рис. 1). Для определения типа трафика используется список фиксированных номеров портов из каталога IANA [6], что работало на тот момент, но уже не так актуально на сегодняшний день. Сопоставлялись данные, полученные от пакета, с набором правил, заданных в процессе анализа, и на основе результата пакет пропускался или блокировался с записью события. Например, «Правило блокировки трафика Telnet» обычно считается описанием пакетов, транспортным протоколом которых является рабочий код, номер порта назначения — 23, и операция блокируется при обнаружении таких пакетов.

Ближе к концу 90х — началу 2000х годов, из-за роста объёма интернет-трафика, актуальными стали ещё две проблемы, решений которых требовало анализа трафика: балансировка нагрузки между серверами, обслуживающими пользователей, и ускорение работы отдельных видов сетевых приложений, которые имели большее значение для пользователей и работы сети по сравнению с другими приложениями. Для решения второй проблемы использовались так называемые прокси-серверы, которые управляют данными, таким образом минимизируя объём трафика. Устройства, предназначенные для решения обеих задач (в частности, функции прокси-сервера), называются контроллерами доставки приложений. В первой половине 2000-х сетевые технологии развивались быстрыми темпами — в частности, в некоторых сетях появились сетевые голосовые обмены (VoIP) и обмены данными, что привело к очередному резкому скачку, который снова потребовал дополнительной модификации сети и используемого ПО. Для развития сетей крупных корпораций необходимо было объединить географически изолированные площади одну локальную сеть, что также поставило новые задачи перед интернет-провайдерами, системными администраторами и специалистами информационной безопасности. Сетевые атаки стали более частыми и сложными, требующими более совершенной защиты, так как злоумышленники научили обходить старые средства

обеспечения безопасности. Для обмена управляющими сигналами и данных VoIP с использованием протоколов как SIP [8] и RTP [9] требовались специальные устройства – пограничные контроллеры сессий [10]. Для решения задачи эффективного обмена данными между различными сегментами распределенной сети, соединенными по каналу с ограниченной пропускной способностью, разработан ряд методик под общим названием Wan Optimizations [11]. Среди этих техник можно указать:

- Дедупликация – Сведите к минимуму повторную передачу данных, сохраняя повторяющиеся элементы данных на обоих концах свопа и затем передавая ссылку на эти данные вместо данных. Он может быть реализован на разных уровнях сетевого стека (в частности TCP и IP).
- Сжатие – передача данных в архивированном виде с последующим разразхивированием.
- Кэширование получаемого содержимого. Было реализовано при помощи прокси-серверов.
- Объединение нескольких пакетов интенсивных сетевых протоколов, таких как CIFS [12], в один. Данные техники впоследствии реализовывались в виде отдельных сетевых устройств [13] и программно, на мощных серверах. В этот период произошли существенные изменения в области сетевой безопасности. Сложность сетевых атак привела к тому, что их стало трудно определять с достаточной точностью для отдельных пакетов, а скорость появления новых атак привела к необходимости реагировать на неизвестные до сих пор их виды. В совокупности это привело к появлению методов безопасности, основанных на анализе поведения при встрече. В то же время стали активнее рождаться вредоносные сайты, заражающие своих посетителей, и тоже генерирующие трафик, только уже вредоносный, а также способы внедрения вредоносного функционала в незараженные сайты. Для защиты от подобных атак потребовалось введение обновленного черного списка сайтов, а также фильтрация и блокировка по URL.

Больше всего анализ трафика развивался 2005-2010 годах, и на это повлияло несколько пунктов:

- Увеличение объёма трафика из-за развития и появления новых сервисов.
- Рост ширины каналов, что приводило к увеличению трафика.
- Увеличение количества разнообразия данных.
- Рост разнообразия сетевых угроз и атак и их количественные характеристики.

В то же время, специфичной областью интересов провайдеров интернета является [14, 15]:

- Гарантия качества связи в часы максимальной нагрузки.
- Получение конкурентного преимущества за счёт возможности предлагать более выгодные индивидуальные тарифы с использованием информации, как конкретный пользователь использует Сеть.
- Регулирование полосы пропускания для некоторых видов трафика.

Одним из самых больших препятствий является трафик от хоста к хосту, и он может использовать значительную часть ширины выданного канала (от 60 до 80 процентов [16]), и как следствие, возникают проблемы с тем, чтобы предоставить достаточное качество сервиса. Из-за этого провайдер интернета должен приходится быстро увеличить ширину выданного канала (как одно из решений).

В другую группу входят компании, предоставляющие различные интернет-услуги, например, с помощью технологии виртуализации сетевых функций. Эти услуги включают в себя:

- облачные сервисы,
- сервисы защиты,
- хранения и др.

Среди проблем этих компаний нередко появлялась проблема регулирования больших объёмов трафика, то есть были необходимы сбалансированность и разумное управление.

1.2 Направления развития технологий анализа сетевого трафика

Есть два основных направления развития.

- Увеличение «глубины» анализа для отдельного сетевого пакета, то есть повышение уровня модели OSI, данные которой анализируются.
- Полнота учета состояния потока, которому принадлежит пакет, а также других потоков, связанных с данными.

В следующих разделах будут рассмотрены обе эти области развития.

1.2.1 Глубина анализа сетевых пакетов

Вдоль этой «оси» последовательно развивались технологии анализа трафика, каждая последующая наследовала часть предыдущих механизмов и добавляла свою. Существует три уровня развития технологии, которые показаны на рисунке 1.

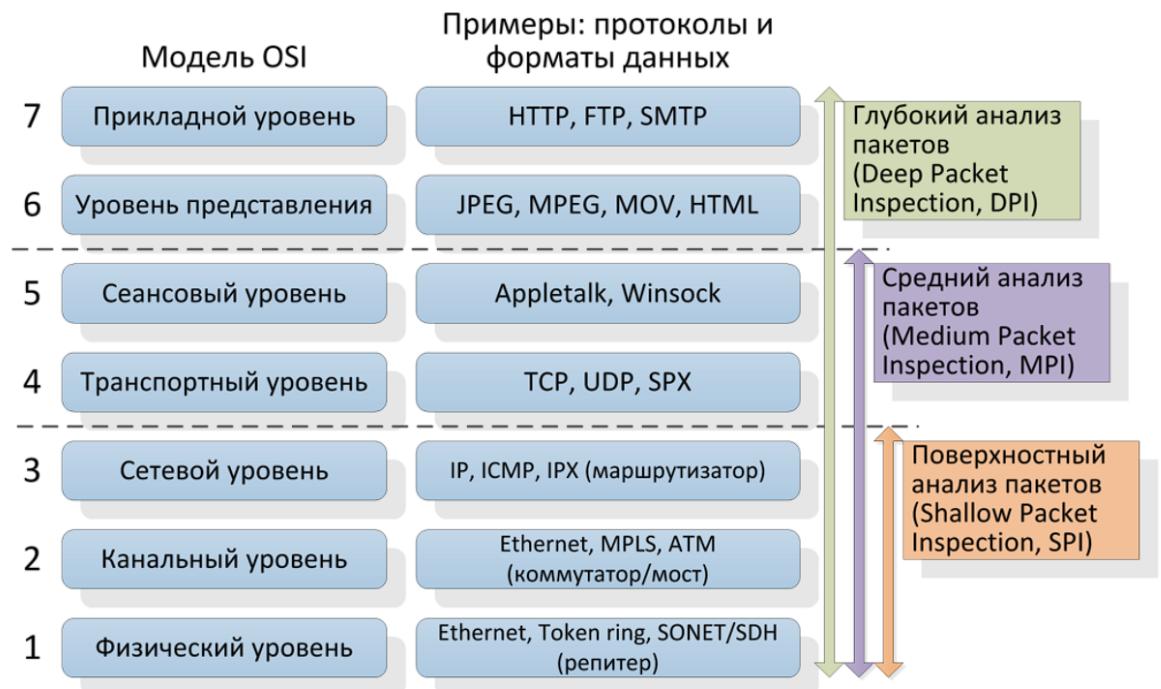


Рисунок 1 – Уровни развития технологии анализа сетевого трафика по «глубине»

Рассмотрим эти уровни более детально.

1.2.2 Поверхностный анализ пакетов (SPI)

Технология анализа трафика полностью основана на многоуровневых заголовках пакетов от 1 до 3 уровней в соответствии с моделью ISO/OSI. Это требует определенных вычислительных ресурсов, которые позволяли бы анализировать большой объем трафика. Эта технология распространена, поскольку на ней основано большинство брандмауэров для операционных систем, маршрутизаторов и других сетевых устройств. На этой основе реализуются списки контроля доступа к сети на уровне адресов и портов интеллектуальной собственности. Таким образом, данная технология подходит для ограничения внешнего доступа к персональным компьютерам и сервисам (портам) внутренних сетей.

1.2.3 Средний анализ пакетов (MPI)

Методы анализа трафика, основанные на сеансах мониторинга и связи, инициированных программой, но установленных промежуточным шлюзом (см. рис. 2). Также используется термин «программный прокси-сервер». По этой методике содержимое пакета анализируется частично и по определенным правилам. Сложные методы анализа типов сигнатур не используются. Устройства, реализующие этот функционал, располагаются между провайдером и конечным пользователем. Эти инструменты анализируют заголовки вплоть до транспортного уровня и сравнивают небольшую часть пакета данных с отдельным разделом в списке анализов, после чего в случае обнаружения следует ответ. Эти списки обычно короче списков контроля доступа и предоставляют более широкий набор функций, в отличие от «включить/отклонить» в случае списков доступа. Вы можете, например,

заблокировать возможность получения флеш-файлов или изображений от определенных веб-сервисов или выполнять определенные команды (на уровне программы) в разных протоколах. Вы можете заблокировать его. Набор протоколов обычно очень ограничен. Межсетевые экраны, использующие эту технологию, относятся ко второму поколению [17].

Эта технология гибче SPI, и также подходит для других задач — кеширование контента, анализ сжатого/зашифрованного трафика, индивидуальный протокол с ограниченным функционалом, запрещающий отдельные заказы. Благодаря подключению в режиме прокси может работать как оптимизатор. Однако основным минусом является плохая масштабируемость: для каждой команды и протокола требуется отдельный порт ввода/вывода. Кроме того, работа в режиме прокси значительно снижает скорость обработки. Для уменьшения использования прокси-серверов используется, например, протокол ICAP [18], который позволяет прокси-серверам отправлять информацию для анализа на другие серверы для обеспечения уровня безопасности или классификации.

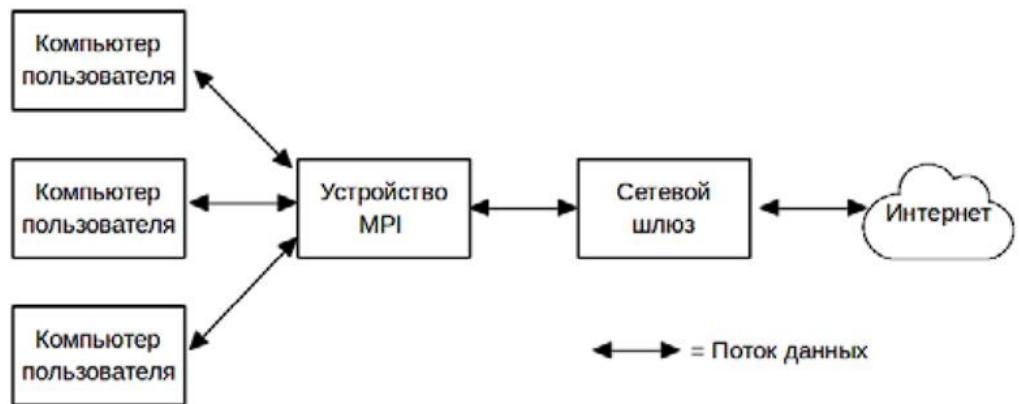


Рисунок 2 - Схема применения устройств анализа на основе технологии MPI.

1.2.4 Глубокий анализ пакетов (DPI)

Иногда употребляют более узкий термин — DPP (Deep Packet Processing), который означает модификацию, фильтрация или перенаправление. Сегодня эти

два термина часто используются взаимозаменяемо [19]. Эта технология представляет собой логическое развитие информационных технологий. При таком подходе анализирующая программа тщательно изучает содержимое каждого пакета. Важным отличием от предыдущих технологий является то, что страничные системы могут принимать решения не только о содержимом пакетов, но и о косвенных характеристиках, присущих конкретным программам и протоколам в сети. Для этого необходим статистический анализ. Например, анализ частоту появления определенных символов, длину пакета, расстояние между последовательными отметками времени пакета и тому подобное, что позволяет получать более точные результаты по сравнению с иными подходами. Кроме того, по сравнению с предыдущими подходами значительно расширился список технических приложений: классификация, ограничение пропускной способности, приоритет, маркировка, кэширование и тому подобное. Технология развивалась в основном за счет быстрого роста вычислительной мощности центральных процессоров, производительности и возможности более полного и точного анализа сетевых данных. В отличие от Instant, эта технология изначально разрабатывалась для высокоскоростной обработки и обнаружения большого количества приложений в реальном времени. Таким образом, решения на базе DIP хорошо растут по ширине канала сети (известные решения работают в каналах порядка 100 Гбит/с) и количеству определенных приложений (в существующих решениях — несколько тысяч). С точки зрения приложения основным компонентом любого страничного решения является модуль классификации, отвечающий за классификацию сетевых потоков.

Важно отметить, что соответствие между классами различных уровней точности не однозначно, что показано на рисунке 3.

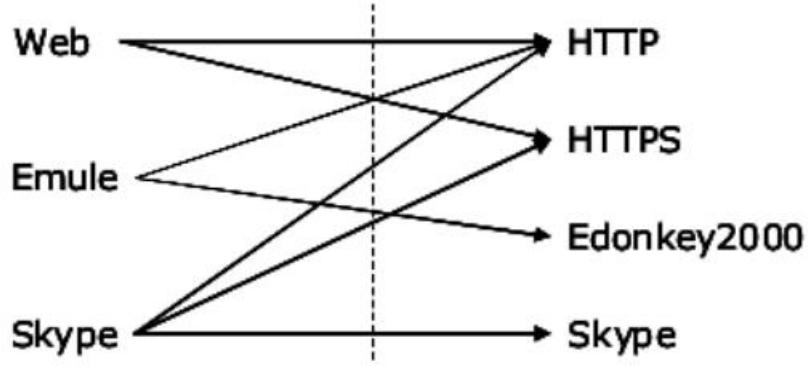


Рисунок 3 - Различие между идентификацией приложений (слева) и протоколов (справа)

В итоге, реализация нюансов, внутренняя структура и связанные с этим функции оборудования были установлены на международном уровне на основе многих стандартов, требований и рекомендаций [20, 21].

1.3 Учёт состояния потока при анализе сетевого трафика.

Вторым направлением развития технологии анализа можно назвать учет состояния протоколов в процессе анализа — так называемые типы анализа без состояния/с сохранением состояния. Это направление актуально только для протоколов, использующих транспортные протоколы, ориентированные на подключение, то есть перед обменом информацией имеет место быть процесс "установления соединения" - участвующие в обмене хосты обмениваются определенной последовательностью пакетов, называемыми "рукопожатиями", а после окончания сессии происходит похожий процесс. Протоколы, ориентированные на соединение, в частности, включают протокол TCP, но не UDP, так как UDP в своей изначальной форме не использует рукопожатия. Однако следует отметить, что поверх UDP может быть реализован другой транспортный протокол с установлением соединения. В качестве примера можно привести протокол Quick UDP Internet Connections (QUIC) [22] — протокол транспортного уровня, устанавливающий соединение с помощью UDP. Поэтому анализ состояния не может быть полностью исключен для пакетов UDP.

Чтобы описать разницу между описанными подходами, необходимо определить понятие «поток пакетов». Известны различные определения этого понятия. Некоторые из наиболее часто используемых перечислены на веб-сайте Центра прикладного анализа интернет-данных (CAIDA) [23]. В этой работе мы будем использовать «однонаправленный поток транспортного уровня» — последовательность пакетов, передаваемых с определенного IP-адреса и порта TCP/UDP на определенный IP-адрес и порт TCP/UDP. Таким образом, поток определяется пятью значениями - <srcIP, SrcPort, dstIP, dstPort, protocol>. Принимая во внимание это определение, можно сформулировать разницу между statefull и stateless подходом. Она состоит в том, что в statefull-подходе принимается во внимание, к какому потоку принадлежит пакет, а также результат анализа предыдущих пакетов этого же потока, если этот пакет не первый. Если пакет первый, то проверяется, что это правильный пакет установления соединения. Также следует отметить, что понятие «statefull» не совсем понятно и может иметь разные градации с разным «state», что приводит к разному балансу точности анализа/ресурсоёмкости/скорости работы [24, 25]. Один из вариантов градации можно увидеть на рис. 4. Перечень уровней учета состояния потока, который там отражен, выглядит следующим образом:

- Анализ отдельных пакетов без учёта потоков и состояний.
- Изучение пакетов в рамках сегментов данных.
- Анализ сообщений в рамках потока (Message Based Per Flow State, MBFS), т.е. произведена сборка IP-фрагментов в IP-пакеты (IP- нормализация) и сборка TCP-сегментов в TCP-сеансы (TCP- нормализация).
- Изучение пакетов, находящихся внутри протоколов.

Пример автомата состояний протокола HTTP приведён на рисунке 4. Вершины соответствуют состояниям, рёбра — условиям перехода, к которым могут относиться приём/отправка сообщения, результаты обработки сообщений, истечение таймаута.

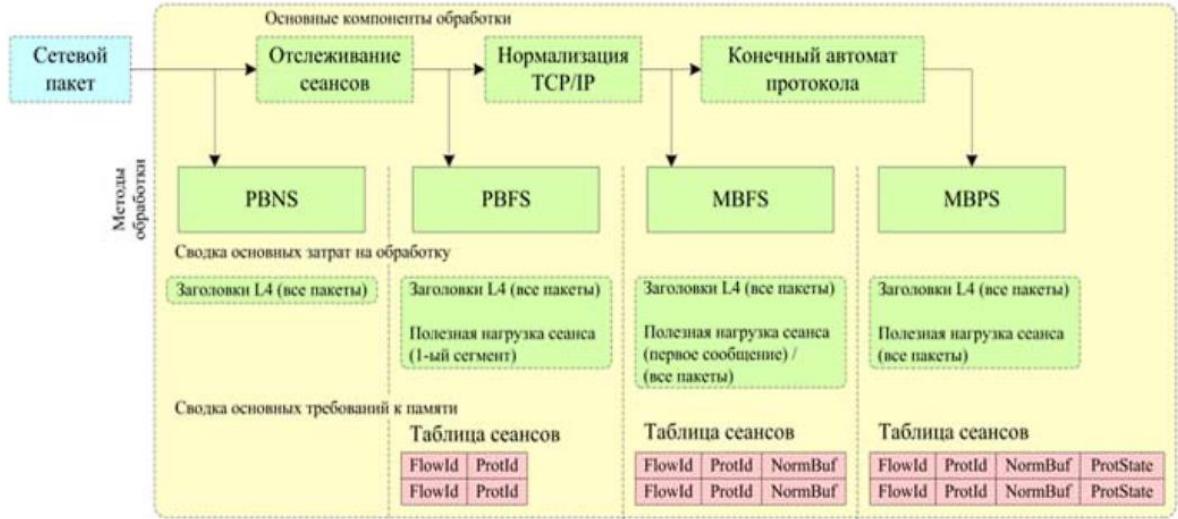


Рисунок 4 - Градации полноты учёта состояния потока.

Базовые реализации технологии DPI часто относятся к stateless-анализу. Этого уровня точности хватает для многих практических приложений и позволяет значительно экономить ресурсы (см. рис. 4). В то же время, существуют задачи, для которых такого уровня точности недостаточно. В то же время есть задачи, для которых этого уровня точности недостаточно. В качестве примеров можно привести две методики, использующие подход с определением состояния-проверку пакетов с заполнением состояния (SPI) и глубокую проверку содержимого (DCI).

1.3.1 Анализ сетевых пакетов с учётом состояния потоков

При использовании SPI программа или устройство при создании соединения удостоверяется, что он соответствует выбранной политике безопасности и запоминает настройки этого соединения до момента, когда он будет за. С помощью такого подхода проверяется корректность подключения – например, отсутствие пакетов на открытом сетевом порту после завершения подключения, так как в ином случае может быть сделан вывод, что что-то идёт не так. Реализации SPI содержатся в большинстве современных маршрутизаторов в виде брандмауэров SPI, что упрощает использование. Этот метод также используется в правительственные брандмауэрах, контрольно-

пропускных пунктах и во многих системах IDS/IPS, что позволяет обеспечивать большую безопасность, требуемую в этих предприятиях. Межсетевые экраны, использующие эту технологию, относятся к третьему поколению [17]. Такой подход позволяет отслеживать не только входящие и исходящие пакеты, но и индивидуальный статус соединения, который хранится в динамических таблицах. По этой причине при анализе следующего пакета могут учитываться не только заданные правила и политики в отношении адреса и содержимого пакета, но и состояние соединения, которому принадлежит пакет, и предыдущий пакет, которому он принадлежит, ссылки, а также другие соединения, связанные с данными.

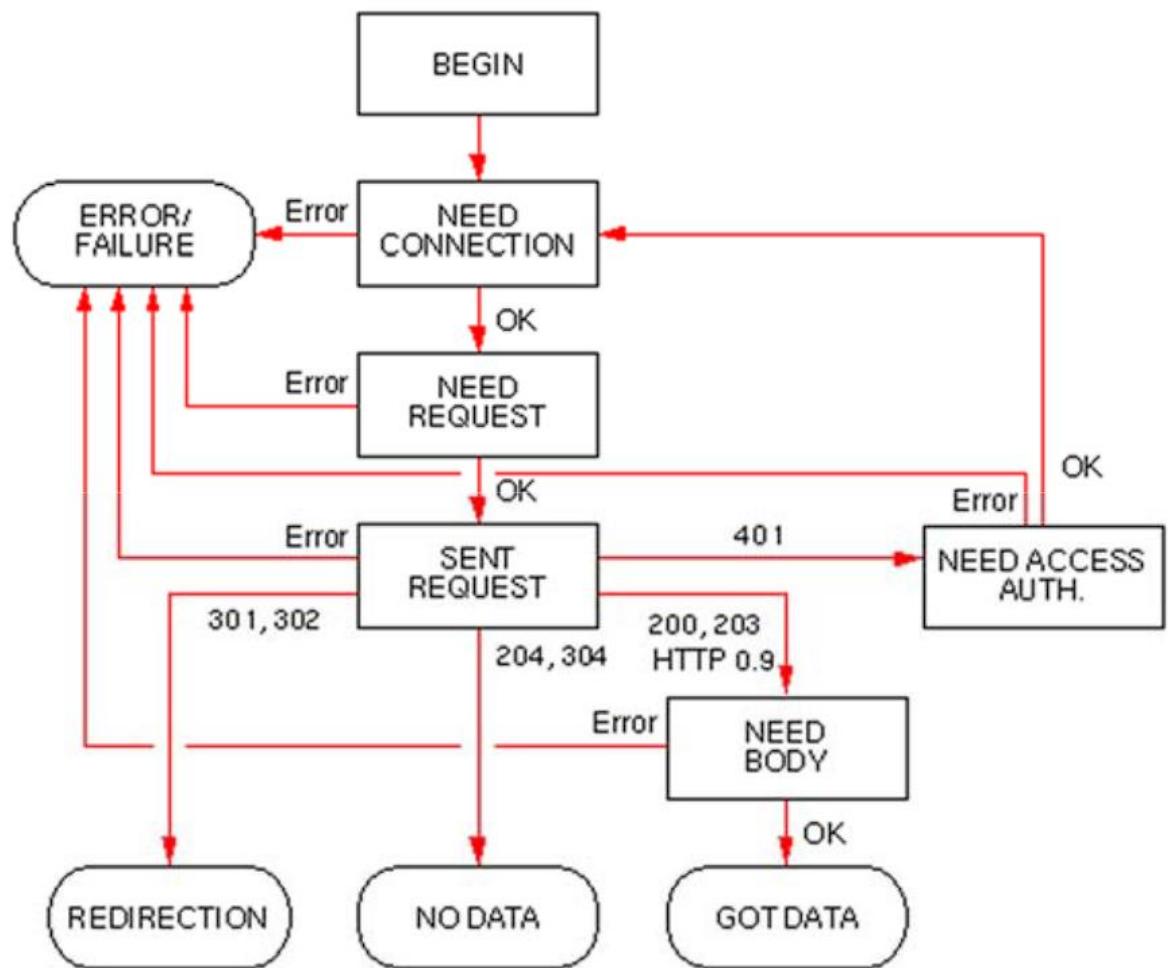


Рисунок 5 - Пример автомата состояний протокола HTTP.

1.3.2 Анализ содержимого сетевых протоколов прикладного уровня

При использовании DCI происходит как анализ протокола передачи данных, так и группировка сегментов данных в группы, отвечающие за воздействие на сервисные объекты, например, протокол передачи. Кроме того, в процессе использования DCI анализ не останавливается на обнаружении протокола, а также пытается идентифицировать программу, которая его использует, и собирает содержимое этого приложения в том виде, в котором он использует программу, за исключением отправки ее по сети. Примером использования данной технологии может служить функция прослушивания VoIP-запросов для перехвата трафика в анализаторе Wireshark [26]. С точки зрения функциональности, которую реализует вклад DCI, разделение (основной функционал DPI) представляет собой набор модулей анализа для различных протоколов прикладного уровня и различных типов данных в различных кодировках (например, MIME [27]), они встречаются. Функции парсинга модуля, с выполнением к результату:

1. Анализ буфера данных (сетевого пакета или собранной сессии), в соответствии с форматом протокола сообщений, описанного, как правило, на одном из специфических языков типа ASN .1 [28] и P4 [29].
2. Сборка протоколов сеансов с установлением соединений и их последующим анализом.

Одним из последних компонентов разработки инструментов DPI/DCI является универсализация и централизация анализа. Это можно было бы назвать «DPI как услуга» — именно под таким названием оно поставлялось [30]. Переход к этому поиску довольно похож на переход к программно-определяемым сетям (SDN) [31] в управлении массовым трафиком, где все решения используются алгоритмами маршрутизации и на уровне его реализации переносятся с конкретными моделями на выделенные устройства. Такие приемы упрощают масштабирование системы и позволяют эффективно расширять функционал без дополнительных работ по масштабированию и реконфигурации оборудования.

Демонстрация — выбор функционала аналогичных протоколов и удаление метаданных в виде редких модулей. Причем эти модули могут быть как чисто программными, так и привязанными к какому-то железу. Использование модулей как составной части систем и управления трафиком позволяет формулировать политики безопасности и другие типы политик в различных терминах, таких как URL-адреса, названия программ, особенности функций в программах управления (например, блокировка голоса в Skype). Это позволяет более гибко настраивать политики обслуживания, что ведёт к увеличению качества сервиса. Типовой пример работы по данной схеме [14] показан на рис. 6, где «Внешний интерфейс» — решение типа «DPI как услуга», правил, при помощи которых анализируется передаваемая информация, «Внутренний интерфейс» — это хост, который сохраняет различные виды информации.

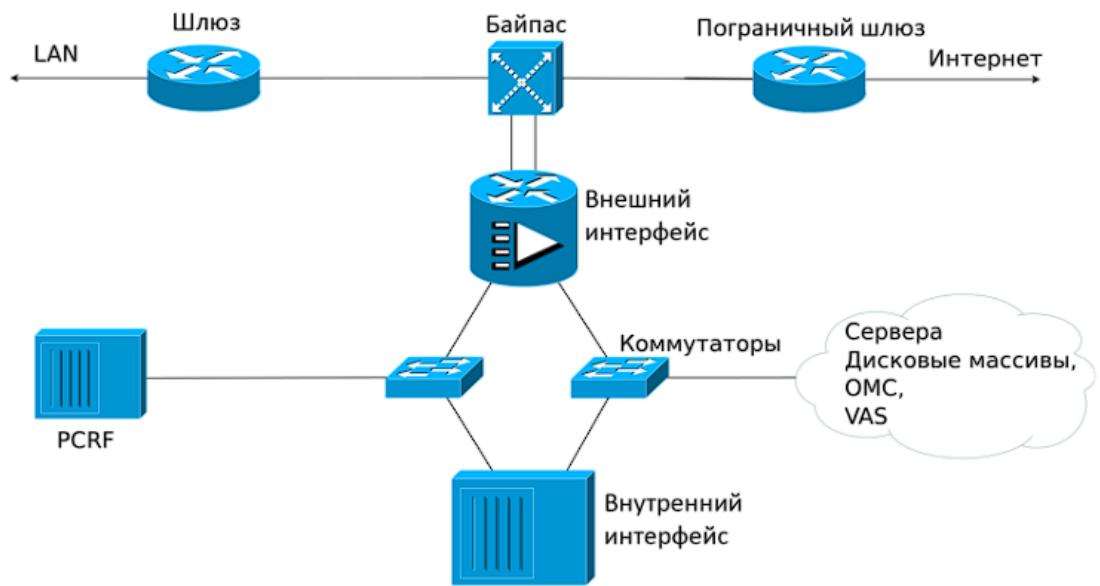


Рисунок 6 - Схема использования системы DPI для применения политик к сетевому трафику.

В следующем разделе диаграмма будет рассматриваться как инфраструктурная часть анализа, так как она одинакова в разных решениях для анализа сетевого трафика с небольшими изменениями. В частности, будут выделены отдельные этапы анализа с кратким описанием их характеристик. Каждый шаг будет обсуждаться более подробно в следующих разделах.

1.4 Общая схема инфраструктурных алгоритмов анализа сетевого трафика.

В общем виде, анализ трафика состоит из нескольких этапов, каждый из которых приводит к повышению уровня представления объекта анализа.

1. Захват пакетов, проходящих через контролируемое сетевое соединение. В результате выполнения данного этапа, будут получены сетевые пакеты, для их дальнейшего анализа. В зависимости от требуемой точности и скорости последующего анализа, а также имеющихся вычислительных мощностей могут использоваться различные подходы.
2. Нарезка, при которой анализируется не все содержимое пакетов, а только какой-то префикс (первые n байт).
3. Выборка, при которой перехватываются не все пакеты, а только их часть, которую можно выбирать по различным условиям, в зависимости от потребностей. В процессе развития технологий было предложено большое количество стратегий селекции [39]. Например, для мониторинга типов трафика подходит вариант с выделением каждого n -го пакета (равномерная выборка), где n можно выбирать в зависимости от соотношения ширины канала и пропускной способности системы анализа.
4. Для задач, для которых нужна наибольшая точность, необходим перехват всех данных всего трафика без потерь — для обозначения этого используется термин глубокий захват пакетов. Объединение пакетов в потоки по некоторым адресным характеристикам (генерация потока [43]), получение нового объекта анализа — сетевого потока. Если пакетные данные не учитываются при дальнейшем анализе, то такой вид анализа называется «анализ потока» — анализ на основе потока (Анализ содержимого сетевых потоков в отличие от анализа на основе пакетов, при котором анализируются пакетные данные). На рис. 7 показаны различия между типичными схемами пакетного и потокового анализа. Потоковый анализ получил широкое распространение из-за значительно меньших требований

к вычислительной мощности и пропускной способности, за счет значительного сокращения объема обрабатываемых данных, пусть этот метод и мог показывать худшие результаты. Этот вид анализа может выполняться как локально [43], так и удаленно от точки сбора данных [44]. Для передачи собранных данных из пункта сбора в пункт анализа используется большое количество протоколов, часть из которых стандартизована в виде IPFIX [45], а часть разрабатывается отдельными производителями — Cisco NetFlow, Juniper Jflow. В рамках этого подхода записи, описывающие поток, могут содержать другой набор данных. Наиболее распространенный набор таких данных следующий:

- IP адреса отправителя и получателя,
- протокол транспортного уровня,
- в случае протоколов TCP/UDP — номера портов отправителя или получателя,
- различные метрики: объём переданной информации, время инициализации и прекращения потоков.

В большинстве реальных задач количество потоков немного меньше количества пакетов (примерно на порядок) из-за большого количества очень коротких потоков, состоящих из нескольких пакетов — флеш-потоков [46]. Для решения этой проблемы было предложено использовать выборку для потоков [41], что позволяло уменьшить значимость этой проблемы. Еще одна особенность этого метода заключается в том, что из-за ограниченной памяти устройство, которое собирает пакеты, не может отслеживать один поток в течение произвольного периода времени.

3. Выполнение классификации по протоколу прикладного уровня или конкретному сетевому приложению. После выполнения этого действия можно провести дополнительную обработку итогового объекта, конкретный вид которого коррелирует с решаемой задачей:

- разбор полей протокола,

- сборка сессии протокола для протоколов с установлением соединения,
- извлечение данных приложения,
- разбор данных приложения.



Рисунок 7 - Различия типичных схем packet (слева) и flow-based (справа) анализа.

Для полноты картины следует сказать, что помимо упомянутых выше подходов на основе пакетов и потоков существует еще один источник данных о сетевом трафике — так называемая База информации управления (MIB) [48] — виртуальная база данных, используемая для управления объектами в коммуникационной сети. Модули накопления, хранения и обмена данными в формате MIB реализованы в большинстве устройств, что упрощает работу с ними. Данные передаются по протоколу SNMP [49]. Данные, полученные таким образом, имеют небольшой объем и неспецифичны для протоколов. Следует отметить, что одной из причин развития MIB и потоковых подходов, несмотря на их относительно невысокую точность, стала продолжавшаяся до сих пор глобальная дискуссия [50] о правомерности и допустимости глубокого анализа трафика с точки зрения нарушения безопасности, прав на неприкосновенность частной жизни и т. д., результаты которых мы можем наблюдать и поныне. На данный момент одним из следствий этого обсуждения является, в частности, то, что в научных работах трафик, который подвергается глубокому анализу,

предварительно проходит процедуру «анонимизации» с использованием специальных средств [51], для сохранения конфиденциальности пользователей, благодаря которым этот трафик был собран. Далее мы уделим внимание отдельным частям приведенной выше общей схемы анализа сетевого трафика, методы, алгоритмы и подходы, а также их особенности и ограничения применимости.

1.4.1 Захват сетевых пакетов

Программные и аппаратные средства, осуществляющие захват трафика относятся к классу сниферов (sniffers). Для этой задачи захвата трафика могут использоваться как специальные сетевые карты. Такие карты, как правило, реализованы на базе FPGA или ASIC и имеют встроенные средства для проставления временных меток (которые очень помогают при анализе трафика), аппаратной фильтрации (что снижает нагрузку), снятия некоторых заголовков низкоуровневых протоколов (что уменьшает объём данных без потери точности), балансировки нагрузки между процессорами на многопроцессорных компьютерах с учётом IP-потоков (что ускоряет анализ), выявления ошибочных и дублирующихся пакетов (что также снижает нагрузку при анализе). При этом вся обработка (в том числе и копирование данных в память компьютера из памяти сетевой карты) осуществляется без привлечения ресурсов ЦПУ. По мере развития технологий многие из описанных свойств реализуются и на базе стандартных сетевых карт. Технология реализации таких дополнительных функций носит название TCP Offload Engine (TOE). Она включает в себя следующие вещи, среди которых стоит отметить:

- Деление объёмных пакетов TCP на небольшие,
- Воссоздание изначальных пакетов, которые были разделены,
- Удостоверение, что контрольные суммы верны в транспортных протоколах,
- Шифрование и дешифрование передаваемой информации.

Основной проблемой стандартных сетевых адаптеров является не скорость передачи данных как таковая, а количество пакетов в единицу времени. Это связано с особенностями внутренней реализации обработчиков пакетов на сетевых картах, драйверов сетевых карт и сетевых стеках программного обеспечения ОС. Другой проблемой является точное представление временных меток, а это имеет значение при выстраивании точной картины.

Также появляются отдельные вызовы, связанные с переходом на сетевые соединения, которые способны обеспечивать более высокие скорости. По большей части эти проблемы возникают из-за:

- Неспособностью аппаратных средств соответствовать имеющейся скорости,
- Архитектурными ограничениями при взаимодействии аппаратуры с ОС и ОС с пользовательскими приложениями,
- Объем памяти, необходимый для хранения полученных данных. Среди проблем наиболее распространённых решений, которые приводят к снижению производительности, можно назвать:
 - Двойное копирование данных пакета (из карты в память ядра, из памяти ядра в память пользовательского процесса).
 - Большое число прерываний от сетевой карты (на каждый пакет, чтобы он был скопирован в буфер ядра).
 - Недостаточное использование параллелизма на уровне отдельных ядер и процессоров (по умолчанию все прерывания обрабатываются одним ядром).
 - Проблемы с синхронизацией при доступе к данным из нескольких потоков выполнения. Если полученные данные необходимо обрабатывать в нескольких потоках, между этими потоками возникает ситуация конкуренции за ресурсы, что в свою очередь требует дополнительного участия ЦПУ/ОС. В зависимости от количества копий пакетных данных, которые выполняются в процессе перехвата, решения делятся следующим образом:

- 0-копия (zero-copy). Для поддержки этого способа необходима поддержка со стороны сетевой карты — в ней должен быть DMA-контроллер.
- 1-копия. Для реализации такого подхода возможны несколько вариантов — разработка анализатора на уровне ядра, что является очень сложной задачей, либо прямое отображение памяти ядра в память пользовательского процесса.
- 2 копии. Стандартное решение на основе LibPcap или WinPcap. Для решения этих задач реализован ряд специализированных драйверов и сетевых стеков, к которым относятся, например, коммерческое решение Sniffer10G от Emulex и Myricom, а также открытая разработка PF_RING от Ntop. Эти решения используют схему кольцевого буфера как более эффективную, а также оптимизированы для многопроцессорных и многоядерных компьютеров, которые сейчас составляют большинство. В частности, они реализуют следующий функционал:
 - Обработка перехвата пакетов с использованием большого числа потоков исполнения.
 - Балансировка нагрузки между ядрами (одно ядро — одна входная очередь).
 - Фильтрация пакетов внутри сетевой карты. Для реализации этих функций используется как аппаратная поддержка со стороны архитектуры, так и поддержка со стороны ОС (специализированное API).

1.4.2 Группировка сетевых пакетов в потоки

Группировка пакетов в потоки — достаточно стандартная и простая операция. Основное различие между различными реализациями этой функциональности связано с тем, какие поля адресной информации и как их использовать для идентификации потока. Наиболее часто используемое определение потока было дано ранее. Поскольку он использует 5 полей в качестве ключевой информации для определения того, принадлежит ли

конкретный пакет конкретному потоку, для его обозначения обычно используется термин 5-кортеж. Также иногда используются двусторонние потоки, симметричные перестановке пар $\langle \text{srcIP}, \text{SrcPort} \rangle$ и $\langle \text{dstIP}, \text{dstPort} \rangle$. Модуль, отвечающий за группировку пакета, обычно называют генератором потока. Во время работы этот модуль сохраняет в памяти отображение актуальной ключевой информации о данных конкретных потоков. При появлении нового пакета с ним выполняются следующие операции. 1. Ключевая информация извлекается из пакета, чтобы определить, к какому потоку он принадлежит. 2. Поиск выполняется в текущем наборе потоков. 3. Если поток найден, в данных потока увеличиваются соответствующие счетчики – как правило, они включают в себя время жизни потока, количество пакетов и байтов в потоке. Если поток не найден, создается новая запись потока и в нее добавляется информация о текущем пакете. Результаты оценки ресурсов (из работы [38]), которые нужны для выполнения первые 2 операций, представлены в таблице 1. Абсолютные цифры, приведенные на рисунке, могут быть не совсем актуальными на данный момент, но их ценность заключается, прежде всего, в относительной стоимости операций.

Таблица 1 - Скорость основных операций при анализе передаваемой информации

Операция	Стоимость (такты процессора)
Извлечение идентификатора	79
Поиск или добавление идентификатора	48
Поиск сигнатуры потока с использованием детерминированного конечного автомата (мин.,ср., макс.)	12-4332-8950

Описанная схема хранит в себе значительный минус — предполагается, что модуль имеет бесконечную память, так как нет определения условий завершения потока и поэтому непонятно, когда удалять запись о потоке с отображения. В случае транспортного протокола с установкой соединения (например, TCP) учитывается явная процедура завершения соединения. Если используются протоколы по типу UDP, то такой подход не работает, поэтому, в частности, использовался подход при использовании таймера — для примера, соединение прекращалось после 10 минут. Тот же подход используется для слишком длинных потоков TCP [43].

1.4.3 Классификация сетевого трафика.

Перечислим объекты, которые получаются на выходе алгоритма, насколько они пригодны для последующей классификации. По этому критерию выделяют три основных варианта классификации. Они перечислены ниже в порядке увеличения «точности» классификации:

- Тип трафика не является достаточно хорошим видом классификации и, как правило, либо не подлежит последующему анализу, либо подлежит достаточно простой более доказательной классификации. В зависимости от приложения могут быть разные типы. В качестве примера можно привести:
- Передача данных между конечными пользователями, трансляция графической информации в режиме реального времени,
- Трафик, который можно распознать, как вредоносный, и трафик, генерируемый незаражёнными пользователями,
- Высокий процент оригинального трафика, если используется система для контроля оригинальности.
- Используемый протокол прикладного уровня (идентификация протокола) содержит достаточно информации и может использоваться напрямую, например, в системах сбора и мониторинга статистики для повышения уровня точности. Основным методом дальнейшей обработки является парсинг протокола, включающий в себя две основные функции - сборку

сеанса прикладного уровня, при необходимости протокол извлечения данных из отдельных полей (метаинформация уровня протокола).

- Приложение, передающее данные, дает максимально детализированный уровень классификации.

В разных приложениях результаты анализа и классификации трафика используются по-разному. Например, в случае системы защиты от вредоносного кода под протоколом можно понимать протокол управления и контроля (C&C) ботнета, а под приложением — конкретный вирус. Соответственно, извлекаемая метаинформация — это команды ботнета, передаваемые им данные, а целью анализа является уточнение его функциональности, оценка распространенности и изучение возможностей его деактивации. В случае системы профилей пользователей для последующего показа таргетированной рекламы (например, IMarker) в качестве протокола может выступать HTTP, в качестве приложения может выступать браузер, а объектом анализа является запрос пользователя к поисковой системе, который подвергается дальнейшему анализу текста для извлечения ключевых слов. Выбор конкретной прикладной задачи может существенно повлиять как на выбор алгоритма классификации, так и на его параметры и производительность. В качестве примера можно рассмотреть следующее сравнение. В случае системы статистики алгоритм классификации обычно работает последовательно над пакетами каждого потока «до первого срабатывания». Схема такой классификации приведена на рисунке 8.

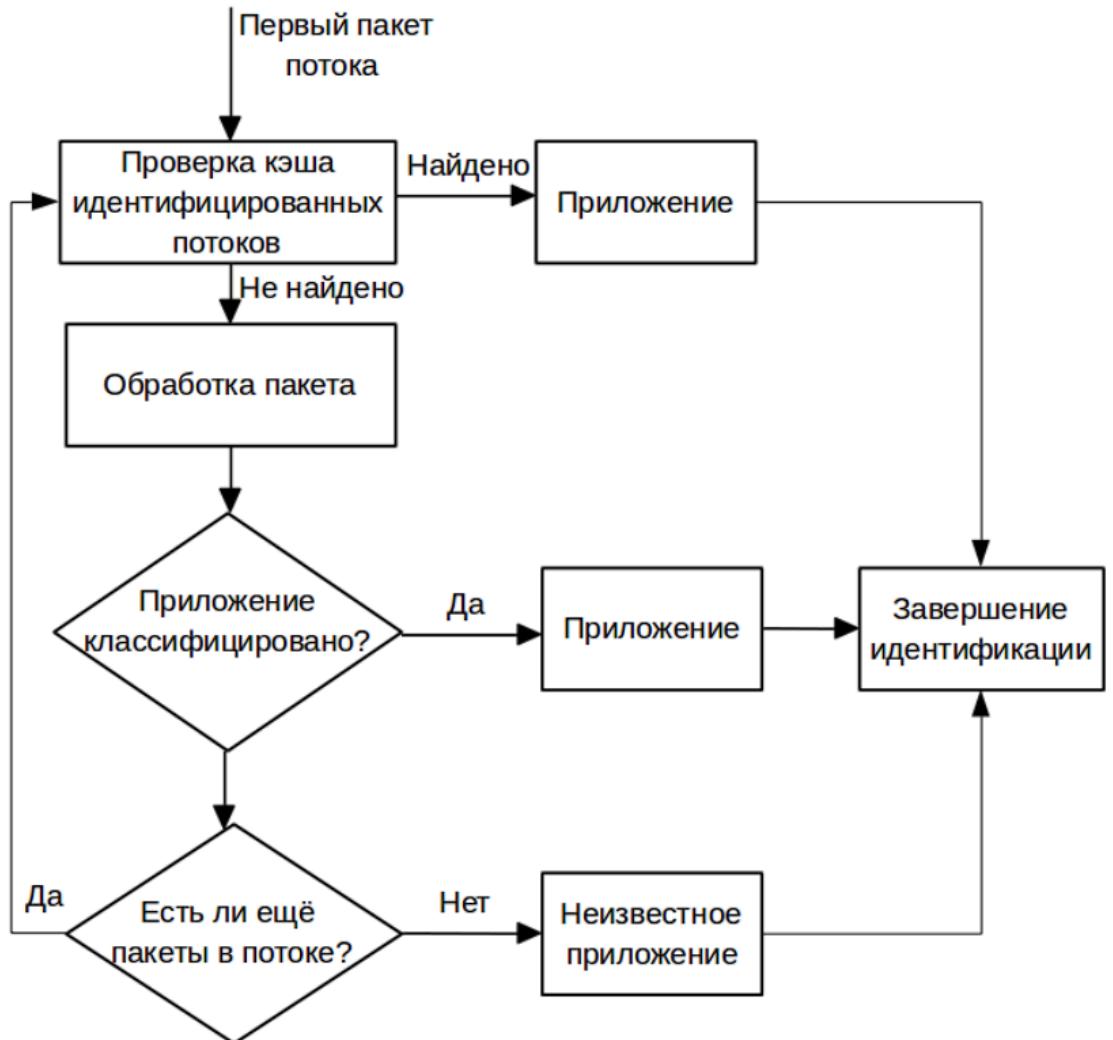


Рис. 8 - Схема классификации «до первого срабатывания».

Однако в случае систем анализа с использованием конкретных слов этот метод не подходит, так как в одном и том же потоке разные слова находятся в разных пакетах и в этом случае этот поток будет классифицироваться как несколько классов. В целом очевидно, что первый подход гораздо продуктивнее, так как приходится анализировать гораздо меньшие объемы данных. Кроме того, в ряде подходов для дополнительного ускорения анализируется не все содержимое пакета, а только часть его префикса (по аналогии со слайсингом). В [38] была исследована взаимосвязь размера префикса пакета и точность классификации по протоколам, а также скорость работы классификатора. Результаты представлены на рисунке 9, где на левом графике ошибки

классификации обозначены как неправильно классифицированные, а трафик, который не удалось классифицировать как неизвестный.

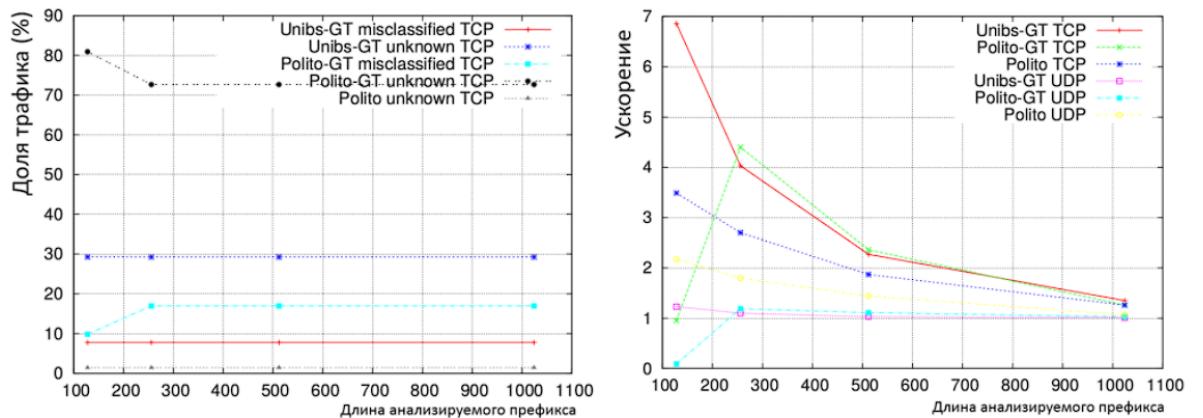


Рис. 9 - Оценка влияния длины префикса на точность классификации (слева) и скорость (справа).

Рисунок 9 - Оценка влияния длины префикса на точность классификации (слева) и скорость (справа).

Классический подход к классификации — анализ содержимого пакетов (на основе полезной нагрузки). При этом, как правило, осуществляется поиск так называемых «сигнатур» (сигнатурных подходов) — характерных признаков, которые создаются заранее для каждого приложения или их групп. Классификация может выполняться как на уровне отдельных пакетов (stateless analysis), так и учитываться состояние потока (statefull analysis). Для повышения точности распознавания в некоторых подходах используются уточненные «подписи» на основе автоматов состояний протоколов (см. рис. 5). Как показано на рисунке 9, классификация является наиболее загруженным алгоритмом анализа сетевых пакетов. В силу исторических причин из-за недостатка мощностей активно использовался выбор источника данных, который поступал в качестве входа на классифицирующий алгоритм. При этом выбор осуществлялся так, чтобы получаемые данные без потери точности занимали бы меньше пространства памяти. Эта группа подходов (в отличие от «сигнатурного») относится к классу логических выводов.

1.4.4 Подходы на основе вывода.

Все подходы на основе вывода можно разделить на группы по двум основным параметрам:

- используемые для вывода данные,
- используемый для их анализа алгоритм. Все виды данных, в свою очередь, можно разделить на:
 - характеристики отдельных пакетов в рамках отдельного потока (packet based),
 - характеристики потоков в целом (flow based). К первой группе относятся подходы, использующие такие характеристики как: временные промежутки между пакетами, последовательности размеров пакетов [56], и др. Ко второй группе относятся два основных подхода.
 - Анализ портов - идентификация происходит при помощи номера порта на основе базы данных характерных статических портов, использующих протоколы, зарегистрированные в IANA. Этот способ считается неэффективным, так как на данный момент существует большое количество протоколов с динамическими номерами портов. В частности, к таким протоколам относятся практически все реализации P2P.
 - Различные методы при помощи анализа метаданных о том, насколько много трафика генерирует различные устройства: в какое время и в каком количестве устройство в сети информацией участвовало в обмене данными и т.д. Впоследствии эта информация анализировалась с учётом созданных правил. Этот подход был описан в работе [57]. По одному из них алгоритмы анализа данных делятся на два основных:
 - сравнение с тем или иным видом заранее созданного шаблона,
 - подход на основе машинного обучения и последующего распознавания.

В настоящее время методы на основе машинного обучения развиваются очень активно. Причиной развития является доступность большого количества

различных данных для обучения (социальные сети, крупные БД и т.д). На данный момент эта группа методов включает в себя большое количество алгоритмов: байесовские сети, деревья принятия решений, методы опорных векторов, методы k-средних и др. Эти методы делятся на группы по методу обучения [58], который используется для их конфигурирования:

- классификация,
- кластеризация,
- ассоциирование,
- численное предсказание.

1.4.5 Методы на основе сигнатур

По мнению экспертов, недостаток этих методов - их высокая ресурсоемкость, связанная с необходимостью просмотра больших объемов данных. Однако, в настоящее время вычислительные мощности позволяют использовать более точную и не основывающуюся на выводе сигнатурные методы. Эти методы делятся между собой на две большие группы:

- поиск строк (string matching)
- поиск регулярных выражений (regexp matching).

В процессе разработки для поиска строк использовалось большое количество различных алгоритмов поиска строк с различными преимуществами и недостатками, что определило область их применения [59,60]. В [61] проведен обзор и сравнение большого количества методов поиска строк по тому, как реализован алгоритм сравнения с имеющимися сигнатурами. Выделено 4 группы методов:

- Последовательное сравнение со всеми сигнатурами.
- Дерево сравнений.
- Декомпозиция - при её использовании некоторые части классифицируются отдельно друг от друга, с последующим воссозданием изначальной картины.

- Ассоциативный доступ (Tuple Space), при котором сигнатуры разбиваются на группы бит, с которыми проводятся операции сравнения.

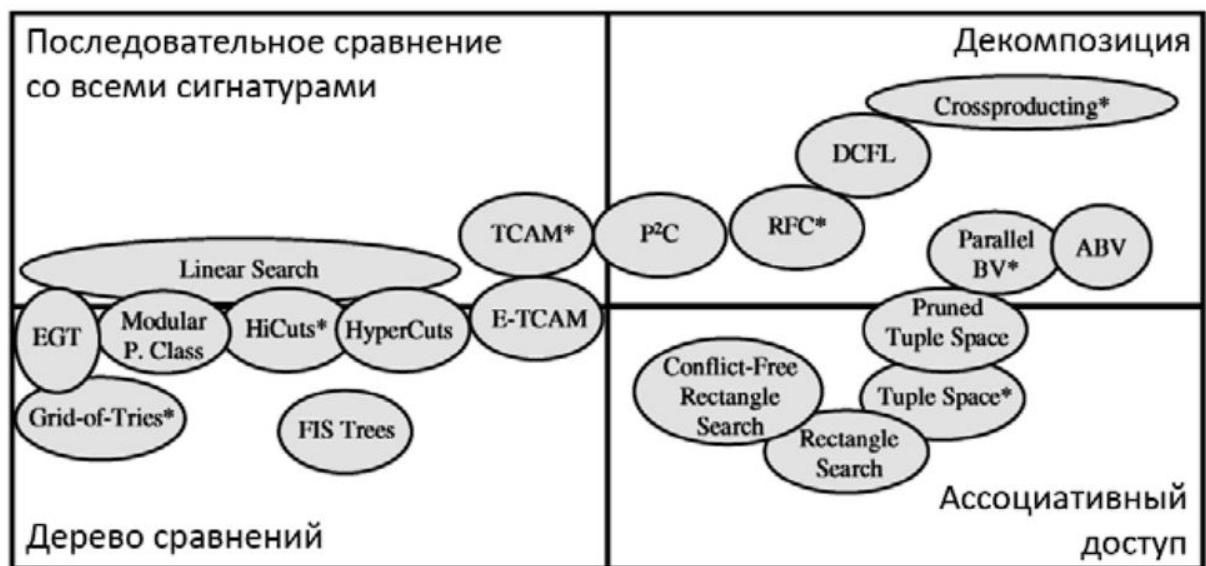


Рисунок 10 - Распределение алгоритмов поиска строковых сигнатур по данным группам.

1.4.6 Сигнатуры на основе регулярных выражений.

С ростом количества протоколов и их сложности строковое представление оказалось недостаточно выразительным, и поэтому для описания подписей стали использовать обычные языки в виде грамматик и регулярных выражений. Чтобы эффективно искать скрытые данные, регулярный язык описания сигнатуры представляется в виде конечного автомата. Два основных типа автоматов – детерминированные и недетерминированные. Эти две идеи имеют свои достоинства и недостатки. Такие подходы могут не сработать, если подписи были разбиты на несколько пакетов в сети IP или TCP. В этом случае необходимо перед поиском сигнатуры выполнить дефрагментацию IP и нормальную работу TCP соответственно. Основным достоинством конечных автоматов (НКА, НКА) является их компактность: объем памяти равен количеству символов, входящих в регулярное выражение. Несмотря на это, конечным автоматам может потребоваться до $O(N)$ обращения к памяти для обработки каждого символа.

входа в память. N - число состояний автомата [63] По этой причине возможности использования ЧПУ для высоконагруженных систем ограничены. Демагогические конечные автоматы (DKA, DFA) требуют одного доступа к памяти для каждого входного символа. Неудобства могут возникнуть из-за их большого размера: количество состояний DKA может расти экспоненциально, «экспоненциальный взрыв», и ограничивается $O(2l)$, где l - общая длина регулярных выражений в каноническом представлении. Исследование, проведенное в [38], показало влияние различных типов регулярных выражений на рост размера автомата. Результаты представлены на рис. 11. С точки зрения их влияния на размер автомата были выделены 3 типа:

- выражения, которые относились анализу начала пакета;
- выражения, которые относились к началу пакета и при этом имели в себе звёздочку Клини;
- выражения, которые не относились к началу и при этом имевшие в себе звёздочку Клини (*).

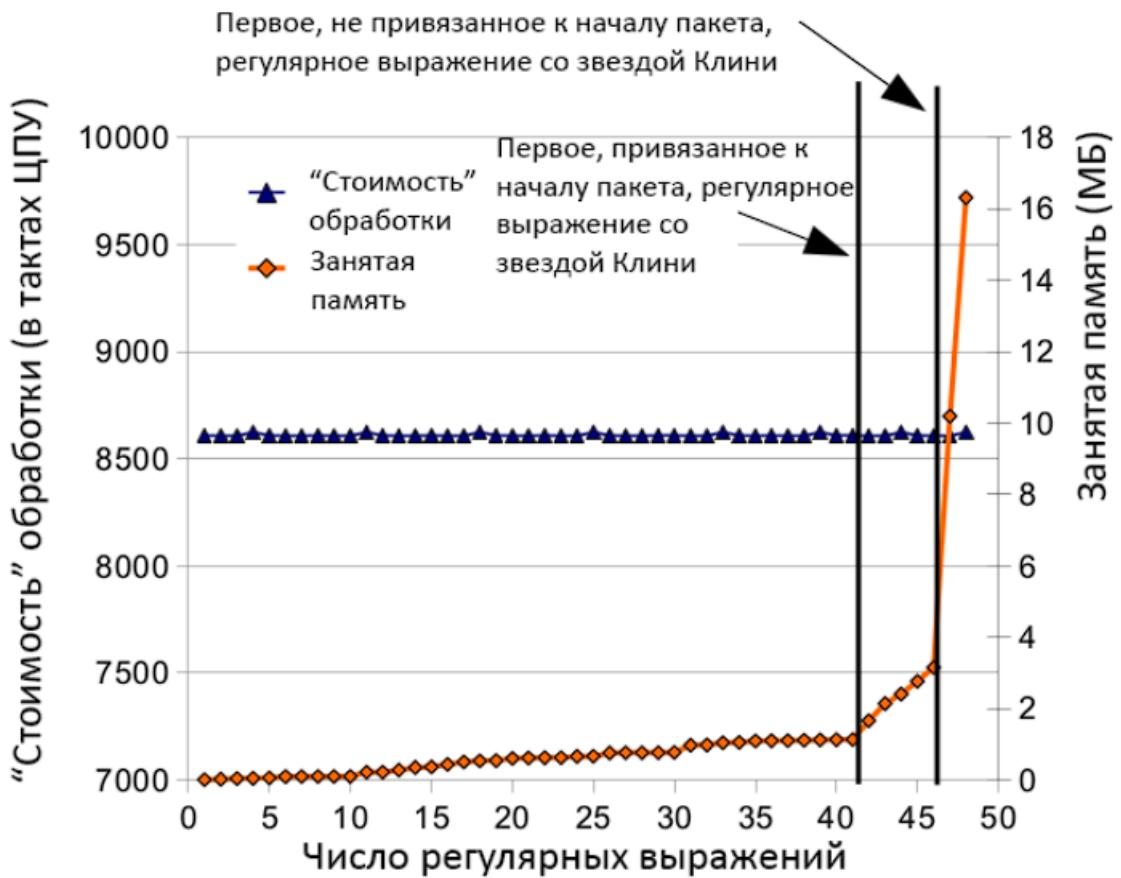


Рисунок 11 - Резкое увеличение DFA при присоединении регулярных выражений, использующих звёздочку Клини.

Для уменьшения размера автоматов часто используются различные виды сжатия. Такие машины называются Сжатые ЦФА (Compressed DFA, dfa). В табл. 2 приведено сравнение трех основных типов автоматов по размеру и поисковой производительности, взятое из [38]. Эти автоматы были построены с использованием регулярных выражений классификатора L7[63]. Чтобы предотвратить экспоненциальный рост размера, детерминированные автоматы были разделены на 4 части.

Таблица 2 - Сравнение размеров и скорости работы основных видов конечных автоматов.

Алгоритм	Стоимость в тактах ЦПУ (мин,ср., макс.)	Количество автоматов	Размер автомата

НКА	$2.2 \cdot 10^{**4}$, $4.1 \cdot 10^{**7}$, $8.9 \cdot 10^{**7}$	1	509 Кб
ДКА	52, $2.5 \cdot 10^{**4}$, $3.6 \cdot 10^{**4}$	4	230 Мб
Сжатый ДКА	268, $1.2 \cdot 10^{**5}$, $1.7 \cdot 10^{**5}$	4	53 Мб

По причине мягких требований к памяти, конечные автоматы (и их модификации) получили гораздо большее распространение в быстродействующих системах анализа. На сегодняшний день системы для классификации трафика предъявляют высокие требования к скорости обработки данных и к количеству используемых в обработке регулярных выражений. Однако ни DKA, ни НКА не могут одновременно удовлетворить требования к скорости и объему памяти, и по этой причине сейчас исследуются возможности гибридных представлений. Автоматы представляют собой таблицы состояний, в каждой ячейке которых находится список возможных переходов из этого состояния в другое. Именно поэтому два основных направления работы направлены на уменьшение количества состояний и переходов соответственно.

1.4.7 Анализ данных в разных представлениях

Одной из важных проблем контентных классификаторов является тот факт, что одни и те же данные могут кодироваться по-разному при передаче по сети. В частности, под «различными взглядами» в этом разделе есть следующие аспекты:

- Различные методы кодировки, в частности для текстовых данных — ASCII и Unicode кодировки, а для бинарных данных — различные транспортные

кодировки, например представление в виде текста (binary-to-text), примером которых является Base64.

- Архивированность информации, чтобы обеспечить меньшее объём передаваемого трафика.
- Шифрование данных для обеспечения безопасности, например использование криптографических алгоритмов RC4 и AES в протоколах SSL/TLS.

По данным различных исследований, сжатый и шифрованный трафик (иногда используется общий термин «непрозрачный», opaque) составляет всё большую долю от всех сетевых потоков данных [54]. Причиной этому являются:

- увеличение использования сервисов видео-хостингов, которые используют архивирование графической информации,
- увеличение доли P2P-сервисов, трафик которых нередко идёт зашифрованным,
- переход большинства веб-сервисов на защищённые протоколы (HTTPS),
- использование архивирования трафика HTTP в большинстве веб-ресурсов.

Проблема классификации этих видов трафика имеет несколько аспектов:

- Для корректной классификации такого трафика требуется дополнительный функционал.
- Неудачная попытка классифицировать трафик «в лоб» значительно снижает общую производительность класса. Необходимо внимательно изучить все пакеты данных, которые проходят через большую часть машины и результат может быть отрицательным или положительным. То есть это «худший случай», характеристики производительности алгоритмов классификации которого существенно ниже среднего (см. табл. 2).

Для решения первого аспекта проблемы используются несколько подходов:

- Генерация копий сигнатур, которые подвергаются различным видам сжатия и кодирования.
- Использование модулей, которые распаковывают/перекодируют данные перед их классификацией. Этот метод имеет те же ограничения, что и предыдущий, а также плохо масштабируется. Кроме того, этот метод повышает уязвимость системы к атакам zip-бомбы [70], при которых размер распакованных данных превышает размер сжатых данных на несколько порядков.
- Установка системы анализа на месте или после инструмента декомпрессии/расшифровки данных. Примером такого инструмента является прокси-сервер.

1.4.8 Классификация угроз

Статистическое обнаружение аномалий является одним из таких подходов, где используется классификация угроз, а не идентификация/классификация протоколов. Его использование подразумевает обучение модели на трафике, который не содержит атак. После чего система будет способна находить отклонения в реальном трафике. Такие подходы называются «обнаружение статистических аномалий». Такой метод широко используется в работе систем, защищающих от таких атак как DDoS и IDS.

1.5 Требования, предъявляемые к современным средствам анализа содержимого сетевого трафика

С учетом приведенного обзора основных алгоритмов и схем анализа сетевого трафика можно сформулировать ряд функциональных и нефункциональных требований к современным системам анализа сетевого трафика. Все требования можно разделить на подсистемы, к которым они относятся, и отдельно выделить те, которые относятся ко всей системе в целом:

- Поддержка масштабирования по пропускной способности анализируемого канала передачи данных.

- Минимизация числа перестановок пакетов в рамках отдельных потоков.
- Возможность добавления программ для предварительной обработки пакетов, осуществляющей перед передачей подсистеме классификации.
- Поддержка разбора всех протоколов ниже сетевого уровня, встречающихся в контролируемом канале (MPLS, VLAN и т.д.) Это необходимо, для обеспечения попадания всех пакетов одного потока в одну очередь обработки при выполнении балансировки нагрузки (хеширование должно выполняться на уровне IP-пакета).
- Задействование для сохранения анализируемых пакетов и режима zero-copy буферного кольца, если сетевое оборудование обеспечивает поддержку этого, или 1-copy, если оборудование не содержит таких функций.
- Для эффективного использования ресурсов многопроцессорных и многоядерных машин необходима поддержка того или иного типа технологии RSS.
- Средства для указывания класса самой важной информации, при помощи которой возможно понять, относится ли пакет к сегменту данных.
- Максимальное количество одновременно обрабатываемых потоков и время жизни каждого отдельного потока в условиях ограниченных ресурсов памяти.
- Для обработки сжатых данных необходима возможность одновременного отслеживания потока, который представлен как в сжатом, так и в разжатом виде.
- Встроенная защита от атак типа «zip-бомба».
- Отслеживание факта связанности потоков (например, потока управления и потока данных в случае FTP), в частности, для уточнения классификации.
- Алгоритм поиска сигнатур в среднем должен иметь сложность больше или равной линейной, а желательно и в самом плохом случае для того,

чтобы обеспечить сохранение работоспособности системы к атакам злоумышленников.

- Расширяемый набор «сигнатур» для поддержки новых протоколов, их групп и сетевых приложений.
- Хорошая масштабируемость по памяти при росте количества «сигнатур».
- Возможность анализа данных, представленных в различных кодировках.

1.6 Эволюция методов классификации трафика

Методы классификации сетевого трафика развивались и модифицировались на протяжении многих лет. В первую очередь это связано с требованиями и ограничениями, накладываемыми сетью. Изменение устройства сетевого трафика и особенностей его передачи приводит к тому, что старые методы классификации становятся неэффективными или просто непригодными для использования. С другой стороны, развитие методов классификации и оборудования, на котором может работать система, позволяет использовать больше признаков и более развитые способы их применения для принятия решений.

1.6.1 Классификация по номерам портов

Первые системы классификации трафика были основаны на извлечении номеров портов из пакетов и сравнении их со списком IANA (Internet Assigned Numbers Authority, «Администрирование адресного пространства Интернета»). IANA выделяет и регистрирует номера портов, используемые для определенных конкретных целей, например, порт 80 выделяется для протокола HTTP. Информация о протоколе уже может быть использована для приблизительного определения типа активности пользователя. Этот метод классификации работает очень быстро и не требует хранения данных о потоке, он прост в вычислительном

отношении. Это делает его удобным, например, для использования в межсетевых экранах для фильтрации трафика. Однако он имеет ряд существенных недостатков, что по мере развития Сетевого устройства негативно сказывается на результатах его работы.

1.6.2 Глубокий анализ пакетов

Следующим шагом в развитии классификаторов интернет-трафика стало использование технологии DPI. Фильтрация сетевых пакетов в этом случае осуществляется по ихному содержимому, то есть проводится анализ не только заголовков, но и всего трафика на уровнях модели OSI от второго и выше. Этот метод показывает высокую точность работы, а полученная с его помощью разметка часто принимается за эталон для данных с неизвестными классами. Для классификации с помощью DPI создается библиотека сигнатур и шаблонов пакетов, и для каждого пакета в этой библиотеке ищутся совпадения.

Было замечено, что некоторые проприетарные протоколы передают информацию на битовом уровне, что привело к созданию инструментов, работающих и на этом уровне.

При всех своих преимуществах метод DPI сталкивается с существенными проблемами в своей работе. Невозможность работы с зашифрованным трафиком, доля которого в Интернете растет каждый год и высокие требования к ресурсам. В процессе хранения данных пакета и библиотеки символов требуется достаточно большой объем памяти, а с увеличением количества известных классов увеличивается размер этой библиотеки. Это увеличивает время поиска в ней сходства. Из-за этого данный метод не подходит для работы в высокоскоростных сетях в режиме реального времени. Также, существует определенная сложность в создании и поддержании стабильности библиотеки сигнатур для непрерывного роста количества протоколов или приложений на сайте.

И еще один момент, который необходимо учитывать – это защита приватности пользователей. Сети - проблема, актуальная для всех систем,

использующих полезную нагрузку пакетов. Этот аспект следует учитывать при создании, обучении и эксплуатации систем глубокого анализа пакетов.

Некоторые способы ограничивают объём использования данных, например первые 40 бит, но это не решает проблему полностью.

1.6.3 Стохастический анализ пакетов

Стохастический анализ пакетов (SPI, Stochastic packet inspection) для классификации пакетов изучает статистические свойства их содержимого. Например, в [5] для исследования случайности распределения первых байтов полезной нагрузки пакета используется критерий Пирсона хи-квадрат. Это создает модель синтаксиса протокола, используемого приложением. В [6] потоки определяются как зашифрованные или незашифрованные в зависимости от энтропии первого пакета. В [7] вычисление энтропии первых байтов полезной нагрузки идентифицирует тип содержимого как текст, двоичный файл или зашифрованный файл, что позволяет установить приоритет передачи некоторых файлов. Однако такую классификацию сложно назвать точной или подробной, поскольку для одного и того же приложения могут использоваться все типы контента. Кроме того, хотя в стохастическом анализе используются более простые операции, чем в глубоком пакетном анализе, он все же использует для анализа большой объем памяти. В связи с этим данный метод не получил широкого распространения.

1.6.4 Использование машинного обучения для классификации трафика

Текущие тенденции сетевого трафика, широкое использование шифрования, рост скорости передачи данных, постоянное появление новых классов трафика – всё это требовало появления новых способов классификации. Этот метод позволяет во многом облегчить работу с созданием наборов отличительных признаков классов, автоматизируя этот процесс на основе

анализа большого количества примеров этих классов (который гораздо проще собрать, чем разобрать вручную). Помимо этого, большинство предложенных методов работают с общими характеристиками потоков, а не с полезной нагрузкой пакетов, что решает проблемы, связанные с шифрованием и защитой данных. Он также обеспечивает преимущества в быстрой классификации и уменьшает объем памяти, необходимый для принятия решений.

Далее будут подробно рассмотрены методы машинного обучения для классификации сетевого трафика: типы классификации, используемые модели и признаки, а также наборы данных, на которых модели обучаются и тестируются.

1.7 Типы классификации

Трафик в сети можно классифицировать по отдельным пакетам, и методы классификации на основе портов и DPI способны решить эту проблему, но сейчас в большинстве работ классификация выполняется для потоков. Здесь и далее потоком являются пять значений: <

IP-адрес источника, IP-адрес получателя, порт источника, порт получателя, тип транспортного протокола>.

Есть методы, когда запросы отправителя и ответы получателей интерпретируются как два разных встречных потока. Более распространенным решением является объединение этих потоков в единый двунаправленный поток. Таким образом, поскольку Интернет-поток обычно подразумевает одно завершение процесса обмена между отправителем и получателем (клиентом или сервером) в целом не возникает проблем с отнесением всего потока в целом к одному классу.

Из-за того, что потоки отличаются по продолжительности и объему передаваемых данных, иногда выделяют самый маленький или самый большой поток. По причине существенной разницы в объеме потоков результаты классификации иногда проверяются отдельно по доле правильно классифицированных потоков и количеству байтов. Не менее важным является и длина потока. Элементы потока или фрагмент потоков, состоящий из

небольших пакетов, могут нести недостаточно информации для определения класса, поэтому требуют особого подхода и внимания.

Виды классификации трафика могут проводиться онлайн или офлайн, то есть в режиме реального времени. Классификация осуществляется по решению задачи. Сбор данных об использовании сети для глобального перераспределения ресурсов, сбор статистики использования сети и получение информации о активности пользователя для выставления ему счетов за интернет-услуги и расчета этих цен — всё это не требует срочного ответа или обработки. Другой задачей является обеспечение качества обслуживания пользователя, выявление атак и угроз, оперативное перераспределение ресурсов. Здесь требуется максимально быстро реагировать на ситуацию. При этом классификатор не может ждать окончания потока и оперировать полной информацией о нем, а вынужден ограничиться только частью информации, например первыми N пакетами. Основываясь на используемую модель классификатора, необходимо учитывать ограничения по использованию ресурсов системы и быстро принимать решение о потоке. Более того, это означает, что в этих случаях решение обычно принимается одновременно для нескольких/многих потоков данных сразу и поэтому накладывается ограничение на объем оперативной памяти, выделяемой для обработки потока.

Выбор набора классов для классификации трафика зависит от решаемой задачи. Примеры включают следующее:

1) Классификация по протоколам прикладного уровня (HTTP, SMTP, SSH и др.) [8, 9]. Обычно выбирают протоколы прикладного уровня, поскольку такая классификация наиболее ценна с практической точки зрения.

2) Классификация по приложениям, генерирующими интернет-трафик (Skype, Torrent, браузер и др.) [10, 11]. Это определяет активность пользователя и позволяет формировать его профиль, ограничивать активность конкретных приложений, решать маркетинговые задачи.

3) Классификация по типам действий пользователя (просмотр в Интернете, загрузка файлов, просмотр видео и т. д.) [12, 13]. В этом случае определяется не

конкретное используемое приложение, а тип активности пользователя. В некотором смысле это обобщение предыдущего типа классификации.

На сегодняшний день существует множество других [14, 15] подходов к определению набора классов для классификации интернет-трафика, но эти три являются самыми популярными и часто рассматриваются в научных статьях.

Различные классы трафика [16] и классификации трафика устройств интернета вещей [14] имеют свои особенности, выделяющие их в отдельную задачу. Они имеют большую популярность новых и/или специальных протоколов, а также другие сценарии использования или другой масштаб распространении.

2. СПЕЦИАЛЬНЫЙ РАЗДЕЛ

2.1 Методы машинного обучения, используемые для классификации траффика

Задача классификации сетевого трафика, как и другие задачи классификации, обычно рассматривается как задача обучения с учителем, поэтому при её решении используются соответствующие методы машинного обучения. Среди них можно выделить:

- наивный байесовский классификатор;
- метод опорных векторов;
- метод k-ближайших соседей;
- деревья принятия решений (с разными алгоритмами построения дерева: CART, C4.5, C5.0);
- методы бэггинга (случайный лес);
- методы бустинга (Adaboost, XGBoost);
- разные виды нейронных сетей: CNN, CNN+RNN, CNN+LSTM, SAE.

Рассмотрим примеры и результаты применения вышеупомянутых методов в исследованиях по теме классификации сетевого трафика.

2.1.1 Наивный байесовский классификатор

Наивный байесовский классификатор – простой вероятностный классификатор, основанный на применении Теоремы Байеса со строгими (наивными) предположениями о независимости. По теореме Байеса,

$$P(A|B) = (P(B|A)*P(A) / P(B)),$$

где А – класс, В – признак. Для предсказания неизвестного класса вычисляется его апостериорная вероятность. Все признаки считаются независимыми, вероятности напрямую вычисляются из обучающих данных (дискретные значения) или оцениваются через нормальное распределение.

2.1.2 Метод опорных векторов

В методе опорных векторов (SVM, support vector machine) каждый объект данных представляет из себя точку в p -мерном пространстве (где p - количество признаков), и алгоритм пытается построить гиперплоскость размерности ($p-1$), максимально эффективно разделяющую точки, относящиеся к разным классам.

2.1.3 Метод k-ближайших соседей

Метод k-ближайших соседей присваивает каждому классифицируемому примеру то значение класса, которое наиболее распространено среди его k ближайших согласно выбранной функции расстояния соседей, классы которых известны. В качестве функции расстояния часто выбирается евклидова метрика. Такой подход не даёт высокую точность сам по себе, но может применяться в сочетании с кластеризацией, как в [17] для улучшения результатов выделения из трафика отдельных неизвестных классов.

2.1.4 Дерево принятия решений

Дерево принятия решений (decision tree) представляет собой бинарное дерево, в вершинах которого записаны атрибуты (признаки), по которым различаются различные ситуации, на рёбрах – значения этих атрибутов, а в листьях – значения целевой функции. Чтобы классифицировать новый случай, надо спуститься по дереву до листа и выдать соответствующее значение.

Для построения дерева могут применяться разные алгоритмы, в работах по классификации сетевого трафика можно найти примеры использования:

- C4.5 [8, 18-20] – улучшенная версия одного из базовых алгоритмов построения деревьев ID3 (Iterative Dichotomiser 3), которая может использовать как дискретные, так и непрерывные признаки, позволяет задавать веса для признаков, и производит прореживание (pruning) для построенных деревьев с целью их оптимизации. Для выбора признака для разбиения используется информационный выигрыш, основанный на энтропии.

- C5.0 [10, 21] – оптимизация алгоритма C4.0, дающая преимущество по скорости работы и используемой памяти, строящая деревья, сравнимые по эффективности, но меньшего размера.
- CART [22] (Classification and Regression Trees) строит бинарное дерево решений с использованием критерия Джини.

Деревья принятия решений и решения, построенные на их основе, являются одним из самых популярных способов решения задачи классификации трафика, так как среди их достоинств можно перечислить:

- отсутствие необходимости в специальной подготовке данных (нормализация, добавление фиктивных переменных для приведения примеров к единому размеру и др.) как при работе с нейронными сетями (см. [23]),
- способность работы с разными типами переменных (как категориальными, так и интервальными),
- способность работать с большими объёмами данных,
- хорошие результаты классификации [11, 18],
- высокую скорость работы и низкую вычислительную сложность предсказания результата построенным деревом [18],
- простоту мультиклассовой классификации (в сравнении, например, с SVM).

2.1.5 Бэггинг

Бэггинг (bagging от bootstrap aggregating) – это способ композиции нескольких более простых классификаторов в один с целью повысить его стабильность и точность. Особое распространение получил алгоритм Random Forest [9, 11, 13, 24, 25], заканчивающийся использованием комитетного (ансамблевого) дерева решений. Суть данного метода состоит в использовании множества деревьев, каждое дерево само по себе является классификатором с низкой точностью. Однако благодаря тому, что используется большое количество деревьев точность существенно возрастает. Классификация

осуществляется следующим образом: группе деревьев присваивается классификация конкретного объекта и каждое отдельное дерево классифицирует объект по заданным классам. Далее объекту присваивается тот класс, который выберет наибольшее количество деревьев

2.1.6 Бустинг

Бустинг (boosting) – способ организации классификаторов, позволяющий упорядочить и обучить более простые классификаторы (чаще всего это решающие деревья небольшой глубины) таким образом, чтобы результирующий классификатор показывал лучшие результаты.

Примерами алгоритмов бустинга являются:

- AdaBoost (Adaptive Boosting) [13, 18, 26] – регулирует веса в процессе обучения, чтобы примерам, на которых ошибся прошлый классификатор, придавалось большее значение;
- XGBoost [24] – ошибка минимизируется алгоритмом градиентного спуска.

По качеству и скорости работы буст примерно сравним с бэггингом (зависит от конкретных данных, пример такого сравнения можно найти в [13]).

Видно, что из рассмотренных на данный момент моделей машинного обучения для классификации сетевого трафика чаще всего и с наибольшим успехом используются деревья решений (с разными алгоритмами построения), а также их комбинации методами бэггинга или бустинга.

2.1.7 Нейронные сети

Искусственные нейронные сети состоят из нескольких слоев искусственных нейронов, каждый из которых получает на вход несколько числовых значений и преобразует их в выходное значение в соответствии со своими внутренними правилами (заранее заданная функция активации и вычисляемые веса входных параметров).

Существуют модели так называемых рекомбинантных нейронных сетей с обратной связью, где сигнал от выходных нейронов или скрытого слоя нейронов частично передается обратно на входные нейроны входного слоя, но чаще всего нейронная сеть образована прямой связью. распространение сети. Значения для нейронов в каждом слое рассчитываются на основе значений во время процесса предыдущего слоя, называемого прямым распределением алгоритмов. В процессе обучения изменение настроек направлено на уменьшение штрафной функции (значения функции). Алгоритм, используемый в процессе тренировки с корректирующим весом, распространение ошибок (обратное распространение).

Сверточные нейронные сети (CHC), использующие набор небольших ядер для преобразования поступающей информации и уменьшения количества извлекаемых признаков, показали свою эффективность для работы с большим количеством параметров во внутренних слоях сети или для определения признаков, инвариантных относительно перевода [12, 23, 27-30].

Рекуррентные сети (RNN), как было сказано ранее, можно использовать для работы с признаками, имеющими зависимости во времени (например, в случае, когда текущий ответ сети должен зависеть не только от текущих признаков, но и от некоторых из них). признаки, использованные при принятии предыдущего решения). Для этого сеть особым образом сохраняет часть извлеченной информации для дальнейшего использования. Наиболее часто используемым представителем является LSTM (долговременная кратковременная память от англ. Long Short-term memory) [27, 28].

Автоэнкодеры представляют собой нейронные сети, состоящие из двух частей: первая половина сети учится кодировать входную информацию в сжатом виде, а вторая половина учится максимально точно воссоздавать эту информацию, используя сжатое представление. Поскольку полученный метод кодирования эффективно работает только для того типа данных, на котором была обучена сеть, остальные данные будут воссозданы с ошибкой, что позволяет использовать для классификации автокодировщики [12, 31, 32].

Генеративно-состязательная сеть (GAN, Generative adversarial network) представляет собой совокупность двух нейронных сетей, одна из которых учится генерировать выборки данных, похожие на реальные, а вторая учится различать эти выборки. В процессе совместного обучения повышается качество работы обеих этих сетей, что позволяет генерировать искусственные данные, практически неотличимые от реальных. Этот подход можно использовать для генерации дополнительных примеров, если другим способом невозможно получить их достаточное количество [9].

Также есть примеры совместного использования нескольких типов нейронных сетей для получения лучших результатов, в частности, CNN+RNN [14, 27, 33].

2.2 Признаки сетевого трафика, используемые для его классификации

Алгоритмы машинного обучения для своей работы нуждаются в получении признаков классифицируемых примеров, на основе которых будет приниматься решение. Для задачи классификации сетевого трафика можно выделить два основных способа их получения:

- признаки выделяются на специальном этапе подготовки данных на основе некоторой внешней информации;
- признаки выделяются из данных самой моделью в процессе глубокого обучения.

Первый подход требует от экспериментатора дополнительных усилий, но позволяет лучше контролировать процесс и использовать любую доступную информацию и ее производные (например, рассчитанные на основе доступной статистики). Второй подход требует минимальных действий при подготовке данных (нормализация данных, унификация длины примеров, переупорядочивание данных при необходимости) и может позволить найти неочевидные на первый взгляд зависимости, но набор извлекаемых им признаков не всегда лучшее или минимальное для решения задачи. Кроме того, второй подход обычно требует больше данных для обучения.

По содержанию признаков их можно несколько условно разбить на следующие классы:

- данные пакета;
- метаинформация о пакете;
- временные характеристики;
- информация о потоке.

К первой категории относится содержимое полезной нагрузки пакета или все байты пакета без разделения на заголовок и данные. Чаще всего используется при глубоком обучении модели классификации [12, 23, 27]. Паттерны, извлекаемые из содержимого пакетов, очень информативны (такой подход в некотором смысле является автоматизированным аналогом DPI). В [12] показано, что такой метод можно использовать и для классификации зашифрованного трафика, но объем пакетной информации, используемой для его классификации, достаточно велик (1480 байт).

В разных исследованиях проблемы классификации трафика используются разные категории признаков. Задача отбора и отбора признаков очень актуальна и решается практически в каждом новом исследовании. Существуют статьи [34] и инструменты [35], посвященные описанию и получению максимально возможного количества доступных признаков для пакетов в потоке данных.

2.3 Выбор и подготовка набора данных, их сравнение

Препятствием для прямого сравнения разных подходов в работе исследовательских групп по классификации трафика является не только наличие нескольких различных принципов классификации (по протоколам/приложениям/типам приложений), но и отсутствие единого общепризнанного набора данных, на котором будут тестироваться предлагаемые методы.

Получение большой и репрезентативной выборки данных для обучения и тестирования моделей — первая проблема, возникающая при решении задачи классификации данных. Каждая исследовательская группа должна найти

подходящий для себя источник данных, убедиться в правильности его разметки согласно поставленной задаче и оценить его полноту относительно возможных вариантов организации трафика в сети. Количество существующих приложений/протоколов огромно, поэтому провести полную классификацию вряд ли возможно. Обычно исследователи выбирают только определенный набор возможных вариантов (наиболее частые/характерные/представляющие наибольший интерес) и работают только с ним. Однако в этом случае возникает проблема обработки данных, не входящих в этот набор классов. В некоторых работах эти данные просто не рассматриваются, но такой подход неизбежно сталкивается с проблемами при работе с реальными данными. Еще одной проблемой является частое появление новых приложений/протоколов, которые нужно либо своевременно отражать в данных, либо обрабатывать особым образом. Также следует учитывать, что от метода сбора данных зависит не только их репрезентативность, но и вероятностное распределение классов, что также может повлиять на конечный результат.

2.3.1 Основные аспекты выбора набора данных

При выборе или получении данных особое внимание следует уделять следующим аспектам.

2.3.2 Получение правильной разметки данных

При использовании систем, использующих методы машинного обучения для классификации сетевого трафика, помимо тестовой выборки, для оценки результатов необходима достаточно большая обучающая выборка с правильно отмеченными ответами. Соответственно, возникает проблема получения этой разметки. Здесь может быть несколько источников.

а) Сторонние системы классификации. Например, вы можете использовать методы, которые разбирают и анализируют всю информацию, содержащуюся в пакете, для создания более легких и оперативных классификаторов. Под это

определение подходят системы DPI, такие как Wireshark [36], nDPI [37] и т. д. Сравнение некоторых доступных для использования систем DPI было сделано в [38, 39]. Для проверки правильности разметки инструментов предлагается проверять их работу на наборе данных, для которых специальная программа регистрирует генерирующие их приложения, что позволяет добиться заведомо высокого качества разметки. Исследование показывает, что использование инструментов DPI дает высокую, но не идеальную точность определения приложений.

Соответственно, при таком подходе следует учитывать, что точность такой разметки, а соответственно и обученных на ней алгоритмов, ограничивается точностью этих сторонних методов. Кроме того, этот метод может быть неприменим при работе с зашифрованным трафиком.

б) Получение данных в контролируемых условиях. В этом случае необходимо организовать сбор информации максимально близко к пользователю и поставить перед ним задачу генерировать только заранее заданный трафик. Наиболее распространенной проблемой в данном случае является фильтрация фонового трафика, доля которого может доходить до 70%. Для контроля поступления трафика можно либо собирать его при выполнении специальных скриптов, генерирующих только определенные его классы (ISCX, NIMS), либо параллельно со сбором трафика запускать специальные программы, позволяющие определить его источник (набор данных UPC).

в) Генерация искусственных данных на основе существующих. Еще один способ получить дополнительные данные для тех классов, в которых этих данных недостаточно для обучения модели, — дополнить обучающую выборку искусственными данными. Например, в [9, 28] для этой цели используется LSTM. Как показано, этот метод позволяет повысить качество построенной модели, особенно в тех случаях, когда некоторые из классов недостаточно представлены в обучающей выборке. Однако для его использования нужен хорошо обученный генератор данных, что является отдельной задачей, которую также необходимо решить. Кроме того, необходимо следить за тем, чтобы

полученные таким образом данные не были излишне однообразными и соответствовали реальному положению дел в сети.

2.3.3 Доступность данных

Из-за шифрования данных или из-за необходимости соблюдения конфиденциальности пользователей некоторые данные в пакетах могут быть искажены или недоступны. Для примера, при публикации снимков сетевых маршрутов может выполняться специальное кодирование IP-адресов или удаляться полезная нагрузка пакетов. Такие действия могут создавать трудности, особенно при разметке данных, но в первую очередь следует учитывать этический вопрос. Один из способов найти компромисс в этой ситуации — собрать статистику поведения реальных пользователей Интернета и написать специальные алгоритмы-боты, которые будут максимально правдоподобно имитировать такое поведение, не выдавая при этом никакой личной информации, пусть этот способ и всё равно является имитацией, из-за чего могут возникать дополнительные сложности.

Такой подход весьма распространён, так как он позволяет получить хорошие наборы данных для разнообразных задач, связанных с анализом трафика. Однако одним из серьёзных минусов является значительная упорядоченность и детерминированность действий по сравнению с реальными пользователями, действия которых куда более хаотичны, не говоря уже о потенциальных проблемах в реальной сети, что может нарушить анализ реальных статистических признаков данных.

2.3.4 Место получения данных

Данные, получаемых для анализа, зависит от места их сбора. Чтобы получить данные как можно ближе к пользователям, необходимо решить вопрос шифрования данных, а сбор информации от провайдера на глобальных маршрутизаторах позволяет получить больший объем разнообразных и

репрезентативных сведений. Это также может привести к тому, что такие особенности как распределение времени в потоке, изменение длины пакетов при туннелировании, наличие только одного направления потока данных на маршрутизаторе для сбора информации, повлияют на обучение модели машинного обучения. Лучшей стратегией будет использование данных из той же точки сети.

В [40] исследовано влияние характеристик сети на работу класса, в частности показано падение результатов работы классификаторов при тестировании трафика из сети с иной средой (отличающейся от той, в которой модель была обучена).

2.3.5 Репрезентативность набора данных

После выбора системы используемых классов и разметки данных необходимо убедиться, что полученный набор данных является репрезентативным, то есть содержит достаточное количество разнообразных примеров для каждого класса (по возможности охватывает все возможные ситуации). Необходимо нивелировать перекос в количестве данных по разным классам, чтобы предотвратить переобучение классификатора. Некоторые источники данных требуют особого внимания и предварительной обработки на этом этапе. Например, если набор данных поступает всего от нескольких пользователей, то нужно постараться сделать так, чтобы модель смогла вычленить из данных глобальные признаки, а не пытаться определить специфику работы каждого конкретного пользователя.

2.4 Используемые общедоступные наборы данных

Поскольку сам процесс получения большого количества релевантных данных, пусть и неразмеченных, может быть затруднителен, в исследованиях нередко используются широко доступные трассы сетевого трафика, которые можно найти в интернете. Обычно такие трассы содержат достаточно большой

объём данных и различаются по месту и способу их получения, а также предоставленной разметке (если она есть).

3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

3.1 Python

В настоящее время Python является языком программирования универсального уровня. Он относится к интерпретируемым языкам. Язык программирования Python является многомерным. Он поддерживает объектно-ориентированный и структурный подходы к функциональному и аспектному программированию. Python использует динамическую типизацию.

Python часто используется в сфере искусственного интеллекта, особенно с помощью таких библиотек: scikit-learn, pandas, numpy и keras.

Использование ИИ для создания приложений отличается от обычной разработки. При этом работа с ИИ предполагает наличие специального стека технологий и специальных навыков, которые необходимы для работы с ИИ. Помимо этого, разработка приложения на основе ИИ требует глубоких знаний.

Для осуществления идеи, связанной с использованием искусственного интеллекта, нужен надежный, гибкий язык программирования с богатым набором инструментов. Для разработки различных проектов ИИ Python является таким языком.

Для разработчиков Python помогает эффективно трудиться во всех областях: от разработки до поддержки. Данный язык имеет свои специфические черты и является лучшим выбором для проектов по машинному обучению, ИИ. Он простой в понимании; у него есть хорошие библиотеки для обучения и работы с ИИ. У разработчиков есть свои идеи. Что это. Это все благодаря всему этому Python является одним из самых популярных языков программирования в мире, причем не только для разработки программ на языках ML и AI.

На Python легко писать лаконичный и понятный код. Вы можете полностью сосредоточиться на задачах, которые вы решаете с помощью ML, не беспокоясь о технических нюансах языка.

Кроме того, Python прост и в освоении. Этот код понятен человеку, который пишет его для себя, и облегчает моделирование обучаемых машин.

Python более интуитивен, чем многие другие языки программирования. Также существует множество систем, библиотек и расширений для реализации задуманных функций.

Язык программирования Python широко распространен. Он может решить множество проблем, обучать машины и быстро создавать прототипы для дальнейшей отладки.

Создание алгоритма для ИИ и обучения машины — сложная задача, требующая много времени. Но чтобы программистам было легче находить оптимальные пути решения задачи, нужна хорошо структурированная и безопасная среда.

Программы и библиотеки на Python помогают существенно сократить время разработки приложений.

Python, обладающий большим количеством технологий, имеет обширную библиотеку искусственного интеллекта и обучающихся машин.

Мультиплатформенность (в нашем случае) — это свойство языка программирования или программного обеспечения и позволяет разработчикам переносить его на различные машины без изменений.

В основе популярности Python лежит то, что он не зависит от платформы, поскольку поддерживает многие из них, включая Linux и Windows.

Формулы Python могут быть использованы для создания программ на большинстве операционных систем, а это означает что программа программы Windows может легко распространяться и использоваться в этой системе без специального программного обеспечения.

На данный момент разработчики программного обеспечения обычно используют такие сервисы, как Google или Amazon для своих вычислительных нужд. Не только они имеют возможность пользоваться своими машинами с высокими графическими процессорами (GPU), но и могут использовать их машины для обучения своих моделей.

3.2 Scikit-learn

Scikit-learn — это один из наиболее широко используемых пакетов Python для науки о данных и машинного обучения. Он позволяет выполнять различные операции и предоставляет различные алгоритмы. Scikit-learn также предлагает отличную документацию по своим классам, методам и функциям, а также описание используемых алгоритмов.

Scikit-Learn поддерживает:

- предварительную обработку данных;
- уменьшение размерности;
- выбор модели;
- регрессии;
- классификации;
- кластерный анализ.

Он также предоставляет несколько наборов данных, которые вы можете использовать для тестирования ваших моделей.

Scikit-learn не реализует все, что связано с машинным обучением. Например, он не имеет комплексной поддержки для:

- нейронных сетей;
- самоорганизующихся карт (сетей Кохонена);
- обучения ассоциативным правилам;
- обучения с подкреплением (reinforcement learning).

Scikit-learn основан на NumPy и SciPy.

Scikit-learn — это пакет с открытым исходным кодом. Как и большинство материалов из экосистемы Python, он бесплатный даже для коммерческого использования. Он лицензирован под лицензией BSD.

3.3 Pandas

Для обработки и анализа данных используется программная библиотека pandas, которая написана на Python. Панды работают с данными, используя

библиотеку NumPy. Это инструмент более низкого уровня. Существует специальная структура данных и операции, которые используются при работе с числами или временными рядами. Как имя библиотеки, так и его название являются производными от эконометрического термина «панельные данные», используемого для описания многомерных структурированных наборов информации. А новая лицензия BSD распространяется и на pandas.

Основополагающее приложение для обеспечения рабочей среды на Python не только собирает данные, но и анализирует их с помощью более специфичных языков статистической обработки (таких как Research и Octave).

Набор для очистки и первичной оценки данных по общим показателям, таким как средний, квартильный или другой; это не статистический пакет в полном смысле слова. Однако типы наборов DataFrame и Printers используются в качестве входных данных во многих модулях анализа данных и машин обучения (SciPy, Scikit-Learn).

Основные возможности библиотеки:

- Объект *DataFrame* для манипулирования индексированными массивами двумерных данных.
- Инструменты для обмена данными между структурами в памяти и файлами различных форматов
 - Встроенные средства совмещения данных и способы обработки отсутствующей информации
 - Переформатирование наборов данных, в том числе создание сводных таблиц
 - Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных
- Вставка и удаление столбцов данных
- Возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение» (англ. *split-apply-combine*).
- Слияние и объединение наборов данных

- Иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности
- Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее.

Библиотека оптимизирована для высокой производительности, наиболее важные части кода написаны на Cython и Си.

3.4 NumPy

NumPy — это библиотека Python, которая используется для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй. Полное название библиотеки — Numeric Python extensions или «Python Numeric Extensions».

Эта библиотека имеет несколько важных функций, которые сделали ее популярным инструментом. Во-первых, его исходный код находится в свободном доступе на GitHub.

Во-вторых, библиотека написана на языках С и Fortran. Это компилируемые языки (языки программирования, текст которых преобразуется в машинный код — набор инструкций для конкретного типа процессора. Преобразование происходит с помощью специальной программы-компилятора, благодаря которой вычисления в компилируемых языках выполняются быстрее), в которых вычисления выполняются намного быстрее и эффективнее, чем в интерпретируемых языках (языках программирования, которые не заточены под конкретный тип процессора и могут выполняться на разных типах устройств). Сам Python относится к этим языкам.

Где используется NumPy:

- Научные вычисления. NumPy пользуются ученые для решения многомерных задач в математике и физике, биоинформатике, вычислительной химии и даже когнитивной психологии.

- Создание новых массивных библиотек. На основе NumPy появляются новые типы массивов, возможности которых выходят за рамки того, что предлагает библиотека. Например, библиотеки Dask, CuPy или XND.
- Data Science. В основе экосистемы для анализа данных лежит NumPy. Библиотека используется на всех этапах работы с данными: извлечение и преобразование, анализ, моделирование и оценка, презентация.
- Machine Learning. Библиотеки для машинного обучения scikit-learn и SciPy тоже работают благодаря вычислительным мощностям NumPy.
- Визуализация данных. По сравнению непосредственно с Python возможности NumPy позволяют исследователям визуализировать наборы данных, которые гораздо больше по размеру. Например, библиотека лежит в основе системы PyViz, которая включает в себя десятки программ для визуализации.

3.5 Keras

Keras — это открытая библиотека, написанная на Python и обеспечивающая взаимодействие с искусственными нейронными сетями. Это дополнение к фреймворку TensorFlow. Keras является масштабируемым и модульным, что делает его универсальным продуктом, который позволяет быстро и эффективно проектировать сети глубокого обучения. Он был создан в рамках исследовательских усилий проекта ONEIROС. Открытая нейроэлектронная интеллектуальная операционная система для роботов), а ее основным автором и сопровождающим является Франсуа Шолле, инженер Google.

Данная библиотека включает в себя множество реализаций основных элементов для проектирования нейронных сетей, среди них: уже готовые обученные модели, масштабируемые слои, целевые и передаточные функции, оптимизаторы и множество инструментов для обработки данных, в том числе визуальных и текстовых. На сегодняшний день Keras является частью пакета

TensorFlow и так же распространяется на свободной основе. При этом имеет множество форумов по поддержки пользователей.

3.6 TensorFlow

TensorFlow — это библиотека машинного обучения от компании Google, распространяемая по открытой лицензии Apache 2.0. TensorFlow позволяет создавать нейросетевые модели и обучать их целью решения различных задач искусственного интеллекта, таких как: автоматизация поиска и классификация изображений. Google очень широко использует данную библиотеку, как в проведении собственных исследований, так и для разработки продуктов. Основной API для работы с библиотекой реализован для Python, также есть реализации для R, C Sharp, C++, Haskell, Java, Go и Swift.

Является продолжением закрытого проекта DistBelief. TensorFlow изначально разрабатывался командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0.

TensorFlow был открыт для бесплатного доступа 9 ноября 2015 года. TensorFlow — это система машинного обучения Google Brain второго поколения. В то время как эталонная реализация работает на одном устройстве, TensorFlow может работать на многих параллельных процессорах, как ЦП, так и ГП, полагаясь на архитектуру CUDA для поддержки вычислений общего назначения на ГП. TensorFlow доступен для 64-разрядных версий Linux, macOS, Windows и для мобильных вычислительных платформ, включая Android и iOS.

Расчеты TensorFlow выражаются потоками данных через граф состояний. Название TensorFlow происходит от операций с многомерными массивами данных, которые также называют «тензорами».

3.7 Набор данных

Разработанные нейросети используют набор данных (dataset) <https://www.kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps?resource=download>.

3.8 Метод опорных векторов

Машины опорных векторов — семейство алгоритмов бинарной классификации, обучаемых с учителем, которые используют линейное деление пространства признаков с помощью гиперплоскости.

Основная идея метода заключается в отображение векторов пространства признаков, представляющих классифицируемый объекты, в пространство более высокой размерности. Это связано с тем, что в пространстве большей размерности линейная разделимость множества оказывается выше, чем в пространстве меньшей размерности. Причины этого интуитивно понятны: чем больше признаков используется для распознавания объектов, тем выше ожидаемое качество распознавания.

После перевода в пространство большей размерности, в нём строится разделяющая гиперплоскость. При этом все векторы, расположенные с одной «стороны» гиперплоскости, относятся к одному классу, а расположенные с другой — ко второму. Также, по обе стороны основной разделяющей гиперплоскости, параллельно ей и на равном расстоянии от неё строятся две вспомогательные гиперплоскости, расстояние между которыми называют **зазор**.

Задача заключается в построении разделяющей гиперплоскость таким образом, чтобы максимизировать зазор — область пространства признаков между вспомогательными гиперплоскостями, в которой не должно быть векторов. Предполагается, что разделяющая гиперплоскость, построенная по данному правилу, обеспечит наиболее уверенное разделение классов и минимизирует среднюю ошибку распознавания.

Векторы, которые попадут на границы зазора (т.е. будут лежать на вспомогательных гиперплоскостях), называют **опорными векторами** (что и дало название методу).

Теория опорных векторов была разработана В.Н. Вапником в 1990г.

Первая реализованная модель машинного обучения реализует классификацию при помощи метода опорных векторов.

Листинг 1 – Классификация при помощи SVM.

```
!pip install pyspark

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from functools import reduce
from pyspark.sql.functions import isnan, when, count, col
import matplotlib.pyplot as plt

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will
list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext

spark =
SparkSession.builder.master('local[*]').config("spark.driver.memory",
"15g").appName('TrafficAnalysisUsingPySpark').getOrCreate()
print(spark)

# Load csv to Spark DataFrame
```

```
TRAFFIC_DATA = "/input/Dataset-Unicauga-Version2-87Atts.csv"

traffic_df = spark.read.options(header='True',inferSchema='True') \
    .csv(path=TRAFFIC_DATA)

# Display the schema of DataFrame
traffic_df.printSchema()

# Show the first three rows (too many columns!)
# The columns have trailing whitespaces!
# traffic_df.show(3)

# Show dataframe columns
current_columns = traffic_df.columns

new_columns = list(map(lambda item : item.replace(
    ",","_").replace(".","_").upper().strip(),current_columns))

final_df = reduce(lambda data, idx:
    data.withColumnRenamed(current_columns[idx], new_columns[idx]),
    range(len(current_columns)), traffic_df)

final_df.printSchema()

# Count rows of df
final_df.count()

# Display newly modified columns
final_df.columns
```

```
# Create a subtable by selecting some columns
```

```
sub_df = final_df.select('FLOW_ID',  
                        'SOURCE_IP',  
                        'SOURCE_PORT',  
                        'DESTINATION_IP',  
                        'DESTINATION_PORT',  
                        'PROTOCOL',  
                        'TIMESTAMP',  
                        'FLOW_DURATION',  
                        'TOTAL_FWD_PACKETS',  
                        'TOTAL_BACKWARD_PACKETS',  
                        'TOTAL_LENGTH_OF_FWD_PACKETS',  
                        'TOTAL_LENGTH_OF_BWD_PACKETS',  
                        'FLOW_BYTES_S',  
                        'FLOW_PACKETS_S',  
                        'AVERAGE_PACKET_SIZE',  
                        'LABEL',  
                        'PROTOCOLNAME')  
sub_df.show(10)
```

```
# Check out what protocol name we have
```

```
sub_df.groupBy("PROTOCOLNAME").count().show()
```

```
# Calculate count, mean, stddev, min and max for FLOW_DURATION
```

```
sub_df.select('FLOW_DURATION').describe().show()
```

```
# Check for any NaN values
```

```
sub_df.select([count(when(isnan(c), c)).alias(c) for c in  
sub_df.columns]).show()
```

```
# Show distinct values
sub_df.select('PROTOCOLNAME').distinct().collect()

socmed = ['TWITTER','INSTAGRAM','FACEBOOK']

records = sub_df.filter(sub_df.PROTOCOLNAME.isin(socmed))

records.show(5)

records_df = records.toPandas()

records_df

records_df['TIMESTAMP'] =
pd.to_datetime(records_df['TIMESTAMP'],format= '%d/%m/%Y %H:%M:%S' )

records_df.head()

# Create a bar plot in pandas
records_df["PROTOCOLNAME"].value_counts().plot.bar()

# Use seaborn to plot the same graph above
import seaborn as sns
plt.figure(figsize=(10,4))
sns.countplot(x = 'PROTOCOLNAME', data = records_df)

import networkx as nx

G = nx.Graph()
```

```

G = nx.from_pandas_edgelist(records_df[records_df['PROTOCOLNAME']=='INSTAGRAM'], 'SOURCE_IP', 'DESTINATION_IP')

# G = nx.from_pandas_edgelist(records_df[records_df, 'SOURCE_IP',
'DESTINATION_IP']

plt.figure(figsize=(30, 30))
nx.draw_networkx(G, with_labels=True,node_size=60,font_size=12)

# nx.draw_shell(G, with_labels=True,node_size=60,font_size=12)
plt.figure(figsize=(30, 30))
nx.draw_spring(G, with_labels=True,node_size=60,font_size=12)

# Print out network edges
[e for e in G.edges]

plt.figure(figsize=(30, 30))
nx.draw_kamada_kawai(G, with_labels=True,node_size=60,font_size=12)

import plotly.express as px

fig =
px.line(records_df[records_df['PROTOCOLNAME']=='INSTAGRAM'],
x='TIMESTAMP', y="AVERAGE_PACKET_SIZE",
title="Average Packet Size vs Time")
fig.show()

from sklearn.metrics import explained_variance_score
from sklearn.model_selection import train_test_split

```

```
from sklearn.svm import SVC  
from sklearn.impute import SimpleImputer  
from sklearn.preprocessing import LabelEncoder  
from sklearn.metrics import accuracy_score  
  
# Label encode protocol name  
encoder = LabelEncoder().fit(records_df['PROTOCOLNAME'])  
records_df['PROTOCOLNAME'] =  
encoder.fit_transform(records_df['PROTOCOLNAME'])  
records_df['PROTOCOLNAME']  
  
X = records_df.drop(columns = ['FLOW_ID',  
'SOURCE_IP',  
'SOURCE_PORT',  
'DESTINATION_IP',  
'DESTINATION_PORT',  
'PROTOCOL',  
'TIMESTAMP',  
'LABEL',  
'PROTOCOLNAME'])  
Y = records_df['PROTOCOLNAME']  
  
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,  
random_state=3)  
  
imputer = SimpleImputer()  
X_train = imputer.fit_transform(X_train)  
X_test = imputer.transform(X_test)  
  
model = SVC()
```

```
model.fit(X_train,Y_train)

pred = model.predict(X_test)
explained_variance_score(pred,Y_test)

accuracy_score(pred,Y_test)
```

На рисунке 12 можно увидеть наглядную визуализацию того, какие пакеты встречались чаще всего.

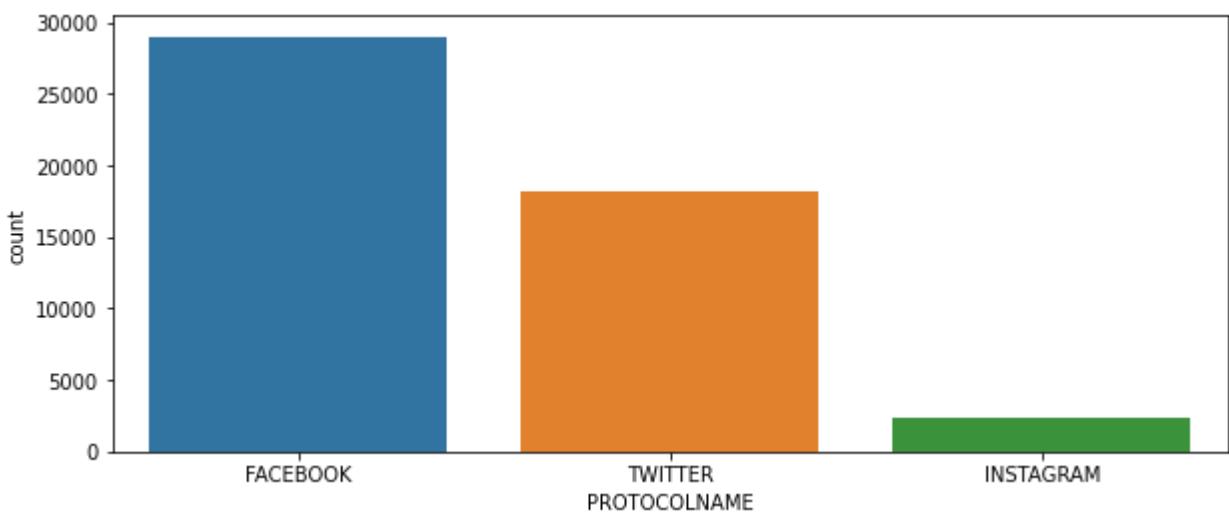


Рис. 12 – Пакеты, чаще всего встречающиеся в наборе данных

На рис.13 можно увидеть, между какими компьютерами были переданы данные, что может оказаться очень полезным, например, для контроля пользователей сети и расследования инцидентов.

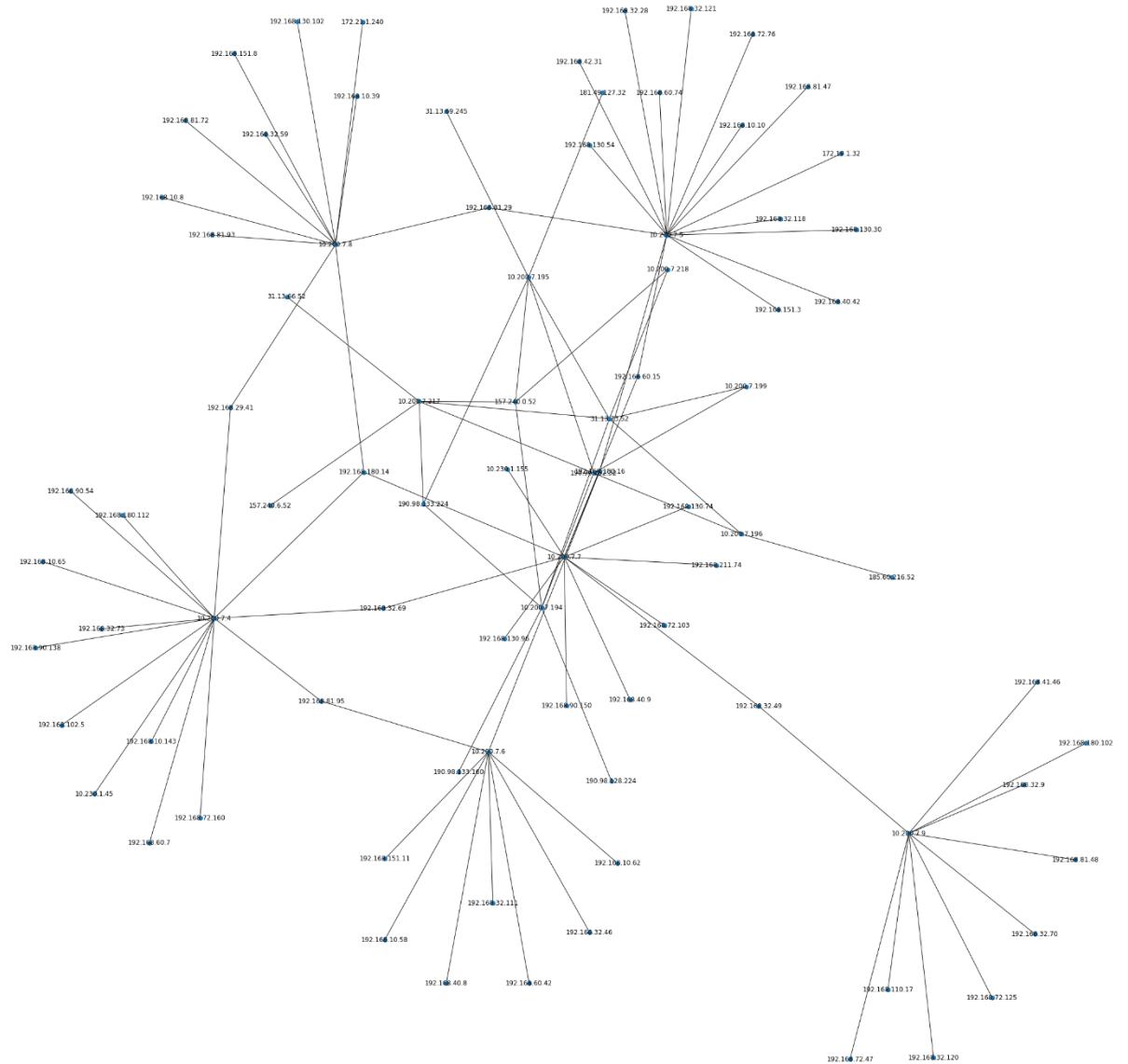


Рис. 13 – Граф коммуникаций между компьютерами в сети.

Реализованная модель имеет точность классификации 0.6513779923556628.

3.9 Рекуррентная нейронная сеть

Следующая модель использует рекуррентную нейронную сеть с 5 слоями. Данная модель показала более высокую точность, а также я добавил визуализацию наиболее часто отправляющих и принимающих сетевые пакеты IP-адресов.

```
import time
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
import os

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

t1 = time.time()

path = '../input/ip-network-traffic-flows-labeled-with-87-apps/Dataset-Unicauga-Version2-87Atts.csv'

dataset = pd.read_csv(path)

dataset

dataset.shape

# Histogram on Source.IP
Sour_feat = pd.DataFrame(dataset['Source.IP'].value_counts()[:30])
plt.figure(figsize=(20,10))
plt.plot(Sour_feat)
plt.xticks(rotation=90)
```

```

plt.xlabel('Source.IP', {'fontsize':15})
plt.ylabel('Counts', {'fontsize':15})
plt.title('Top 30 Counts in Source.IP\n', {'fontsize':20})
plt.grid()
plt.savefig('hist Source.IP.png')
Sour_feat = Sour_feat.reset_index()['index'].values

# Histogram on Destination.IP
Dest_feat = pd.DataFrame(dataset['Destination.IP'].value_counts()[:30])
plt.figure(figsize=(20,10))
plt.plot(Dest_feat)
plt.xticks(rotation=90)
plt.xlabel('Destination.IP', {'fontsize':15})
plt.ylabel('Counts', {'fontsize':15})
plt.title('Top 30 Counts in Destination.IP\n', {'fontsize':20})
plt.grid()
plt.savefig('hist Destination.IP.png')
Dest_feat = Dest_feat.reset_index()['index'].values

Dest_feat

Sour_feat

# Filtering the dataset to contain only 30 frequently reported IP address in
Source.IP and Destination.IP
f_dataset = dataset[dataset['Destination.IP'].isin(Dest_feat) &
dataset['Source.IP'].isin(Sour_feat)].reset_index()
f_dataset = f_dataset.drop('index', axis=1)

# making dummies

```

```
dum_s = pd.get_dummies(f_dataset['Source.IP'])

dum_d = pd.get_dummies(f_dataset['Destination.IP'])

label = pd.get_dummies(f_dataset['ProtocolName'])

dum_s.shape

dum_d.shape

label.shape

f_dataset.columns

# removing columns
f_dataset = f_dataset.drop(f_dataset.select_dtypes(include =
['object']).columns, axis = 1)

f_dataset =
f_dataset.drop(['Source.Port','Destination.Port','L7Protocol','Protocol'], axis = 1)

f_dataset.columns

f_dataset.shape

p_dataset = pd.concat([f_dataset, dum_s, dum_d], axis=1)

# normalizing the data
scaler = MinMaxScaler()
n_dataset = scaler.fit_transform(p_dataset)

n_dataset.shape
```

```

# splitting the dataset
X = n_dataset
y = label

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.4,
random_state = 101)

print(X_train.shape,'n',y_train.shape,'n',X_test.shape,'n',y_test.shape,'n')

# defining the model
model = tf.keras.Sequential(
    layers=[tf.keras.layers.Dense(100, input_shape=[X.shape[1]]),
            tf.keras.layers.Dense(100, activation='tanh'),
            tf.keras.layers.Dense(100, activation='tanh'),
            tf.keras.layers.Dense(100, activation='tanh'),
            tf.keras.layers.Dense(y.shape[1], activation='softmax')])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy',tf.keras.metrics.Precision(),tf.keras.metrics.Recall()])

model.summary()

# training the model
t2 = time.time()
history = model.fit(X_train, y_train, validation_split=0.2, epochs=100,
verbose=0)
t3 = time.time()

```

```
# training figures  
plt.figure(figsize=(10,10))  
  
plt.subplot(411)  
plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.legend(['loss', 'val_loss'])  
plt.xlabel('epochs')  
plt.ylabel('loss')  
plt.grid()  
  
plt.subplot(412)  
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.legend(['accuracy', 'val_accuracy'])  
plt.xlabel('epochs')  
plt.ylabel('accuracy')  
plt.grid()  
  
plt.subplot(413)  
plt.plot(history.history['precision'])  
plt.plot(history.history['val_precision'])  
plt.legend(['precision', 'val_precision'])  
plt.xlabel('epochs')  
plt.ylabel('precision')  
plt.grid()  
  
plt.subplot(414)  
plt.plot(history.history['recall'])
```

```

plt.plot(history.history['val_recall'])

plt.legend(['recall', 'val_recall'])

plt.xlabel('epochs')

plt.ylabel('recall')

plt.grid()

plt.savefig('training.png')


# prediction

loss, accuracy, precision, recall = model.evaluate(X_test, y_test, verbose=0)

print('loss:    {} \naccuracy: {} \nprecision: {} \nrecall:   {}'.format(loss,
accuracy, precision, recall))



# time taken

t4 = time.time()

print('run time      = {} sec'.format(int(t4 - t1)))

print('training time = {} sec'.format(int(t3 - t2)))

print('pre-processing time = {} sec'.format(int(t2 - t1)))



y_true = y_test.idxmax(
    axis='columns'
).reset_index().drop('index', axis=1).rename(columns={'0':'ProtocolName'})


y_pred = pd.DataFrame(pd.DataFrame(model.predict(X_test),
                                    columns = y_test.columns)
                      .idxmax(axis='columns'), columns=['ProtocolName'])


conf_mat = pd.DataFrame(confusion_matrix(y_true, y_pred,
                                         labels = label.columns),

```

```
    columns = label.columns,  
    index = label.columns)  
  
plt.figure(figsize=(24,20))  
  
sns.heatmap(conf_mat,  
            cmap = 'gray',  
            linecolor = 'white',  
            linewidths = 0.01,  
            annot=True)  
  
plt.title("confusion matrix", {'fontsize':35})  
plt.xlabel('y_pred', {'fontsize':20})  
plt.ylabel('y_true', {'fontsize':20})  
plt.savefig('confusion matrix.png')  
  
print(classification_report(y_true, y_pred))
```

На рис. 14 и рис. 15 можно увидеть, с каких IP-адресов чаще всего отправлялись пакеты, и какие IP-адреса чаще всего получали пакеты.

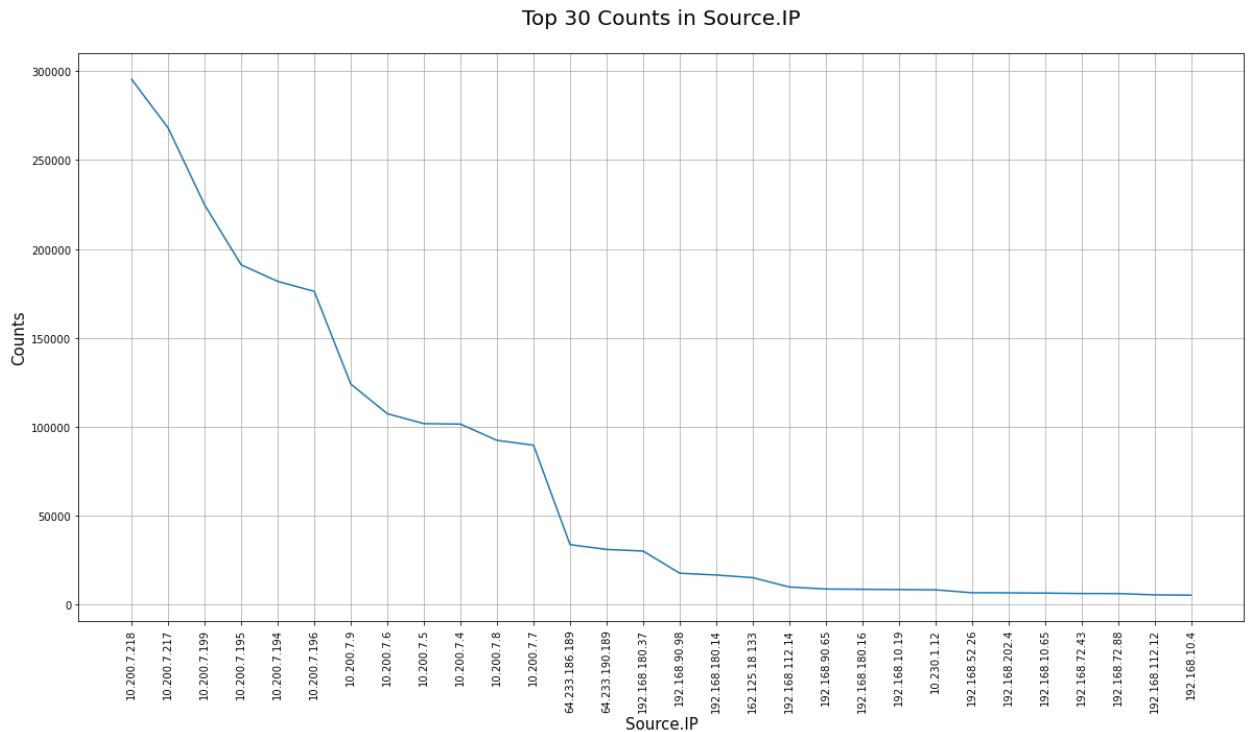


Рис. 14 – Какие IP-адреса чаще всего были указаны как Source.IP в сетевых пакетах.

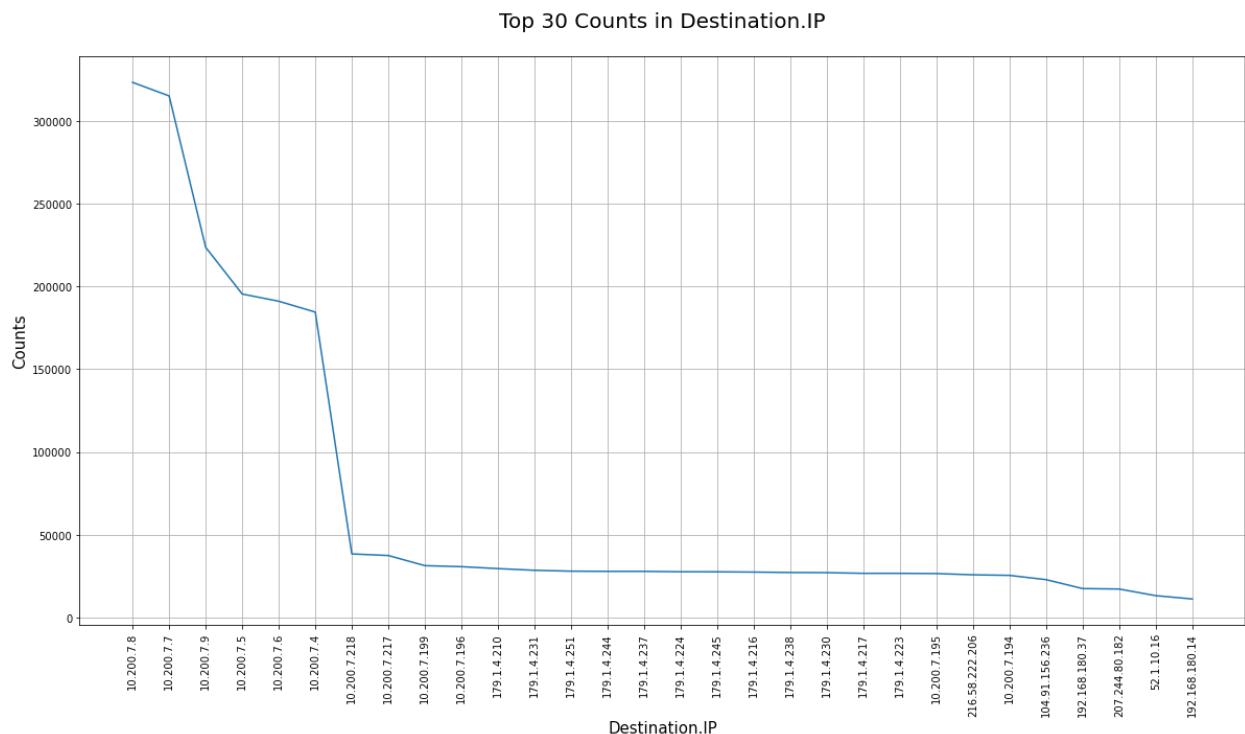


Рис. 15 – Какие IP-адреса чаще всего были указаны как Destination.IP в сетевых пакетах.

На рис. 16 мы можем увидеть, как увеличивались или уменьшались метрики нейронной сети с каждой новой эпохой.

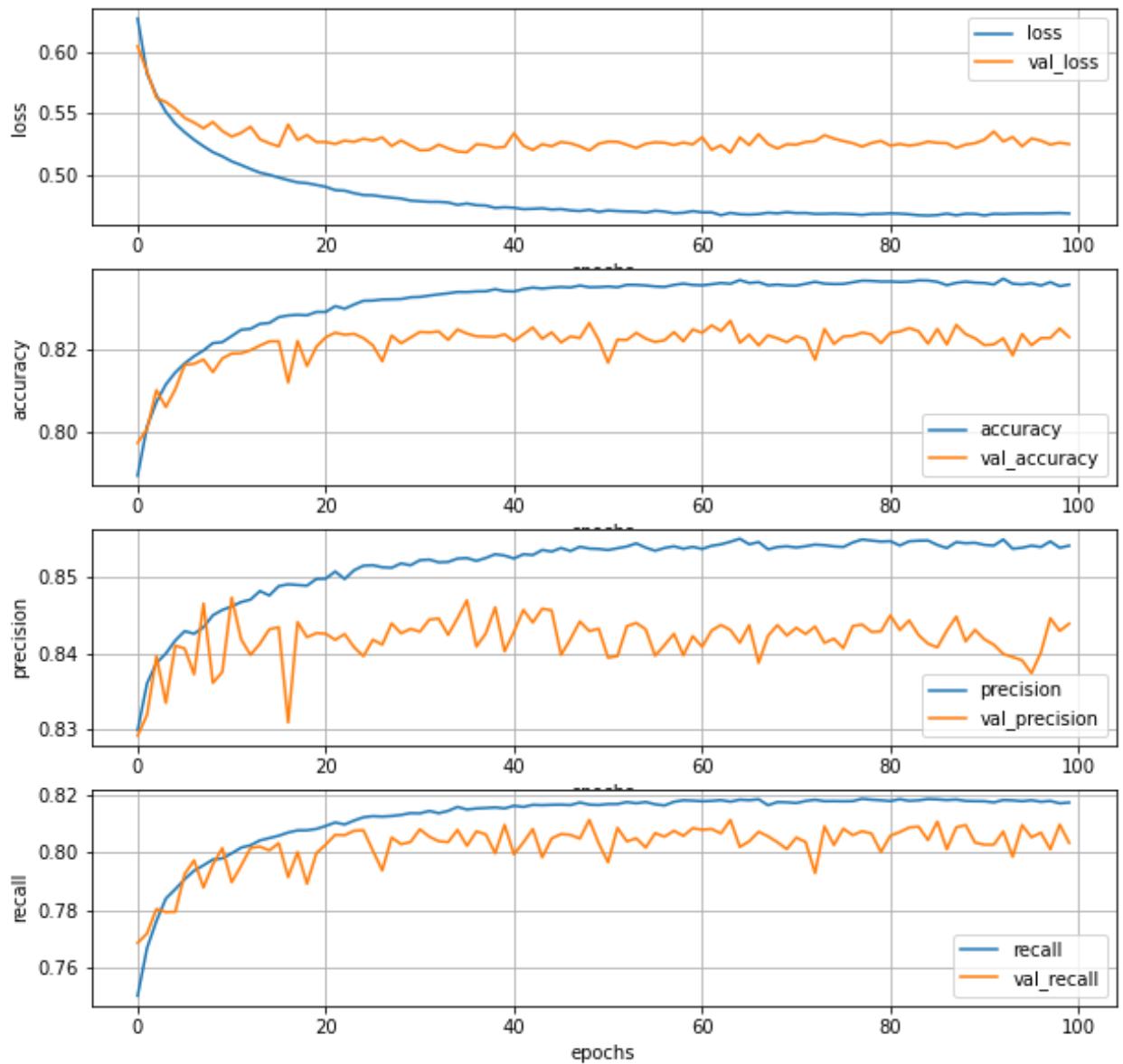


Рис. 16 – Изменение метрик нейронной сети на протяжении всех эпох.

Также в таблице 3 можно увидеть отчёт об общей точности реализованной нейронной сети.

	precision	recall	f1-score	support
accuracy			0.83	404480
macro avg	0.40	0.27	0.30	404480

weighted avg	0.81	0.83	0.81	404480
-----------------	------	------	------	--------

Вывод

Таким образом, в разделе 3 были реализованы две модели машинного обучения для классификации трафика. Для их реализации был выбран язык Python, который лучше всего подходит для задач машинного обучения, так как включает в себя множество библиотек. Также были приведены и обоснованы результаты работ моделей, благодаря которым уже можно делать выводы о том, какие ресурсы чаще всего используют пользователи сети, а также оптимизировать сеть в соответствии с собранными данными.

Список литературы

- [1]. Sniffer. https://www.opennet.ru/base/sec/arp_snif.txt.html.
- [2]. Wireshark. <https://www.wireshark.org/>.
- [3]. NAT. <https://tools.ietf.org/html/rfc1918>.
- [4]. Software NAT. <http://www.nat32.com/v2/>.
- [5]. DDoS. <http://ddos-protection.ru/chto-takoe-ddos>.
- [6]. IANA Service Name and Transport Protocol Port Number Registry.
<http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.
- [7]. Д. Виньяр. Deep Packet inspection: Technology and products. Workshop «DPI: architecture and experience архитектура и опыт», «Peter-Service », 3.12.2013.
- [8]. SIP. <https://www.ietf.org/rfc/rfc3261.txt>.
- [9]. RTP. <https://www.ietf.org/rfc/rfc3550.txt>.
- [10]. Session Border Controller. <http://www.voip-info.org/wiki/view/Session+Border+Controller>.
- [11]. Wan Optimizations.
<http://searchenterprisewan.techtarget.com/definition/WAN-optimization>.
- [12]. CIFS. <https://msdn.microsoft.com/en-us/library/ee442092.aspx>.
- [13]. MiddleBoxes. <https://tools.ietf.org/html/rfc3234>.
- [14]. DPI. <http://nag.ru/articles/article/22432/dpi.html>.
- [15]. Ekonomika programmnykh i apparatnykh DPI na primere Cisco SCE i SKAT <http://nag.ru/articles/article/28436/ekonomika-programmnyih-i-apparatnyih-dpi-na-primere-cisco-sce-i-skat.html>.
- [16]. Platformy glubokogo analiza trafika i upravleniya trafikom prilozhenii. http://www.inlinetelecom.ru/solutions/access_network/platform_depth_analysis_of_traffic_and_traffic_control_applications/.
- [17]. David L. Cannon. CISA Certified Information Systems Auditor Study Guide, 2nd Edition, 2008, ISBN: 978-0-470-23152-4
- [18]. ICAP. <https://tools.ietf.org/html/rfc3507>.

- [19]. Sergei Medvedev. Deep Packet Inspection (DPI). Seminar «Zhivye vstrechi», Krasnoyarsk, 18.01.2014.
- [20]. Recommendation ITU-T Y.2770, Requirements for deep packet inspection in next generation networks, edition 1.0, 2012.11.20
- [21]. Recommendation ITU-T Y.2771, Framework for deep packet inspection, 2014.07.01
- [22]. QUIC. <https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00>.
- [23]. CAIDA Flow Types. <https://www.caida.org/research/traffic-analysis/flowtypes/>.
- [24]. P. Filimonov, M. Ivanov. Sovremennye podkhody k klassifikatsii trafika fizicheskikh kanalov seti Internet, Trudy 18-oi Mezhdunarodnoi konferentsii «Raspredelennye komp'yuternye i kommunikatsionnye seti: upravlenie, vychislenie, svyaz» (DCCN-2015), 19 - 22 oktyabrya 2015 g, str. 466-474.
- [25]. F. Risso, M. Baldi, O. Morandi, A. Baldini, P. Monclus, “Lightweight, payload-based traffic classification: An experimental evaluation” in Proc. IEEE ICC, 2008, pp. 5869– 5875.
- [26]. Wireshark, VoIP calls. https://wiki.wireshark.org/VoIP_calls.
- [27]. MIME. <https://tools.ietf.org/html/rfc2045>.
- [28]. ASN.1. <https://tools.ietf.org/html/rfc6025>.
- [29]. P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker. P4: Programming protocol-independent packet processors. SIGCOMM Computer Communications Review, 2013.
- [30]. A. Bremler-Barr, Y. Harchol, D. Hay, Y. Koral. Deep packet inspection as a service. In CoNEXT, pages 271–282, 2014.
- [31]. SDN. <https://tools.ietf.org/html/rfc7426>.
- [32]. Qosmos ixEngine. <http://www.qosmos.com/products/deep-packet-inspection-engine/>.
- [33]. Ipoque PACE. <https://www.ipoque.com/products/pace>.

- [34]. Windriver Content Inspection Engine.
http://www.windriver.com/products/product-overviews/PO_Wind-River-Content-Inspection-Engine.pdf.
- [35]. Procera PacketLogic Content Intelligence.
<https://www.proceranetworks.com/content-intelligence.html>.
- [36]. Cisco NBAR. <http://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html>.
- [37]. Junos OS Next Generation Application Identification.
https://www.juniper.net/documentation/en_US/junos15.1x49/topics/concept/services-application-identification-techniques-understanding.html.
- [38]. Cascarano N, Ciminiera L, Risso F. Optimizing deep packet inspection for high-speed traffic analysis. Network System Manager. 2011; 19(1):7–31.
- [39]. Duffield N., Lund C. “Predicting Resource Usage and Estimation Accuracy in an IP Flow Measurement Collection Infrastructure”. ACM Internet Measurement Conference, 2003.
- [40]. Callado A., Kamienski C., Szabo G., Gero B., Kelner J., Fernandes S., Sadok D. A Survey on Internet Traffic Identification; Communications Surveys & Tutorials, IEEE Volume 11, Issue 3, 3rd Quarter 2009 Page(s):37 – 52.
- [41]. Duffield, N.; Lund, C.; Thorup, M., “Learn more, sample less: control of volume and variance in network measurement”, IEEE Transactions in Information Theory, vol. 51, no. 5, pp. 1756-1775, 2005.
- [42]. PSAMP. <http://www.rfc-editor.org/rfc/rfc5476.txt>.
- [43]. Se-Hee Han, Myung-Sup Kim, Hong-Taek Ju and James W. Hong, “The Architecture of NGMON: a Passive Network Monitoring System for High-Speed IP Networks”, Accepted to appear in the Proc. of the 13th IFIP /IEEE International Workshop on Distributed Systems:Operations & Management (DSOM 2002), Montreal, Canada, October 21-23, 2002.
- [44]. InveaTech FlowMon. <https://www.invea.com/en/products/flowmon>.
- [45]. IPFIX. <https://tools.ietf.org/html/rfc5101>.

- [46]. M.-S. Kim, Y. J. Won, and J. W. Hong. Characteristic analysis of internet traffic from the perspective of flows. *Comp. Comm.*, 29(10):1639–1652, 2006.
- [47]. R. Sommer and A. Feldmann. NetFlow: Information loss or win? In ACM SIGCOMM Internet Meas. Workshop, 2002.
- [48]. MIB. <https://tools.ietf.org/html/rfc3418>.
- [49]. SNMP. <https://www.ietf.org/rfc/rfc1157.txt>.
- [50]. Colin J. Bennett, Andrew Clement, Kate Milberry. Introduction to Cyber-Surveillance. *Cyber-Surveillance in Everyday Life*. Vol. 9, No 4 (2012)
- [51]. T. Farah, and L. Trajkovic, "Anonym: A tool for anonymization of the Internet traffic." In IEEE 2013 International Conference on Cybernetics (CYBCONF), 2013, pp. 261-266.
- [52]. F. Risso, and L. Degioanni, "An Architecture for High Performance Network Analysis", Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001), Hammamet, Tunisia, July 2001.
- [53]. PF_RING ZC. http://www.ntop.org/products/packet-capture/pf_ring/.
- [54]. Andrew M White, Srinivas Krishnan, Michael Bailey, Fabian Monroe, and Phillip Porras. Clear and Present Data: Opaque Traffic and its Security Implications for the Future. NDSS, 2013.
- [55]. Network Intrusion Detection Signatures.
<http://www.symantec.com/connect/articles/network-intrusion-detection-signatures-part-five>.
- [56]. Yuji Waizumi, Yuya Tsukabe, Hiroshi Tsunoda, and Yoshiaki Nemoto. Network Application Identification Based on Communication Characteristics of Application Messages. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering* Vol:3, No:12, 2009
- [57]. T. Karagiannis, K. Papagiannaki and M. Faloutsos. BLINC: Multilevel traffic classification in the dark. In Proc. of ACM SIGCOMM, August 2005
- [58]. Nguyen, T. T. T. and Armitage, G.2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials* 10, 4, (2008), 56-76.

- [59]. T. AbuHmed, A. Mohaisen, and D. Nyang, “A survey on deep packet inspection for intrusion detection systems,” CoRR, vol. abs/0803.0037, 2008. [Online]. Available: <http://arxiv.org/abs/0803.0037>.
- [60]. Koloud Al-Khamaiseh, Shadi ALShagarin. A Survey of String Matching Algorithms. Int. Journal of Engineering Research and Applications. ISSN : 2248-9622, Vol. 4, Issue 7(Version 2), July 2014, pp.144-156.
- [61]. D. E. Taylor, “Survey and taxonomy of packet classification techniques,” ACM Comput. Surv., vol. 37, no. 3, pp. 238–275, 2005.
- [62]. L7-filter. <http://l7-filter.sourceforge.net/>.
- [63]. J. E. Hopcroft and J. D. Ullman, “Introduction to Automata Theory, Languages, and Computation,” Addison Wesley, 1979.
- [64]. S. Kumar and P. Crowley. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. In Proc. of SIGCOMM, 2006.
- [65]. D. Ficara, S. Giordano, G. Prociassi. An Improved DFA for Fast Regular Expression Matching. In Proc. of SIGCOMM, 2008.
- [66]. F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. In Proc. of ANCS, 2006.
- [67]. S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese. Curing Regular Expressions Matching Algorithms From Insomnia. In Proc. of ANCS, 2007.
- [68]. R. Smith, C. Estan, S. Jha, and S. Kong. Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata. In Proc. of ACM SIGCOMM, 2008.
- [69]. C. Liu, J. Wu. Fast Deep Packet Inspection with a Dual Finite Automata. IEEE Transactions on Computers, Vol. 62.
- [70]. Zip Bomb. <http://xeushack.com/zip-bomb/>