Министерство образования и науки Российской Федерации
Московский физико-технический институт
(национальный исследовательский университет)

Физтех-школа радиотехники и компьютерных технологий
Кафедра системного программирования (ИСП РАН)

Выпускная квалификационная работа бакалавра

Автоматическая генерация сигнатур сетевых протоколов

Автор:
Студент Б01-009а группы
Дурнов Алексей Николаевич

Научный руководитель:
канд. физ.-мат. наук
Гетьман Александр Игоревич

Москва 2024

Аннотация

Автоматическая генерация сигнатур сетевых протоколов

Дурнов Алексей Николаевич

Данная работа посвящена исследованию и разработке методов автоматической генерации сигнатур сетевых протоколов. В ходе работы был проведен анализ существующих подходов к генерации сигнатур. На их основе был выбран наиболее универсальный формат сигнатур для сетевого трафика. Были изучены ограничения представленных методов генерации сигнатур и выбран для последующих исследований метод LASER (Application Signature ExtRaction), основанный на алгоритме LCS (Longest Common Subsequence). По их результатам было установлено, что сборка TCP-сессии, частичная или полная, негативно сказывается на результатах классификации для данного метода: полученные сигнатуры являются слишком специфичными и не обладают достаточной предсказательной способностью. Наилучшие результаты классификации показал стандартный метод LASER без сборки TCP-сессии: средневзвешенное значение F1-меры составило 0.95. Также было показано, что недостаточно простой постобработки для выделения минимального покрывающего набора сигнатур. Разработанный генератор сигнатур и классификатор были встроены как модули в систему анализа сетевого трафика, разрабатываемую в ИСП РАН.

Содержание

1 Введение

Интернет-провайдеры и сетевые администраторы хотят идентифицировать тип сетевого трафика, который проходит через их сеть, для того, чтобы предоставлять своим клиентам лучший сервис, предлагая им высокое качество обслуживание (QoS), а также планировать свою инфраструктуру и управлять ею. Сетевой трафик можно классифицировать как в соответствии с использующимся протоколом, так и в соответствии с использующимся приложением. Вторая классификация более трудоёмкая, но позволяет решать более широкий спектр задач: формирование трафика в сети, улучшение качества обслуживание, а также предоставление более детального биллинга.

В области классификации сетевого трафика было проведено множество исследований, что привело к разработке многих методов. Самый наивный метод классификации сетевого трафика это идентификация по номеру порта. На заре развития интернета многие клиент-серверные приложения имели четко определенные номера портов сервера, которые определяются Internet Assigned Numbers Authority (IANA) [1]. К примеру, DNS использует номер порта сервера 53, SMTP использует номер порта сервера 25, HTTP - номер 80 и т.д. Таким образом, используя известные номера портов, можно классифицировать трафик. Однако современные приложения используют динамическое распределение портов и туннелирование трафика, например, по протоколу HTTP, поэтому этот метод на данный момент не обладает необходимой точностью [2]. Чтобы преодолеть эти ограничения идентификации были введены более совершенные методы [3].

В рамках подхода глубокого анализа пакетов (DPI) анализатор просматривает содержимое каждого пакета полностью. Эта технология получило очень широкое распространение, так как эти методы являются точными для идентификации трафика, несмотря на свою трудо- и ресурсоёмкость. Ведущий подход, использующийся в DPI для классификации трафика, основан на сопоставлении сигнатур полезной нагрузки. Под сигнатурой понимается часть данных полезной нагрузки, которая является статичной и различимой для приложений/протоколов и может быть описана, как последовательность строк или шестнадцатеричных чисел.

Подход, альтернативный DPI, основан на статистическом анализе. Статистический анализ реализуются в основном с помощью алгоритмов машинного обучения [4]. Для этого подхода используются уже косвенные признаки пакетов и потоков, такие как задержки между пакетами, размеры пакетов и другие. Классификация сетевого трафика с использованием данного подхода подвержена следующим проблемам. Во-первых, приложения могут иметь схожую статистику, поэтому их потоки трудно отличить с помощью таких свойств. Во-вторых, эти признаки являются динамическими величинами и зависят от самой сети. Несмотря на эти проблемы, этот подход может применяться для зашифрованного сетевого трафика. Так как в первом приближении можно считать, что полезная нагрузка зашифрованного трафика представляет собой белый шум, поэтому сигнатурный подход не применим.

Чтобы точно классифицировать поток, система DPI должна искать сигнатуры в полезной нагрузке пакетов. Такой подход не нов, системы обнаружения вторжения (IDS) с помощью сигнатур находят интернет-червей и другой трафик, угрожающий безопасности [5–7]. По сравнению с проблемой генерации сигнатур червей проблема генерации сигнатур безопасного трафика имеет несколько отличий. Во-первых, общие подстроки в прикладных протоколах обычно очень короткие, а некоторые составляют всего несколько байт, в то время как общие подстроки у червей обычно достигают несколько десятков или сотен байт. Некоторые методы [5,6] оказываются неэффективными, когда общие подстроки короткие. Во-вторых, разные потоки в одном и том же приложении могут иметь разные общие подстроки. Например, в P2P приложении некоторые потоки используются для обмена одноранговой информацией, а другие потоки используются для обмена данными. Они могут использовать разные протоколы и иметь разные общие подстроки.

Поначалу сигнатуры извлекались вручную, но и сейчас часто пишутся в том числе на заказ [8]. Постоянное появление новых приложений и их частые обновления подчёркивают необходимость автоматической генерации сигнатур, так как ручная операция извлечения сигнатур занимает много времени, а также может быть разница в качестве сигнатур в зависимости от оператора извлечения.

Методы автоматической генерации сигнатур должны быть основаны не на семантическом анализе протоколов, так как, хотя они и повышают точность сигнатур, но не могут быть применены к анализу высокоскоростного трафика в режиме реального времени [9].

Поэтому данная работа посвящена исследованию различных методов автоматической генерации сигнатур полезной нагрузки для классификации сетевых протоколов и сложностям, связанным с классификацией приложений.

2 Постановка задачи

Целью данной работы является разработка и реализация метода автоматической генерации сигнатур полезной нагрузки сетевого трафика для классификации этого трафика в соответствии с использующимся протоколом.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести исследование литературы по соответствующей теме.
2. Собрать набор сетевых трасс для последующего тестирования.
3. Выбрать формат сигнатуры сетевых протоколов.
4. Разработать алгоритм генерации сигнатур.
   (a) Рассмотреть существующие методы для автоматической генерации сигнатуры.
   (b) Выделить ограничения рассмотренных методов.
   (c) Выбрать оптимальный метод для дальнейшего исследования.
5. Разработать классификатор сетевого трафика для проверки сгенерированных сигнатур.
   (a) Выбрать метрики, по котором можно оценить качество сигнатур.
   (b) Реализовать алгоритм сопоставления сигнатур для выбранного формата сигнатуры.
6. Встроить генератор сигнатур и классификатор как модули в систему анализа сетевого трафика, разрабатываемую в ИСП РАН.

3 Обзор существующих решений

3.1 Формат сигнатур

Прежде чем говорить непосредственно о методах автоматической генерации сигнатур, стоит сначала понять, какие бывают сигнатуры и в каком формате они могут быть представлены.

Существует несколько представлений сигнатур. Некоторые из этих видов использовались для представлений сигнатур червей. В работе [7] приводится следующая классификация:

- **Конъюнктивная сигнатура**: состоит из набора подстрок (или токенов). Полезная нагрузка соответствует ей, если все токены в наборе были найдены в любом порядке.
- **Сигнатура, представленная последовательностью токенов**: состоит из упорядоченного набора токенов. Полезная нагрузка соответствует сигнатуре, если она содержит всю последовательность токенов в том же порядке.
- **Байесовская сигнатура**: состоит из набора токенов, каждый из которых связан с оценкой, и общего порогового значения. Байесовские сигнатуры обеспечивают вероятностное сопоставление: вычисляется вероятность сопоставления, используя оценки присутствующих токенов в полезной нагрузки. Если результирующая вероятность превышает пороговое значение, то считается, что полезная нагрузка совпала с сигнатурой.

При использовании любого из этих определений в качестве определения сигнатуры приложения/протокола возникают некоторые специфические проблемы:

1. **Разнообразие потоков**: в рамках одного приложения/протокола могут использоваться множество потоков, имеющие разные общие подстроки. Например, протокол FTP использует разные потоки: управления и данных. Кроме того приложения будут обновлять свои протоколы: потоки различных версий могут существенно отличаться и существовать одновременно. Ни сигнатура, представленная токен-последовательностью, ни конъюнктивная сигнатура не могут выражать несколько наборов общих подстрок.
2. **Взаимоисключающее свойство некоторых подстрок в прикладных протоколах**. Например, протокол HTTP имеет две последовательности подстрок {'GET' 'HTTP/1.1'} и {'POST' 'HTTP/1.1'}. Подстроки 'GET' и 'POST' не будут быть в одном и том же потоке в протоколе HTTP. Две общие подстроки являются взаимоисключающими, но байесовские сигнатуры не могут выражать взаимоисключающее свойство.

Определение сигнатуры, основанное на регулярных выражениях, становится очень распространённым в классификации приложений [10–12]. Однако процесс сопоставления регулярных выражений требует огромной вычислительной мощности, которая слабо масштабируется для классификации сетевого трафика в режиме реального времени. Способ построения регулярного выражения оказывает непосредственное влияние на классификацию потоков и на общую производительность сопоставления. Несмотря на это, некоторые системы DPI используют регулярные выражения для представления сигнатур приложений. Система обнаружения/предотвращения вторжений Snort (IDS/IPS) [13] имеет множество сигнатур приложений и предлагает пользователю возможность вставлять новые регулярные выражения по требованию.

Почти неизвестно алгоритмов извлечения сигнатур в виде регулярных выражений, а те что есть - основаны на строках и небольшом подмножестве операций регулярных выражений. Поэтому дальше будем рассматривать сигнатуры только в виде строк.

Среди возможных описаний строковых сигнатур, набор последовательностей подстрок является наилучшим определением сигнатуры. Полезная нагрузка соответствует сигнатуре, если она содержит какую-то последовательность подстрок из этого набора. Такое определение решает описанные выше проблемы.

Введем следующее определение: уровень поддержки последовательности подстрок (support) равен отношению количества хостов, использующих соответствующее приложение или протокол и трафик которого содержит эту последовательность подстрок к общему количеству хостов, использующих соответствующее приложение или протокол.

При заданном формате, в общем случае, не любая последовательность подстрок из набора охватывает все хосты. В части из дальнейших рассматриваемых методов считается, что сигнатура состоит из одной последовательности подстрок, уровень поддержки которой равен 1, то есть она присутствует в трафике каждого хоста, использующего рассматриваемый протокол или приложение. Этот показатель можно использовать как параметр в некоторых методах.

3.2 Структура сигнатур

Большинство форматов сигнатур в предыдущих работах [9, 14, 15] представляют собой простые подстроки, которые часто появляются в полезной нагрузке. Следовательно, всё ещё существует вероятность того, что извлеченные сигнатуры полезной нагрузки могут быть не специфичными для конкретного протокола, некоторые могут принадлежать и другому протоколу. Это называется избыточностью сигнатур.

Для улучшения качества сигнатур предлагается выделить три типа сигнатур [16, 17]:

1. сигнатура содержимого (полезной нагрузки),

2. сигнатура пакета,

3. сигнатура потока.

Сигнатура содержимого определяется как различимая и уникальная подстрока полезной нагрузки, состоящая из непрерывных символов или шестнадцатеричных значений. Очень трудно обеспечить уникальность с помощью одной подстроки. Например, такие строки «GET» или «HTTP», которые часто встречаются в HTTP, не могут служить конечными сигнатурами, так как они не различают приложения.

Сигнатура пакета состоит из серии сигнатур содержимого, которые появляются в одном пакете. Так как классификация может выполняться без накопления пакетов, то есть без сбора потока, то анализируется всегда хотя бы один пакет. Это значит, что для классификации всегда можно использовать сигнатуры пакетов, и не имеет смысла использовать отдельно сигнатуры содержимого.

Сигнатура потока состоит из серии сигнатур пакетов, которые появляются в одном потоке, где под потоком понимается набор пакетов, имеющий общие IP-адрес источника, IP-адрес назначения, порт источника, порт назначения и используемый протокол транспортного уровня. Сигнатура потока гораздо более специфична для конкретного приложения, чем сигнатура пакета, и значительно повышает точность.

Поэтапный процесс извлечения сигнатур представлен на рис. 1. Данная структура сигнатур обладает свойством вложенности.

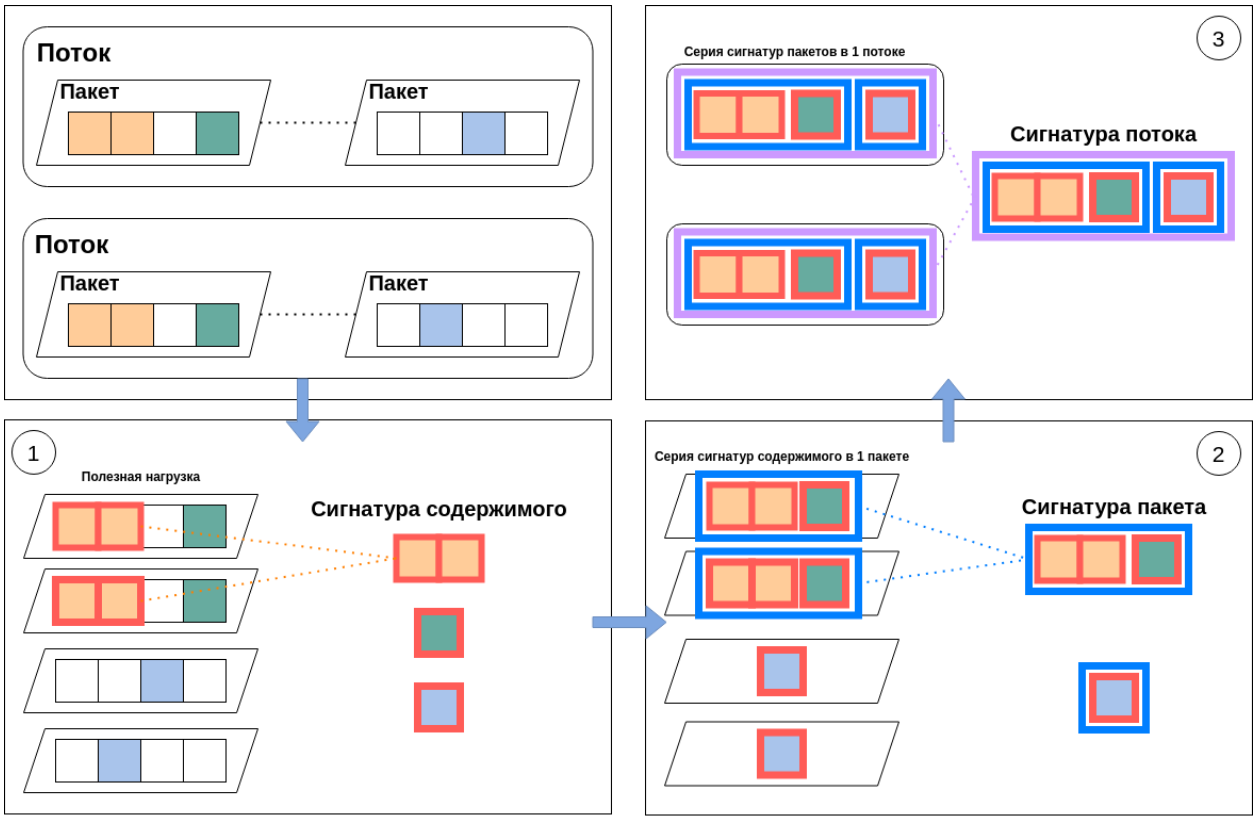

Рисунок 1 – Процесс извлечения предлагаемой структуры сигнатур полезной нагрузки.

3.3 Метрики оценки качества сигнатур

Для оценки качества получаемых сигнатур будем рассматривать результаты мультиклассового классификатора. Выходом такого классификатора будет служить набор классов, сигнатурам которых соответствует полезная нагрузка потока. В реальности классификатор следует останавливать при первом соответствии сигнатуре, но в рамках исследований будет проверяться на соответствие каждой сигнатуре каждого класса.

Рассмотрим матрицу ошибок для мультиклассовой классификации. Так как все последующие показатели удобно рассчитывать для случая бинарной классификации, то на рис. 2 показан переход от мультиклассовой к бинарной классификации.

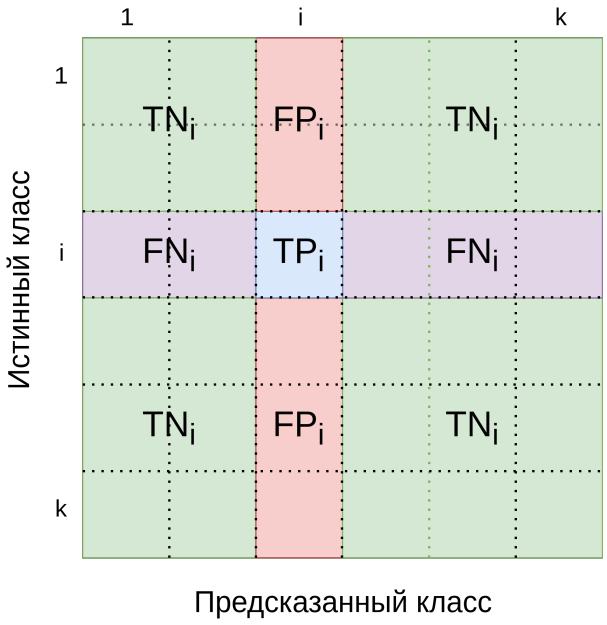

Рисунок 2 – Матрица ошибок в мультиклассовом случае. Переход от мультиклассовой классификации к бинарной.

Для неё вводят следующие четыре категории, к котором можно отнести результат работы классификатора на полученной сигнатуре:

- истинно положительный (TP_i): указывает, что поток правильно классифицирован, как относящийся к классу i.
- истинно отрицательный (TN_i): указывает, что поток правильно классифицирован, как не относящийся к классу i.
- ложно положительный (FP_i): указывает, что поток неправильно классифицирован, как относящийся к классу i.
- ложный отрицательный (FN_i): указывает, что поток неправильно классифицирован, как не относящийся к классу i.

Наиболее часто используемые показатели для классификации трафика определяются следующим образом:

- $Accuracy_i$ (достоверность) - доля правильных классификаций для класса i.

$$accuracy_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

- $Recall_i$ (полнота) - отношение верно классифицированных потоков класса i к общему числу потоков класса i, то есть описывает способность сигнатуры обнаружить данный целевой протокол или приложение.

$$recall_i = \frac{TP_i}{TP_i + FN_i}$$

- Precision (точность) - доля верно классифицированных потоков среди всех потоков, которые классификатор отнёс к классу i.

$$precision_i = \frac{TP_i}{TP_i + FP_i}$$

- $F_1 - score_i$ ($F_1 -$ мера$_i$) - гармоническое среднее между точностью и полнотой.

$$F_1\text{-}score_i = \frac{2 \times recall_i \times precision_i}{recall_i + precision_i}$$

Метрика accuracy может терять свой смысл в задачах с сильно неравными классами. Напротив же recall и precision не зависят от соотношения классов и поэтому применимы в случае несбалансированных классов, что является верным для сетевого трафика. Часто на практике возникает задача найти оптимальный баланс между precision и recall. Для этих целей подходит F-мера, которая достигает максимума при recall и precision равным 1, и стремится к минимуму, если хотя бы один из параметров стремится к нулю.

Для сигнатур полезно ещё ввести такое понятие как:

- Redundancy (избыточность) определяется как:

$$Redundancy = \frac{\text{Число потоков, идентифицированные двумя и более сигнатурами}}{\text{Число потоков, идентифицированные набором сигнатур}}$$

Redundancy имеет значение от 0 до 1, где 0 - наилучшее значение, которое указывает на то, что все сигнатуры набора классифицируют исключительно только свою часть трафика, то есть являются уникальными и незаменяемыми. Если redundancy близка к 1, то в наборе присутствует ненужные сигнатуры, которые

идентифицируют перекрывающийся трафик. По мере увеличения количества сигнатур увеличиваются и накладные расходы системы, поэтому данное значение должно оставаться низким.

3.4 Обзор существующих методов автоматической генерации сигнатур

Рассмотрим несколько существующих методов автоматической генерации сигнатур и опишем основной принцип их работы.

3.4.1 LASER

В статье [9] описан алгоритм LASER (Application Signature ExtRaction), который основан на задаче поиска наиболее длинной общей подпоследовательности LCS (Longest common subsequence). Данный алгоритм автоматически определяет достоверный шаблон в полезной нагрузке пакета без предварительного знания форматов протоколов, то есть генерирует сигнатуру пакета. За шаг извлечения сигнатуры был принят алгоритм LCS, который ранее в основном использовался для сопоставления последовательностей ДНК в приложениях биоинформатики [18]. Он был модифицирован под текущие задачи.

Вводятся ограничения на модификацию LCS:

- **Количество пакетов в потоке**. Нет необходимости проводить проверку над всеми пакетами в наборе, потому что сигнатура существует в нескольких начальных пакетах потока.
- **Минимальная длина подстроки**. Стоимость сопоставления сигнатур пропорциональна их длинам. Сгенерированная сигнатура состоит из последовательности подстрок. Чтобы избежать каких-либо тривиальных сигнатур, минимальная граница длины подстроки должна рассматриваться как ограничение для модифицированного алгоритма LCS. Имея это ограничение длины, предотвращается включение однопозиционных и многопозиционных символов в последовательность общих строк, например символа '/' в HTTP пакетах.
- **Сравнение размера пакетов**. Это увеличивает вероятность нахождения надёжной сигнатуры, если пакеты сгруппированы по назначению (например, управляющий трафик или трафик загрузки) и характеристикам трафика. Одной из таких характеристик является размер пакета. Объём данных при установлении соединения небольшой. Конкретная сигнатура существует только в первых пакетах. Поэтому сравнения небольших пакетов установки соединения и пакетов загрузки нежелательно для генерации надёжной сигнатуры.

Поясним идею алгоритма. На вход алгоритма подаётся набор потоков, в которых содержится хотя бы по packet_constraint (параметр алгоритма) пакетов. Выбираются

два потока, для которых происходит полный попарный перебор пакетов, и если разница размеров пакетов в паре меньше threshold (параметр алгоритма), то вызывается функция LASER для этой пары пакетов (поиск наиболее длинной общей подпоследовательности). Выбирается самая длинная общая подпоследовательность среди всех результатов вызовов LASER. Затем происходит процесс уточнения: проходим по всем оставшимся потокам, вызываем функцию LASER для всех пакетов из выбранного потока и текущей сигнатуры, среди результатов выбирается самая длинная общая подпоследовательность. Когда потоки закончились, возвращаем полученный результат.

Функция LASER представляет собой алгоритм Needleman-Wunsch, в котором используется матрица направлений, но в результат попадают не все подстроки, а только те, что длиннее min_substring_length_constraint (параметр алгоритма).

Таким образом, у алгоритма LASER 3 параметра настройки:

- packet_constraint (количество пакетов в потоке)
- threshold (порог разницы размеров для сравнения пакетов)
- min_substring_length_constraint (минимальная длина подстроки)

3.4.2 AutoSig

Следующий рассматриваемый метод называется AutoSig [14, 15]. Каждый поток из выборки обрабатывается как последовательность байт из первых N байт полезной нагрузки. Данный алгоритм генерирует сигнатуру-потока, являющуюся набором общих последовательностей подстрок. Приведем описание работы алгоритма.

Сначала алгоритм делит содержимое потоков на небольшие блоки содержимого фиксированного размера, которые называются шинглами. Чтобы исключить шумы, шинглы делят на разные группы в зависимости от их смещения. Шинглы делят с помощью перекрывающихся окон фиксированной длины $2W$, как показано на рис. 3. Размер области перекрытия между двумя последовательными окнами равен W. Шингл будет считаться общим, если он встретился в одном и том же окне более, чем в $R \cdot N$ потоках, где N - количество потоков, R - пороговое значение выбора общего шингла.

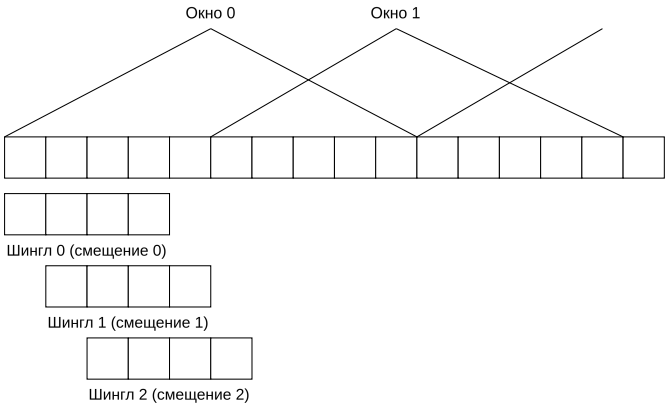

Рисунок 3 – Схема выделение шинглов и окон в алгоритме AutoSig.

С помощью адаптивного алгоритма слияния перекрывающиеся общие шинглы объединяются в подстроки. Для оценки их слияния используется показатель

$$sim(x, y) = \frac{|flows(xy)|}{|flows(x) \cup flows(y)|}$$

где $flows(x)$ - потоки, содержащие шингл x, а $flows(xy)$ - потоки, содержащие шинглы x и y вместе (смежные или перекрывающиеся, иначе $sim(x, y) = 0$). Объединение шинглов x и y происходит, если $sim(x, y) > S$, где S - предопределенное пороговое значение. Наконец, строится дерево подстрок для организации общих подстрок в конечные сигнатуры. Пример такого дерева представлен на рис. 4.

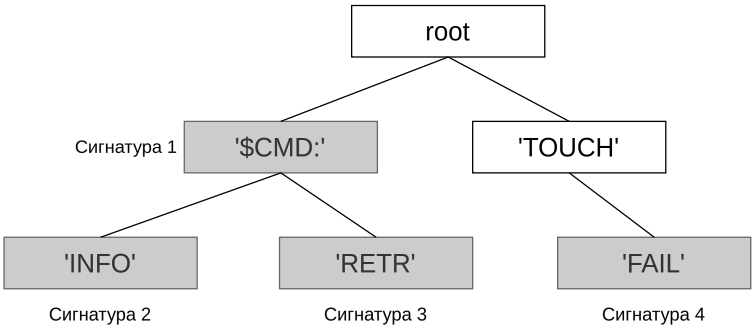

Рисунок 4 – Дерево подстрок в алгоритме AutoSig.

Корневой узел дерева подстрок является пустой подстрокой. Серые узлы соответствуют узлам сигнатурам. Каждый путь от узла сигнатуры к корневому узлу соответствует последовательности подстрок, находящиеся в узлах вдоль этого пути (исключая корневой узел). Дерево подстрок обладает следующими свойствами:

- Каждый узел соответствует общей подстроке и набору потоков F. Каждый поток в этом наборе содержит общую подстроку. Общая подстрока корневого узла является пустой строкой, будем считать, что она содержится во всех потоках.

- Если узел P является родительским узлом узла A, тогда набор потоков F(A) является подмножеством набора потоков F(P). Это значит, что поток, принадлежащий узлу F(A), не только содержит общую подстроку узла A, но также содержит общую подстроку своего родительского узла P. Например, на рис. 4 потоки узла ‘INFO’ содержат подстроки ‘INFO’ и ‘$CMD:’.

- Если набор потоков узла является строгим надмножеством объединения наборов потоков его дочерних узлов, узел является узлом сигнатуры. В противном случае это не узел сигнатуры. Корневой узел не является узлом сигнатуры. Листья являются узлами сигнатуры. Например, на рис. 4 узел ‘FAIL’ является узлом сигнатуры, а узел ‘TOUCH’ - нет. Потоки, содержащие подстроку ‘TOUCH’, также содержат подстроку ‘FAIL’. Вместо двух последовательностей подстрок

{‘TOUCH’, ‘FAIL’} и {‘TOUCH’} генерируется только последовательность подстрок {‘TOUCH’, ‘FAIL’}.

Построение дерева сигнатур начинается с пустого корня. Затем общие подстроки сортируются в порядке убывания в соответствии с количеством потоков, в которых они присутствуют. Потом эти подстроки вставляются в дерево одна за другой.

Для вставки новой подстроки, сначала проверяется является ли набор потоков нового узла подстроки подмножеством набора потоков текущего узла. Если это верно, то вставка завершается неудачей. В противном случае пытаемся рекурсивно вставить новый узел подстроки под каждым дочерним узлом текущего узла. Если все вставки завершатся неудачей, новый узел подстроки будет вставлен как ребенок текущего узла.

Каждый путь от узла сигнатуры к корневому узлу соответствует последовательности подстрок. Для генерации последовательностей подстрок используется алгоритм поиска в глубину со стеком для записи пути. После извлечения последовательностей подстрок в дереве каждая последовательность подстрок переупорядочивается в соответствии со смещением в потоке каждой подстроки.

3.4.3 SigBox

Теперь рассмотрим метод, который называется SigBox [19]. Данный метод основан на алгоритме последовательных шаблонов. Конечная цель этого алгоритма состоит в том, чтобы найти частые подпоследовательности в наборе входных последовательностей. Применяется один и тот же алгоритм для извлечения трех типов сигнатур: содержимого, пакета и потока. Изменяется лишь то, какие последовательности подаются на вход алгоритму и чем является элемент этой последовательности.

В случае извлечения сигнатуры содержимого последовательность содержимого является полезной нагрузкой пакета. Следовательно, элемент последовательности представляет собой однобайтный символ полезной нагрузки пакета.

В случае извлечения сигнатуры пакета последовательность пакетов представляет собой серию сигнатур содержимого, находящаяся в одной и той же полезной нагрузки пакета. Следовательно, элемент последовательности пакетов представляет собой индивидуальную сигнатуру содержимого.

В случае извлечения сигнатуры потока последовательность потока представляет собой серию сигнатур пакетов, расположенных в одной и той же полезной нагрузки потока. Следовательно, элемент последовательности потока представляет собой индивидуальную сигнатуру пакета.

Опишем работу алгоритма последовательных шаблонов. Сначала извлекаются подпоследовательности длиной 1 из всех последовательностей и сохраняются в наборе подпоследовательностей длиной 1, L_1. Из подпоследовательностей длиной $k-1$ извлекаются все подпоследовательности кандидаты длины k, увеличивая длину до тех пор, пока не будут извлечены все новые подпоследовательности (кандидаты). Этот итерационный процесс состоит из двух частей. Сначала из текущего множества исключаются

кандидаты, которые не удовлетворяют минимальной поддержке (1.0). Затем извлекаются кандидаты длиной k с помощью кандидатов длиной $(k-1)$. В качестве последнего шага проверяется связь включения между подпоследовательностями; если связь найдена, включенные подпоследовательности удаляются.

Чтобы подсчитать уровень поддержки подпоследовательности для заданного множества кандидатов, необходимо сначала найти множество всех хостов для множества кандидатов, а затем найти множество хостов, последовательность которых содержит заданную подпоследовательность. Значение поддержки для подпоследовательности равняется отношению мощностей найденных множеств. В [19] использовался наивный алгоритм сопоставления, однако авторы советовали заменить на более быстрые алгоритмы [20–23].

Для извлечения кандидатов длины k используется набор подпоследовательностей длины $k-1$, L_{k-1}. Сравниваются все возможные пары из L_{k-1}. Если $k=2$, то подпоследовательности в паре объединяются как подпоследовательность длиной 2. В противном случае, проверяется, что две подпоследовательности перекрываются $k-2$ символами. В этом случае две подпоследовательности длины $(k-1)$ объединяются, образуя единую подпоследовательность длины k.

Предложенная система генерирует сигнатуры содержимого, затем генерирует сигнатуры пакетов и, наконец, генерирует сигнатуры потоков, используя на каждом этапе описанный выше алгоритм.

4 Исследование и построение решения задачи

4.1 Сбор данных для дальнейшего тестирования

В данной работе генерация сигнатуры приложений рассматриваться не будет, так как большинство современных приложений использует шифрование, а административные методы, позволяющие дешифровать поступающий трафик, остаются за рамками данного исследования. Однако были выбраны такие протоколы, которые покрывают все основные возможные проблемы, возникающие при генерации сигнатур приложений, которые были описаны раннее.

Будут рассматриваться следующие протоколы: BitTorrent [24], DNS, FTP, HTTP, IMAP, POP3, SMTP. HTTP - типичный представитель текстового протокола, который используется не только веб-приложениями, но и в качестве туннеля. FTP использует множественное подключение (как минимум двойное), при этом один канал является управляющим, а через остальные происходит передача данных. DNS является представителем бинарного протокола, использующий UDP. IMAP, SMTP, POP3 - почтовые протоколы с маленькими размерами пакетов. BitTorrent - P2P протокол для кооперативного обмена файлами.

Исследуемый сетевой трафик снимался с кампусной сети ИСП РАН. Затем с помощью Wireshark [25] этот трафик был разбит по протоколам и результатом его работы были .pcap - файлы, которые содержали в себе сессии определенного протокола, захваченные в течение исследуемого сетевого взаимодействия. Пример выходного .pcap файла для HTTP можно увидеть на рис. 5.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	37.29.40.247	83.149.199.97	HTTP	1008	GET /api/tasks HTTP/1.1
2	0.017280	83.149.199.97	37.29.40.247	TCP	935	80 → 60785 [PSH, ACK] Seq=1 Ack=951 Win=1452 Len=877 [TCP seg…
3	0.017352	83.149.199.97	37.29.40.247	HTTP	78	HTTP/1.1 200 OK (application/json)
4	2.342100	37.29.40.247	83.149.199.97	HTTP	1008	GET /api/tasks HTTP/1.1
5	2.360254	83.149.199.97	37.29.40.247	TCP	935	80 → 60785 [PSH, ACK] Seq=898 Ack=1901 Win=1452 Len=877 [TCP …
6	2.360306	83.149.199.97	37.29.40.247	HTTP	78	HTTP/1.1 200 OK (application/json)
7	4.731904	37.29.40.247	83.149.199.97	HTTP	1008	GET /api/tasks HTTP/1.1
8	4.748144	83.149.199.97	37.29.40.247	TCP	935	80 → 60785 [PSH, ACK] Seq=1795 Ack=2851 Win=1452 Len=877 [TCP…
9	4.748264	83.149.199.97	37.29.40.247	HTTP	78	HTTP/1.1 200 OK (application/json)
10	7.039584	37.29.40.247	83.149.199.97	HTTP	1008	GET /api/tasks HTTP/1.1
11	7.056378	83.149.199.97	37.29.40.247	TCP	935	80 → 60785 [PSH, ACK] Seq=2692 Ack=3801 Win=1452 Len=877 [TCP…
12	7.056378	83.149.199.97	37.29.40.247	HTTP	78	HTTP/1.1 200 OK (application/json)

Рисунок 5 – Пример работы Wireshark.

В таблицах 1 и 2 представлены характеристики полученных данных для генерации сигнатур и их тестирования. Классы протоколов дополнительно не балансировались. В случае обучающего датасета нас интересуют потоки, содержащие хотя бы 5 пакетов, так как в слишком коротких потоках сигнатуры может не быть. Для тестирования был добавлен дополнительный класс «other», в который включены все остальные не рассматриваемые протоколы, и к которому классификатор будет относить все потоки, которым не соответствует ни одна сигнатура.

Таблица 1 – Данные для генерации сигнатур.

Протокол	Размер, МБ	Количество пакетов	Количество потоков	Avg bytes/pkt	Avg pkts/stream	Количество потоков: ≥ 5 pkts
bittorrent	272,4	240159	708	1134	339	214
dns	130,7	1283082	204150	102	6	11027
ftp	0,86	16959	735	51	23	725
http	1811,5	799062	13710	2268	58	1500
imap	22,0	27702	66	793	419	65
pop	0,06	919	59	64	16	40
smtp	13,5	59121	1120	229	53	799

Таблица 2 – Данные для тестирования сигнатур.

Протокол	Размер, МБ	Количество пакетов	Количество потоков	Avg bytes/pkt	Avg pkts/stream
bittorrent	1,26	9409	876	133	11
dns	53,3	664809	27800	80	24
ftp	0,19	4000	294	48	14
http	1494	406030	4607	3681	88
imap	3,38	10587	143	320	74
other	1119	1235122	18846	906	66
pop	0,02	344	21	58	16
smtp	5,8	34428	1018	170	34

4.2 Особенности при генерации сигнатур

1. Необходимо рассматривать только двунаправленные потоки, так как для генерации должны использоваться однородные потоки. Например, в случае HTTP набор общих подстрок для запрос и ответов разный. Поэтому необходимо либо уметь разделять потоки по направлениям и генерировать для каждого направления свою сигнатуру, либо не учитывать направление вовсе и генерировать общую сигнатуру. В первом случае, если не проводить семантический анализ протокола, то можно считать, что первый захваченный пакет в соединение идет от клиента к серверу, но это верно только для случая когда известно начало соединения. Это накладывает большое ограничение на выборку данных для генерации. Поэтому дальше будем рассматривать только двунаправленные потоки.

2. Сигнатура обычно располагается в начале потока, поэтому не имеет смысла анализировать весь поток. Подбор константы первых анализируемых пакетов или байт полезной нагрузки является эвристическим. При этом привязку лучше делать по количеству пакетов из-за сильного разброса размеров пакетов разных протоколов, так как согласно [9] сигнатура содержится в первых 10 пакетах потока.

3. Необходимо ввести следующее эвристическое ограничение на сигнатуры, чтобы исключить тривиальные сигнатуры из рассмотрения: общая длина одной последовательности подстрок не меньше 8 символов и состоит хотя бы из 2 подстрок, каждая из которых не меньше 3 символов.

4. Алгоритм AutoSig ввел дерево подстрок. Данная структура хороша тем, что увеличивает мощность получаемых сигнатур, это позволяет выделить несколько последовательностей подстрок, которые могут охватить те ситуация, в которых приложение имеет сильно разные потоки (например, поток управления и поток данных в FTP). Однако в том в виде, в котором это дерево представлено в работе [14], получается сильно избыточный результат. Если узел является сигнатурным и не листом, то по свойству этого узла его набор потоков является строгим надмножеством наборов потоков любых других сигнатурных узлов, находящихся в его поддереве (например, листы, которые всегда являются сигнатурными), таким образом, любые пути из сигнатурных узлов поддерева являются избыточными, так как если нашлась последовательность подстрок соответствующая сигнатурному узлу из рассматриваемого поддерева, то найдётся в потоке и последовательность подстрок, соответствующая рассматриваемому узлу. При этом специфичность нашей сигнатуры за счёт этих сигнатурных узлов не увеличивается, так как для совпадения со сигнатурой достаточно совпадения последовательности подстрок, соответствующей нашему узлу, которая уже включена в другие. А значит поддерево можно удалить. Однако если не выполняется строгость надмножества, то последовательности строк поддерева не будут включаться друг в друга, а полнота покрытия потоков сохраниться, при этом вырастит специфичность (чем длиннее последовательность подстрок, тем специфичнее сигнатура). Именно поэтому при таком условии узел не становится сигнатурным.

4.3 Выбор алгоритма для автоматической генерации сигнатур

В таблице 3 проведен сравнительный анализ рассмотренных ранее методов

Таблица 3 – Сравнительный анализ методов автоматической генерации сигнатур.

Показатель\Метод	LASER (LCS)	AutoSig	SigBox
Тип сигнатуры	сигнатура пакета (сигнатура потока)	сигнатура потока	сигнатура потока
Формат сигнатуры	последовательность подстрок	набор последовательностей подстрок	последовательность подстрок
Тип ограничения	количество пакетов (размер полезной нагрузки)	размер полезной нагрузки	размер полезной нагрузки
Агрегация потоков	не хранятся все потоки	хранятся все потоки	хранятся все потоки
Сборка сессии	не требуется	требуется	не требуется
Параметр поддержки	отсутствует	присутствует	не является параметром алгоритма

Тип сигнатуры. LASER извлекает наибольшую сигнатуру пакета, в то время как AutoSig и SigBox извлекают сигнатуры потоков. Можно также применить шаг генерации алгоритма LASER - LCS, не только к пакету, но и ко всему потоку целиком, получив сигнатуру потока.

Формат сигнатуры. AutoSig в отличии от LASER и SigBox генерирует не одну последовательность подстрок, а сразу несколько, используя дерево сигнатур. Но как

отмечалось уже раннее, это дерево сигнатур может быть избыточным.

Тип ограничения. Для алгоритмов AutoSig и SigBox используется ограничение по размеру полезной нагрузке потока, т.е. потоки должен быть не меньше n-байт, но при этом использоваться будут только первые k-байт. В свою очередь LASER использует аналогичное ограничение по числу пакетов.

Агрегация потоков. Для алгоритма LASER нет необходимости хранить все потоки одновременно, так как при генерации одновременно используется не более 2 потоков. А в случае AutoSig и SigBox необходимо хранить все потоки одновременно, так как используется частотное распределение для построение сигнатур путём увеличения коротких последовательностей.

Сборка сессии. Несмотря на схожесть алгоритмов AutoSig и SigBox, SigBox не требует сборки сессии из-за иерархического построения сигнатуры (содержимого -> пакета -> потока). Для LASER сборка не требуется, так как он не работает на уровне потока.

Параметр поддержки. Этот параметр помогает бороться с шумами (потоки, которые сильно выбиваются по своему набору подстрок). В LASER этот параметр не был заложен, его можно принимать константой равной 1.0. В AutoSig этим параметром может выступать параметр «S» (пороговое значение слияния). Этот параметр регулирует мощность набора сигнатур, если он равен 1.0, то набор будет состоять не более, чем из 1 последовательности подстрок. В SigBox этот параметр зафиксирован и равен 1.0, таким образом выходной сигнатурой SigBox тоже будет состоять не более, чем из 1 последовательности подстрок.

Таким образом, для дальнейших исследований был выбран алгоритм LASER, за свою простоту и не требовательность к хранению всех потоков одновременно, что является важным параметром для интеграции в систему анализа сетевого трафика. Также алгоритм LASER позволяет взять свой шаг генерации - алгоритм LCS, и попробовать получить сигнатуру потока.

4.4 Реализация генератора и классификатора

Был реализован генератор сигнатур с возможностью замены основного алгоритма генерации сигнатур. В данной работе будет использоваться алгоритм LASER и его модификации.

Классификатор был также реализован с возможностью замены алгоритма сопоставления сигнатуры с полезной нагрузкой. Он поддерживает сопоставление сигнатур пакетов, так и сигнатур потоков.

На описанных раннее данных были запущены генератор сигнатур и классификатор. Полученные результаты представлены на рис. 6 и в табл. 4.

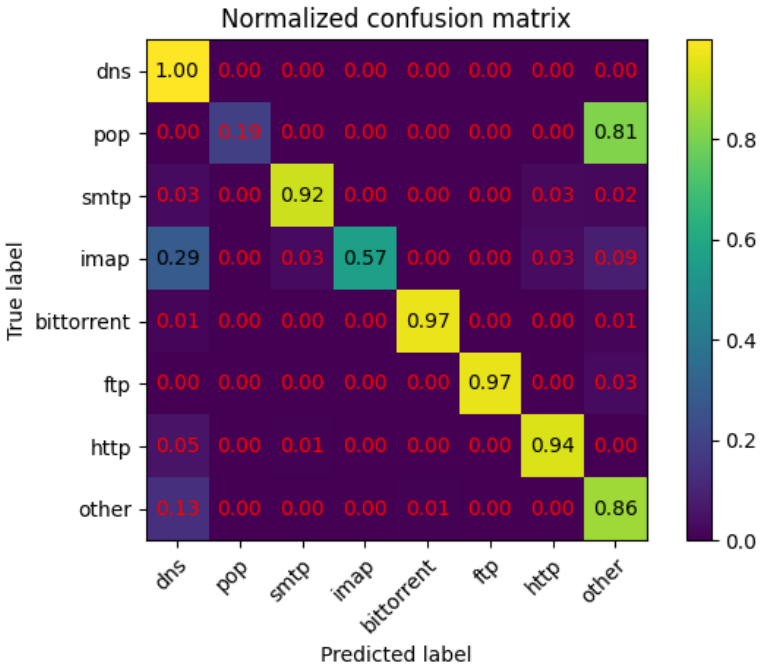

Рисунок 6 – Матрица ошибок для алгоритма LASER.

Таблица 4 – Результаты классификации для алгоритма LASER.

protocol	precision	recall	f1-score	support
dns	0.93	1.00	0.97	27801
pop	0.80	0.19	0.31	21
smtp	0.94	0.92	0.93	1076
imap	1.00	0.57	0.72	175
bittorrent	0.92	0.97	0.95	886
ftp	0.99	0.97	0.98	295
http	0.99	0.94	0.96	4888
other	0.99	0.86	0.92	12094
accuracy			0.95	47236
macro avg	0.95	0.80	0.84	47236
weighted avg	0.95	0.95	0.95	47236

Все протоколы, кроме POP3 и IMAP, отлично классифицируются: средневзвешенное значение F1-меры составило 0.95. Такой результат можно объяснить малым количеством потоков для генерации сигнатур этих протоколов. Из матрицы ошибок также можно увидеть, что сигнатуры DNS не являются специфичными.

Рассмотрим среднее количество потоков необходимое для генерации сигнатуры. Согласно табл. 5 наибольшее среднее количество потоков, потребовавшееся для генерации 1 сигнатуры, равно 80,5 для FTP. Это объясняется тем, что в FTP есть два разных потока: управления и данных, и алгоритм LASER не способен генерировать сигнатуру для сильно разных потоков в отличии от AutoSig и SigBox, поэтому генератор дожидается того момента, пока не придут потоки одного типа. Малое количество сигнатур

POP3 объясняется малым количеством потоков в начальных данных. Все остальные результаты согласуются с [15].

Таблица 5 – Среднее количество потоков необходимое для генерации сигнатуры.

Протокол	Количество сигнатур	Среднее количество потоков на 1 сигнатуру
bittorrent	51	4,1
dns	682	16,1
ftp	9	80,5
http	145	10,3
imap	10	6,5
pop	2	20,0
smtp	167	4,8

4.5 Влияние постобработки на результаты классификации

Стоит отметить большое количество сигнатур DNS, которые естественным образом перестают быть специфичными и являются избыточными. Необходимо выполнять постобработку для генерируемых сигнатур.

Постобработка будет состоять из следующих шагов:

1. удаление дубликатов
2. удаление надмножеств

Первый шаг очень естественный. На втором шаге предлагается удалить надмножества. Согласно раннее описанному недостатку введенному дереву сигнатур в AutoSig, если представлять набор последовательностей подстрок (все сигнатуры можно объединить в один набор) в виде дерева, то не имеет смысла иметь в наборе сигнатуры, путь от корня до узлов которых встречаются другие сигнатурные узлы. Если эта сигнатура была найдена в полезной нагрузке, то это значит, что и любая её часть была найдена в полезной нагрузке. Удаляя такие сигнатуры сильно уменьшается количество сигнатур в наборе, при этом качество классификации не изменяется.

Если бы удалялись подмножества, т.е. оставлялись самые специфичные сигнатуры, то количество сигнатур в наборе изменялось меньше, а качество классификации меняется (обычно) в худшую сторону, так как остаются слишком специфичные сигнатуры.

Результаты постобработки приведены в табл. 6.

Приблизительно половина всех удаленных сигнатур была дубликатами, остальные - надмножеством. Можно увидеть, что такая простая постобработка не помогла сильно уменьшить избыточность сигнатур, т.е. найти минимальное покрывающее множество. Поэтому в дальнейших исследованиях планируется рассмотреть более мощные методы.

Таблица 6 – Количество сигнатур после постобработки.

Протокол	Количество сигнатур		Избыточность	
	Было	Стало	Было	Стало
bittorrent	51	7	0.53	0.53
dns	682	64	0.99	0.99
ftp	9	8	0.98	0.98
http	145	47	1.00	1.00
imap	10	4	1.00	0.79
pop	2	1	1.00	0.00
smtp	167	26	0.99	0.99

4.6 Влияние сборки сессии на результаты классификации

Теперь рассмотрим модификацию алгоритма LASER. Шагом генерации сигнатуры в нем является алгоритм LCS, который выделяет общие подстроки у двух потоков байт. В оригинальном алгоритме шаг генерации применялся к пакетам, но теперь будем применять к потоку.

Будут исследоваться следующие две модификации:

1. с частичной сборкой сессии
2. с полной сборкой сессии

Под частичной сборкой сессии понимается, что полезная нагрузка пакетов собирается в один поток байт до первого пакета другого направления. Для UDP полезная нагрузка просто конкатенируется, а для TCP также восстанавливаться правильный порядок пакетов.

На рис. 7 и в табл. 7 представлены результаты для частичной сборки сессии. Результаты получились аналогичными оригинальному алгоритму, небольшое отклонение связано с изменением значения числа «пакетов» для рассмотрения. Оно было изменено таким образом, чтобы число потоков удовлетворяющему требованию на размер потока, осталось примерно таким же.

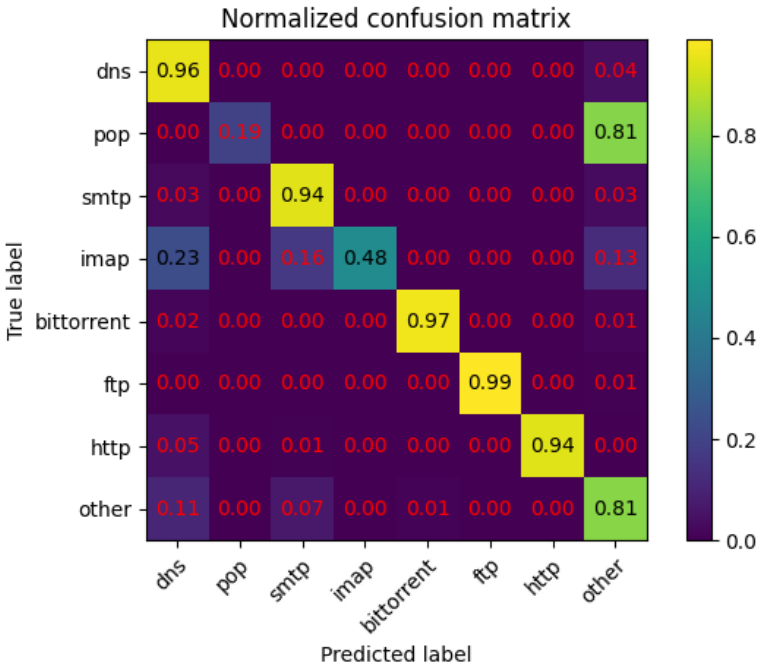

Рисунок 7 – Матрица ошибок для алгоритма LASER с частичной сборкой сессии.

Таблица 7 – Результаты классификации для алгоритма LASER с частичной сборкой сессии.

protocol	precision	recall	f1-score	support
dns	0.92	0.96	0.94	10565
pop	0.67	0.19	0.30	21
smtp	0.78	0.94	0.85	1464
imap	0.91	0.48	0.62	189
bittorrent	0.95	0.97	0.96	853
ftp	0.99	0.99	0.99	288
http	1.00	0.94	0.97	4890
other	0.88	0.81	0.85	4804
accuracy			0.92	23074
macro avg	0.89	0.79	0.81	23074
weighted avg	0.92	0.92	0.92	23074

На рис. 8 представлены результаты для случая полной сборки сессии. Как можно видеть выделенные сигнатуры получились не специфичными. Связано это с привязкой уже к размеру рассматриваемой полезной нагрузки. Значение равное 1 КБ было выбрано согласно [14]. Это связано с тем, что положение сигнатуры все же привязано к определенным номерам пакетов, а из-за разного распределения размеров для протоколов, нельзя использовать одно значение размера полезной нагрузки для всех протоколов.

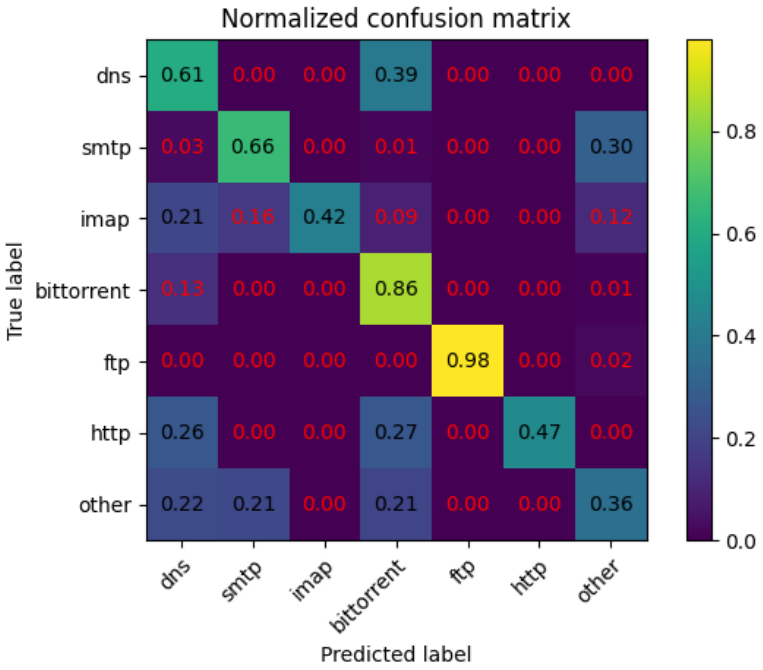

Рисунок 8 – Матрица ошибок для алгоритма LASER с полной сборкой сессии.

5 Описание практической части

5.1 Формат хранения сигнатуры

Выбранный формат сигнатуры - это набор последовательности подстрок. Представим требования хранения выбранного формата сигнатур в памяти:

1. Результатом работы оригинального алгоритма LASER является одна последовательность подстрок. Предполагается, что подстроки отделены специальным символом, но так как в бинарных протоколах используются все значения байта, то нельзя найти такой символ. Однако предположим, что такой символ найдется и заметим, что в момент уточнения сигнатуры, алгоритм LCS, который лежит в основе LASER, не чувствителен к наличию этого символа во входном потоке байт, так как это специальный символ и во втором потоке байт его нет. Значит последовательность подстрок необходимо хранить последовательно без разделяющего символа. Это также уменьшит размер матрицы направлений.

2. Для того, чтобы постоянно не рассчитывать общую длину последовательности подстрок, будем хранить этот размер.

3. Для того, чтобы иметь доступ к самим подстрокам, например, с целью напечатать их, будем хранить смещения на каждую подстроку.

4. Необходимо также хранить и само количество подстрок, чтобы определить количество смещений.

5. Так как строка в `C/C++` должна быть нуль-терминирована, то в конце всех подстрок будет стоять ‘\0’. Сами подстроки дополнительно на ‘\0’ не оканчиваются.

6. Так же для быстрого доступа к подстроке и быстрого расчёта её длины, будем ещё будем хранить одно дополнительное смещение на ‘\0’, то есть суммарно $n+1$ смещение, где n - количество подстрок. Каждое смещение рассчитывается относительно начала последовательности подстрок. Тогда размер подстроки можно легко вычислить через разницу смещений. Первое смещение всегда равно 0, его можно не хранить, но хранится для однородности вычислений.

7. Так как размер последовательности подстрок небольшой, то такое представление последовательности подстрок должно быть cache-friendly. Значит для представления набора последовательностей подстрок каждая последовательность должна быть локализирована.

8. Естественно надо хранить количество последовательностей и их смещения для доступа к ним.

Таким образом, данным требованиям удовлетворяет представление на рис. ??.

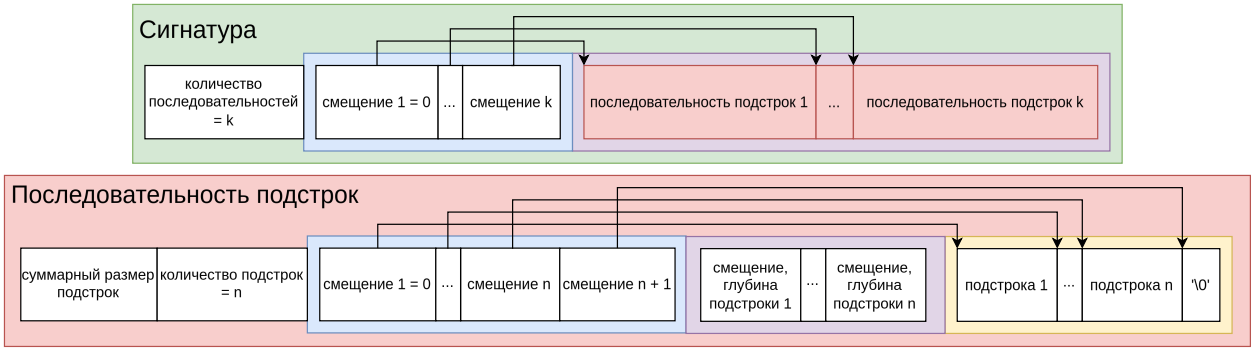

Рисунок 9 – Формат хранения сигнатуры.

Также в это представление были заложены специальные поля для каждой подстроки: смещение и глубина. Информация о смещении указывает начальную позицию для поиска подстроки в полезной нагрузке пакета, а информация о глубине указывает, как далеко поиск должен продолжаться от местоположения смещения. Если этой информации нет, то значения данных полей соответственно равны 0 и -1. Глубина равная -1 означает, что поиск требуется осуществлять до конца полезной нагрузки. Данные поля помогают ускорить процесс поиска подстрок при сопоставлении сигнатуры.

5.2 Интеграция в систему анализа сетевого трафика

Система анализа сетевого трафика, разрабатываемая в ИСП РАН, состоит из обрабатывающих модулей. В рамках данной задачи использовалось несколько схем обрабатывающих модулей, которые объединяются в схемы обработки. На схемах ниже оранжевым обозначены модули, которые уже были реализованы в системе, а фиолетовым - те, которые появились в процессе работы над задачей.

Сначала использовались схемы без сборки TCP-сессий: для них использовался оригинальный алгоритм LASER.

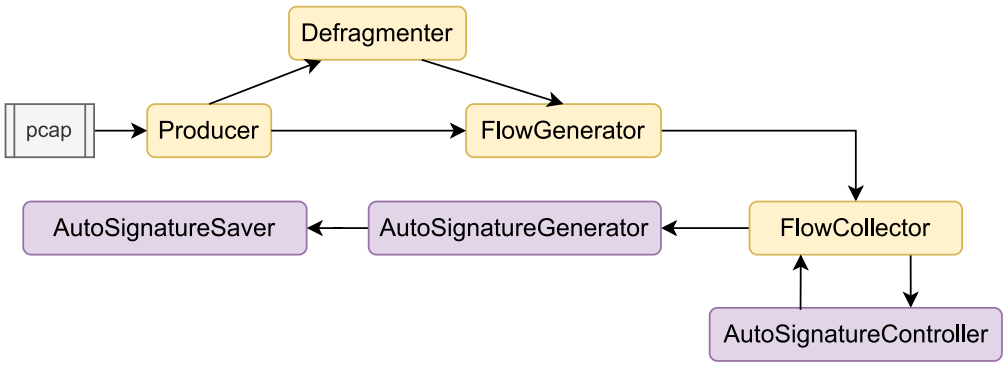

Рисунок 10 – Схема генерации сигнатуры без сборки TCP-сессий.

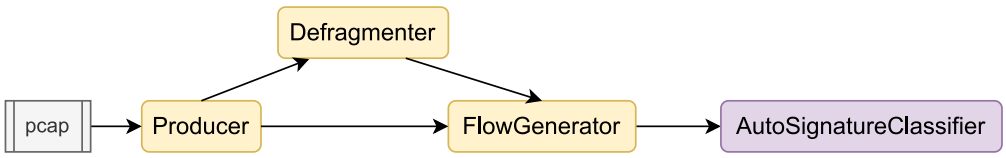

Рисунок 11 – Схема классификации трафика без сборки TCP-сессий.

Поговорим про каждый модуль более подробно:

- **Producer**: обрабатывает поступающий ему на вход pcap-файл, разделяя его на пакеты. Также может поставлять пакеты с интерфейса устройства в режиме реального времени.
- **Defragmenter**: осуществляет дефрагментацию IP пакета, если он был разделён на фрагменты.
- **FlowGenerator**: приписывает каждому пакету номер потока, к которому он принадлежит по IP-адресу источника, IP-адресу назначения, порту источника, порту назначения и используемый протокол транспортного уровня. Как уже и отмечалось ранее, потоки, идущие в два направления, объединены.
- **FlowCollector**: сохраняет приходящие пакеты и перенаправляет их в модуль AutoSignatureController и ждёт от него сигнала. Как только приходит сигнал от AutoSignatureController с идентификатором потока, FlowCollector отправляет все сохранённые пакеты этого потока в модуль AutoSignatureGenerator.
- **AutoSignatureController**: собирает статистику по каждому потоку и отправляет сигнал FlowCollector с идентификатором того потока, который выполнил некоторые требования выбранного алгоритма. Например, для LASER это ограничение по пакетам: в потоке должно присутствовать хотя бы N пакетов, последующие пакеты будут проигнорированы.
- **AutoSignatureGenerator**: выполняет генерацию сигнатуры на основе приходящих пакетов и передаёт полученную сигнатуру в AutoSignatureSaver.
- **AutoSignatureSaver**: выполняет десериализацию сигнатуры и сохраняет в файл в формате .json.
- **AutoSignatureClassifier**: классифицирует приходящий трафик, сопоставляя полезную нагрузку пакета с набором сигнатур.

Для других модификаций алгоритма LASER использовалась схема со сборкой TCP-сессии (полной или частичной).

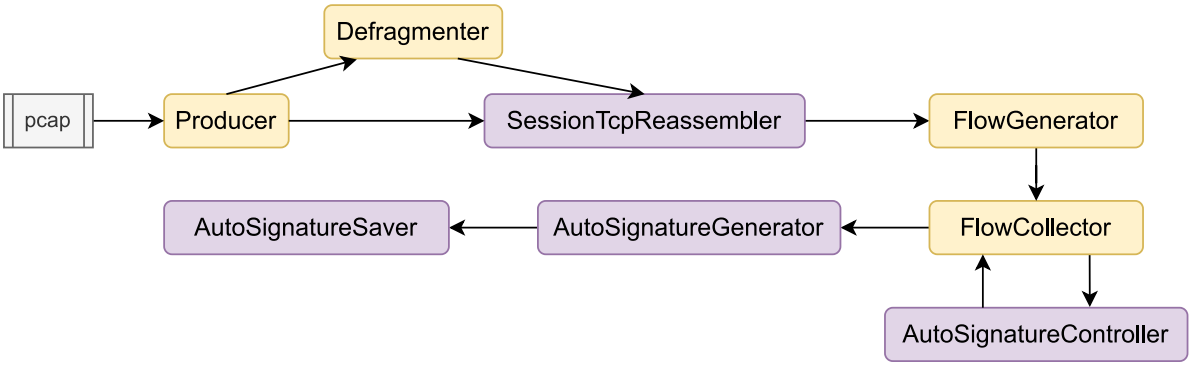

Рисунок 12 – Схема генерации сигнатуры со сборкой TCP-сессий.

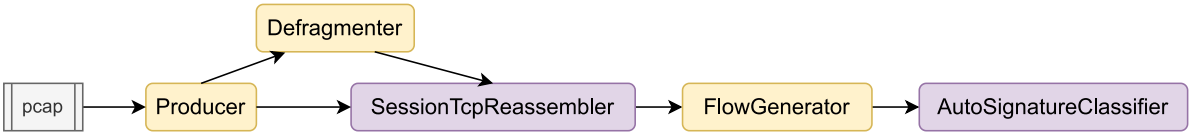

Рисунок 13 – Схема классификации трафика со сборкой TCP-сессий.

Отличия от предыдущих схем:

- Перед FlowGenerator стоит модуль SessionTcpReassembler. Он собирает TCP-сессию и отправляет полезную нагрузку дальше после завершения сборки (или при срабатывании настраиваемого таймера), все последующие модули работают с этой полезной нагрузкой. Если пришёл UDP-пакет, то просто пропускается дальше.
- AutoSignatureController использует ограничение на необходимую длину полезной нагрузки.
- AutoSignatureClassifier использует уже сигнатуры-потоков, так как сопоставление происходит по полезной нагрузке потока.

В ходе данной работы были разработаны следующие модули:

1. AutoSignatureGenerator,
2. AutoSignatureClassifier,
3. AutoSignatureController,
4. AutoSignatureSaver,
5. SessionTcpReassembler.

Так как все модули системы были реализованы на языке программирования `C++`, то новые модули были реализованы также на нём. `C++` позволяет писать высокопроизводительные программы, что является необходимостью для обработки трафика в режиме реального времени.

Также для совместимости с SessionTcpReassembler были модифицированы модули FlowGenerator и FlowCollector, которые теперь позволяют работать не только с пакетами в отдельности, но и с полезной нагрузкой сессии.

В конечном итоге после внедрения модулей было получено 2 пайплайна. Первый позволяет генерировать сигнатуры потоков на основе их полезной нагрузки, а второй использует эти сигнатуры для классификации трафика. Разделение на 2 пайплайна выполнено из соображения их независимости.

6 Заключение

В данной работе был разработан и реализован автоматический генератор сигнатур на основе полезной нагрузки сетевого трафика и классификатор трафика, основанный на сопоставлении этих сигнатур, в соответствии с использующимся протоколом.

Были выполнены следующие задачи:

1. Проведено исследование литературы по соответствующей теме.
2. Собран набор сетевых трасс для генерации и классификации.
3. Выбран формат сигнатуры сетевых протоколов.
4. Рассмотрены ограничения выбранных методов и реализован один из них.
5. Разработан классификатор сетевого трафика для проверки сгенерированных сигнатур.
6. Рассмотрено влияние сборки TCP-сессии и постобработки на результат классификации.
7. Встроены генератор сигнатур и классификатор как модули в систему анализа высокоскоростного сетевого трафика, разрабатываемую в ИСП РАН.

Планируемые будущие исследования по данной теме:

1. Реализация и сравнение других методов генерации сигнатур: AutoSig и SigBox.
2. Реализация модуля уточнения положения сигнатуры и его влияние на точность.
3. Рассмотрение возможности применения машинного обучения для отбора сигнатур при постобработке результатов.

Список литературы

[1] Internet Assigned Numbers Authority. — https://www.iana.org, дата обращения 11.06.2024.

[2] Dusi Maurizio, Crotti Manuel, Gringoli Francesco, Salgarelli Luca. Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting // Computer Networks. — 2009. — Vol. 53, no. 1. — P. 81–97.

[3] Гетьман А. И., Евстропов Е. Ф., Маркин Ю. В. Анализ сетевого трафика в режиме реального времени: обзор прикладных задач, подходов и решений. // Препринт ИСП РАН. — 2015. — Т. 28. — С. 1–52.

[4] Erman Jeffrey, Mahanti Anirban, Arlitt Martin. Qrp05-4: Internet traffic identification using machine learning // IEEE Globecom 2006 / IEEE. — 2006. — P. 1–6.

[5] Singh Sumeet, Estan Cristian, Varghese George, Savage Stefan. Automated Worm Fingerprinting. // OSDI. — Vol. 4. — 2004. — P. 4–4.

[6] Kim Hyang-Ah, Karp Brad. Autograph: Toward Automated, Distributed Worm Signature Detection. // USENIX security symposium / San Diego, CA. — Vol. 286. — 2004.

[7] Newsome James, Karp Brad, Song Dawn. Polygraph: Automatically generating signatures for polymorphic worms // 2005 IEEE Symposium on Security and Privacy (S&P'05) / IEEE. — 2005. — P. 226–241.

[8] Перспективный мониторинг. — https://amonitoring.ru/service/snort/, дата обращения 11.06.2024.

[9] Park Byung-Chul, Won Young J, Kim Myung-Sup, Hong James W. Towards automated application signature generation for traffic identification // NOMS 2008-2008 IEEE Network Operations and Management Symposium / IEEE. — 2008. — P. 160–167.

[10] Szabó Géza, Turányi Zoltán, Toka László, Molnár Sándor and. Automatic protocol signature generation framework for deep packet inspection // 5th International ICST Conference on Performance Evaluation Methodologies and Tools. — 2012.

[11] Wang Yu, Xiang Yang, Zhou Wanlei, Yu Shunzheng. Generating regular expression signatures for network traffic classification in trusted network management // Journal of Network and Computer Applications. — 2012. — Vol. 35, no. 3. — P. 992–1000.

[12] Vinoth George C, Ewards V. Efficient regular expression signature generation for network traffic classification // International Journal of Science and Research (IJSR). — 2013.

[13] Snort. — https://www.snort.org, дата обращения 11.06.2024.

[14] Ye Mingjiang, Xu Ke, Wu Jianping, Po Hu. Autosig-automatically generating signatures for applications // 2009 Ninth IEEE International Conference on Computer and Information Technology / IEEE. — Vol. 2. — 2009. — P. 104–109.

[15] Santos Alysson Feitoza. Automatic Signature Generation : Ph.D. thesis / Alysson Feitoza Santos ; Federal University of Pernambuco. — 2009. — https://www.cin.ufpe.br/~tg/2009-1/afs5.pdf.

[16] Goo Young-Hoon, Shim Kyu-Seok, Lee Su-Kang, Kim Myung-Sup. Payload signature structure for accurate application traffic classification // 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS) / IEEE. — 2016. — P. 1–4.

[17] Shim Kyu-Seok, Goo Young-Hoon, Lee Dongcheul, Kim Myung-Sup. Automatic Payload Signature Update System for the Classification of Dynamically Changing Internet Applications // KSII Transactions on Internet and Information Systems (TIIS). — 2019. — Vol. 13, no. 3. — P. 1284–1297.

[18] Ning Kang, Ng Hoong Kee, Leong Hon Wai. Finding patterns in biological sequences by longest common subsequencesand shortest common supersequences // Sixth IEEE Symposium on BioInformatics and BioEngineering (BIBE'06) / IEEE. — 2006. — P. 53–60.

[19] Shim Kyu-Seok, Yoon Sung-Ho, Lee Su-Kang, Kim Myung-Sup. SigBox: Automatic Signature Generation Method for Fine-Grained Traffic Identification. // Journal of Information Science & Engineering. — 2017. — Vol. 33, no. 2.

[20] Karp Richard M, Rabin Michael O. Efficient randomized pattern-matching algorithms // IBM journal of research and development. — 1987. — Vol. 31, no. 2. — P. 249–260.

[21] Boyer Robert S, Moore J Strother. A fast string searching algorithm // Communications of the ACM. — 1977. — Vol. 20, no. 10. — P. 762–772.

[22] Xie Linquan, Liu Xiaoming, Yue Guangxue. Improved pattern matching algorithm of BMHS // 2010 Third International Symposium on Information Science and Engineering / IEEE. — 2010. — P. 616–619.

[23] Zhou Yansen, Pang Ruixuan. Research of Pattern Matching Algorithm Based on KMP and BMHS2 // 2019 IEEE 5th International Conference on Computer and Communications (ICCC) / IEEE. — 2019. — P. 193–197.

[24] Bittorent. — https://www.bittorent.com, дата обращения 11.06.2024.

[25] Wireshark. — https://www.wireshark.org, дата обращения 11.06.2024.