

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 и приложений

**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 и приложений

*Дурнов Алексей Николаевич*

Краткое описание задачи и основных результатов, мотивирующее прочитать весь текст

# Содержание

<b>1</b>	<b>Введение</b>	<b>4</b>
<b>2</b>	<b>Постановка задачи</b>	<b>5</b>
<b>3</b>	<b>Обзор существующих решений</b>	<b>6</b>
3.1	Формат сигнатур . . . . .	6
3.2	Структура сигнатур . . . . .	6
3.3	Метрики оценки качества сигнатур . . . . .	7
3.4	Обзор существующих методов автоматической генерации сигнатур . . . .	8
3.4.1	LASER . . . . .	9
<b>4</b>	<b>Исследование и построение решения задачи</b>	<b>12</b>
<b>5</b>	<b>Описание практической части</b>	<b>13</b>
<b>6</b>	<b>Заключение</b>	<b>14</b>

# 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 обслуживания (QoS), а также планировать свою инфраструктуру и управлять ею. Сетевой трафик можно классифицировать как в соответствии с используемым протоколом, так и в соответствии с используемым приложением. Вторая классификация более трудоёмкая, но позволяет решать более широкий спектр задач: формирование трафика в сети, улучшение качества обслуживания, 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. Однако современные приложения используют динамическое распределение портов и туннелирования трафика, например, по протоколу НТТР, данный метод даёт очень плохие и неточные результаты. Чтобы преодолеть эти ограничения идентификации были введены более совершенные методы.

Первый подход основан на сопоставлении сигнатур полезной нагрузки. Сигнатура полезной нагрузки - это часть данных полезной нагрузки, которая является статичной и различимой для приложений и может быть описана, как последовательность строк или шестнадцатиричных чисел. Второй подход основан на алгоритмах машинного обучения. Для этого подхода используются такие признаки потоков и пакетов, как задержки между пакетами, размеры пакетов и другие, а полезная нагрузка пакетов не анализируется, поэтому он менее точен, чем сигнатурный подход. Однако он может применяться для зашифрованного сетевого трафика, так как в приближении полезная нагрузка зашифрованного трафика представляет собой белый шум, поэтому сигнатурный подход невозможен. Также существуют и гибридные методы, которые используют эти два подхода вместе.

Методы, которые анализируют полезную нагрузку пакетов, являются очень эффективными и точными для идентификации трафика. Их часто называют глубокой проверкой пакетов (DPI). DPI является очень трудоемким и ресурсоемким процессом. Система DPI должна искать сигнатуры в полезной нагрузке пакетов, чтобы точно классифицировать поток. Такой подход не нов, системы обнаружения вторжения (IDS), основанные на DPI, с помощью сигнатур находят интернет-червей и другой трафик, угрожающий безопасности. Далее рассматриваемые методы будут сосредоточены на идентификации приложений среди безопасного трафика.

Поначалу сигнатуры извлекались вручную. 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.

Поэтому данная работа посвящена исследованию различных методов автоматической генерации сигнатур полезной нагрузки для классификации сетевого трафика по протоколам и приложением.

## 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 используемым протоколом или приложением в режиме реального времени.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести исследование литературы по соответствующей теме.
2. Собрать набор сетевых трасс для последующего тестирования и сравнение методов.
3. Выбрать оптимальный метод для автоматической генерации сигнатур.
  - (а) Выбрать оптимальный набор параметров метода для каждого тестируемого протокола и приложения, если метод обладает настраиваемыми параметрами.
  - (б) Найти ограничения рассматриваемых методов.
4. Встроить генератор сигнатур и классификатор как модули в систему анализа высокоскоростного 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 будут представлены.

Существует несколько представлений сигнатур. Некоторые из этих видов использовались для представлений сигнатур червей. Однако для обычного трафика можно выделить два основных представления:

1. сигнатуры, представленные регулярными выражениями
2. сигнатуры, представленные строками

Использование регулярных выражений для описания сигнатур приложений становится очень распространённым в классификации потоков. Однако процесс сопоставления регулярных выражений требует огромной вычислительной мощности, которая не масштабируется для идентификации 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) имеет более 1000 подписей приложений и предлагает пользователю возможность вставлять новые регулярные выражения по требованию.

Представление сигнатур в виде строк это компромисс между мощностью выражения и эффективностью сопоставления. Также такой подход позволяет преобразовывать сигнатуры, представленные строками, в регулярные выражения.

Будем дальше рассматривать сигнатуры в виде строк, преобразование в регулярные выражения останется за рамками данной работы.

### 3.2 Структура сигнатур

Большинство форматов сигнатур в предыдущих работах 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 приложения, некоторые могут принадлежать и другому приложению. Это называется избыточностью сигнатур.

Выделим три типа сигнатур:

1. сигнатура содержимого (полезной нагрузки),
2. сигнатура пакета,
3. сигнатура потока.

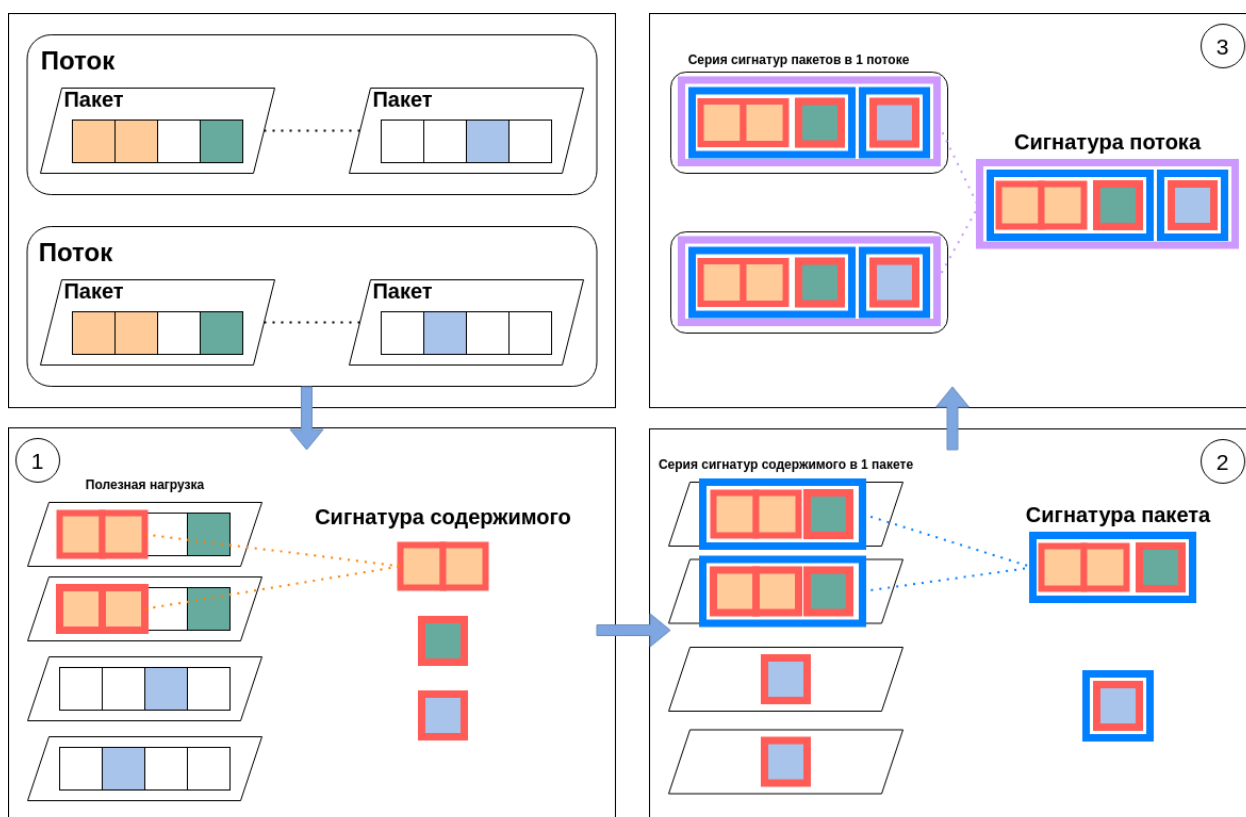


Рис. 1: Процесс извлечения предлагаемой структуры сигнатур полезной нагрузки

Сигнатура содержимого определяется как различимая и уникальная подстрока полезной нагрузки, состоящая из непрерывных символов или шестнадцатеричных значений. На самом деле уникальность с помощью одной подстроки тяжело обеспечить, 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, т.е. без сбора потока, то анализируется всегда хотя бы один пакет. Это значит, что для классификации не имеет смысла использовать отдельно сигнатуру содержимого.

Сигнатура потока состоит из серии сигнатур пакетов, которые появляются в одном потоке, где под потоком понимается набор пакетов, имеющих одни и те же 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.

### 3.3 Метрики оценки качества сигнатур

Для оценки качества получаемых сигнатур рассмотрим матрицу ошибок: 4 стандартные категории, к которым можно отнести результат работы классификатора на полученной сигнатуре. В нашем случае рассматриваемый класс это целевой протокол или приложение. Под классификацией трафика будем понимать классификацию конкретного пакета или потока в зависимости от того, какой уровень сигнатур используется.

	Принадлежит классу (P)	Не принадлежит классу (N)
Предсказана принадлежность к классу (T)	TP	TN
Предсказано отсутствие принадлежности к классу (F)	FP	FN

- истинно положительный (TP): указывает, что трафик правильно классифицирован, как относящийся к определенному классу.
- истинно отрицательный (TN): указывает, что трафик правильно классифицирован, как не относящийся к определенному классу.
- ложно положительный (FP): указывает, что трафик неправильно классифицирован, как относящийся к определенному классу.
- ложный отрицательный (FN): указывает, что трафик неправильно классифицирован, как не относящийся к определенному классу.

Наиболее часто используемые показатели для классификации трафика определяются следующим образом:

- Ассигасу (достоверность) =  $\frac{TP+TN}{TP+TN+FP+FN}$  - доля правильных классификаций.
- Recall (полнота) =  $\frac{TP}{TP+FN}$  - отношение верно классифицированного трафика определенному классу к общему числу трафика этого класса, т.е. описывает способность сигнатуры обнаружить данный целевой протокол или приложение.
- Precision (точность) =  $\frac{TP}{TP+FP}$  - доля верно классифицированного трафика среди всего трафика, который классификатор отнёс к этому классу.
- F-мера =  $2 \frac{recall \cdot precision}{recall + precision}$  - гармоническое среднее между точностью и полнотой. Метрика ассигасу 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 правдой 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:

$$\text{Redundancy} = \frac{\text{Объём трафика идентифицированный двумя и более сигнатурами}}{\text{Объём трафика идентифицированный набором сигнатур}}$$

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, т.е. являются уникальными и незаменимыми. Если redundancy близка к 1, то в наборе присутствуют 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

Перед тем как начать извлекать сигнатуры из пакетов или потоков, сначала производится сбор очищенных пакетов, т.е. сырых пакетов, которые относятся только к целевому протоколу или приложению



### 3.4.1 LASER

В статье [1] описан алгоритма 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, т.е. генерирует сигнатуру пакета. За шаг извлечения сигнатуры был принят алгоритм LCS, который в основном использовался для сопоставления последовательностей ДНК в приложениях биоинформатики. Он был модифицирован под наши задачи. Вводятся ограничения на модификацию LCS:

- **Количество пакетов в потоке.** Поток это набор пакетов, имеющих одни и те же IP-адрес источника, IP-адрес назначения, порт источника, порт назначения и используемый протокол. 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 длине. 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, мы предотвращаем 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 такой характеристик является размер пакета. Из графика видно, что 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.

Приведём сам алгоритм из статьи:

```
1 procedure Signature_Generation ()
2   Flow_Pool {F1[]...Fx[]} <- Santized_packet_collector
3   F1[] <- Iterate, packet dump for Flow 1
4   F2[] <- Iterate, packet dump for Flow 2
5   while i from 0 to #_packet_constraint do
6     while j from 0 to #_packet_constraint do
7       if |F1[i].packet_size - F2[j].packet_size| < threshold
8         result_LCS <- LASER (F1[i], F1[j])
9         LCS_Pool{} <- Append result_LCS, end if
10    j++, end while
11  i++, end while
12  S <- select the longest from LCS_Pool
13  while i from 0 to number of rest flows of Flow_Pool do
14    Fi <- select one from the rest of Flow_Pool
15    result_LCS <- LASER (S, Fi)
16    S <- select the longest from result_LCS
17  i++, end while
18  return S
19
```

```

20 procedure LASER (PacketA[1...m], PacketB[1...n])
21   PacketA [m...1] <- Reverse byte stream
22   PacketB [n...1] <- Reverse byte stream
23   Matrix [m][n]
24   while i from 0 to m do
25     while j from 0 to n do
26       if i = 0 or j = 0, then Matrix[i][j] = 0;
27       else if PacketA[i] = PacketB[j], then
28         Matrix [i][j] <- 'Diagonal',
29         Matrix [i][j] = Matrix [i-1][j-1] + 1;
30       else if Matrix[i-1][j] >= Matrix[i][j-1], then
31         Matrix[i][j] <- 'Up',
32         Matrix[i][j] = Matrix [i-1][j];
33       else
34         Matrix[i][j] <- 'Up',
35         Matrix[i][j] = Matrix [i][j-1];
36     end while
37   end while
38   i <- m-1; j <- n-1 /* Tracking */
39   while Matrix[i][j] != 0 do
40     if Matrix[i][j] = 'Left', then
41       j--
42     else if Matrix[i][j] = 'Up', then
43       i--
44     else if Matrix[i][j] = 'Diagonal', then do
45       Substring <- Append PacketA[i]
46       if Matrix[i-1][j-1] != 'Diagonal', then
47         Substring <- Append special break point character (e.g. '/')
48     i--; j--, end while
49   while tokenizing substring based on break point do
50     if token_length > minimum_substring_length_constraint
51       then, result_LCS <- Append token_substring
52   end while
53   return result_LCS

```

Поясним работу алгоритма. К моменту генерации сигнатуры у нас есть набор потоков, в которых содежится хотя бы по `#_packet_constraint` (параметр нашего алгоритма) пакетов (строка 2). Дальше из них произвольно выбираются два потока (строки 3-4). Затем происходит полный попарный перебор пакетов из двух потоков, и в случае, если разница размеров пакетов в паре меньше `threshold` (параметр нашего алгоритма), то вызывается функция `LASER` для этой пары пакетов (поиск наиболее длинной общей подпоследовательности), а результат работы заносится в пул (строки 5-11). Из этого пула выбирается самая длинная общая подпоследовательность (строка 12). 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 (строки 13-17). Когда потоки в пуле закончились, возвращаем полученный результат (строка 18).

Функция `LASER` представляет собой алгоритм Needleman-Wunsch, в котором используется матрица направлений. Так как пакеты развернуты (строки 21-22), то на самом деле матрица заполняется с конца (строки 23-37). Затем обходится в обратном порядке, по направлениям и собирается общая подпоследовательность (строки 38-48). Дальше из общей подпоследовательности, состоящая из токенов, выкидываются те, что короче `minimum_substring_length_constraint` (параметр нашего алгоритма) (строки 49-

53). И возвращаем полученный результат (строка 50).

Таким образом, у LASER 3 параметра настройки алгоритма:

- `#_packet_constraint` (количество пакетов в потоке)
- `threshold` (порог разницы размеров для сравнения пакетов)
- `minimum_substring_length_constraint` (минимальная длина подстроки)

### **3.4.2 AutoSig**

Следующий рассматриваемый метод называется AutoSig [2, 3].

## 4 Исследование и построение решения задачи

Здесь надо декомпозировать большую задачу из постановки на подзадачи и продолжать этот процесс, пока подзадачи не станут достаточно простыми, чтобы их можно было бы решить напрямую (например, поставив какой-то эксперимент или доказав теорему) или найти готовое решение.

## 5 Описание практической части

Если в рамках работы писался какой-то код, здесь должно быть его описание: выбранный язык и библиотеки и мотивы выбора, архитектура, схема функционирования, теоретическая сложность алгоритма, характеристики функционирования (скорость/память).

## 6 Заключение

Здесь надо перечислить все результаты, полученные в ходе работы. Из текста должно быть понятно, в какой мере решена поставленная задача.

## Список литературы

- [1] 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.
- [2] 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>.
- [3] 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.
- [4] Cheng MU, HUANG Xiao-hong, Xu TIAN et al. Automatic traffic signature extraction based on fixed bit offset algorithm for traffic classification // The Journal of China Universities of Posts and Telecommunications. — 2011. — Vol. 18. — P. 79–85.
- [5] 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.
- [6] Szabó Géza, Turányi Zoltán, Toka László et al. Automatic protocol signature generation framework for deep packet inspection // 5th International ICST Conference on Performance Evaluation Methodologies and Tools. — 2012.
- [7] Vinoth George C, Edwards V. Efficient regular expression signature generation for network traffic classification // International Journal of Science and Research (IJSR). — 2013.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] Tharp Justin, Suh Sang C, Cho Hyeonkoo, Kim Jinoh. Improving signature quality for network application identification // Digital Communications and Networks. — 2019. — Vol. 5, no. 3. — P. 139–146.
- [12] 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.
- [13] 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.

- [14] Boyer Robert S, Moore J Strother. A fast string searching algorithm // Communications of the ACM. — 1977. — Vol. 20, no. 10. — P. 762–772.
- [15] Xie Linqun, Liu Xiaoming, Yue Guangxue. Improved pattern matching algorithm of BMHS // 2010 Third International Symposium on Information Science and Engineering / IEEE. — 2010. — P. 616–619.
- [16] 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.
- [17] Internet Assigned Numbers Authority. — <https://www.iana.org>, дата обращения 25.01.2024.
- [18] Snort. — <https://www.snort.org>, дата обращения 25.01.2024.
- [19] Bittorent. — <https://www.bittorent.com>, дата обращения 25.01.2024.
- [20] Wireshark. — <https://www.wireshark.org>, дата обращения 25.01.2024.