­

Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра автоматизации систем вычислительных комплексов

Гудис Александр Вячеславович

**Исследование метода Отелло хеширования для доступа к таблицам классификации в архитектуре сетевого процессорного устройства**

КУРСОВАЯ РАБОТА

**Научный руководитель:**

к.ф-м.н, доцент

Волканов Дмитрий Юрьевич

Москва, 2024

**Аннотация**

В данной работе рассматривается сетевое процессорное устройство — ключевой элемент программируемых коммутаторов, отвечающий за обработку сетевого трафика. В процессе этой обработки каждый пакет данных проходит несколько этапов, одним из которых является классификация. Классификация пакета заключается в поиске соответствующего правила, определяющего, как следует изменить его заголовки и на какой порт направить. Эти правила хранятся в соответствующих таблицах классификации, выбор типа которых зависит от множества факторов. Поиск по таким таблицам является вычислительно трудной задачей, поэтому разработка более эффективных методов её выполнения остаётся актуальной. В работе предлагается использование метода хеширования данных для организации таблиц классификации типа MAC-VLAN. Рассматривается и изучается метод Отелло хеширования для представления такой таблицы. Сравнительные экспериментальные исследования с уже существующими классическими методами хеширования для доступа к таблицам классификации показали, что Отелло хеширование обладает

…………….

высокой эффективностью в рамках поставленной задачи, а также гибкостью настройки, что делает их применимыми в широком спектре сценариев.

Оглавление

[Введение 5](#_Toc194659511)

[1. Цель работы 6](#_Toc194659512)

[2. Предметная область 7](#_Toc194659513)

[2.1 Сетевое процессорное устройство 7](#_Toc194659514)

[2.2 Классификация пакетов 8](#_Toc194659515)

[2.3 Хеширование данных 9](#_Toc194659516)

[3. Постановка задачи 12](#_Toc194659517)

[3.1 Задачи таблицы классификации 12](#_Toc194659518)

[3.2 Задача представления таблицы классификации 13](#_Toc194659519)

[4. Обзор существующего решения 14](#_Toc194659520)

[4.1. Значимые критерии 14](#_Toc194659521)

[4.2. Отелло хеширование 14](#_Toc194659522)

[5. Исследование и решение задачи 16](#_Toc194659523)

[5.1. Основные понятия 16](#_Toc194659524)

[5.2. Алгоритм поиска 16](#_Toc194659525)

[5.3. Алгоритм построения структуры 16](#_Toc194659526)

[5.4. Алгоритм вставки 16](#_Toc194659527)

[5.5. Алгоритм удаления 16](#_Toc194659528)

[5.6. Группа параллельных Отелло 16](#_Toc194659529)

[6. Экспериментальное исследование 17](#_Toc194659530)

[6.1. Подраздел 17](#_Toc194659531)

[Заключение 18](#_Toc194659532)

[Литература 18](#_Toc194659533)

# Введение

В современном мире сетевые технологии играют важнейшую роль в обеспечении коммуникации и передачи данных. Развитие технологий и растущие потребности общества требуют более эффективных решений в области сетевых взаимодействий. Ключевыми устройствами, отвечающими за высокую пропускную способность и минимизацию задержек в компьютерных сетях, являются коммутаторы и маршрутизаторы. Они выполняют обработку сетевых пакетов, определяют их маршруты, управляют приоритетами передачи данных, выявляют и блокируют вредоносный трафик и выполняют другие важные функции.

В настоящее время активно разрабатываются программируемые коммутаторы, отличительной чертой которых является реализация обработки пакетов на уровне программного обеспечения, а не аппаратного. Их центральный компонент — сетевое процессорное устройство (СПУ) [1] — позволяет гибко настраивать логику обработки данных, обеспечивая при этом высокую производительность. Такие коммутаторы являются важной частью программно-конфигурируемых сетей (SDN), но также находят применение и в традиционных сетевых архитектурах.

Использование программной обработки сетевых пакетов в СПУ позволяет поддерживать широкий спектр протоколов и работать на различных уровнях модели OSI. Однако такая гибкость сопровождается увеличенными требованиями к памяти, необходимой для хранения алгоритмов обработки, а также снижением скорости обработки пакетов. Для решения этой проблемы требуется оптимизация структур данных, отвечающих за хранение правил маршрутизации и обработки.

Одним из ключевых методов оптимизации является хеширование данных, которое позволяет каждой хранимой единице назначить уникальный идентификатор, упрощающий и ускоряющий её поиск. Введение хеширования в процесс классификации пакетов может существенно сократить время его выполнения, приближая его к константному, однако это требует дополнительных затрат памяти.

Основная сложность хеширования связана с коллизиями, возникающими, когда разные данные получают одинаковый идентификатор. Чем больше выделенной памяти, тем ниже вероятность таких коллизий. Однако существуют различные методы хеширования, позволяющие минимизировать избыточное использование памяти без значительного увеличения времени поиска. В данной работе рассмотрен и изучен один из таких методов.

# Цель работы

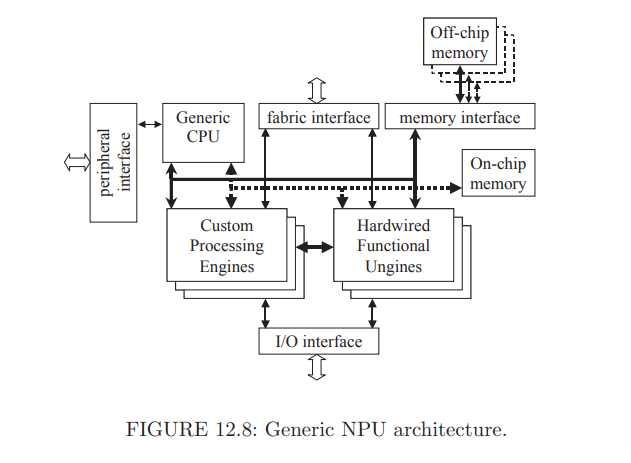
Цель данной работы — изучение метода Отелло хеширования и разработка подхода к представлению таблиц классификации на его основе.

Для достижения этой цели планируется:

1. Провести анализ и изучить теоретическую составляющую данного метода, предложенную авторами в их научной статье [7]
2. Разработать модель представления таблиц классификации с использованием этого метода хеширования;
3. Реализовать предложенную модель на практике;
4. Выполнить экспериментальный и сравнительный с другими классическими методами хеширования анализ эффективности.

# Предметная область

## 2.1 Сетевое процессорное устройство



Картинка из литературы на лето, страница 440. Не уверен в том, что она подходит

Сетевое процессорное устройство (СПУ) — это специализированный микропроцессор, предназначенный для обработки и управления сетевым трафиком в программируемых коммутаторах. Архитектура СПУ представляет собой баланс между высокой производительностью, характерной для узкоспециализированных решений, и гибкостью настройки, присущей процессорам общего назначения. Благодаря этому СПУ поддерживает широкий спектр сетевых стандартов и протоколов, обеспечивая при этом высокую пропускную способность.

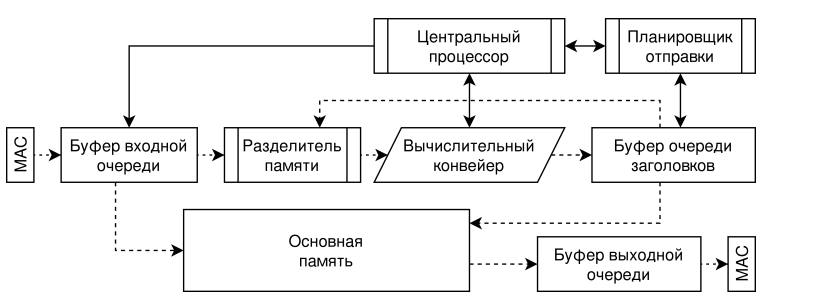


Рис. 1: Пример архитектуры СПУ

Как правило, обработка трафика в СПУ осуществляется с использованием конвейера вычислительных ядер. Такой подход позволяет разбить процесс обработки пакетов на последовательные этапы, что значительно ускоряет обработку данных. Однако задержка на любой из стадий конвейера может повлиять на общую производительность системы. Обработка сетевого пакета в СПУ включает следующие этапы:

1. Прием пакета с входного порта;
2. Извлечение заголовка пакета и временное сохранение его содержимого во внутренней памяти;
3. Классификация пакета, заключающаяся в поиске соответствующего правила в таблице классификации;
4. Модификация заголовка в соответствии с найденным правилом;
5. Объединение измененного заголовка с телом пакета;
6. Передача пакета в систему управления очередями;
7. Отправка пакета на выходной порт согласно определенному правилу.

Третий шаг, классификация пакета - является одним из самых ресурсоемких этапов. Он представляет собой поиск по соответствующей таблице классификации, содержащей правила обработки пакетов.

## 2.2 Классификация пакетов

Таблица классификации представляет собой структуру данных, содержащую информацию о способе обработки пакетов определенного типа. Записи в таблице, называемые правилами, могут быть упорядочены по приоритету. Каждое правило включает в себя несколько полей: ключ, определяющий условия сопоставления, и значение, задающее соответствующие действия.

Ключ правила формируется на основе определенных частей заголовков пакета, таких как исходный и целевой адреса, порты, тип протокола и другие параметры. В зависимости от задачи ключ может быть задан как точное значение (константа) или как шаблон (wild-card), например, префикс IP-адреса. Поиск по таблице классификации может выполняться либо по полному соответствию, либо по частичному, например, по наиболее длинному совпадающему префиксу.

Значение правила играет разную роль: оно может представлять собой константу, используемую при обработке пакета, либо ссылку на исполняемый код, предназначенный для обработки пакетов данного типа. Такой подход особенно распространен в таблицах потоков (Flow table).

В данной работе рассматриваются таблицы классификации типа «MAC-VLAN», работающие на втором (канальном) уровне модели ISO-OSI.

|  |  |
| --- | --- |
| MAC-VLAN table | |
| MAC-VLAN | port |
| xx:xx:xx:xx:xx:xx-xxx  (x – 4 бита) | 1 |
| … | … |
| xx:xx:xx:xx:xx:xx-xxx | n |

Таблица 1: Таблица классификации «MAC-VLAN»

Они позволяют сопоставлять MAC-адреса с виртуальными сетями VLAN, обеспечивая эффективную маршрутизацию трафика.

Правила в таких таблицах имеют следующий формат:  
<{MAC-addr, VLAN-id}, port-mask>, где:

* {MAC-addr, VLAN-id} — ключ, представляющий собой 60-битную строку, полученную путем объединения MAC-адреса и идентификатора VLAN;
* port-mask — значение, представленное 8-битной строкой, указывающей, на какие порты следует направить пакет после обработки.

Поиск в таблице классификации MAC-VLAN осуществляется по точному совпадению ключа.

## 2.3 Хеширование данных

Хеш-таблица [2; 3] представляет собой структуру данных, организованную в виде ассоциативного массива, который обеспечивает эффективное хранение и извлечение пар ключ-значение. Простейший вариант такой таблицы показан на рисунке 2. На этапе инициализации массив заполняется специальными значениями, обозначающими пустые ячейки (NULL).

Основой работы хеш-таблицы являются одна или несколько хеш-функций, преобразующих множество ключей в множество хеш-кодов. Хеш-код представляет собой индекс массива, по которому можно получить доступ к соответствующей записи.

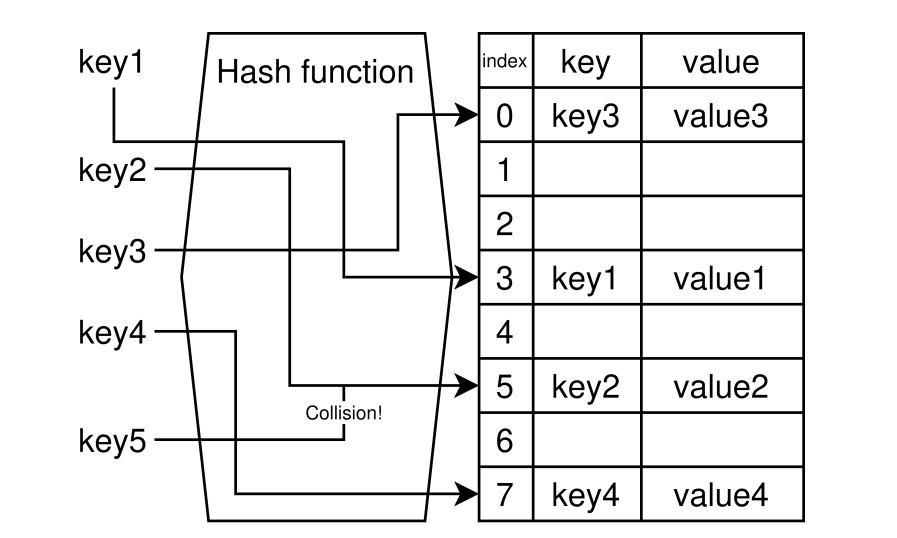


Рис. 2: Пример хеш-таблицы



?????

Для использования в структурах данных хеш-функции должны обладать рядом свойств:

1. Фиксированная длина выходных данных – все хеш-коды находятся в заранее заданном диапазоне;
2. Сюрьективность – каждому ключу соответствует определенный хеш-код, который не обязательно является уникальным;
3. Равномерное распределение – хеш-коды должны быть распределены как можно более равномерно по множеству ключей;
4. Лавинный эффект – небольшое изменение входных данных должно приводить к значительному изменению хеш-кода, что обеспечивает учет каждого бита ключа.

В теории хеш-функции позволяют получать доступ к данным за константное время, не зависящее от объема хранимой информации. Если удается подобрать хеш-функцию, обеспечивающую биективное отображение ключей в хеш-коды, такое хеширование называют идеальным. Однако при расширении множества ключей идеальное хеширование, как правило, теряет свои преимущества.

Одна из ключевых проблем при работе с хеш-таблицами — коллизии, возникающие, когда разные ключи получают одинаковый хеш-код. В таких ситуациях необходимо применять методы разрешения коллизий, чтобы корректно сохранить и извлечь данные. Наиболее распространенный способ — использование цепочек: массив ячеек хранит ссылки на связанные списки, в которых размещаются записи с одинаковым хеш-кодом, однако со временем такая структура данных может деградировать и перестать обладать стандартными преимуществами хеш-таблиц и станет более похожа на классический список.

Для минимизации коллизий важным свойством хеш-функции является равномерность распределения хеш-кодов, что повышает эффективность и отказоустойчивость хеш-таблицы. Однако в некоторых случаях коллизии могут накапливаться до такой степени, что добавление новых записей становится невозможным. В таких ситуациях применяют перехеширование — смену хеш-функции и пересборку структуры.

Дополнительный инструмент для улучшения распределения хеш-кодов — соль (salt), представляющая собой дополнительный константный параметр хеш-функции. Использование соли увеличивает количество возможных вариантов хеширования и повышает вероятность выбора оптимальной функции для текущего множества ключей.

# Постановка задачи

## 3.1 Задачи таблицы классификации

Таблица классификации MAC-VLAN представляет собой структурированный набор правил, организованный в соответствии с определенным методом представления. Этот метод решает три основные задачи: поиск правила по заданному ключу, добавление новой записи и удаление существующей. В рамках описания данных задач используются следующие обозначения:

Набор правил:

*,*

Где

Правило представляет собой совокупность ключа (key) и значения (value), либо является пустым NULL и в классификации не участвует. Ключ является битовым значением заданной длины , значение также является полем определенной длины и соответствует порту в таблице классификации.

**Задача построения структуры данных**

**Дано:**

Набор правил таблицы классификации MAC-VLAN

**Требуется:**

Построить структуру данных Отелло для последующих операций поиска в ней

**Задача поиска правила**

**Дано:**

Ключ

**Требуется:**

Найти правило вида , если оно представлено в исходной таблице, в противном случае вернуть NULL

**Задача вставки правила**

**Дано:**

Правило , ранее не присутствовавшее в таблице

**Требуется:**

Получить такую таблицу , что и для любого . Новая таблица содержит добавленное правило и доступна его операция поиска

**Задача удаления правила**

**Дано:**

Ключ

**Требуется:**

Получить такую таблицу , что и для любого . В новой таблице будет отсутствовать удалённое правило и его поиск возвращает результат NULL.

## 3.2 Задача представления таблицы классификации

**Дано:**

Набор констант:

где S – операция поиска, I – операция вставки, D – операция удаления

**Требуется:**

Разработать способ представления таблицы классификации , такой чтобы для любого

*,*

# **Обзор существующего решения**

## 4.1. Значимые критерии

При выборе основного метода для его исследования в качестве курсовой работы были учтены следующие параметры:

* количество обращений к памяти во время процедуры поиска;
* количество вызовов хеш-функций во время процедуры поиска;
* количество обращений к памяти во время процедуры вставки;
* количество вызовов хеш-функций во время процедуры вставки;
* количество обращений к памяти во время процедуры удаления;
* количество вызовов хеш-функций во время процедуры удаления;
* ожидаемая максимально допустимая плотность записей.

## 4.2. Отелло хеширование

Отелло-хеширование[9] в своей классической реализации без дополнительных улучшений алгоритма является структурой данных, которая позволяет разделить множество ключей или правил на два подмножества, опираясь при этом на использование двудольного графа, двух битовых массивов и пары различных хеш-функций. Размер этих массивов, как и диапазон значений, возвращаемых хеш-функциями, соответствует размеру долей графа. Каждый ключ отображается на ребро графа, при этом его вершины определяются значениями хеш-функций, вычисленных от ключа. Важно, чтобы граф оставался ацикличным. Доли графа связаны с соответствующими массивами, а каждая вершина, обладающая определенным цветом в соответствии с алгоритмом раскраски, отображается на конкретную ячейку своего массива.

Для задачи классификации сетевых пакетов Отелло хеш используется как базовая структура данных при реализации алгоритма POG «Parallel Othello Group» или же «группа параллельно работающих Отелло». При этом рассматривается ПКС (программно конфигурируемая сеть), состоящая из коммутаторов и единого контроллера, отвечающего за принятие решения о маршрутизации путей пакетов. В результате Отелло хеширование работает в двух различных плоскостях: плоскость данных и плоскость управления.

* **Структура запроса Othello (**Othello query structure**)**, реализованная в коммутаторе. Выполняет только запросы имен. Выделяется своей эффективностью по используемой памяти, времени выполнения запроса и полным отсутствием операций обновления.
* **Структура управления Othello (**Othello control structure**)**, реализованная в контроллере. Используется для операций добавления, удаления, изменения элементов структуры Отелло и рассылки этой информации на маршрутизаторы.

В Отелло хешировании принадлежность ключа к определённому подмножеству определяется с помощью побитового сложения (XOR) значений в соответствующих ячейках массивов. Повторное условно параллельное применение данного подхода позволяет разделить множество ключей на различных групп.

При вставке нового ключа сначала вычисляются значения хеш-функций, определяющие вершины графа и адреса ячеек в массивах. Затем проверяется, приведёт ли добавление нового ребра между этими вершинами к образованию цикла. Если цикл не возникает, ребро добавляется в граф. Далее, для корректного распределения и сохранения логики разбиения, изменяются значения битов в массивах вдоль достижимых путей от новых вершин. Такая коррекция возможна именно при отсутствии циклов.

При удалении достаточно просто убрать ребро из графа — при этом изменение битов в массивах не требуется.

Основным преимуществом Отелло хеширования является использование небольшого объема занимаемой памяти при хранении битовых массивов для эффективной работы в задаче классификации, а также фиксированное число вызовов хеш-функций и битовых операций XOR.

# **Исследование и решение задачи**

## 5.1. Основные понятия

Из постановки задачи известно, что множество всех имен (правил) в таблице классификации есть *,* пусть *.*

Отелло классифицирует *n* имен в два непересекающихся множества:

Отелло можно расширить для классификации имен в d (d > 2)непересекающихся множеств. Подобная модификация структуры данных будет рассмотрена в разделе 5.6

Как структура данных Отелло представляет собой кортеж из 7 значений, то есть каждое из которых определяется следующим образом:

* Целые числа *ma* и *mb*, описывающие размер Отелло, то есть битовых массивов, использующихся для классификации;
* Пара равномерных случайных хеш-функций *ha, hb*, сопоставляющих имена с целыми значениями *{0,1,··· ,ma−1}* и *{0,1,··· ,mb−1}* соответственно. Выбор хеш-функций является принципиально важным фактором для успешной работы структуры поиска;
* Битовые массивы *a* и *b*. Длины равны *ma* и *mb* соответственно;
* Двудольный граф G. Во время построения и обновления Отелло G используется для определения значений в a и b.

Возможны различные варианты указания длин массивов *a* и *b*. Требуется, чтобы , наилучшим вариантом является . Данное предположение будет проверено опытным путём

## 5.2. Алгоритм поиска

Для ключа *k* по соответствующему алгоритму вычисляется значение При . Если . Если Значения битовых массивов определяются во время построения Отелло, так что можно вычислить следующим образом:

## 5.3. Алгоритм построения структуры

**Операция построения. Сonstruct.**

**Сложность:** О(n)

**Дано:**

Два набора имён , находящиеся в таблице классификации .

**Требуется:**

Построить структуру Отелло содержащую ациклический граф.

Граф G используется для определения пары хеш-функций и значений массивов .

.

Вершина , , что соответствует битам Существует ребро тогда и только тогда, когда такое что и , где – множество всех ключей ТК.

Для каждой вершины, которая связана хотя бы с одним ребром, соответствующий бит в массиве устанавливается в 0 или 1. Вершина, связанная с битом 0, окрашена в белый цвет, а вершина, связанная с битом 1, окрашена в черный цвет. Для вершин, которые не имеют связанных ребер, значение соответствующих битов может быть установлено в 0 или 1 произвольно, поскольку они не влияют ни на какое значение .

**Фаза I: Выбор пары хэш-функций.**

На этой фазе Отелло находит пару хэш-функций < На каждой попытке построения случайным образом выбираются две хеш-функции и создаётся граф G. Далее используется поиск в глубину (DFS) на G, чтобы проверить, содержит ли он цикл. Если два или более имени генерируют ребра с одинаковыми двумя конечными точками, будем считать, что есть цикл. Если G является циклическим, алгоритм будет выбирать другую пару хеш-функций до тех пор, пока не будет найден ациклический G.

**Фаза II: Вычисление битовых массивов.**

На этой фазе устанавливаются a и b.Сначала значения в a и b помечаются как неопределенные, что соответствует серому цвету вершин графа. Затем для каждого ребра в порядке DFS обхода:

* Если ни одно из значений не было установлено, тогда ;
* Если назначено только одно из , всегда можно назначить другое значение так, чтобы;
* Поскольку G ацикличен, следуя порядку DFS,мы никогда не увидим такое ребро, что и a[i], и b[j] имеютустановленныезначения.­

## 5.4. Алгоритм вставки

**Операция вставки. Insert.**

**Сложность:** амортизированная О(1).

**Дано:**

Структура Отелло , добавляемое правило с ключом и принадлежностью к множеству X или Y.

**Требуется:**

Добавить ключ в Отелло так, чтобы стала доступна его операция поиска, возвращающая корректный результат.

Сначала алгоритм вычисляет ребро которое должно быть добавлено к G для , где При этом G можно разложить на компоненты связности, от расположения которых относительно добавляемого ребра возможны различные случаи вставки.

**Случай 1**

В битовых массивах есть неустановленные индексы, что соответствует серым вершинам в графе. Предположим, что в добавляемом ребре одна из вершин уже существует и покрашена, а другая нет. В таком случае действия эквиваленты алгоритму построения и значение пустой вершины будет установлено так, чтобы ребро соответствовало добавляемого правила.

**Случай 2**

принадлежат одному и тому же связному компоненту. Добавление ребра к графу может привести к появлению цикла. Необходимо повторно выбрать пару хеш-функций пока не будет найдена допустимая пара хеш-функций для нового набора имен , то есть заново применить операцию построения.

**Случай 3**

находятся в двух разных связных компонентах. Объединяя два связных компонента и новое ребро, мы получаем один связный компонент, который по-прежнему ацикличен, однако возможны подслучаи.

**Случай 3.1**

Добавляемое ребро удовлетворяет значению и перекраска/перестройка компоненты связности не требуется, так как можно однозначным образом классифицировать правило при операции поиска.

**Случай 3.1**

Если добавляемое ребро не удовлетворяет раскраске какой-либо компоненты связности, то необходимо обойти её в порядке DFS и произвести перекраску вершин. Утверждается, что хотя бы одна из двух компонент связности точно сохраняет свою раскраску

## 5.5. Алгоритм удаления

## 5.6. Группа параллельных Отелло

# **Экспериментальное исследование**

**<…>**

# Подраздел

**<…>**

# Заключение

**<…>**

# Литература

[1] Об одном подходе к построению сетевого процессорного устройства / С. О. Беззубцев [и др.] // Моделирование и анализ информационных систем. — 2019. — Т. 26, № 1. — С. 39—62.

[1] Dong, C., Wang, F, Feng, D., 2021. Dxhash: A scalable consistent hash based on the pseudo-random sequence. arXiv preprint arXiv:2107.07930.

[2] Maurer W. D., Lewis T. G. Hash table methods // ACM Computing Surveys (CSUR). — 1975. — Т. 7, № 1. — С. 5—19.

[3] A survey on learning to hash / J. Wang [и др.] // IEEE transactions on pattern analysis and machine intelligence. — 2017. — Т. 40, № 4. — С. 769—790.

[2] Ahmadi, M. and Wong, S., 2007, January. Hashing functions performance in packet classification. In Proceedings of the International Conference on the Latest Advances in Networks (ICLAN’07) (pp. 127-132).

[3] Dharmapurikar, S., Krishnamurthy, P. and Taylor, D.E., 2003, August. Longest prefix matching using bloom filters. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (pp. 201-212).

[4] Charles, D. and Chellapilla, K., 2008, September. Bloomier filters: A second look. In European Symposium on Algorithms (pp. 259-270). Berlin, Heidelberg: Springer Berlin Heidelberg.

[5] Shi, S., Qian, C., 2020. Ludo hashing: Compact, fast, and dynamic keyvalue lookups for practical network systems. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 4(2), pp.1-32.

[6] Pagh, R., Rodler, F.F., 2001, August. Cuckoo hashing. In European Symposium on Algorithms (pp. 121-133). Berlin, Heidelberg: Springer Berlin Heidelberg.

[7] Yu, Y., Belazzougui, D., Qian, C., Zhang, Q., 2018. Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing. IEEE/ACM Transactions on Networking, 26(3), pp.1151-1164.

[8] Александров А.В., Волканов Д.Ю. Модифицированный метод хеширования данных для доступа к таблице классификации MACVLAN в сетевом процессорном устройстве // Программные системы и инструменты. Тематический сборник № 24. Т. 24. – М.: ООО «МАКС Пресс», 2024. – С. 6–16. DOI: 10.29003/m4299.978-5-317-07288-9

[9] Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing / Y. Yu [и др.] // IEEE/ACM Transactions on Networking. — 2018. — Т. 26, № 3. — С. 1151— 1164.