МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В. ЛОМОНОСОВА
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ
КАФЕДРА АВТОМАТИЗАЦИИ СИСТЕМ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Александров Алексей Владимирович

Методы хеширования данных для доступа к таблицам классификации в сетевом процессорном устройстве

ДИССЕРТАЦИЯ

Научный руководитель:
Доцент, к.ф.-м.н. Волканов Дмитрий Юрьевич

Москва, 2023

Аннотация

В данной работе рассматривается сетевое процессорное устройство, особый компонент программируемых коммутаторов, которые выполняют обработку сетевого трафика. В ходе данной обработки каждый сетевой пакет проходит несколько этапов, одним из которых является классификация.

Классификация пакета представляет поиск подходящего правила, которое определяет, как необходимо модифицировать заголовки пакета и на какой порт его отправлять. Правила хранятся в таблицах классификации, тип которых зависит от многих факторов. Поиск по таблице является трудоемкой задачей, и нахождение способов более эффективного ее решения является актуальным на сегодняшний день.

В работе предлагается применение методов хеширования данных в таблицах классификации типа MAC-VLAN. В работе описано три способа представления таблиц классификации, использующих принципы нескольких методов хеширования данных, что позволяет им компенсировать недостатки друг друга.

Экспериментальное исследование выявило, что описанные способы обладают высокой эффективностью в рамках поставленной задачи, а так же гибкостью конфигурации, позволяющей использовать данные решения в широком ряде случаев.

Оглавление

Введение

В современном обществе сетевые технологии играют ключевую роль в обеспечении потребностей сетевой связи и передачи данных. Человеческий прогресс и развитие новых технологий создали востребованность в более эффективных сетевых коммуникациях. Центральными устройствами в компьютерных сетях, обеспечивающими высокую пропускную способность и минимальное время задержки, являются коммутаторы и маршрутизаторы. Данные устройства выполняют обработку сетевых пакетов, которые направляют их по необходимой точке назначения, разделяют их по приоритету, выделяют и ликвидируют вредоносные пакеты и прочее.

В настоящее время ведется разработка программируемого коммутатора, Их ключевой особенностью является механизм обработки пакетов, реализованный на уровне программного обеспечения вместо аппаратного. Центральный модуль программируемого коммутатора, сетевое процессорное устройство (СПУ) [1], позволяет гибко настраивать поведение узла в сети, обеспечивая при этом высокую производительность. Программируемые коммутаторы являются неотъемлемой частью программно-конфигурируемых сетей, новой вехи сетевых технологий, однако находят применение и в традиционных сетях.

Программная обработка сетевых пакетов в СПУ позволяет покрыть широкий спектр поддерживаемых протоколов, а так же работать на разных уровнях сетевой модели OSI. Цена за подобную универсальность — повышенные требования к памяти, в вмещающий заданный алгоритм, а так же понижение скорости обработки. Нивелировать данную проблему можно за счет усовершенствования структур данных, хранящих правила обработки различных пакетов.

Хеширование данных — это метод, присваивающий каждой хранимой единице данных свой идентификатор, по которому можно найти место в памяти, где эти данные хранятся. Хеширование позволит упростить и ускорить процесс классификации пакета, являющийся одним из самых затратных этапов обработки. Его внедрение в этап классификации позволит достигнуть выполнения этого этапа за константное время, однако оно стоит еще больших накладных расходов на память.

Проблема хеширования заключается в коллизиях, событиях, когда у разных единиц данных оказывается один идентификатор. Чем больше неиспользуемой памяти, тем ниже вероятность коллизии. Однако существуют различные методы хеширования, позволяющие минимизировать накладные расходы на память без существенного изменения времени поиска. В данной работе рассматриваются подобные методы хеширования.

Структура работы. В главе 1 описываются цели работы. В главе 2 находится описание предметной области, в рамках которой проводится исследование. В главе 3 математически описаны исследуемые объекты и сформулирована математическая постановка задачи. В главе 4 находится обзор существующих методов решения поставленной задачи. В главе 5 предлагаются способы решения задачи, основанные на результатах обзора. В главе 6 представлено описание имитационной модели, использовавшейся в исследовании. В главе 7 описан ход экспериментального исследования, его результаты и следующие выводы.

Глава 1

Цель работы

Целью работы является исследование различных методов хеширования данных и разработка способов представления таблиц классификации на их основе.

Для проведения исследования планируется:

1) Выполнить обзор методов хеширования и выбрать методы для дальнейшего исследования;

2) Разработать способы представления таблиц классификации на основе выбранных методов хеширования;

3) Реализовать предложенные способы;

4) Провести экспериментальное исследование предложенных способов;

Глава 2

Предметная область

2.1 Сетевое процессорное устройство

Сетевое процессорное устройство — это специализированный микропроцессорный элемент, который предназначен для обработки и управления сетевым трафиком в программируемых коммутаторах. Пример архитектуры СПУ представлен на рисунке 1. Они представляют собой компромисс высокой производительности, которой отличаются устройства с узкоспециализированной архитектурой, а так же гибкостью настройки, присущей устройствам на базе ядер общего назначения. Особая архитектура СПУ позволяет поддерживать различные стандарты и протоколы, при этом обеспечивая высокую пропускную способность.

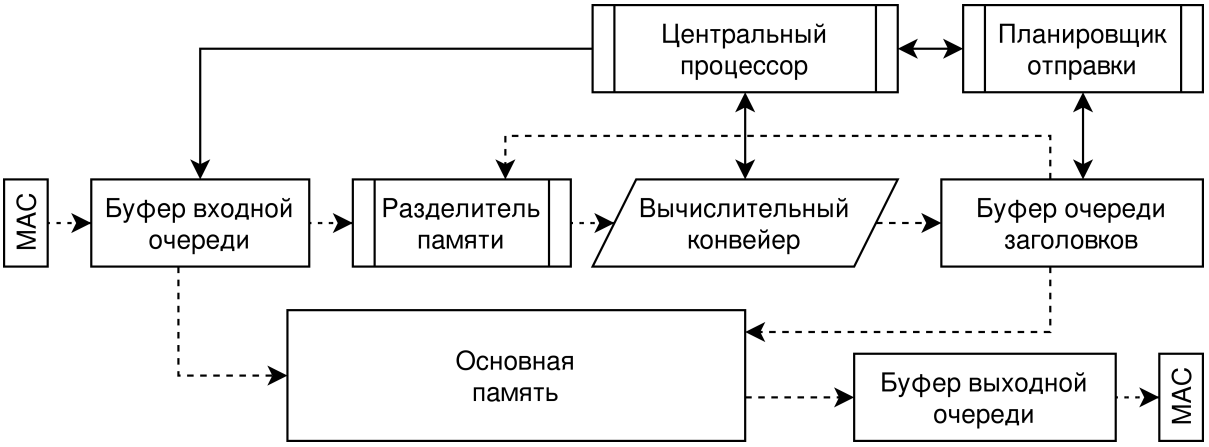

Рис. 1: Пример архитектуры СПУ

Как правило, обработка трафика на СПУ занимается конвейер вычислительных ядер. Данное архитектурное решение позволяет разделить программу обработки пакетов на стадии, что существенно ускоряет обработку трафика. Однако критичным становится время пребывания пакета на каждой стадии: задержка на одной из них приводит к задержке на всем конвейере. Обработка сетевого пакета разделяется на следующие этапы:

1) принятие пакета с входных портов;

2) отделение заголовка пакета от тела, сохранение последнего во внутренней памяти;

3) классификация пакета, представляющая собой поиск правила по таблице классификации;

4) модификация заголовка согласно найденному правилу;

5) объединение модифицированного заголовка с телом пакета;

6) передача пакета на устройство управления очередями;

7) отправка пакета на порт согласно найденному правилу.

Классификация пакета является одним из самых ресурсоемких этапов. Он представляет собой поиск по таблице классификации, содержащей правила обработки пакетов.

2.2 Классификация пакетов

Таблица классификации — это структура данных, хранящая информацию о способе обработки пакета соответствующего типа. Записи в таблице называются правилами, они могут быть упорядочены по приоритету. Каждое правило состоит из нескольких полей, включая поля для сопоставления условий, именуемые ключом, и поля для определения соответствующих действий, именуемые значением.

Ключ правила включает в себя определенные части заголовков пакета, такие как исходный и целевой адреса, а так же порты, тип протокола и другие атрибуты. В общем случае ключ может представлять собой либо константу, либо подставляемое значение (`wild-card`), как, например, префикс IP-адреса. Поиск правила также разделен на поиск по полному соответствию и поиск по частичному соответствию, например, по длиннейшему префиксу.

Значение правила может иметь различные роли. Может быть как константой, применяемой при дальнейшей обработке пакета, так и указателем на начало соответствующего кода, обрабатывающего данный тип пакетов. Последний вид значений правила применяется в таблицах потоков (Flow table).

В данной работе рассматриваются таблицы классификации типа «MAC-VLAN». Данная таблице работает на втором (канальном) уровне системы ISO-OSI. Использование данной таблицы помогает «привязать» MAC-адреса к определенным виртуальным сетям VLAN. Правила данной таблицы имеют следующий вид: `<{MAC-addr, VLAN-id}, port-mask>`, где `{MAC-addr, VLAN-id}` — ключ правила, битовая строка длинной 60 символов, являющаяся конкатенацией полей MAC-адреса и идентификатора VLAN; `port-mask` — значение правила, битовая строка длинной 8 символов, представляющая маску портов, на которые должен быть сетевой пакет после модификации. Поиск в таблице осуществляется по полному соответствию с ключом.

2.3 Хеширование данных

Хеш-таблица [2; 3] представляет собой ассоциативный массив данных, который позволяет эффективно хранить и извлекать пары ключ-значение. Пример простейшей хеш-таблицы изображен на рисунке 2. После инициализации структуры, массив заполнен специальными значениями — пустыми значениями, именуемыми `NULL`.

Хеш-таблица основана на одной или нескольких хеш-функциях, отображениях множества ключа в множество хеш-кодов. Хеш-кодом называется индекс, по которому доступна ячейка используемого массива, где сохранена соответствующая запись.

Свойствами хеш-функций, важными для использования в структурах данных, являются:

1) фиксированная длина выходных данных, т.е. все хеш-коды находятся в некотором диапазоне;

2) сюръективность — каждому ключу соответствует некоторый не обязательно уникальный хеш-код;

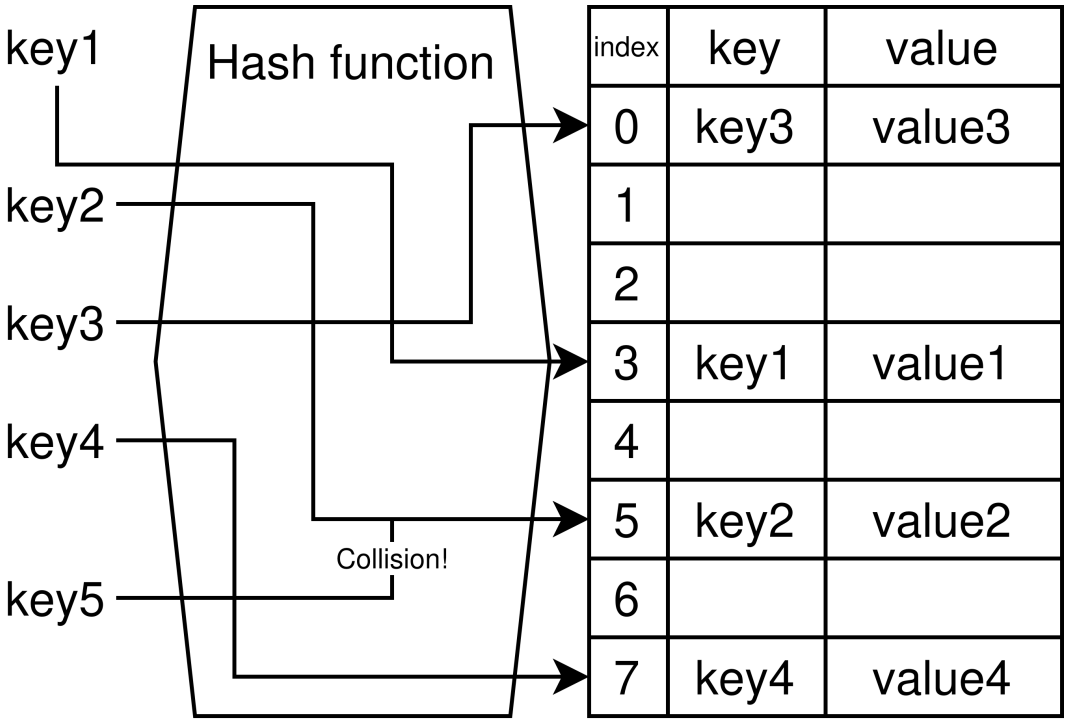

Рис. 2: Пример хеш-таблицы

3) равномерность распределения хеш-кодов для любой выборки ключей;

4) лавинный эффект — при небольшом изменении ключа значительное изменение хеш-кода, каждый бит ключа важен.

Теоретически, хеш-функции позволяют получить доступ к данным за константное относительно размеров области памяти время. При правильном подборе функции для конкретного множества ключей можно достигнуть биективности, такое хеширование называется идеальным. Но при расширении данного множества идеальное хеширование в общем случае теряет свои свойства.

При использовании динамически изменяемых множеств ключей возникает проблема коллизий, событий, когда разные ключи имеют одинаковый хеш-код. В таком случае необходимо принять меры для разрешения коллизии и корректно сохранить все значения. Для решения данной проблемы существуют различные подходы к заполнению массива записей хеш-таблицы. Самой распространенной практикой использования является применение односвязных списков в качестве записей массива, а уже сами записи вместе с ключами хранятся в ячейках списка.

Основным полезным свойством хеш-функции, вновь из-за коллизий, является равномерность распределения хеш-кодов. Такое свойство минимизирует коллизии, а значит, повышает эффективность и отказоустойчивость структуры.

В самых плачевных ситуациях возможен случай, при котором добавление дополнительной записи из-за коллизий невозможно. Тогда обычно меняют хеш-функцию, пересобирая структуру. В таких ситуациях помогает соль — дополнительный константный операнд хеш-функции. Ее применение значительно увеличивает набор возможных хеш-функций, а значит и шанс найти более подходящую для текущей выборки ключей.

Глава 3

Постановка задачи

3.1 Задачи таблицы классификации

Таблица классификации MAC-VLAN представлена упорядоченным набором правил и способом представления. Способ представления решает три задачи: поиск правила по ключу, вставку записи и удаление записи. В описании задач используются следующие обозначения: Набор правил:

$$\mathcal{R} = \langle r_0, r_1, ..., r_{n-1} \rangle,$$

где

$$\mathcal{R}[i] = r_i := \langle k_i, v_i \rangle \,|\, \text{NULL};$$

$$k_i \in \{0,1\}^{s_k}; \quad v_i \in \{0,1\}^{s_v}; \quad \text{NULL} \in \{0,1\}^{s_k+s_v}.$$

Задача поиска правила

Дано:
Ключ k^*
Требуется:
Найти правило $r = \langle k^*, v \rangle \in \mathcal{R}$ при наличии такого правила

Задача вставки правила

Дано:
Ключ k^*, значение v^*
Требуется:
Вернуть $\mathcal{R}^*$, где $\exists\, j:\ \mathcal{R}[j] = \langle k^*, v \rangle \,|\, \text{NULL}; \quad \mathcal{R}^*[j] = \langle k^*, v^* \rangle$, и
$\forall\, r \in \mathcal{R} \backslash \{\mathcal{R}[j], \text{NULL}\}:\ r \in \mathcal{R}^*$

Задача удаления правила

Дано:
Ключ k^*
Требуется:
Вернуть $\mathcal{R}^*$, где $\exists\, j:\ \mathcal{R}[j] = \langle k^*, v \rangle \,|\, \text{NULL}; \quad \mathcal{R}^*[j] = \text{NULL}$, и
$\forall\, r \in \mathcal{R} \backslash \{\mathcal{R}[j], \text{NULL}\}:\ r \in \mathcal{R}^*$

3.2 Задача представления таблицы классификации

Дано:

Набор констант:

$$\begin{aligned} M_{\mathbf{S}},\, M_{\mathbf{I}},\, M_{\mathbf{D}} \in \mathbb{N} &\quad \text{— ограничения на обращения к памяти;} \\ H_{\mathbf{S}},\, H_{\mathbf{I}},\, H_{\mathbf{D}} \in \mathbb{N} &\quad \text{— ограничения на вызовы хеш-функций;} \\ N \in \mathbb{N} &\quad \text{— максимальное количество правил;} \\ F \in (0,1) &\quad \text{— максимальная плотность ненулевых правил.} \end{aligned}$$

Требуется:

Разработать способ представления ТК $\mathcal{T} = \langle \mathbf{S}, \mathbf{I}, \mathbf{D} \rangle$, такой чтобы

$$\forall\, n < N, \quad \forall\, \mathcal{R} = \langle r_0, r_1, ..., r_{n-1} \rangle, \quad \frac{\text{rules}(\mathcal{R})}{n} \leq F :$$

$$\begin{gathered} \text{memAcc}(\mathbf{S}) \leq M_{\mathbf{S}}, \quad \text{memAcc}(\mathbf{I}) \leq M_{\mathbf{I}}, \quad \text{memAcc}(\mathbf{D}) \leq M_{\mathbf{D}}, \\ \text{hashCall}(\mathbf{S}) \leq H_{\mathbf{S}}, \quad \text{hashCall}(\mathbf{I}) \leq H_{\mathbf{I}}, \quad \text{hashCall}(\mathbf{D}) \leq H_{\mathbf{D}}. \end{gathered}$$

Глава 4

Обзор методов хеширования

4.1 Критерии обзора

В данном обзоре сравниваются методы хеширования по следующим пунктам:

- 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.

Для всех критериев, кроме последнего, считается асимптотическая оценка в среднем и худшем случае от параметров метода.

4.2 Описание алгоритмов

4.2.1 Линейное зондирование

Линейное зондирование [4] является одной из самых простых методов разрешения коллизий. Метод использует массив записей, инициализированный пустыми значениями, и одну хеш-функцию. Записи располагаются в массиве таким образом, что среди ячеек, сдвиг которых от вычисленного хеш-кода подчиняется линейной зависимости, нет пустых вплоть до индекса расположения искомой записи.

Во время вставки и поиска записи вычисляется хеш-код ключа. Если ячейка по соответствующему индексу пуста или содержит искомый ключ, выполняются необходимые действия, иначе проверяются ячейки ниже по массиву. При достижении конца массива проверка ячеек продолжается с его конца, массив замкнут сам на себя.

Алгоритм удаления должен происходить таким образом, чтобы удаляемая запись не создавала пустой ячейки между вычисленным хеш-кодом и индексом расположения для все оставшихся записей.

Метод хорош тем, что обладает легко реализуемыми алгоритмами вставки и поиска, однако удаление наоборот, усложнено требованиями структуры используемого массива. Также он страдает от кластеризации записей — локализованных скоплений.

4.2.2 Квадратичное зондирование

Квадратичное зондирование [5] призвано разрешить проблему кластеризации, возникающую в его линейном аналоге. Метод использует массив записей, инициализированный пустыми значениями, и одну хеш-функцию. Записи располагаются в массиве таким образом, что среди ячеек, сдвиг которых от вычисленного хеш-кода подчиняется квадратичной зависимости, нет пустых вплоть до индекса расположения искомой записи.

Во время вставки и поиска записи вычисляется хеш-код ключа. Если ячейка по соответствующему индексу пуста или содержит искомый ключ, выполняются необходимые действия, иначе проверяются ячейки по сдвигу 1, затем 4, 9 и так далее согласно квадратичному закону. При достижении конца массива проверка ячеек продолжается с его конца, массив замкнут сам на себя.

Такая структура массива помогает избежать кластеризации, однако цена за это — практическая невозможность удаления ячеек. Выполнение корректного удаления из-за разного шага во время вставки записей представляется крайне сложной задачей, требующей перестройки всего массива в общем случае.

4.2.3 Кукушкин хеш

Кукушкин хеш [6] разрешает коллизию путем добавления альтернативного места в памяти для каждой записи. Он использует массив записей, инициализированный пустыми значениями, и две хеш-функции. Записи располагаются в массиве согласно одному из двух допустимых хеш-кодов.

Во время вставки добавляемая запись всегда размещается по первому хеш-коду. Если эта ячейка была не пуста, то располагаемая в ней запись перемещается согласно своему второму хеш-коду. При повторной коллизии процесс повторяется до тех пор, пока не будет встречена пустая ячейка.

Во время поиска и удаления поочередно проверяются оба хеш-кода, затем выполняется необходимая операция.

Наличием двух хеш-функций и применением алгоритма достигается повышение максимальной плотности записей в массиве при небольшом усложнении алгоритмов.

4.2.4 Универсальный хеш

Универсальный хеш [7] расширяет идею использования множества хеш-функций. Он использует массив записей, инициализированный пустыми значениями, и несколько хеш-функций, количество которых параметризуемо. Записи располагаются в массиве согласно одному из допустимых хеш-кодов.

При вставке записи случайным образом выбирается хеш-код, по нему в массив вставляется запись, если ячейка пуста. Иначе выбирается иной хеш-код до тех пор, пока не будет найдена пустая ячейка.

При поиске и удалении происходит последовательная проверка всех хеш-кодов, пока не будет найдена запись с искомым ключом, поле чего выполняется необходимая операция.

4.2.5 Павлиний хеш

Павлиний хеш [8] использует множество массивов записей и хеш-функций, количество которых равно. Массивы расположены в иерархическом порядке согласно их размерам, убывающим по геометрической прогрессии. Они инициализированы пустыми значениями. Каждому массиву соответствует своя хеш-функция, возвращающая значения в диапазоне его размера. Записи расположены в одном из массивов, согласно вычисленным хеш-кодам.

При вставке записи ячейки, соответствующие хеш-кодам, поочередно проверяются в массивах согласно их иерархии. Проверка должна происходить по всем массивам. Это связано со случаем, когда при удалении одной записи хеш-код другой в вышестоящем массиве освобождается. В таком случае, после вставки записи в пустую ячейку, запись в нижестоящем массиве с тем же ключом должна быть заменена на пустую.

При поиске и удалении массивы также проверяются в порядке иерархии, пока не будет найдена искомая ячейка или пока не проверены все массивы.

4.2.6 Отелло хеш

Отелло хеш [9] разделяет выборку ключей на два подмножества. Он использует двудольный граф, два битовых массива и две хеш-функции. Размер массивов и диапазон возвращаемых значений хеш-функций совпадает с размером долей графа. Каждый ключ ассоциируется с ребром графа, его вершины определяются результатами хеш-функций от данного ключа. В самом графе не должно быть циклов. Доли графа ассоциируются с массивами, а каждая вершина — с ячейками соответствующего массива. Принадлежность ключа к тому или иному подмножеству определяется посредством двоичного сложения с соответствующими ячейками массива. Множественное применение данного метода позволит получить распределение ключей на 2^q множеств.

Во время вставки вычисляются хеш-коды, определяющие вершины и ячейки массивов. После проверяется, не создаст ли новая грань из соответствующих вершин цикла в графе и, если нет, грань добавляется. В соответствии с подмножеством ключа все биты в массивах, соответствующие достижимым вершинам в графе, меняют значения для соблюдения корректности. Это возможно всегда при отсутствии циклов в графе.

При поиске и удалении выполняется вычисление хеш-кодов ключа и бинарное сложение соответствующих битов в массивах. В зависимости от его результата определяется принадлежность ключа к одной из подгрупп. При удалении соответствующее ребро убирается из графа, при этом необходимости изменять массивы нет.

4.3 Результаты обзора

Согласно результатам обзора на таблице 1 для реализации выбраны Линейное зондирование, Кукушкин хеш, Универсальный хеш и Отелло хеш.

Линейное зондирование обладает высоким показателем плотности записи, однако встречается с проблемой кластеризации, когда множество правил располагаются в локальной части массива, сильно повышая время выполнения процедур. Кукушкин хеш обладает интересным принципом «вытеснения» записей, однако без улучшений это не дает сильного преимущества. Оба этих метода могут быть улучшены объединением с Универсальным хешированием, которое само по себе требует значительного количества хеш-функций для достижения высокой плотности массива. Отелло хеширование также имеет пространство для улучшения, уменьшив количество обращений к памяти путем увеличения размерности элементов массива вместо много кратного применения данного метода.

Таблица 1: Результаты хеширования

		Линейное зонд.	**Квадрат. зонд.**	**Кукушкин хеш**	**Универс. хеш**	**Павлиний хеш**	**Отелло хеш**
Поиск	Обр. к памяти	ср. $\underline{O}(\frac{n}{N})$ худ. $\underline{O}(N)$	ср. $\underline{O}(\frac{n}{N})$ худ. $\underline{O}(N)$	1–2	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\log(N))$
	Выч. хешей	1	1	1–2	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\lvert H\rvert)$	2
Вставка	Обр. к памяти	ср. $\underline{O}(\frac{n}{N})$ худ. $\underline{O}(N)$	ср. $\underline{O}(\frac{n}{N})$ худ. $\underline{O}(N)$	ср. $\underline{O}(1)$ худ. $\underline{O}(N)$	ср. $\underline{O}\left(\frac{n}{N\cdot\lvert H\rvert}\right)$ худ. $\underline{O}(N)$	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\log(N))$
	Выч. хешей	1	1	ср. $\underline{O}(1)$ худ. $\underline{O}(N)$	ср. $\underline{O}\left(\frac{n}{N\cdot\lvert H\rvert}\right)$ худ. $\underline{O}(N)$	$\underline{O}(\lvert H\rvert)$	2
Удаление	Обр. к памяти	ср. $\underline{O}(\frac{n}{N})$ худ. $\underline{O}(N)$	—	1–2	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\log(N))$
	Выч. хешей	ср. $\underline{O}(\frac{n}{N})$ худ. $\underline{O}(N)$	—	1–2	$\underline{O}(\lvert H\rvert)$	$\underline{O}(\lvert H\rvert)$	2
Ожид. max $\frac{n}{N}$		1	~ 1	< 0.4	< 1	< 0.8	< 1

Обозначения:

n — количество записей в таблице; N — размер используемого массива;
$|H|$ — количество хеш-функций, используемых структурой.

Глава 5

Реализация композиций методов

5.1 Композиции методов

В ходе проведения работы было установлено, что совместное применение принципов из нескольких методов хеширования позволяет компенсировать их недостатки и повысить эффективность. В этой главе описаны предложенные способы представления таблицы классификации MAC-VLAN, представляющие собой композиции выбранных методов хеширования.

5.2 «Универсальное зондирование»

Способ, представляющий собой линейное зондирование с несколькими хеш-функциями, количество которых параметризуемо. У каждой записи имеется несколько точек доступа во внутренний массив, при этом располагаются они не дальше, чем значение некоторой переменной. Принципиальная схема способа расположена на рисунке 3.

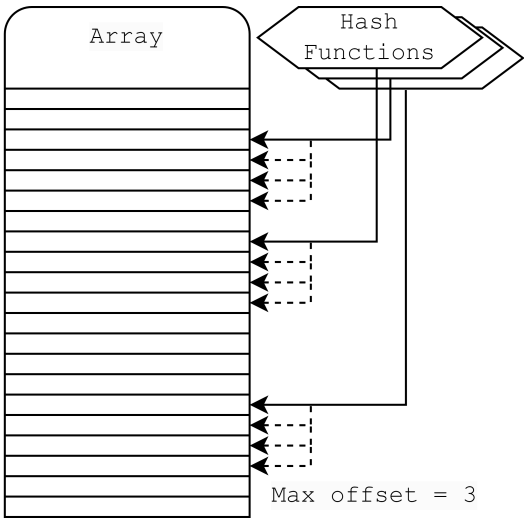

Рис. 3: «Универсальное зондирование»

Инициализация

Для инициализации способа требуется массив записей A размера m, множество хеш-функций F и целочисленная неотрицательная переменная S. Массив замкнут сам на себя, то есть для последней ячейки следующая — первая.

Алгоритм вставки

1) вычисление хеш-кодов вставляемого ключа;

2) проверка всех ячеек по индексам хеш-кодов со сдвигом не более S;

3) если при проверке найдена ячейка, которая содержит вставляемый ключ, то значение записи заменяется на новое, алгоритм успешно завершается;

4) если при проверке найдена пустая ячейка (и не найдено ячейки с вставляемый ключом), то она заполняется вставляемой записью, алгоритм успешно завершается;

5) происходит проверка ячеек по индексам хеш-кодов со сдвигом более S;

6) когда найдена пустая ячейка, она заполняется вставляемой записью, максимальный сдвиг S обновляется, алгоритм успешно завершается;

7) если сдвиг достиг размера массива m, алгоритм завершается неудачей.

Алгоритм поиска

1) вычисление хеш-кодов искомого ключа;

2) проверка всех ячеек по индексам хеш-кодов со сдвигом не более S;

3) если при проверке найдена ячейка, которая содержит искомый ключ, возвращается данная ячейка, алгоритм успешно завершается;

Алгоритм удаления

1) выполняется алгоритм поиска;

2) найденная ячейка заполняется пустым значением.

5.3 Универсальное хеширование «с вытеснением»

Способ, объединяющий Универсальный и Кукушкин хеши. Он использует множество хеш-функций, количество которых параметризуемо. У каждой записи имеется несколько точек доступа во внутренний массив. Для разрешения коллизий возможно вытеснение записи, если другого места нет, после чего вытесненная запись вновь проходит процесс вставки. Принципиальная схема способа расположена на рисунке 4.

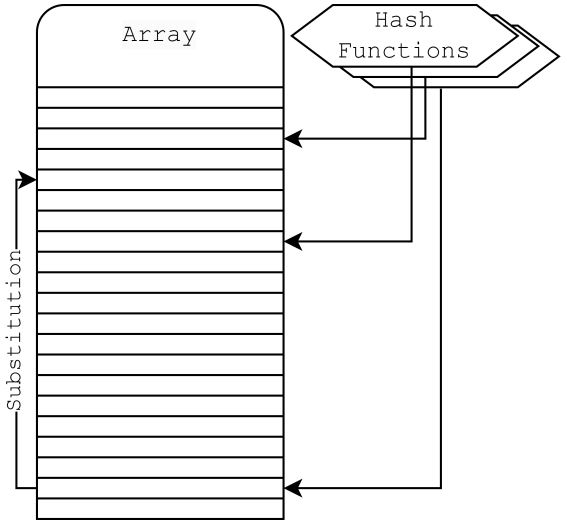

Рис. 4: Универсальное хеширование «с вытеснением»

Инициализация

Для инициализации способа требуется массив записей A размера m и множество хеш-функций F.

Алгоритм вставки

1) вычисление хеш-кодов вставляемого ключа;

2) проверка всех ячеек по индексам хеш-кодов;

3) если при проверке найдена ячейка, которая содержит вставляемый ключ, то значение записи заменяется на новое, алгоритм успешно завершается;

4) если при проверке найдена пустая ячейка (и не найдено ячейки с вставляемый ключом), то она заполняется вставляемой записью, алгоритм успешно завершается;

5) если вставки так и не произошло, запись вставляется в ячейку по одному из хеш-кодов, для «вытиснутой» записи повторяется вставка;

6) после вытеснения последовательность выбора функций произвольно меняется.

Алгоритм поиска

1) вычисление хеш-кодов искомого ключа;

2) проверка всех ячеек по индексам хеш-кодов;

3) если при проверке найдена ячейка, которая содержит искомый ключ, возвращается данная ячейка, алгоритм успешно завершается;

Алгоритм удаления

1) выполняется алгоритм поиска;

2) найденная ячейка заполняется пустым значением.

5.4 Отелло хеширование «множественное»

Способ представляет из себя модифицированный Отелло хеш. Он использует один двудольный граф и одну пару хеш-функций, а так же множество пар битовых массивов, которые могут быть интерпретированы как пара массивов чисел в диапазоне некоторой степени двойки. Это позволяет распределять ключи по гораздо большему количеству подмножеств без увеличения количества вызовов хеш-функций. Принципиальная схема способа расположена на рисунке 5.

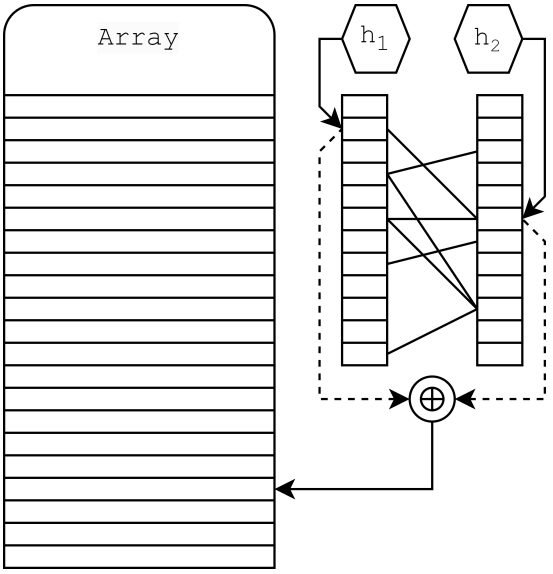

Рис. 5: Отелло хеширование «множественное»

Инициализация

Для инициализации способа требуется массив записей A размера $m \leq 2^q$, пара хеш-функций f_1, f_2, двудольный граф и пара массивов размером B_1 и B_2 совпадающие с долями графа и диапазоном чисел от 0 до 2^q.

Алгоритм вставки

1) вычисление хеш-кодов h_1 и h_2 вставляемого ключа;

2) происходит проверка, не создаст ли новое ребро, соответствующее хеш-кодам, цикла в графе и его добавление;

3) если создается цикл длинны 2, то вероятно, в массиве A имеется запись со вставляемым ключом, тогда происходит замена значения записи на новое, алгоритм успешно завершается;

4) произвольный выбор свободной ячейки из массива записей с индексом w и вставка записи в нее;

5) вычисление $v = B_1[h_1] \oplus B_2[h_2] \oplus w$;

6) произвольный выбор вершины, соответствующей h_1 или h_2 (выгоднее выбрать ту, из которой доступно меньшее количество вершин);

7) выполнение побитовой двоичной суммы с v для всех ячеек битовых массивов, которые ассоциированы с достижимыми вершинами из выбранной.

Алгоритм поиска

1) вычисление хеш-кодов h_1 и h_2 искомого ключа;

2) вычисление $w = B_1[h_1] \oplus B_2[h_2]$;

3) если в $A[w]$ содержится искомый ключ, запись возвращается как результат выполнения.

Алгоритм удаления

1) выполняется алгоритм поиска;

2) найденная запись заменяется на пустую, из графа удаляется соответствующее ребро.

Глава 6

Программная реализация

6.1 Используемая хеш-функция

Для работы описанных способов требуется применение хеш-функции с особыми свойствами. В отличие от тяжеловесных криптографических хеш-функций, она должна обладать высокой скоростью вычисления хеш-кода, при этом не теряя свойств равномерности распределения образов и лавинного эффекта. На ее роль подходит функция, описанная в статье [10].

Семейство хеш-функций

$$\mathbf{H}_n^m \subset \{f : v \to w\}; \quad v \in \{0,1\}^n, w \in \{0,1\}^m$$

состоит из функций, отображающих множество бинарных векторов размера n во множество бинарных векторов размера m. Функция $h \in \mathbf{H}_n^m$ описывается как

$$h(v) = A \otimes (v \oplus s),$$

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \quad s = \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} \quad A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}; \quad v_i, s_i, a_{j,i} \in \{0,1\} \mid i = \overline{1,n}, j = \overline{1,m},$$

где $\cdot \otimes \cdot$ и $\cdot \oplus \cdot$ — операции матричного перемножения и векторного сложения соответственно, вычисляемые по модулю 2.

Вектор соли s и матрица A являются параметрами хеш-функции и задаются произвольно.

Подобные функции, при использовании в СПУ могут быть реализованны как аппаратно, так и программно. В обоих случаях худшее время вычисления функции, за исключением издержек, не превышает $(n \cdot m^2) \cdot \beta$, где β — время выполнения битовой операции в СПУ.

6.2 Описание имитационной модели

Имитационная модель (ИМ) написана на языке Python версии 3.11. Она реализует поведение способов представления таблицы классификации при заданных условиях. Иерархия ИМ представлена на рисунке 6.

В ходе работы процедур способа возвращается следующая информация:

- тип процедуры;
- количество записей в таблице на момент выполнения процедуры;
- наличие искомого ключа во время на момент выполнения процедуры;

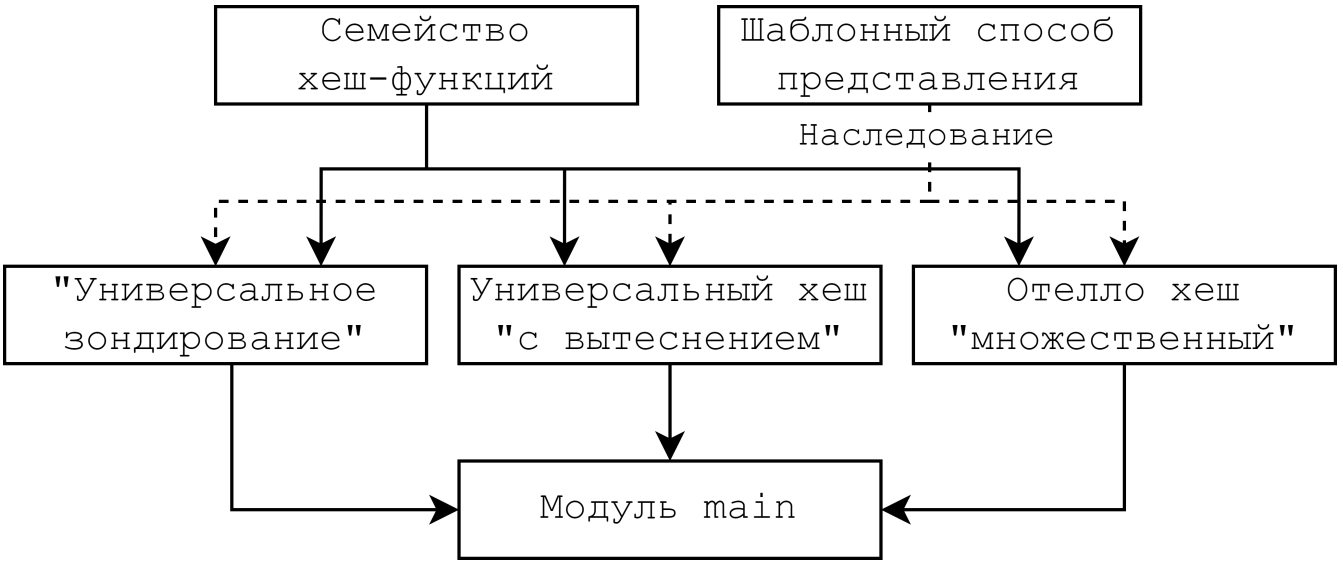

Рис. 6: Иерархия имитационной модели

- количество обращений к памяти;
- количество вычислений хеш-функции;
- результат процедуры, успех или неудача.

Глава 7

Экспериментальное исследование

7.1 Цель исследования

Целью исследования является измерение количества обращений к памяти и вызовов хеш-функции во время выполнения операций при следующих условиях:

- плотность записей в таблице достигает значения 0,9;
- необходимо получение как средних, так и максимальных показателей количества операций;
- для каждой процедуры выполняется измерение как с присутствующей, так и с отсутствующей записью в таблице;
- исследование проводится до достижения количества записей в 65 тысяч правил и 250 тысяч правил.

7.2 Методика экспериментов

Для достижения плотности 0.9 на наборах в 65 тысяч и 250 тысяч правил были выбраны размеры массивов 72 тысячи и 277 тысяч соответственно. Способ «Универсального зондирования» представлен версией, использующей 10 хеш-функций, способ Универсального хеширования «с вытеснением» — версией, использующей 20 хеш-функций.

Для каждого из представленных способов проводится постепенное заполнение таблицы классификации правилами. Последовательность действий следующая:

- вставка правила, отсутствующего в таблице;
- вставка правила, отсутствующего в таблице;
- вставка правила, присутствующего в таблице (модификация правила);
- поиск отсутствующего в таблице правила;
- поиск присутствующего в таблице правила;
- удаление отсутствующего в таблице правила;
- удаление присутствующего в таблице правила;

Данные действия повторяются до достижения требуемого количества правил в таблице.

7.3 Результаты экспериментов

Ниже представлены значения количества операций в процедурах по достижению плотности данных 0,9. На рисунках 7 и 8 изображено количество обращений к памяти при 65 и 250 тысячах правил соответственно. По ним видно, что средние показатели при присутствующем ключе для всех способов не превышают 20 обращений к памяти. При отсутствующем ключе для каждого метода показатели разнятся, но не превышают 150 обращений.

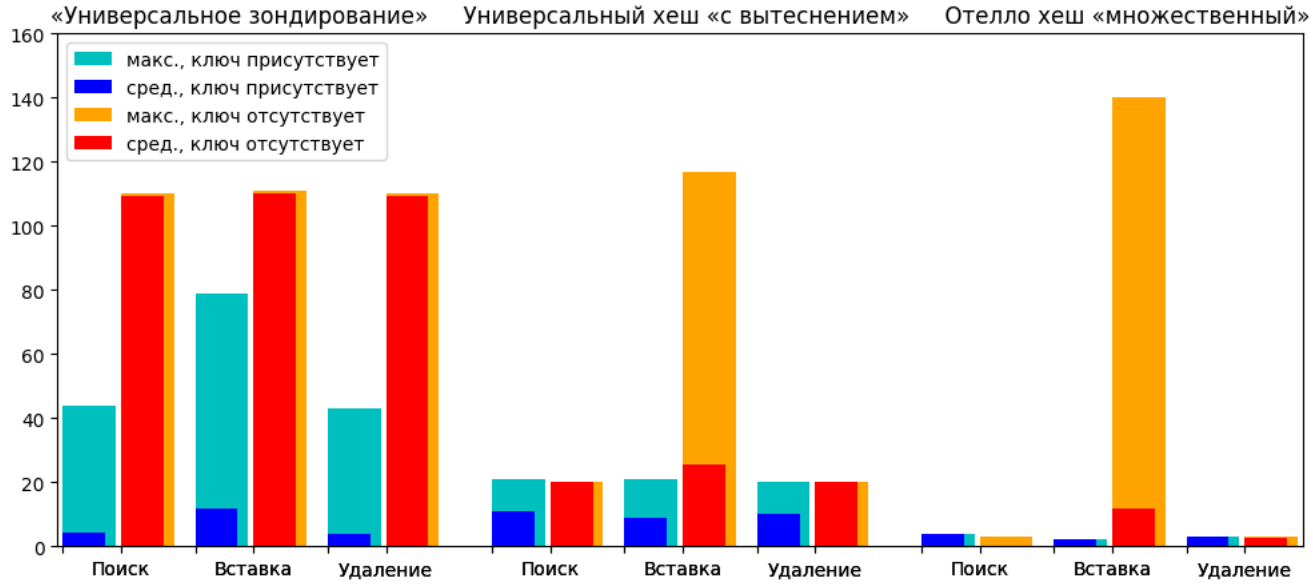

Рис. 7: Количество обращений к памяти при 65000 правил

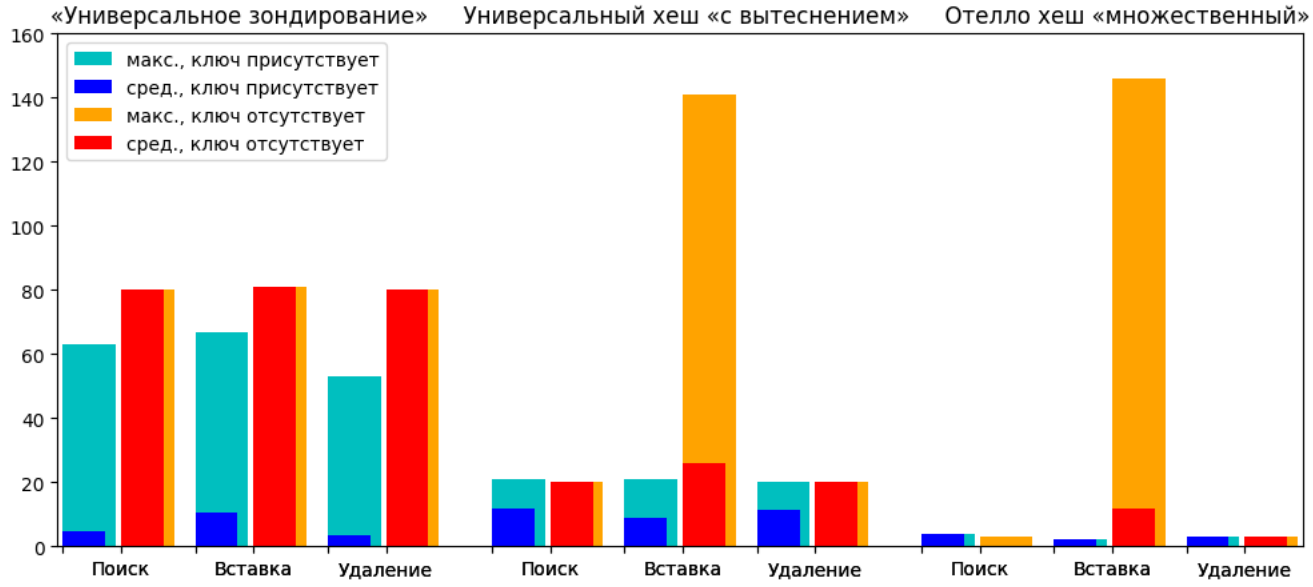

Рис. 8: Количество обращений к памяти при 250000 правил

На рисунках 9 и 10 изображено количество вызовов хеш-функции при 65 и 250 тысячах правил соответственно. По ним видно, что количество вычислений хеш-функции строго ограничено количеством хеш-функций во всех случаях, кроме вставки прежде отсутствующего правила в способе Универсального хеширования «с вытеснением», для которого в среднем значение приближено к количеству хеш-функций, а при худшем случае не превышает 130 вычислений.

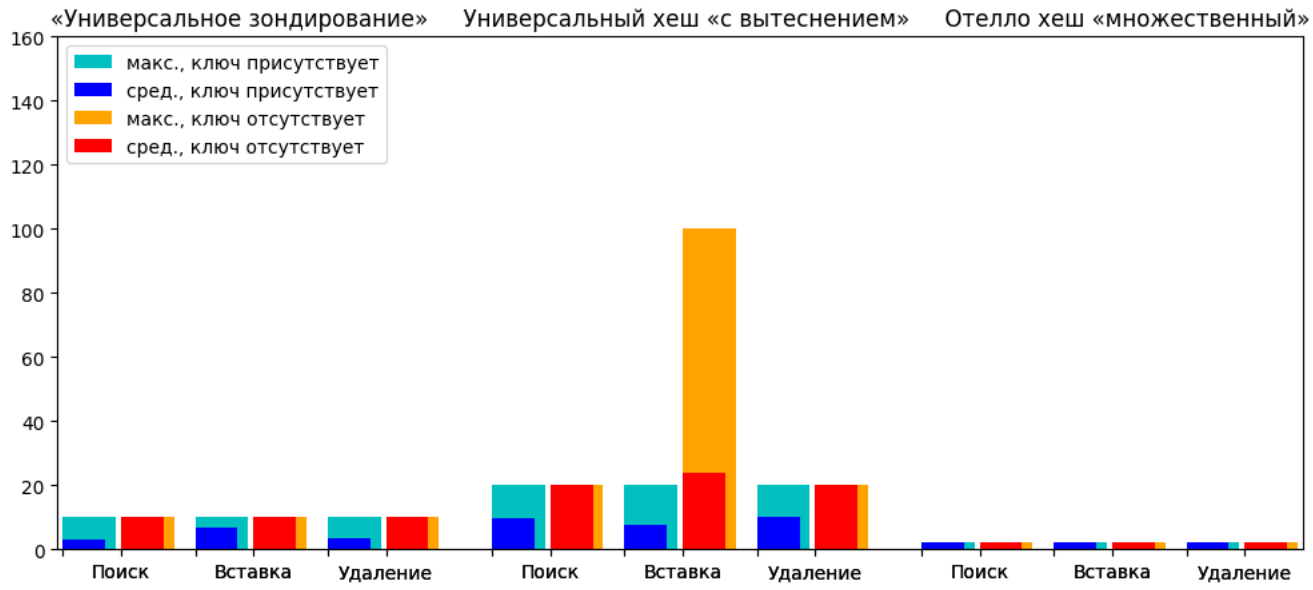

Рис. 9: Количество вызовов хеш-функции при 65000 правил

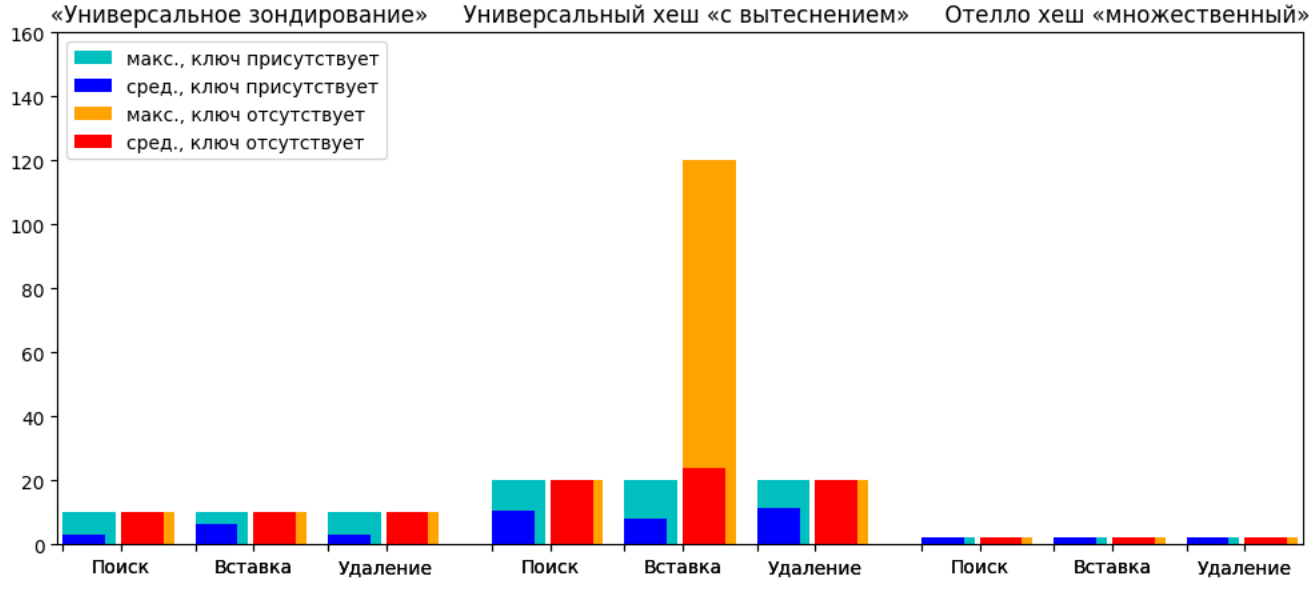

Рис. 10: Количество вызовов хеш-функции при 250000 правил

В приложении А находятся графики для показателей плотности меньше 0,9.

7.4 Выводы

По результатам исследований можно сделать следующие выводы:

- способ «Универсального зондирования» отличается стабильностью производительности на всех процедурах, строго ограниченным количеством вызовов хеш-функции, а обращение в локализованные участки массива правил позволят эффективно использовать кеширование;
- способ Универсального хеширования «с вытеснением» стабилен и строго ограничивает количество вызовов хеш-функции и обращений к памяти для всех процедурах, кроме добавления нового правила, его применение будет целесообразным на всокопроизводительных узлах с больной пропускной способностью;
- способ Отелло хеширования «множественного» показывает еще большую стабильность на тех же процедурах и еще больший отрыв вставки нового правила, его применение будет целесообразным на узлах с предварительно заданными таблицами классификации.

Заключение

В данной работе были исследованы различные методы хеширования данных и были разработаны способы представления таблиц классификации.

Выполнены следующие задачи:

1) на основе обзора методов хеширования данных предложены способы представления таблиц классификации;

2) было проведено экспериментальное исследование, показавшее эффективность данных способов;

3) по результатам исследования была опубликована работа [11].

На основе результатов экспериментов было установлено, что разработанные способы различными полезными свойствами для процесса классификации.

Наибольшей скоростью поиска обладает способ Отелло хеширования «множественного», однако цена за это — дополнительные расходы на память для хранения параметров двудольного графа и массивов чисел, необходимых для получения адреса правила, размер которых растет со скоростью $\underline{O}(n \cdot \log(n))$ относительно требуемого количества правил, что может быть критичным. Поиск у остальных способов также строго ограничен параметрами, так что при нежелательном превышении возможно перестроить таблицу.

Вставка для каждого способа требовательна по обращениям к памяти. Лучшими средними показателями также обладает Отелло хеширование «множественное», остальные способы строго ограничены по количеству вызовов хеш-функций. У «Универсального зондирования» максимальное количество обращений к памяти увеличивается по мере заполнения таблицы, что однако можно своевременно обнаружить, предотвращая отказ работы СПУ.

Удаления во всех способах работают также, как и поиск, за исключением дополнительного обращения к памяти, если ключ был найден.

В качестве дальнейших исследований возможно расширение набора исследуемых методов хеширования, попытка внедрения технологии консистентного хеширования и проведение экспериментов на других типах таблиц классификации.

Список литературы

[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.

[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.

[4] *Larson P.-Å.* Linear hashing with overflow-handling by linear probing // ACM Transactions on Database Systems (TODS). — 1985. — Т. 10, № 1. — С. 75—89.

[5] Frequent item set generation using hashing-quadratic probing technique / M. Krishnamurthy [и др.] // European Journal of Scientific Research. — 2011. — Т. 50, № 4. — С. 523—532.

[6] *Pagh R.*, *Rodler F. F.* Cuckoo hashing // Journal of Algorithms. — 2004. — Т. 51, № 2. — С. 122—144.

[7] *Etzel M.*, *Patel S.*, *Ramzan Z.* Square hash: Fast message authentication via optimized universal hash functions // Annual International Cryptology Conference. — Springer. 1999. — С. 234—251.

[8] *Kumar S.*, *Turner J.*, *Crowley P.* Peacock hashing: Deterministic and updatable hashing for high performance networking // IEEE INFOCOM 2008-The 27th Conference on Computer Communications. — IEEE. 2008. — С. 101—105.

[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.

[10] *Ramakrishna M.*, *Fu E.*, *Bahcekapili E.* A performance study of hashing functions for hardware applications // Proc. of Int. Conf. on Computing and Information. — Citeseer. 1994. — С. 1621—1636.

[11] *Александров А. В.*, *Волканов Д. Ю.* Исследование хеширующих поисковых структур данных для таблицы классификации MAC-VLAN сетевого процессорного устройства // Ломоносовские чтения. Научная конференция. 20 марта – 3 апреля 2024 г. : тезисы докладов. — 2024. — С. 106—107.

Приложение А

Дополнительные графики

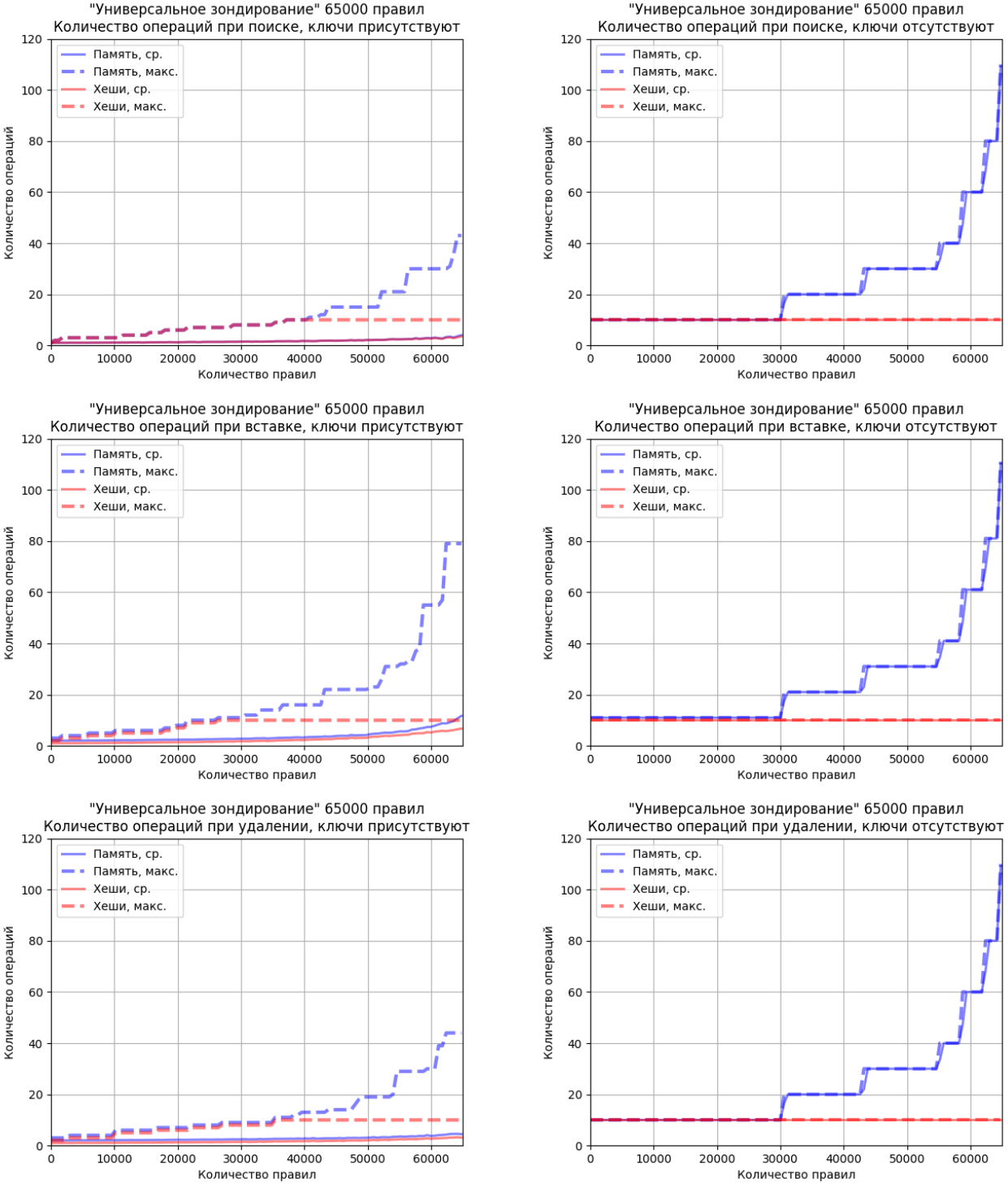

Рис. 11: Результаты «Универсального зондирования» для набора в 65000 правил

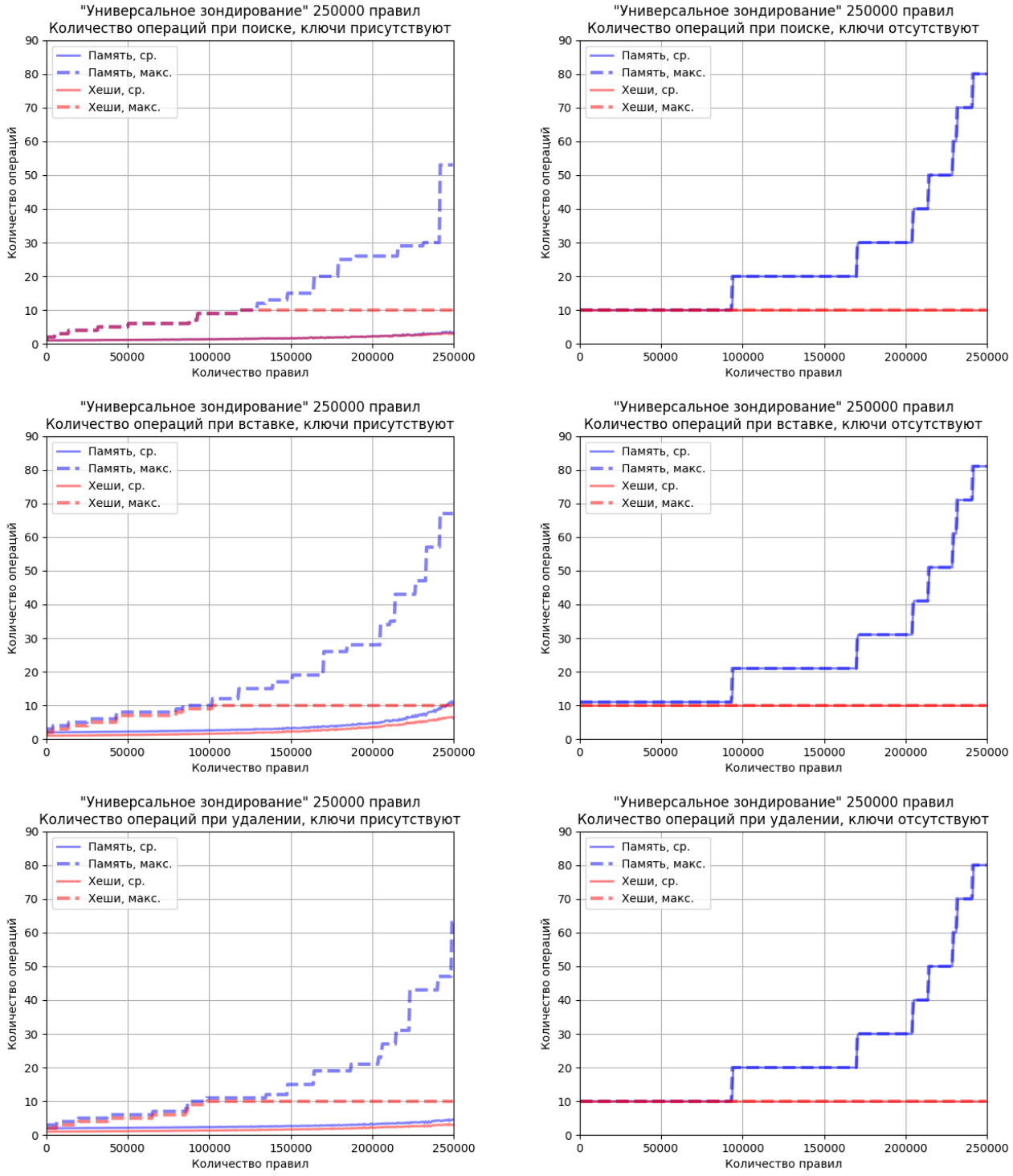

Рис. 12: Результаты «Универсального зондирования» для набора в 250000 правил

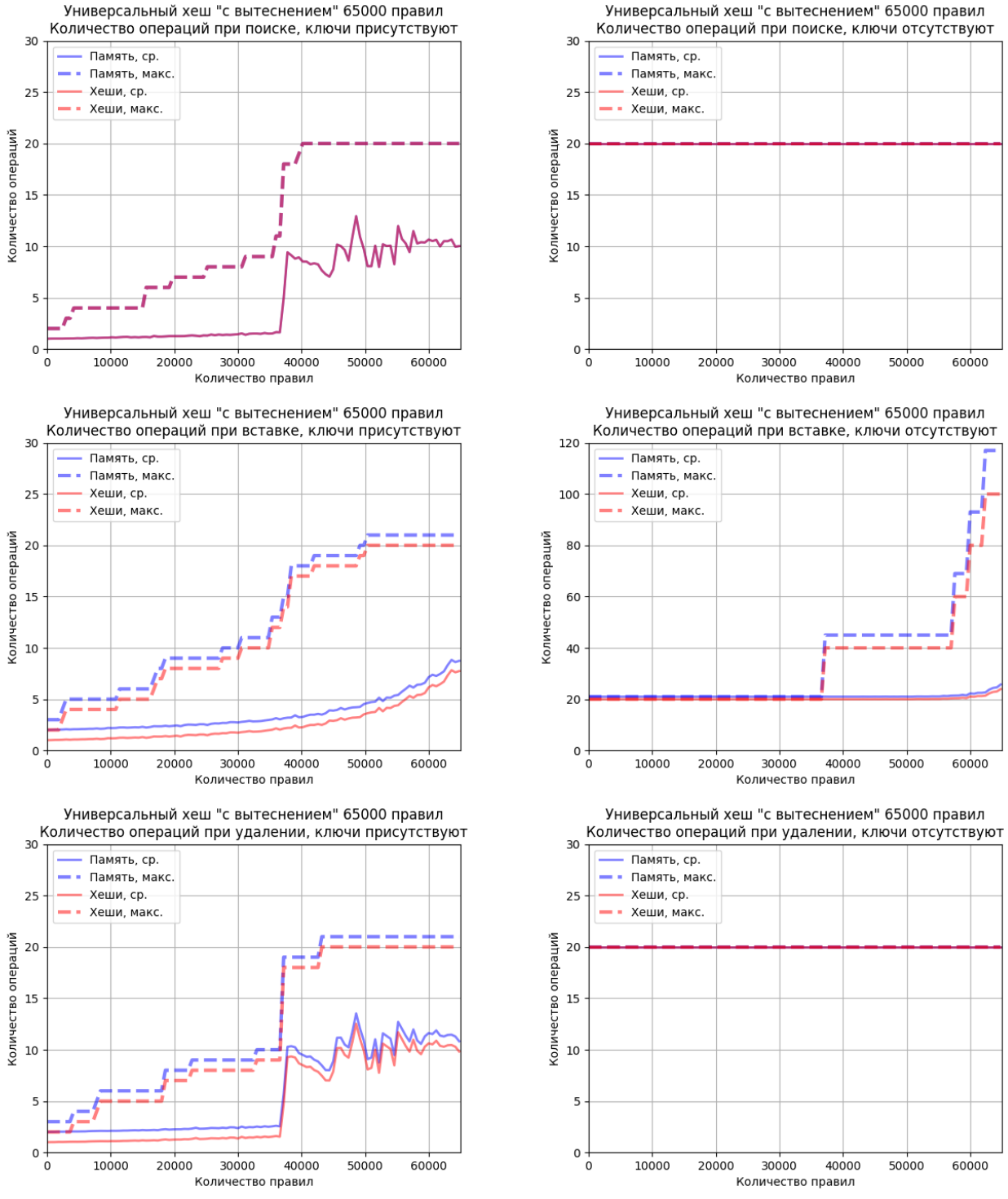

Рис. 13: Результаты Универсального хеширования «с вытеснением» для набора в 65000 правил

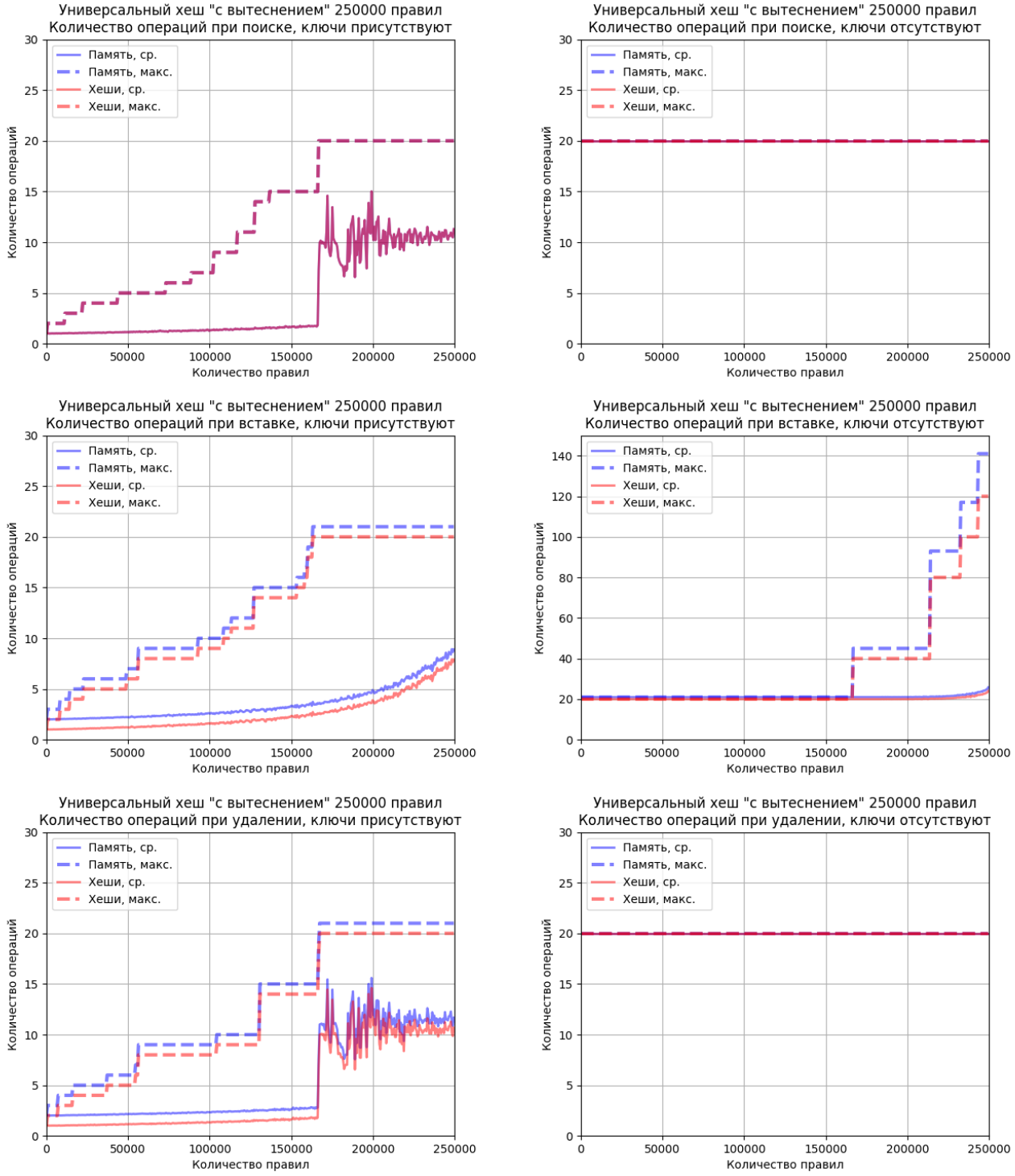

Рис. 14: Результаты Универсального хеширования «с вытеснением» для набора в 250000 правил

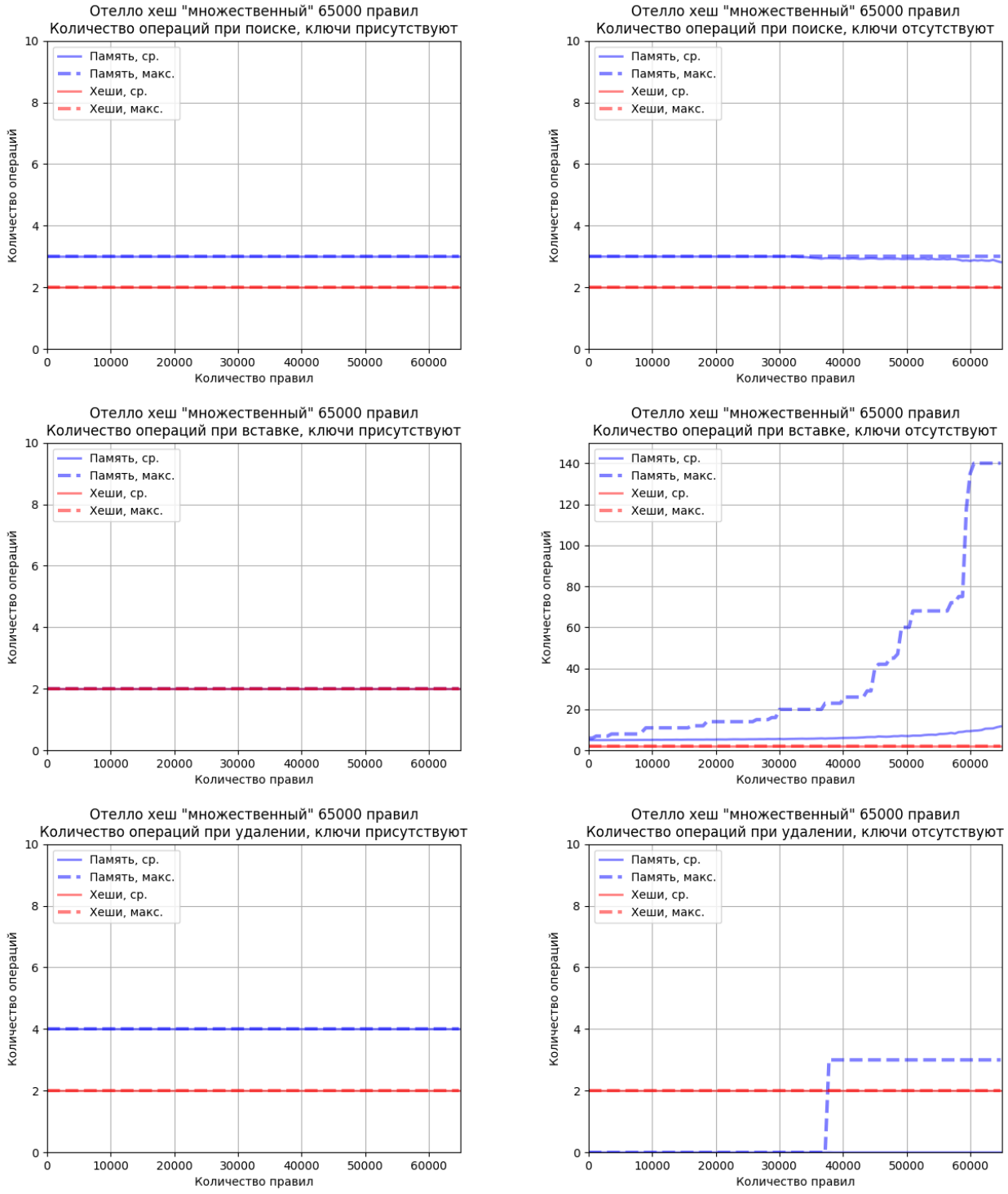

Рис. 15: Результаты Отелло хеширования «множественного» для набора в 65000 правил

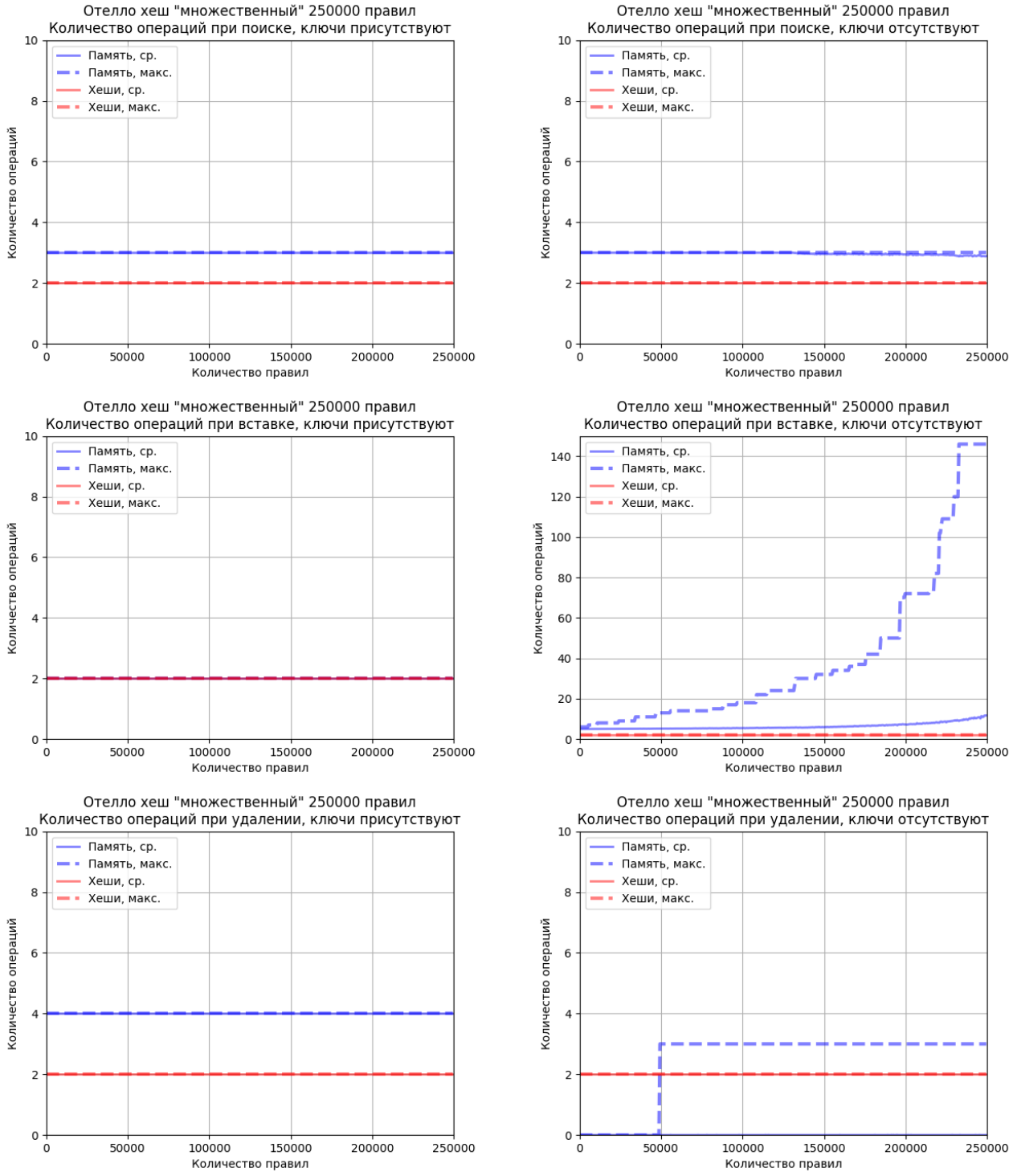

Рис. 16: Результаты Отелло хеширования «множественного» для набора в 250000 правил