ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ НАУКИ
САНКТ-ПЕТЕРБУРГСКОЕ ОТДЕЛЕНИЕ МАТЕМАТИЧЕСКОГО ИНСТИТУТА ИМЕНИ
В. А. СТЕКЛОВА РОССИЙСКОЙ АКАДЕМИИ НАУК

На правах рукописи

Чуприков Павел Сергеевич

ТЕОРЕТИЧЕСКИЙ И ЭМПИРИЧЕСКИЙ АНАЛИЗ ФУНДАМЕНТАЛЬНЫХ ПРОБЛЕМ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СЕТЕЙ И РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

РЕЗЮМЕ
диссертации на соискание ученой степени
кандидата компьютерных наук НИУ ВШЭ

Санкт-Петербург — 2019

Диссертационная работа выполнена в Федеральном Государственном бюджетном учреждении науки Санкт-Петербургское отделение Математического института имени В. А. Стеклова Российской академии наук

Научные руководители:

- Николенко Сергей Игоревич, к.ф.-м.н., ПОМИ РАН, научный сотрудник лаборатории Математической логики
- Коган Кирилл, PhD, IMDEA Networks Institute, Research Assistant Professor

1. Введение

Данный раздел сначала кратко обозначает тему данной диссертации — существующие ограничения систем обработки больших данных и роль в этих ограничениях компьютерных сетей — а затем вводит цели и задачи исследования.

1.1. Тема диссертации и ее актуальность

Термин “большие данные” (англ. *Big Data*) обычно относится к таким массивам данных, объем которых превышает возможности анализа и хранения типичных вычислительных платформ. Ожидается, что в промежутке с 2013-ого по 2020 год глобальный объем данных вырастет экспоненциально с 4.4 зеттабайт до 44 зеттабайт [1]. Из-за большого размера данных и ограничений пропускной способности систем хранения данные приходится распределять между несколькими точками. В данном случае компьютерная сеть выступает *связующим звеном* между распределенными экземплярами приложений, работающих с большими данными.

Объем и разнообразие *больших данных* быстро увеличивается. Как одно из следствий, в сфере облачных вычислений выделенное оборудование все больше заменяется динамически распределяемыми виртуальными вычислительными ресурсами, которые оплачиваются исходя из запрошенного объема, а не по факту реального использования, вследствие чего потребители ресурсов часто платят больше, чем необходимо. В последнее время стали очень популярны *бессерверные вычисления*, и многие игроки на рынке облачных вычислений предложили бессерверные решения: AWS Lambda [2], Google Cloud Functions [3], Azure Functions [4] и другие.

Несмотря на то, что бессерверные вычисления — это только частный случай облачных вычислений, и они не способны обеспечить любую желаемую функциональность, они представляют собой важный случай, способный существенно сократить расходы пользователей. В частности, парадигма “функция как услуга” (англ. *function as a service*, FaaS) добавляет новые трудности к задаче распределения ресурсов. Последние работы только начинают задавать важные теоретические вопросы о бессерверных вычислениях, изучая последние в контексте теории массового обслуживания [5] и измеряя важные показатели уже существующих решений [6]. В данной диссертации сделан следующий шаг к теории бессерверных вычислений: предложена новая целостная формализация модели распределения ресурсов для бессерверных вычислений, представлены новые алгоритмы распределения ресурсов и проведен их подробный теоретический анализ.

В дополнение к необходимости более гибких методов распределения ресурсов, современные объемы больших данных и чувствительные к задержкам целевые функции представляют собой другие серьезные ограничения для существующей парадигмы облачных вычислений. Один из важных недостатков последней — использование сети исключительно как связующей инфраструктуры. Хотя традиционные сети в большинстве своем работают только с частичной информацией от потоков данных (с заголовками пакетов), в некоторых случаях сетевые элементы уже способны обрабатывать потоки данных целиком со скоростью интерфейса (например, шифрование). Поэтому, если внутренняя структура пакета сделана открытой, становится возможным проводить предобработку передаваемых данных перед тем, как они обработаются облачными вычислительными

ресурсами. Согласно [7, 8] средний размер конечного результата типичной задачи вычисления-агрегации составляет 40.3% от размера исходных данных для Google, 8.2% для Yahoo и 5.4% для Facebook. В настоящей диссертации мы используем вышеперечисленные свойства потоков данных для уменьшения нагрузки на традиционные облачные ресурсы. Нами разработаны два новых метода для достижения данной цели: (1) промежуточная агрегация данных внутри сети и (2) обработка данных внутри сетевых элементов; оба метода существенно расширяют текущие возможности компьютерных сетей.

1.2. Цели и задачи исследования

Данный раздел определяет конкретные задачи, которые необходимо выполнить для достижения целей данной диссертации: управление буфером, классификация пакетов, экономичные бессерверные вычисления, и промежуточная агрегация потоков данных.

Управление буфером несколько характеристик пакета. В тот момент, когда *несколько потоков данных* соединяются в сетевом устройстве, соответствующие пакеты данных, ожидающие дальнейшей обработки и передачи, хранятся в едином сетевом буфере. Буферы контролируются *алгоритмами управления буфером*, которые играют существенную роль в оптимизации желаемых целевых функций. Продвинутые экономические модели привносят новые задачи, в частности, задачу максимизации взвешенной пропускной способности (англ. *weighted throughput*); существующие алгоритмы, как правило, не поддерживают такие целевые функции. Точно так же остается неучтенным разнообразие требований к сложности обработки пакетов (англ. *processing requirements*), многократно усугубляющееся необходимостью обрабатывать данные. В предыдущих работах рассматривалась только одна из двух характеристик: в [9, 10] изучается поведение одной очереди с различными требованиями к обработке в то время, как [11] рассматривает пакеты различного веса для FIFO очередей; кроме того, более сложные архитектуры буферов [12] (комбинированный ввод-вывод) и [13] (перекрестная архитектура) были рассмотрены для различающихся весов пакетов. Взаимодействие же двух характеристик и их влияние на максимизацию взвешенной пропускной способности ранее не изучалось. С целью построения теоретической основы выбора оптимального алгоритма управления буфером в данной модели в настоящей диссертации мы решаем следующие задачи для случая буфера с одной очередью: (1) построить модель, учитывающую и веса пакетов, и их требования к обработке; (2) разработать новые алгоритмы управления буфером и оценить их производительность в худшем случае; (3) оценить поведение алгоритмов на реалистичных наборах данных (входных потоках пакетов).

Классификация пакетов: распределение ресурсов между устройствами и переиспользование LPM инфраструктуры. Базовым элементом *обработки одного пакета* в сетевом устройстве является *классификация пакетов*. Задача общей обработки данных внутри устройства привносит новые требования к гибкости и добавляет сложности к свойствам, реализуемым при помощи классификации. Текущие подходы направленные на отдельные устройств делятся на программные и на использующие аппаратные средства. Первые включают специальные деревья принятия реше-

ний [14, 15], хэширование [16] и кодирование [17], в целом данные подходы требуют изменений в алгоритме классификации. Вторые же опираются на троичную ассоциативную память (англ. *ternary content-addressable memory* или *TCAM*) и оптимизируют ее объем [18, 19] и, конкретно, представление диапазонов [20]. На сетевом уровне (между устройствами), две имеющиеся работы [21, 22] предлагают алгоритмы, который дуплицируют правила и сложны вычислительно. Крайне желательно, чтобы новый уровень сложности обработки данных мог быть поддержан без значительных расходов на обновление инфраструктуры, например в виде TCAM. В частности, потребуется решить две задачи: во-первых, найти способ реализовать обобщенную тернарную классификацию, используя традиционное представление классификаторов в виде префиксных правил, в которых приоритет имеет префикс максимальной длины (англ. *longest-prefix match* или *LPM*), не зависящий от конкретной реализации; во-вторых, найти эффективную схему совместного использования ресурсов нескольких устройств, которая необходима для представления больших классификаторов в рамках ограниченных возможностей отдельных устройств.

Бессерверные вычисления: эффективное по стоимости выделение ресурсов. Необходимый шаг к эффективным бессерверным вычислениям состоит в построении реалистичной модели, которая бы включила все расходы и ограничения управления ресурсами. Среди этих расходов и ограничений можно назвать *стоимость выделения ресурса*, *стоимость поддержания ресурса* и, главное, *время, требуемое для выделения ресурса*. Для построенной модели затем следует разработать эффективные алгоритмы, требующие минимум предположений о свойствах приходящих запросов и имеющие гарантии производительности, которые можно было бы использовать в экономически значимых ситуациях. В отличие от существующих моделей, которые либо основаны на жестких правилах [23] и требуют глубокого понимания закономерностей поведения входящих запросов, либо основаны на методах машинного обучения [24], что предполагает, что эти закономерности будут схожи с увиденным во время обучения модели, в предложенной модели нам удалось доказать ряд оценок и результатов в худшем случае, независимо от входящего потока запросов. Ещё одна важная побочная цель состоит в том, что при этом обязательно нужно контролировать задержку в обработке запросов, привносимую предложенными решениями, и оценить, каким образом она влияет на целевую функцию.

Агрегация данных: планирование с учетом и сетевой инфраструктуры и приложений. Для того чтобы извлечь пользу от промежуточной агрегации данных наилучшим образом требуется обеспечить оптимальное взаимодействия между двумя уровнями: сетевым со своими ограничениями и задачами и уровнем приложения со своими собственными задачами и спецификой поведения. В данный момент обмена информацией между ними практически не происходит, и если приложение пытается агрегировать слишком много данных в одном узле, оно может существенно снизить производительность всей системы, например вследствие TCP-incast [25] или перегрузки сетевого канала. Наиболее близкое к нам исследование [26] привязано к фиксированной тополгии сети; [27] исследует техническую сторону поддержки промежуточных агрегаций, и [28] рассматривает исключительно время обработки и не затрагивает ограничения сетевой инфраструктуры. Задача состоит в поиске абсолютно необходимого минимума информации, который нужно потре-

бовать от каждого уровня, чтобы сделать возможным эффективное планирование агрегации.

2. Основные результаты и выводы

Основные результаты, достигнутые при выполнении поставленных задач диссертации, подытожены ниже. Краткий обзор каждого результата, а также его роль в теме исследования, представлены далее в Разделе 4.

2.1. Результаты об алгоритмах управления буфером

Управление буфером с двумя характеристиками пакетов. Задача управления буфером была рассмотрена для одной очереди, содержащей пакеты, различающиеся как по своей относительной ценности (весу), так и по объему работы [29]. Ранние работы [9] работали либо с одной, либо с другой из двух характеристик. Нижние оценки на конкурентность (см. [30]) были доказаны для нескольких естественных алгоритмов, основанных на приоритетных очередях. Главный результат — это верхняя оценка $\left(1 + \frac{W+2}{V}\right)$ для частного случая, в котором допускаются только два возможных значения ценности; здесь W — максимальный объем работы, а 1 и V — два возможных значения ценности. Доказательство основано на индуктивном рассуждении и соответствии между пакетами оптимального и рассматриваемого алгоритма (схожее с использованным в [9]), учитывающего пакеты с ценностью 1 особым образом.

2.2. Результаты о классификации сетевых пакетов

Алгоритмы преобразования тернарных классификаторов в LPM. Два алгоритма, а именно `MinGroupPartition` и `MaxCoveragePartition`, были разработаны для задачи преобразования обобщенных тернарных классификаторов пакетов в эквивалентную группу более ограниченных LPM классификаторов [31]. Основой данных преобразований стала найденная взаимосвязь между способностью сетевых устройств конструировать сложные ключи для запросов и теории порядков (более конкретно с [32]). Было показано, что `MinGroupPartition` минимизирует количество групп, а `MaxCoveragePartition` максимизирует число правил, представленных заданным числом LPM групп. В результате эти алгоритмы делают возможным представление выразительных тернарных классификаторов посредством традиционных LPM ресурсов существующей инфраструктуры.

Сложная классификация пакетов по всей сети. Разработана простая и эффективная с точки зрения памяти схема распределения правил классификации вдоль пути потока данных, названная `OneBit` [33]. Данная схема требует всего лишь одного бита метаданных и, в отличие от более ранних схем, не опирается на эвристики, тривиальна вычислительно (линейна по числу правил), а также допускает полиномиальную процедуру проверки возможности распределения правил классификации для нескольких потоков. Аналогично упомянутым выше алгоритмам преобразования, `OneBit` позволяет реализовать очень сложные правила без всякой замены инфраструктуры.

2.3. Результаты об эффективных бессерверных вычислениях

Эластичное выделение ресурсов. В рамках задачи эластичного выделения ресурсов был исследован баланс между возможностью задерживать обработку пользовательских запросов и эффективным использованием доступных ресурсов [34]. Использованная модель включила в себя стоимость выделения α, стоимость поддержания ресурса β, $\beta < 1$, и время выделения ресурса (нормализованное к 1). Было показано, что простой жадный алгоритм NRAP не более чем $2 \cdot \left(1 + \frac{\alpha}{1-\alpha-\beta}\right)$-конкурентен (англ. -*competitive*) в случае, когда $\alpha < 1 - \beta$. Для случая $\alpha \geq 1 - \beta$ для другого параметризованного алгоритма ρ-AAP, который активно пользуется возможностью задерживать обработку, доказана $\frac{\rho}{1-\rho}\left(\left\lceil \frac{\rho\alpha}{1-\beta} \right\rceil + 1\right)$-конкурентность (с несколько увеличенным размером буфера). В сравнении с ранними работами [35], во многом полагавшихся либо на историю запросов, либо на детальную информацию об их поведении, гарантии качества алгоритмов, доказанные в настоящей диссертации, даны для худшего случае и, как следствие, верны для любой последовательности входящих запросов.

2.4. Результаты об алгоритмах промежуточной агрегации данных

Планирование задач вычисления-агрегации. В целях оптимизации расположения точек промежуточной агрегации для задач вычисления-агрегации в [36] была поставлена задача минимизации вычисления-агрегации (англ. *compute-aggregate minimization* или CAM). В предложенной постановке были компактно учтены характеристики как компьютерной сети (топология и стоимость передачи), так и приложения (функция размера агрегации). Приводится обоснование достаточности данных характеристик для построения *плана агрегаций*, который эффективно использует промежуточные агрегации и минимизирует суммарную стоимость передачи данных. Также в рамках данной диссертации доказывается сложность аппроксимации CAM в неассоциативном случае и характеризуется связь между CAM и задачей о нахождении дерева Штейнера минимального веса. Введенное в диссертации определение плана агрегаций нисколько не ограничивает, в какой момент будут происходить агрегации, таким образом абстрагируясь и от сети, и от приложения. В сравнении с ранней работой, использующей промежуточные агрегации [26], предложенный подход является более общим; в частности, он не накладывает ограничений на топологию сети.

3. Публикации и апробация работы

Каждый из результатов, представленных в предыдущем разделе, был ранее опубликован в одной из рецензируемых научных работ, представленных ниже.

Публикации повышенного уровня:

- Chuprikov P., Nikolenko S. I., Davydow A., Kogan K. Priority Queueing for Packets with Two Characteristics* // IEEE/ACM Transactions on Networking. 2018. vol. 26 (1). pp. 342-355.

*Автор диссертации является главным автором данной статьи

Вклад автора диссертации: Теоремы с 1-ой по 3-ю, характеризующие алгоритмы, основанные на приоритетной очереди Теорема 7, устанавливающая общую нижнюю оценку; Теорема 11, предоставляющая нижние оценки для основанных на приоритетной очереди алгоритмов в случае двух значений ценности; Теорема 12 демонстрирующая верхнюю границу $\mathtt{PQ}_{v,-w}$ в случае двух значений ценности; и, Теорема 17 дающая нижние оценки для алгоритмов, основанных на приоритетной очереди с β-выталкиванием.

- Chuprikov P., Kogan K., Nikolenko S. General Ternary Bit Strings on Commodity Longest-prefix-match Infrastructures* // Proceedings of IEEE ICNP 2017.

  Вклад автора диссертации: Теорема 1 устанавливающая необходимое и достаточное условие для префиксно-переупорядочиваемости; алгоритм `PrefixToLPM` и доказательство его корректности в Теореме 2; алгоритм `MinGroupPartition` и доказательство его корректности изложенное в Теореме 3; Теорема 4 демонстрирующая сложный пример для MinGR; алгоритм `MaxCoveragePartition` и его анализ в Теореме 5; раскрывающая биты эвристика описанная в Разделе V; Наблюдения 3 и 4 связывающие префиксно-переупорядочиваемость с дизъюнктностью по правилам; реализация вышеперечисленных алгоритмов.

- Chuprikov P., Nikolenko S., Kogan K. On Demand Elastic Capacity Planning for Service Autoscaling* // Proceedings of IEEE INFOCOM 2016.

  Вклад автора диссертации: Теоремы 1 и 2 утверждающие неконкурентность в отсутствие буферов; нижняя оценка в в случае буферов и малой стоимости выделения из Теоремы 3; алгоритм `NRAP` и границы его конкурентнсти в Теоремах 4 и 5; алгоритм ρ-`AAP` и его анализ в Теоремах 6 и 7; Теоремы с 8 по 11 предоставляющие гарантии на задержку для `NRAP` и ρ-`AAP`; Теоремы 12 и 13 показывающие общие нижние оценки в модели с ограниченной задержкой (англ. *Bounded Delay*—BD); Теорема 14, ограничивающая сверху конкурентность `NRAP` в BD модели; Теорема 15 с общей нижней оценкой в модели BD с ограниченными ресурсами (англ. *Limited* BD—LBD); Теоремы 16 и 17 дающие оценки на конкурентность `NRAP` в LBD модели;

Публикации стандартного уровня:

- Chuprikov P., Davydow A., Kogan K., Nikolenko S. I., Sirotkin A. Formalizing Compute-Aggregate Problems in Cloud Computing // Proceedings of SIROCCO 2018.

  Вклад автора диссертации: формализация планирования задач вычисления-агрегирования и сложность аппроксимации в неассоциативном случае (Раздел 3 работы); теоремы 2, 6 и 7 связывающие задачу CAM с задачей о нахождении дерева Штейнера минимального веса (Разделы 4.1 и 4.2 соответствующей работы).

Другие публикации:

- Chuprikov P., Kogan K., Nikolenko S. I., How to implement complex policies on existing network infrastructure* // Proceedings of ACM SOSR 2018.

Вклад автора диссертации: алгоритм `OneBit` и гарантии его производительности, представленные в Теореме 5.2; решение задачи $\mathrm{MultiFlow}\mathit{Split}$ рассмотренное в Разделе 6; сравнение производительности `OneBit` с существующими подходами.

Полученные результаты были апробированы двумя путями: во-первых, в рамках реалистичных моделей были получены строгие доказательства производительности алгоритмов в худшем случае; во-вторых, так как худший случай может редко иметь место на практике, предложенные решения также протестированы с помощью синтетических данных. Алгоритмы, работающие с классификаторами (Раздел 4.2) были применены к комплексу тестов под названием *Classbench* [37]. Алгоритмы управления буфером (Раздел 4.1) были опробованы на сетевом трафике, основанном на данных CAIDA [38]. Для тестирования выделения ресурсов (Раздел 4.3) использовались порожденные случайным образом из распределений, схожих с сетевым трафиком [39].

4. Содержание работы

В данном разделе дан обзор каждому из исследовательских проектов, в рамках которых были получены основные результаты диссертации, перечисленные в Разделе 2, а также дано подробное описание того, каким образом были выполнены поставленные в Разделе 1.2 задачи.

4.1. Обработка нескольких потоков данных

Компьютерная сеть должна поддерживать продвинутые экономические модели, выраженные через новые типы целевых функций. Алгоритмы управления буфером играют ключевую роль в оптимизации этих функций. Кроме этого, трафик в сети очень разнообразен, и его характеристики существенно влияют на процесс оптимизации. К сожалению, эти характеристики часто не учитываются управляющими алгоритмами. Переход к обработке данных внутри сети привнесет еще больше разнообразия в объеме работы, приводя к дальнейшему понижению эффективности существующих алгоритмов. В ответ на появляющиеся трудности в статье «Priority Queueing for Packets with Two Characteristics» [29], краткое описание которой дано ниже, изучается эффект привносимый наличием у пакетов двух характеристик, а именно объемом работы и ценности. Результаты данной работы идут первыми среди результатов из Раздела 2.

Рассмотренная далее модель работает с одной очередью, способной хранить B пакетов единичного размера (все приходящие пакеты — единичного размера). Каждый приходящий пакет p обладает двумя характеристиками: *объем работы* (англ. *processing requirements*) $w(p) \in \{1, \dots, W\}$, или просто — *работа*, и также *ценность* (англ. *value*) $v(p) \in \{1, \dots, V\}$; такой пакет p мы будем обозначать как $(w(p) \mid v(p))$. Обе характеристики известны по прибытии. Время поделено на дискретные шаги, и каждый шаг t содержит три фазы: (1) *прием*, когда приходит множество пакетов, и управляющий алгоритм решает, какие их пакетов принять в очередь, возможно предварительно *вытолкнув* другие, принятые ранее; (2) *обработка*, когда один из пакетов в очереди обрабатывается; и (3) *передача*, когда один *полностью обработанный* пакет, т. е. такой пакет p, который был обработан хотя бы $w(p)$ раз, выбран для передачи. Заметим, представленные далее результаты

никоим образом не ограничивают порядок передачи, поэтому решение при передаче обычно тривиально: надо передавать единственный полностью обработанный пакет, если таковой имеется. Цель – придумать такой алгоритм управления буфером, который бы максимизировал суммарную взвешенную пропускную способность, т. е. суммарную ценность всех переданных пакетов.

К сожалению, из-за того, что задача требует принятия промежуточных решений до того, как весь вход становится известен (иначе говоря, задача — *онлайн*), онлайн-алгоритма, который всегда возвращает оптимальное решение, может не существовать. По этой причине, гарантии оптимальности даны относительно неизвестного оптимального решения используя конкурентный анализ [30]. Онлайн-алгоритм ALG называется α-конкурентным для некоторого $\alpha \geq 1$ тогда и только тогда, когда для *любой* конечной последовательности приходящих пакетов σ суммарная взвешенная пропускная способность ALG не более чем в $1/\alpha$ раз меньше суммарной пропускной способности оптимального *всезнающего* алгоритма OPT.

Алгоритмы. В предыдущих работах было показано, что если верно либо $V = 1$, либо $W = 1$, то оптимальными являются алгоритмы, основанные на приоритетной очереди, которые предпочитают либо пакеты с большей ценностью, либо с меньшим объемом работы [9]. Вследствие этого, такого рода алгоритмы кажутся естественными кандидатами и для описанной выше модели, однако в данном случае, выбор правильного приоритета при принятии или обработке не так прост. Далее дано обобщенное понятие основанного на приоритетах алгоритма.

Определение. *Пусть f — некоторая функция пакета, $f(p) \in \mathbb{R}$ (интуиция такая, что более хорошие пакеты имеют большие значения f). Тогда алгоритм $\mathtt{PQ}_f$ определяется следующим образом:*

- $\mathtt{PQ}_f$ жадный, *т. е. он принимает все новые пакеты до тех пор, пока в буфере есть место;*
- $\mathtt{PQ}_f$ сохраняет работу, *т. е. он всегда обрабатывает пакет, если буфер не пуст;*
- $\mathtt{PQ}_f$ *обрабатывает пакеты из буфера в порядке уменьшения значений f;*
- $\mathtt{PQ}_f$ *выталкивает p и принимает новый пакет p' в очередь если буфер полный, p является наихудшим пакетом в буфере и p' лучше чем p: $f(p) = \min_{q \in \mathrm{IB}^{\mathtt{PQ}_f}} f(q)$, и $f(p') > f(p)$, где $\mathrm{IB}^{\mathtt{PQ}_f}$ обозначает множество пакетов в буфере $\mathtt{PQ}_f$ на текущем шаге времени.*

Говоря кратко, при приеме $\mathtt{PQ}_f$ рассматривает новые и ранее принятые пакеты вместе, оставляя в буфере имеющие большее значение f; а при обработке $\mathtt{PQ}_f$ выбирает пакет с наибольшим значением $f(p)$

Существует три подающих надежды кандидата для функции приоритета f. В следующих определениях v и w обозначают, соответственно, ценность и *текущий* объем работы. (1) $\mathtt{PQ}_{v,-w} = \mathtt{PQ}_{v-w/(W+1)}$, предпочитающий пакеты имеющие большую ценность, и предпочитая меньшее время обработки в случае ничьи; (2) $\mathtt{PQ}_{-w,v} = \mathtt{PQ}_{-w+v/(V+1)}$, предпочитающий пакеты с меньшим объемом работы, и предпочитая большую ценность в случае ничьи; (3) $\mathtt{PQ}_{v/w}$ предпочитающий пакеты с наибольшим отношением ценности к объему работы. Примеры поведения каждого алгоритма изображены на Рисунке 1.

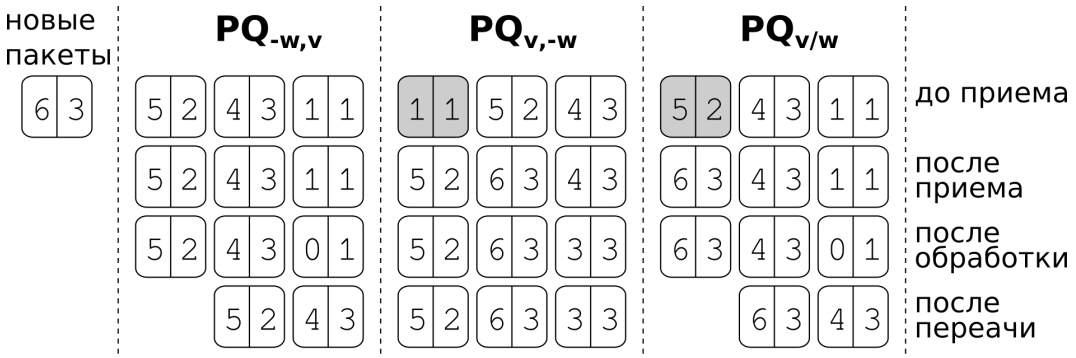

Рис. 1: Пример шага времени для $\mathsf{PQ}_{-w,v}$, $\mathsf{PQ}_{v,-w}$ и $\mathsf{PQ}_{v/w}$.

Нижние оценки. Несмотря на наши ожидания, все три алгоритма показывают плохой результат в терминах конкурентности. Следующая теорема собирает вместе результаты Теорем 1, 2 и 3 из [29] и демонстрирует, что каждый их трех алгоритмов имеет конкурентное соотношение, которое как минимум линейно либо по W либо по V, что свидетельствует о том, что взаимодействие двух характеристик делает задачу управления буфером существенно более трудной.

Теорема. *Для буфера размера B, максимального объема работы W и максимальной ценности V:*

- $\mathsf{PQ}_{-w,v}$ *в точности V-конкурентен;*
- $\mathsf{PQ}_{v,-w}$ *не менее чем* $\left(W \cdot \frac{V-1}{V} - o(1)\right)$*-конкурентен; и*
- $\mathsf{PQ}_{v/w}$ *не менее чем* $\min\{V, W\}$*-конкурентен.*

Оценки представленный выше верны только для конкретных алгоритмов. В частности, из них не следует, что оптимального онлайн-алгоритма не существует. Чтобы доказать последний результат, требуется *общая* нижняя оценка, демонстрирующая такую последовательность входа, которая могла бы «обмануть» *любой* онлайн-алгоритм. В Разделе V [29] представлено несколько общих нижних оценок[†] которые полиномиальны либо по V либо по W. Разница между оценками в ограничениях, накладываемых на реализацию алгоритма: FIFO-порядок обработки, фиксированный размер буфера B, или то, что алгоритм должен быть основан на приоритетной очереди (PQ_f). Нижняя оценка лишенная ограничений намного более слабая, но тем не менее она показывает, что не существует оптимального онлайн алгоритма.

Теорема. *Для буфера размера B, максимального объема работы W и возможных ценностях 1 и $V > 1$, любой детерминированный онлайн-алгоритм не менее чем* $\left(1 + \frac{V-1}{V^2} - O\left(\frac{1}{W}\right)\right)$*-конкурентный.*

Частный случай двух ценностей (верхняя оценка). Нижние оценки говорят нам, что в общем случае алгоритмы, основанные на некоторых естественных приоритетах далеки от оптимальных, и даже от общих нижних оценок. Следующий шаг состоит в том, чтобы добавить какие-либо разумные ограничения в задачу. Один из вариантов — это предположить, что существуют лишь *два допустимых значения ценности*, т. е. либо $v(p) = 1$, либо $v(p) = V$. Данный частный случай соответствует ситуации, когда все пакеты поделены на «обычные» $(w \mid 1)$ и «золотые» высокоприоритетные $(w \mid V)$.

[†]Не являются частью данной диссертации.

Теорема. *Для буфера размера B, максимального объема работы W и возможных ценностях* 1 *и* V*:*

1. $\mathtt{PQ}_{-w,v}$ *как минимум V-конкурентный;*
2. *если $W \geq V$, то* $\mathtt{PQ}_{v/w}$ *как минимум V-конкурентный;*
3. $\mathtt{PQ}_{v,-w}$ *как минимум* $\left(\frac{W}{V} + o(1)\right)$*-конкурентный.*

Во-первых, заметим, что если $W < V$, то $\mathtt{PQ}_{v/w}$ ведет себя в точности как $\mathtt{PQ}_{v,-w}$, т.к. всякий $(w \mid V)$ лучше любого $(w \mid 1)$. Во-вторых, оценка для $\mathtt{PQ}_{v,-w}$ стала значительно менее строгой, и в ней есть некоторая интуиция: если решение обрабатывать некоторый более ценный пакет было неверным, то размер проигрыша в один шаг времени на должен превышать $\frac{V}{W}$ против $\frac{1}{1}$. Оказывается, данная интуиция (почти) верна.

Теорема. *Для буфера размера B, максимального объема работы W и возможных ценностях* 1 *и* V*,* $\mathtt{PQ}_{v,-w}$ *не более чем* $\left(1 + \frac{W+2}{V}\right)$*-конкурентный.*

Доказательство данного результата основано на индуктивном рассуждении, схожим с таковым из [9], которое сравнивает отсортированные по приоритету последовательности пакетов в буферах $\mathtt{PQ}_{v,-w}$ и $\mathtt{OPT}$. Нововведение состоит в построении отображения между "обычными" $(w \mid 1)$ пакетами, обработанными $\mathtt{OPT}$ и шагами времени, в течение которых $\mathtt{PQ}_{v,-w}$ обрабатывал пакеты. Учитывая V-конкуретность $\mathtt{PQ}_{-w,v}$, $\mathtt{PQ}_{v,-w}$ и $\mathtt{PQ}_{-w,v}$ вместе подходят очень близко к нижней оценке[†] $\min\{V, W/V\}$ для алгоритмов, основанных на приоритетной очереди (Теорема 6 в [29]).

Случай бета-выталкивания. В предыдущих рассуждениях решение о выбрасывании только что прибывшего пакета и решение о выталкивании уже принятого рассматривались как равноценные. Однако, существуют причины предпочитать первое второму (к примеру, второе использует больше ресурсов устройства). В работе [9] была рассмотрена модель со стоимостью копирования, призванная учесть данное обстоятельство, вводя штраф за каждый принятый пакет в размере α. Ориентируясь на алгоритм, предложенный в [9], в данной диссертации рассмотрена модифицированная версия алгоритма $\mathtt{PQ}_f$ — $\mathtt{PQ}_f^\beta$, которая выталкивает пакет только в том случае, если новый в β раз лучше ($\beta > 1$).

Теорема. *Для буфера размера B, максимально объема работы W и возможных ценностях* 1 *и V:*

(1) $\mathtt{PQ}_{-w,v}^\beta$ *не менее чем V-конкурентный как в общем случае, так и в случае с двумя ценностями;*

(2) $\mathtt{PQ}_{v/w}^\beta$ *не менее чем* $\min\{V, W\}$*-конкурентный в общем случае, и как минимум V-конкурентный в случае с двумя ценностями если $\beta W \geq V$;*

(3) $\mathtt{PQ}_{v,-w}^\beta$ *не менее чем* $\left(\frac{(V-1)}{V}W - o(1)\right)$*-конкурентный в общем случае, и не менее чем* $\left(\frac{W}{V} + o(1)\right)$*-конкурентный в случае с двумя ценностями.*

Резюме с результатами. Таблица 1 содержит все теоретические результаты, представленные в [29]. В экспериментальном исследовании, основанном на CAIDA [38], пропускная способность $\mathtt{PQ}_{v/w}$ в основном не превышала 90% от оптимальной (мы использовали границу сверху) и $\mathtt{PQ}_{v,-w}$

Алгоритм	Общий случай	Случай двух ценностей	
Общие нижние оценки			
Онлайн-алгоритм	$\frac{\min\{\sqrt[2B]{W}, \sqrt[B]{V}\}-1}{2}$†	$1+\frac{V-1}{V^2}-O\left(\frac{1}{W}\right)$	
Приоритетная очередь	$\sqrt{W}$†	$\min\{V, W/V\}$†	
FIFO онлайн-алгоритм	$\left(\frac{\sqrt{W}}{B}+1-\frac{1}{B}\right)$†	$\frac{V+B-1}{W+B-1}$†	
Верхние и нижние оценки для конкретных алгоритмов			
Алгоритм	Нижняя оценка	Нижняя оценка	Верхняя оценка
$\mathsf{PQ}_{-w,v}$, $\mathsf{PQ}^{\beta}_{-w,v}$	V	V	V
$\mathsf{PQ}_{v,-w}$, $\mathsf{PQ}^{\beta}_{v,-w}$	$\frac{W(V-1)}{V}-o(1)$	$\frac{W}{V}+o(1)$	$1+\frac{W+2}{V}$
$\mathsf{PQ}_{v/w}$, $W \geq V$	V	V	
$\mathsf{PQ}^{\beta}_{v/w}$, $\beta W \geq V$	V	V	
$\mathsf{PQ}_{v/w}$, $W < V$	W	$\frac{W}{V}+o(1)$	$2+\frac{2}{V}$

Таблица 1: Результаты: нижние и верхние оценки.

показывал схожую эффективность при наличии только двух возможных ценностей пакета (см. Рисунок 5 в [29]).

4.2. Классификация пакетов: базовая функция в обработке одного пакета

Как было указано ранее, *классификация пакетов* является одной из ключевых функций процесса обработки одного пакета, и она напрямую влияет на производительность сетевого устройства. Задачей классификации пакетов является распознавание типа *(класса)* пакета с целью применения к нему специфичных для класса действий. Классификация полезна и как форма обработки данных сама по себе, и как способ распознавания отдельных потоков данных.

Входом для процесса классификации служит *заголовок* (англ. *header*) пакета H, представляющий собой последовательность бит $(h_1, \ldots, h_w)$ над $\{0, 1\}$ алфавитом, а выходом является *действие* (англ. *action*), которое должно быть применено к пакету. Упомянутое выше w называется *шириной классификации* (англ. *classification width*). Результат классификации определяется *классификатором пакетов* (англ. *packet classifier*) $\mathcal{K} = (R_1, \ldots, R_N)_{<}$ — упорядоченным (приоритезированным) по $<$ множеством правил. Каждое правило R_i состоит из *фильтра* F_i и *действия* A_i. Фильтр определяет ограничения на заголовок пакета, представленное в виде последовательности тернарных бит $(f_1, \ldots, f_w)$ над алфавитом $\{0, 1, *\}$, где $*$ обозначает «не важно». Говорят, что заголовок $(h_1, \ldots, h_w)$ *удовлетворяет* (англ. *matches*) фильтру $(f_1, \ldots, f_w)$ (или содержащему его правилу) если и только если для любого i либо $h_i = f_i$, либо $f_i = *$. Чтобы классифицировать пакет, выполняется *запрос* (англ. *lookup*) к классификатору, и действие из правила, чей фильтр удовлетворяет заголовку, возвращается в качестве результата. Если же имеется два правила R и R', которые *пересекаются*, т. е. существует заголовок который удовлетворяет обоим, возвращается действие из правила имеющего более высокий приоритет. Игрушечный пример классификатора $\mathcal{K}$ представлен на Рисунке 2, более приоритетное правило всегда расположено выше.

Последующие рассуждения во многом связаны с преобразованием некоторого классификато-

†Не является частью данной диссертации.

$\mathcal{K}$	#1	#2	#3	#4	Действ.
R_1	0	1	0	0	A_1
R_2	0	$*$	$*$	$*$	A_2
R_3	1	0	1	$*$	A_3
R_4	1	$*$	0	$*$	A_4

(а) Исходный классификатор $\mathcal{K}$

$\mathcal{K}^B$	#1	#2	#3	#4	Действ.
R_1^B	0	0	1	0	A_1
R_2^B	0	$*$	$*$	$*$	A_2
R_3^B	1	1	0	$*$	A_3
R_4^B	1	0	$*$	$*$	A_4

(б) А B-переупорядочивание $\mathcal{K}$, где $B = (1, 3, 2, 4)$

Рис. 2: Пример классификатора с четырьмя правилами и шириной заголовка пакета $w = 4$.

ра $\mathcal{K}$ в другой классификатор $\mathcal{K}'$. При этом строго необходимо, чтобы преобразование не меняло поведения $\mathcal{K}$, другими словами, $\mathcal{K}$ и $\mathcal{K}'$ должны быть *эквивалентны*. Формально, два классификатора $\mathcal{K}$ и $\mathcal{K}'$ являются эквивалентными если и только если для каждого заголовка запросы к $\mathcal{K}$ и $\mathcal{K}'$ возвращают один и тот же результат.

Раздел 4.2.1 дает обзор метода, опубликованного в статье «General ternary bit strings on commodity longest-prefix-match infrastructure» [31], и соответствующего второму из результатов перечисленных в Разделе 2. Следующий за ним Раздел 4.2.2 описывает статью под названием «How to implement complex policies on existing network infrastructure» [33], где представлен третий результат из Раздела 2.

4.2.1. Представление обобщенных тернарных правил на LPM инфраструктуре

Классификатор пакета без каких либо дополнительных ограничений называется *тернарным* классификатором. Тернарные классификаторы наиболее выразительны, они встречаются в современных языках для обработки пакетов (к примеру, [40]) и необходимы для некоторых специальных приложений [41]. К сожалению, программные реализации тернарных классификаторов неэффективны: они требуют либо слишком много памяти, либо времени [42]; решения на базе специализированного оборудование *(TCAM)*, будучи чрезмерно дорогими и энергетически затратными, не присутствуют в должном объеме на сетевых устройствах общего потребления. В то же время, как только классификатор удовлетворяет LPM ограничениям, которые определены далее, представить данный классификатор эффективно сразу становится проще, поэтому LPM классификация широко доступна. Формально, *LPM классификатор* должен удовлетворять следующим свойствам: (1) все «$*$» должны идти после всех «0» and «1» (как, например, правила R_2 и R_3 на Рисунке 2(а)), другими словами, классификатор должен быть *префиксным*; (2) среди любых двух пересекающихся правил, то, которое имеет более длинный префикс, всегда должно иметь больший приоритет (к примеру, правила R_2^B и R_3^B на Рисунке 2(б) нарушают данное требование).

Несоответствие между требованиями пользовательских приложений и возможностями инфраструктуры требует решения, которое бы позволило представить любой *тернарный* классификатор $\mathcal{K}$ на *LPM* инфраструктуре, поддерживающей ширину классификации не более w_{LPM} (обычно, w_{LPM} составляет либо 32 либо 128 бит).

Способ предложенный далее не зависит от конкретной реализации LPM классификации. Кратко он состоит в следующем: сначала ширина классификации уменьшается до w_{LPM}, затем, ис-

пользуя оригинальное свойство префиксной переупорядочиваемости, результат преобразуется в префиксный классификатор, и, наконец, правила модифицируются таким образом, чтобы удовлетворять LPM приоритетам. Главный вклад данной работы состоит во втором шаге, с которого начнется дальнейшее изложение.

Префиксная переупорядочиваемость. Сетевые устройства способны конструировать сложные ключи для запросов к классификатору, в частности, они способны изменять порядок бит в заголовке. Переупорядочивание битов в заголовке позволяет тем же самым образом переупорядочить биты в классификаторе, что, в свою очередь, может превратить классификатор в префиксный. Формально, пусть $B = (b_1, b_2, \ldots, b_k)$, где $k \leq w$ и $1 \leq b_i \leq w$, — некоторая последовательность различных индексов бит, задающая их новый порядок. Тогда, для заголовка $H = (h_1, h_2, \ldots, h_w)$ и фильтра $F = (f_1, f_2, \ldots, f_w)$ можно определить $H^B = (h_{b_1}, h_{b_2}, \ldots, h_{b_k})$ и $F^B = (f_{b_1}, f_{b_2}, \ldots, f_{b_k})$, и, соответственно, для правила $R = (F, A)$ положить $R^B = (F^B, A)$. *B-переупорядочиванием* классификатора B называется классификатор $\mathcal{K}^B = (R_1^B, R_2^B, \ldots, R_N^B)$, при каждом запросе в котором, заголовок H заменяется на H^B. К примеру, префиксное $(1, 3, 2, 4)$-переупорядочивание классификатора на Рисунке 2(а) представлено на Рисунке 2(б). Несложно увидеть, что если B — это перестановка, то классификаторы $\mathcal{K}^B$ и $\mathcal{K}$ эквивалентны. Основной вопрос в том, существует ли такая перестановка B множества $\{1, \ldots, w\}$, что $\mathcal{K}^B$ — это префиксный классификатор, т. е. является ли $\mathcal{K}$ *префиксно-переупорядочиваемым*.

Для правила $R = (F, A)$ оказывается полезным рассмотреть множество индексов i таких, что $f_i \neq *$, данное множество обозначается $\mathrm{exact}(R)$. К примеру, на Рисунке 2(а) $\mathrm{exact}(R_4) = \{1, 3\}$. Любопытное наблюдение состоит в том, что если $\mathcal{K}^B$ префиксный, то для любого $R \in \mathcal{K}$ биты из $\mathrm{exact}(R)$ должны в B идти перед битами из $\{1, \ldots, w\} \setminus \mathrm{exact}(R)$; иначе, правило R^B не будет префиксным. Это наблюдение накладывает определенные ограничения на B, которые иногда могут конфликтовать друг с другом, например, в случае с классификатором из $F_1 = (0\ *)$ и $F_2 = (*\ 1)$. В общем случае, если $\mathcal{K}$ содержит два правила R и R' такие, что ни $\mathrm{exact}(R) \subseteq \mathrm{exact}(R')$, ни $\mathrm{exact}(R') \subseteq \mathrm{exact}(R)$ не выполнены, то $\mathcal{K}$ точно *не* префиксно-переупорядочиваемый. Первый результат данного раздела говорит о том, что это условие к тому же достаточно, и, более того, его легко проверить; ниже $\mathrm{exact}(\mathcal{K}) = \{\mathrm{exact}(R) : R \in \mathcal{K}\}$.

Теорема (критерий цепи)**.** *Классификатор $\mathcal{K}$ является префиксно-переупорядочиваемым если и только если для любых $E_1, E_2 \in \mathrm{exact}(\mathcal{K})$ либо $E_1 \subseteq E_2$, либо $E_2 \subseteq E_1$ верно, т. е.,* $\mathrm{exact}(\mathcal{K})$ *можно выстроить в "цепь": $E_{i_1} \subseteq E_{i_2} \subseteq \ldots \subseteq E_{i_{|\mathrm{exact}(\mathcal{K})|}}$. Перестановку индексов можно найти за время $O(|\mathcal{K}| \cdot w)$ (если она существует).*

Как только перестановка B, являющаяся свидетелем префиксно-переупорядочиваемости $\mathcal{K}$, найдена, остается лишь преобразовать $\mathcal{K}^B$ в LPM классификатор, что можно сделать с помощью основанном на боре алгоритме `PrefixToLPM` за время $O(|\mathcal{K}| \cdot w)$ (Теорема 2 в [31]). Стоит заметить, что ни одно из преобразований не увеличивает числа правил.

Далее будут рассмотрены два подхода решающие проблему представления классификаторов, не являющихся префиксно-переупорядочиваемыми. Первый подход ослабляет свойство префиксной переупорядочиваемости и затем пользуется возможностью параллельного выполнения запро-

$\mathcal{K}$	$\#1$	$\#2$	$\#3$	$\#4$	Действ.
R_1	0	0	0	$*$	A_1
R_2	0	0	1	$*$	A_2
R_3	$*$	1	0	0	A_3
R_4	0	0	$*$	$*$	A_4
R_5	$*$	0	1	$*$	A_5
R_6	$*$	1	0	$*$	A_6
R_7	$*$	0	$*$	$*$	A_7

(а) Исходный классификатор $\mathcal{K}$

$\mathcal{K}_1$	$\#1$	$\#2$	$\#3$	$\#4$	Действ.
R_1	0	0	0	$*$	A_1
R_2	0	0	1	$*$	A_2
R_4	0	0	$*$	$*$	A_4
R_7	$*$	0	$*$	$*$	A_7

$\mathcal{K}_2$	$\#1$	$\#2$	$\#3$	$\#4$	Действ.
R_3	$*$	1	0	0	A_3
R_5	$*$	0	1	$*$	A_5
R_6	$*$	1	0	$*$	A_6

(б) Префиксно-переупорядочиваемое разбиение $\mathcal{K}$

Рис. 3: Пример классификатора, не являющегося префиксно-переупорядочиваемым

сов, а второй опирается на дополнительные правила.

Минимальное групповое представление. Предположим, что правила классификатора можно разбить на префиксно-переупорядочиваемые группы. Тогда, каждая группа может быть преобразована в эквивалентный LPM классификатор, используя методы изложенные выше. Так как современные сетевые устройства могут совершать несколько LPM-запросов на скорости интерфейса, можно сделать запрос к каждому из сконструированных классификаторов и вернуть результат с наивысшим приоритетом в качестве ответа. К примеру, классификатор $\mathcal{K}$ на Рисунке 3(а) не является префиксно-переупорядочиваемым, и, тем не менее, он может быть разбит на $\mathcal{K}_1$ и $\mathcal{K}_2$ как на Рисунке 3(б). Оба классификатора, и $\mathcal{K}_1$, и $\mathcal{K}_2$ префиксно-переупорядочиваемы, следовательно, как только они будут преобразованы в LPM форму, всего-лишь два LPM запроса будет достаточно, чтобы представить $\mathcal{K}$. Вследствие того, что число запросов, осуществляемых на скорости интерфейса, ограничено, желательно минимизировать число групп в результате.

Задача (MinGR)**.** *Найти разбиение множества правил заданного классификатора $\mathcal{K}$ на минимальное число дизъюнктных префиксно-переупорядочиваемых групп.*

Ключевой шаг к решению задачи MinGR состоит в том, чтобы перейти от разбиения $\mathcal{K}$ к разбиению $\mathrm{exact}(\mathcal{K})$. Из критерия цепи известно, что подмножество $\mathcal{K}' \subseteq \mathcal{K}$ префиксно-переупорядочиваемо тогда и только тогда, когда $\mathrm{exact}(\mathcal{K}')$ — цепь. Отсюда следует, что имея решение для задачи MinGR можно построить *покрытие цепями* $\mathrm{exact}(\mathcal{K})$ того же размера. И обратно, из любого покрытия $\mathrm{exact}(\mathcal{K})$ цепями можно получить разбиение $\mathcal{K}$, имеющее такое же количество элементов, каждый их которых префиксно-переупорядочиваемый. Для этого нужно лишь сгруппировать правила в соответствии с элементами покрытия. Таким образом, задача MinGR оказывается эквивалентной задаче минимального покрытия $\mathrm{exact}(\mathcal{K})$ цепями.

Заметим, последняя задача выражена исключительно в терминах *частичного порядка* $\mathrm{exact}(\mathcal{K})$ с отношением $\subseteq$, и, оказывается, существует алгоритмическое решение [32] задачи покрытия цепями, работающее за время $O(|\mathrm{exact}(\mathcal{K})|^{5/2})$. Добавляя к этому время на построение частичного порядка и восстановления разбиения правил, получаем алгоритм `MinGroupPartition`. На практике стоит ожидать (и это было подтверждено экспериментально), что $|\mathrm{exact}(\mathcal{K})| \ll |\mathcal{K}|$.

Теорема. *Алгоритм* `MinGroupPartition` *строит оптимальное решение для задачи* MinGR *за время* $O(|\operatorname{exact}(\mathcal{K})|^{5/2} + |\mathcal{K}|^2 w)$.

Тем не менее, число групп, построенное `MinGroupPartition` и, соответственно, число необходимых запросов может быть слишком велико для работы на скорости интерфейса.

Теорема. *Существует классификатор* $\mathcal{K}$ *имеющий* $|\mathcal{K}| = O(\frac{1}{\sqrt{w}} 2^{w/2})$ *правил такой, что оптимальное решение задачи* MinGR *потребует ровно* $|\mathcal{K}|$ *групп.*

Несмотря на то, что вырожденный случай, подобный тому из предыдущей теоремы, вряд ли возникнет на практике, *небольшой* набор правил может оказаться «неприятным» для префиксной переупорядочиваемости и приведет к тому, что ответ полученный `MinGroupPartition` окажется бесполезным. Решением данной проблемы является «смешанное» представление, в котором некоторая часть классификатора реализуется с помощью традиционных (не LPM) методов (к примеру, небольшого TCAM). Задачей в таком случае является минимизация размера этой части при заданном ограничении на число LPM групп.

Задача (MaxCov)**.** *Даны классификатор* $\mathcal{K}$ *и константа* $\beta > 0$*, требуется разбить наибольшее подмножество правил* $\mathcal{K}$ *на не более чем* β *префиксно-переупорядочиваемых групп.*

Для того, чтобы решить задачу MaxCov, алгоритм `MaxCoveragePartition` слегка видоизменяет предложенный в [32] граф, добавляя взвешенные ребра, отвечающие за решения вида «представить все R такие, что $\operatorname{exact}(R) = E$, традиционным образом». В конечном итоге, алгоритм находит паросочетание минимального веса.

Теорема. *Алгоритм* `MaxCoveragePartition` *находит оптимальное решение задачи* MaxCov *за время* $O(|\operatorname{exact}(\mathcal{K})|^2 |\mathcal{K}| + |\mathcal{K}|^2 w)$.

Замечательное свойство двух рассмотренных алгоритмов состоит в том, что они решают проблему представления классификаторов, не являющихся префиксно-переупорядочиваемыми, не увеличивая число правил. Однако, иногда имеет смысл «обменять» некоторое количество памяти на меньшее число групп или меньший размер части, представленной традиционным образом.

Задача ($\mathrm{MaxCov\text{–}m}$)**.** *Даны классификатор* $\mathcal{K}$*, константа* $\beta > 0$ *и максимальное число правил* M*, требуется найти наибольшее подмножество* $\mathcal{K}' \subseteq \mathcal{K}$ *и эквивалентный* $\mathcal{K}'$ *классификатор* $\mathcal{K}^*$ *такой, что* $|\mathcal{K}^*| \leq M$ *и* $\mathcal{K}^*$ *может быть разбит на* β *префиксно-переупорядочиваемых групп.*

Из-за того, что в задаче $\mathrm{MaxCov\text{–}m}$ никак не ограничивается способ построения $\mathcal{K}^*$ из $\mathcal{K}'$, построить оптимальное решение становится трудно. По этой причине, была предложена эвристика, которая поочередно берет произвольное правило R, которое не поместилось ни в одну из групп $\mathcal{K}^*$, и пытается расширить $\operatorname{exact}(R)$, раскрывая $*$-биты в фильтре правила R. Заметим, что раскрытие бит приводит к увеличению числа правил.

Сужение ширины классификации. У нас все еще осталась проблема с тем, что ширина классификации w у классификатора $\mathcal{K}$ может превышать w_{LPM}, поддерживаемую инфраструктурой. Для сужения ширины классификации в [18] авторы предложили использовать свойство *дизъюнктности по правилам* (англ. *rule-disjointness*). Классификатор называется дизъюнктным по правилам на множестве индексов B, если и только если в $\mathcal{K}^B$ любые два правила не пересекаются. Полезный

факт состоит в том, что если $\mathcal{K}$ дизъюнктный на B, то $\mathcal{K}$ эквивалентен $\mathcal{K}^B$, в котором в каждое правило добавлен тест на ложно-положительное совпадение с соответствующим правилом из $\mathcal{K}$. Таким образом, ширина классификации может быть уменьшена как только $|B| < w$. Важное замечание состоит в том, что дизъюнктность по правилам и префиксно-переупоядочиваемость не конфликтуют друг с другом. Однако, остается открытым вопрос в том, какое свойство нужно удовлетворять первым для достижения наилучшего результата.

Реализация и экспериментальная проверка. Перечисленные выше оптимизации были реализованы в независимом от платформы виде, используя инфраструктуру языка P4 [40]. Также, алгоритмы были протестированы на синтетических данных из набора *Classbench* [37]. Лучшие результаты показала эвристика, которая сначала разбивает правила на группы, дизъюнктные по правилам, а затем на префиксно-переупорядочиваемые группы, раскрывая по пути «$*$». В частности, удалось представить, используя лишь 32-битную LPM инфраструктуру от 80% до 99% правил для классификаторов основанных на списках контроля доступа (англ. *Access Control Lists (ACL)*), от 38% до 100% — для основанных на файерволах, и практически 100% — для *IP-Chain* (см. Таблицы I и II в [31]).

4.2.2. Представление сложных классификаторов

В предыдущем разделе задача представления сложных классификаторов на существующей инфраструктуре была рассмотрена с точки зрения одного сетевого устройства. Для того, чтобы справится с ограничениями на размер классификаторов по всей сети, [21] и [22] предложили идею «виртуального конвейера» для совместного использования ресурсов. В частности, [22] предложили задачу разложения классификатора по устройствам вдоль *пути сетевого потока* как базовый элемент в решении задачи для всей сети. Формально, для данного классификатора $\mathcal{K}$ *разложением* (англ. *splitting*) $\mathcal{K}$ является *последовательность* классификаторов, эквивалентная $\mathcal{K}$. Запрос в последовательность $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_l$ осуществляется через последовательность запросов в каждый из $\mathcal{K}_i$ по порядку, при этом результат i-ого запроса применяется к пакету сразу, т. е. перед тем, как выполнить $(i+1)$-ый запрос. Интуиция состоит в том, что если $\mathcal{K}$ представляет собой классификатор, который необходимо применить к данному потоку, тогда $\mathcal{K}_i$ должен быть установлен на i-ом устройстве вдоль пути данного потока. Наконец, *l-разложением* классификатора $\mathcal{K}$ называется разложение $\mathcal{K}$ размера l. Формальная задача следующая:

Задача (FlowSplit)**.** *Даны классификатор $\mathcal{K}$ и последовательность натуральных чисел $c_1, c_2, \ldots, c_l$, требуется найти l-разложение $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_l$ классификатора $\mathcal{K}$ такое, что $\mathcal{K}_i$ содержит не более чем c_i правил.*

Трудность задачи FlowSplit произрастает из пересекающихся правил, т. е. правил, которые могут удовлетворять одному и тому же заголовку. К примеру, если классификатор $\mathcal{K}$ изображенный на Рисунке 4(а) разложен произвольным образом, например, как на Рисунке 4(б), то эквивалентность может быть потеряна: заголовок $(1\ 0\ 1\ 0)$ совпадет дважды, сначала с правилом R_1 и затем с правилом R_2. Возможным решением может служить добавление к $\mathcal{K}_2$ **nop**-правил, которые ничего не делают, как на Рисунке 4(в).

$\mathcal{K}$	#1	#2	#3	#4	Действ.
R_1	$*$	$*$	1	0	A_1
R_2	1	0	$*$	$*$	A_2
R_3	0	0	$*$	$*$	A_3
R_4	$*$	$*$	1	1	A_4

(а) Исходный классификатор $\mathcal{K}$

$\mathcal{K}_1$	#1	#2	#3	#4	Действ.
R_1	$*$	$*$	1	0	A_1
R_3	0	0	$*$	$*$	A_3

$\mathcal{K}_2$	#1	#2	#3	#4	Действ.
R_2	1	0	$*$	$*$	A_2
R_4	$*$	$*$	1	1	A_4

(б) *Не* разложение $\mathcal{K}$

$\mathcal{K}'_2$	#1	#2	#3	#4	Действ.
R_1	$*$	$*$	1	0	**nop**
R_2	1	0	$*$	$*$	A_2
R_3	0	0	$*$	$*$	**nop**
R_4	$*$	$*$	1	1	A_4

(в) Вариация $\mathcal{K}_2$ дополненная **nop**-правилами чтобы сохранить эквивалентность

Рис. 4: Пример разложения классификатора

Существует два подхода к проблеме пересекающихся правил. Palette [21] раскрывает «$*$»-биты таким образом, чтобы правила можно было разложить на непересекающиеся группы, т. е. чтобы никакие $\mathcal{K}_i$ и $\mathcal{K}_j$, $i \neq j$, не имели бы правил, удовлетворяющих одному и тому же заголовку. Так как оптимизационная задача, возникающая при данном подходе, сложна вычислительно, Palette вынужден использовать эвристические алгоритмы. Второй подход, *One Big Switch* (OBS) [22], допускает пересечение правил, находящихся на разных устройствах. Для того, чтобы сконструировать $\mathcal{K}_i$ OBS выбирает многомерный параллелепипед r_i и строит $\mathcal{K}_i$ как «проекцию» на r_i всех оставшихся правил (не покрытых $\mathcal{K}_j$, $j < i$). Для сохранения эквивалентности, в каждый последующий классификатор $\mathcal{K}_j$, $j > i$, добавляется специальное высоко-приоритетное правило, отображающее r_i в **nop** (схожим с Рисунком 4 образом).

Булевая минимизация. Первый метод, предложенный в данной диссертации, берет за основу итеративный подход OBS, но позволяет $\mathcal{K}_i$ представлять произвольное подмножество правил, необязательно только ограниченное параллелепипедом. Ценой дополнительной гибкости является большее число **nop**-правил в последующих классификаторах.

Данный метод интенсивно пользуется *Булевой Минимизацией* (англ. *Boolean Minimization*) [43], и потому назван BM. BM содержит l шагов, i-ый шаг берет в качестве входа классификатор $\mathcal{K}^{(i-1)}$ ($\mathcal{K}^{(0)} = \mathcal{K}$) и, используя эвристику, являющуюся параметром метода, выбирает подмножество $\mathcal{K}'_i \subseteq \mathcal{K}^{(i-1)}$. Далее, в качестве $\mathcal{K}_i$ выбирается $\mathcal{K}^{(i-1)}$, в котором во всех *не* входящих в $\mathcal{K}'_i$ правилах действие заменено на **nop**, и затем применена булева минимизация. Если в результате $\mathcal{K}_i$ вмещается в c_i, то осуществляется переход к $(i+1)$-ому шагу, где $\mathcal{K}^{(i)}$ строится схожим с $\mathcal{K}_i$ образом, но в этот раз действия заменяются на **nop** в правилах *из* $\mathcal{K}'_i$. На Рисунке 4 изображен возможный результат работы BM. Заметим, что эвристика, выбирающее представляемое подмножество, может порождать несколько кандидатов для $\mathcal{K}'_i$.

Один бит метаданных. Можно показать (см. Раздел 4 в [33]) что все три подхода — BM, Palette, и OBS, не сравнимы в общем случае. Тем не менее, они обладают тремя общими недостатками: (1) *резкое увеличение требований к памяти* либо из-за **nop**-правил (OBS, BM), либо вследствие дублирования правил (Palette); (2) *медленное время работы* из-за сложности оптимизационных задач лежащих в основе; и (3) *отсутствие поддержки динамических полей*, т. е. таких полей, которые меняются в результате действий и затем используются для классификации. Метод, позволяю-

щий избежать все три недостатка с помощью лишь одного бита метаданных, представлен далее.

Рассмотрим наивный подход, в котором первые c_1 правил из $\mathcal{K}$ идут в $\mathcal{K}_1$, затем следующие c_2 правил — в $\mathcal{K}_2$, и так далее, пока не кончатся правила. Единственное, что может пойти не так, это совпавший в некотором $\mathcal{K}_i$ заголовок может ошибочно совпасть снова в $\mathcal{K}_j$, $j > i$, т.к. первое совпадение в данной схеме *всегда* правильное. В качестве элементарного решения повторного совпадения, в алгоритме `OneBit` к заголовку добавляется специальный бит **`matched`**, указывающий, совпал ли заголовок с каким либо более ранним правилом, и запрос к классификатору осуществляется только в том случае, если **`matched`** не выставлен. Кроме того, ко всем действиям в каждом из $\mathcal{K}_i$, добавляется выставление **`matched`** в единицу, а в $\mathcal{K}_1$ действием по-умолчанию (применяемое, если заголовок не совпал ни с одним правилом) осуществляется сброс **`matched`** бита. Следующая теорема говорит, что `OneBit` удается избежать всех недостатков предыдущих подходов.

Теорема. *Для экземпляра задачи* FlowSplit $(\mathcal{K}, \{c_i\}_i)$*, в которой* $\sum_i c_i \geq |\mathcal{K}|$*, алгоритм* `OneBit` *успешно находит решение за время* $O(|\mathcal{K}|)$*, которое остается корректным даже при наличии динамических правил, и в котором ровно* $|\mathcal{K}|$ *правил.*

Решение для всей сети. Алгоритм `OneBit` решает задачу представления классификатора только для одного потока и, соответственно, пути. По компьютерной сети же проходит множество потоков данных, и каждый из них классифицируется согласно своему набору правил и проходит свой путь. В [22] следующая задача была поставлена для реализации классификации внутри сети:

Задача (MultiFlowSplit)**.** *Дано множество узлов* V*, вместимости каждого узла* $c : V \to \mathbb{N}$*, и множество* k *пар из пути и классификатора* $(P_i, \mathcal{K}_i)$*, где* P_i *— это последовательность вершин* $(v^i_1, \ldots, v^i_{l_i})$*, требуется найти распределение вместимости* $a : V \times [k] \to \mathbb{N}$*, что* $\sum_{i=1}^{k} a(v,i) \leq c(v)$*, а также решения задач* FlowSplit *со входами* $\mathcal{K} = \mathcal{K}_i$ *и вместимостями* $c_j = a(v^i_j, i)$ *для каждого* $j = 1, \ldots, l_i$*.*

Для решения MultiFlowSplit, авторы `OBS` предложили итеративную эвристику, шаг за шагом уточняющая распределение вместимости между потоками и пытающаяся на каждой итерации решить получающиеся экземпляры FlowSplit, используя `OBS`. К сожалению, неизвестно никакой приемлемой оценки сверху на число итераций, т.к. неизвестно точного критерия того, удастся ли решить FlowSplit или нет. В противовес, простота подобного критерия для `OneBit` позволяет получить решение MultiFlowSplit в один шаг, используя сведение к задаче о максимальном потоке.

4.3. Эластичное выделение ресурсов

Эластичность, которую предлагают облачные технологии, увеличивает гибкость и сокращает операционные расходы. Чтобы извлечь из этого как можно больше пользы, объем выделенных ресурсов должен соответствовать текущей нагрузке настолько близко, насколько возможно. Так как перераспределить ресурсы мгновенно невозможно, требуется планирование ресурсов наперед. Традиционно, планирование было задачей пользователя, однако недавно появилась новая парадигма «плати-за-использование» в форме *бессерверных вычислений* [2, 3, 4], где пользователи платят только за фактически обработанные запросы. В работе «On demand elastic capacity planning for

service auto-scaling» [34] обсуждается вопрос разработки эффективного алгоритма выделения ресурсов в бессерверной постановке, что соответствует четвертому набору результатов Раздела 2.

Описание модели. Время в рассматриваемой модели предполагается дискретным. Хотя каждый запрос может требовать различных ресурсов для обработки (к примеру, память, процессор или пропускную способность сети), вся их совокупность моделируются как единая *виртуальная единица ресурса*. Каждый запрос r имеет свою ценность $v(r)$, нормализованную таким образом, чтобы $\min_r\{v(r)\} = 1$. Запрос r считается обработанным, если для него была выделена ровно одна единица ресурса ровно на один шаг времени, в случае чего он добавляет $v(r)$ к общей выручке. Операционные расходы учитываются следующим образом: стоимость выделения единицы ресурса равна α, а стоимость поддержания единицы ресурса в течение одного шага времени равна β. Наконец, каждый шаг времени t подразделяется на три фазы: (1) *прибытие* (англ. *admission*): прибывает множество новых запросов A_t; (2) *обработка:* выбирается множество запросов P_t, обрабатываемых на данном шаге, $|P_t|$ не должно превышать N_t, — числа выделенных единиц ресурса; (3) *предсказание:* планируется количество единиц ресурса N_{t+1} для следующего шага времени. Заметим, фазы (2) и (3) могут обрабатываться параллельно, однако, необходимо, чтобы предсказание происходило *до* следующего прибытия, чтобы дать время на выделение новых ресурсов. Множество запросов, из которых набирается P_t, зависит от того, возможно ли задерживать обработку запросов, и если да, то на сколько. Изучение фундаментального баланса между задержкой запросов и нехваткой ресурсов для их обработки — главная тема [34].

Целевая функция. Задача — минимизировать $\sum_t(\sum_{r\in P_t} v(r) - \beta N_t - c_t)$, где c_t — это стоимость изменения объема ресурсов с N_t на N_{t+1}. Здесь и далее предполагается линейная модель и стоимость выключения ресурса входит в стоимость выделения: $c_t = \max\{0, N_{t+1} - N_t\}$. В экономически значимой постановке, гарантий на прибыль (расходы) верных лишь в среднем недостаточно, поэтому в данной работе рассматривается худший случай в рамках конкурентного анализа [30]. Формально, *онлайн*-алгоритм ALG называется α-конкурентным, если для любой конечной последовательности запросов σ, достигнутое ALG значение целевой функции $\texttt{ALG}(\sigma)$ составляет как минимум $\texttt{OPT}(\sigma)/\alpha$, где $\texttt{OPT}(\sigma)$ — это значение целевой функции *оптимального оффлайн-алгоритма*. ALG называется *неконкурентным*, если такого α не существует.

Модель без буфера. Если запросы задерживать запрещено, т. е. $P_t \subseteq A_t$, то модель называется BLH (от англ. *BufferLess with Heterogeneous values*). К сожалению, даже если обработка запроса всегда приносит прибыль ($\alpha < 1 - \beta$), конкурентного алгоритма не существует:

Теорема. *Если $\alpha < 1 - \beta$, то в* BLH *любой детерминированный онлайн-алгоритм неконкурентен.*

Теорема выше пользуется неспособностью онлайн-алгоритма предсказать нужный объем ресурсов для первого шага времени, ведь запросов всегда может прийти значительно больше. Для того, чтобы сделать модель более реалистичной, далее вводится верхняя граница B на количество одновременно выделенных ресурсов, переходя таким образом к модели BBLH (от англ. *Bounded BufferLess setting with Heterogeneous values*). Тем не менее, так как B может быть сколько угодно большим, все еще не существует конкурентного алгоритма:

Теорема. *Если $\alpha < 1 - \beta$, то в* BBLH *любой детерминированный онлайн-алгоритм неконкурентен.*

Модель с буфером. Ориентируясь на неконкурентные результаты в отсутствие буфера, в модель вводится *буфер* размера B (совпадает с ограничением на объем выделенных ресурсов). Запросы, пришедшие в A_t, которые не были в тот же момент обработаны в течение шага t, могут быть помещены в буфер (не более B запросов одновременно) для обработки на последующих шагах. Рассматриваются два варианта модели: более общая с произвольными ценностями BH (от англ. *Buffered with Heterogeneous values*) для верхних оценок и с единичными ценностями BU (от англ. *Buffered with Uniform values*) для нижних оценок. Ни в одной из них оптимального онлайн-алгоритма не существует:

Теорема. *Если $\alpha < 1 - \beta$, то в* BU *любой детерминированный онлайн-алгоритм не менее чем $\left(2 - \frac{\alpha}{1-\beta}\right)$-конкурентен.*

Первый из алгоритмов, использующих буфер, называется NRAP (от англ. *Next Round Allocation Policy*).

Определение. NRAP *работает следующим образом:*

- при прибытии: *запросы принимаются до тех пор, пока есть место в буфере, если буфер полный, то запросы с более высокой ценностью всегда выталкивают из буфера запросы с более низкой;*
- при обработке: *на обработку отправляются либо запросы оставшиеся с предыдущего шага времени, либо те запросы, которые вытолкнули первых;*
- при предсказании: *предсказывается k единиц ресурсов на следующий шаг, где k — это число запросов, которые останутся в буфере в конце текущего шага времени (то есть, не считая обрабатываемых сейчас).*

Из определения следует, что NRAP никогда не выделяет больше ресурсов чем необходимо. Однако, из-за того, что новая единица ресурса может выделяться под каждый новый запрос, NRAP может потратить на выделение слишком много. Тем не менее, при условии $\alpha < 1 - \beta$ NRAP обладает константной конкурентностью.

Теорема. *Если $\alpha < 1 - \beta$, то в* BH *алгоритм* NRAP *не более чем $\left(2 \cdot \frac{1-\beta}{1-\alpha-\beta}\right)$-конкурентен.*

Условие $\alpha < 1 - \beta$ необходимо, чтобы NRAP оставался конкурентным, т.к. в общем случае NRAP делает одно выделение для каждого запроса. Если же стоимость выделения становиться выше и стремиться к 1, то конкурентность NRAP стремиться к бесконечности.

Теорема. *Если $\alpha < 1 - \beta$, то в* BU *алгоритм* NRAP *как минимум $\left(1 + \frac{1-\beta}{1-\alpha-\beta}\right)$-конкурентен.*

Чтобы эффективно работать со стоимостью выделения выше чем $1 - \beta$, параметризованный алгоритм ρ-AAP не выделяет единицу ресурса до тех пор, пока не накопит количество ожидающих ресурсов в буфере достаточное, чтобы покрыть обработкой стоимость выделения. Каждая единица ресурса, таким образом, имеет свой собственный набор *прикрепленных* запросов.

Определение. *Алгоритм ρ-*AAP *(от англ. Amortizing Allocation Policy) с параметром $\rho > 1$ работает следующим образом.*

- при прибытии: *входящие запросы принимаются до тех пор, пока количество свободных мест в буфере плюс число уже выделенных единиц ресурсов больше чем* $(B - \lceil \frac{\rho\alpha}{1-\beta} \rceil)/(\lceil \frac{\rho\alpha}{1-\beta} \rceil + 1) - \lceil \frac{\rho\alpha}{1-\beta} \rceil$ *(пусть для простоты положительное целое); в случае полного буфера выталкивать можно только те запросы, которые были приняты на текущем шаге времени.*

- при обработке: *для каждой выделенной единицы ресурсов выбирается один из* прикрепленных *к ней запросов.*

- при предсказании: *пока суммарная стоимость* неприкрепленных *запросов в буфере составляет как минимум* $\lceil \frac{\rho\alpha}{1-\beta} \rceil$*:*

  – *предсказать, что на следующем шаге времени понадобится дополнительная единица ресурсов* v' *;*

  – *прикрепить к* v' *как можно меньшее число неприкрепленных запросов, суммарной стоимостью как минимум* $\lceil \frac{\rho\alpha}{1-\beta} \rceil$*;*

  для каждой выделенной единицы ресурсов, у которой все еще есть прикрепленные запросы, предсказать, что она будет нужна на следующем шаге времени; те единицы ресурсов, которые обработали все свои прикрепленные запросы, больше не понадобятся.

Следующая теорема предоставляет гарантии конкурентности алгоритма ρ-AAP. Для простоты анализа, ρ-AAP предоставляется буфер на константу большего размера, по сравнению с OPT.

Теорема. *Если* $\alpha \geq (1-\beta)$*, то* ρ-AAP *с буфером* $(B + \lceil \frac{\rho\alpha}{1-\beta} \rceil)$ *не более чем* $\frac{\rho}{\rho-1}(\lceil \frac{\rho\alpha}{1-\beta} \rceil + 1)$*-конкурентен в* BH *и не менее чем* $(\lceil \frac{\rho\alpha}{1-\beta} \rceil + 1)$*-конкурентен в* BU *против* OPT *с буфером* B*.*

Анализ задержек. Для гарантий на качество обслуживания необходимо контролировать задержку (англ. *latency*) при обработке запросов. Если запрос r приходит на шаге времени $t_a(r)$, а покидает систему (обработанный или нет) на шаге времени $t_l(r)$, то задержка для данного запроса определяется как $\mathrm{lat}(r) = t_l(r) - t_a(r)$. Для алгоритма ALG есть два способа определить задержку: *задержка в худшем случае* $\mathrm{lat}(\mathtt{ALG}) = \sup_\sigma \max_{r \in \sigma} \mathrm{lat}(r)$; и *задержка в среднем* $\mathrm{lat}_{\mathrm{avg}}(\mathtt{ALG}) = \sup_\sigma (\sum_{r \in \sigma} \mathrm{lat}(r)/|\sigma|)$.

В то время, как задержка алгоритма NRAP оптимальна, ρ-AAP может удерживать запросы неопределенно долго, до тех пор пока не накопит достаточное их число. Чтобы иметь возможность оценить задержку для ρ-AAP, вводится предположение о том, что хотя бы один запрос приходит на каждом шаге времени (справедливое предположение для хоть сколько-нибудь нагруженной системы). Кроме того, рассматривается модифицированная версия ρ-AAP — ρ-AAPm, которая бросает все еще не прикрепленные запросы, если на данном шаге времени не приходит ни один запрос (что означает конец последовательности).

Теорема. *В модели* BH*: (1)* $\mathrm{lat}(\mathtt{NRAP}) = 1$*; (2)* $\mathrm{lat}_{\mathrm{avg}}(\mathtt{NRAP}) = 1$*; (3)* $\mathrm{lat}(\rho\text{-}\mathtt{AAPm}) = \left\lceil \frac{\rho\alpha}{1-\beta} \right\rceil$*; и, наконец, (4)* $\mathrm{lat}_{\mathrm{avg}}(\rho\text{-}\mathtt{AAPm}) \geq \frac{1}{2}\lceil \frac{\rho\alpha}{(1-\beta)} \rceil + \frac{1}{2}$.

Модель с ограниченной задержкой. Вместо того, чтобы оценивать задержку алгоритмов *a posteriori*, максимально допустимая задержка может быть встроена в модель. А именно, в модели BD (от англ. *Bounded Delay*) у каждого запроса есть *дедлайн* обработки $d(r) \geq 1$, т. е., если запрос r приходит на шаге времени $t_a(r)$, то он *автоматически* бросается в конце шага $t_a(r) + d(r)$. Объем единиц ресурсов и размер буфера положены неограниченными. Тем не менее, существует неявное ограничение для буфера: максимальное количество запросов в шаг времени умножить на максимальный дедлайн. При слишком большой стоимости выделения, конкурентных алгоритмов не существует (в отличие от предыдущей модели).

Теорема. *Если $\alpha > 1-\beta$, то при* BD *любой детерминированный онлайн-алгоритм неконкурентен.*

Если буфер не ограничен, то алгоритм NRAP никогда не выталкивает запросы и, следовательно, имеет возможность обработать все запросы с предыдущего шага. Как результат, NRAP — конкурентный с константным множителем:

Теорема. *Если $\alpha < 1-\beta$, то в* BD *любой детерминированный алгоритм не менее чем $\left(1 + \frac{\alpha}{2(1-\alpha-\beta)}\right)$-конкурентный, а* NRAP *— не более чем $\left(1 + \frac{\alpha}{1-\beta-\alpha}\right)$-конкурентный*

Последняя рассмотренная модель — LBD (от англ. *Limited resource Bounded Delay*), дополняет BD верхней границей на число одновременно выделенных ресурсов B. Эффект данного изменения на конкурентность относительно незначительный.

Теорема. *Если $\alpha < 1 - \beta$, то в* LBD *любой детерминированный онлайн-алгоритм как минимум $(2 + \frac{\alpha}{1-\beta-\alpha})$-конкурентный, в то время, как* NRAP *не более чем $3(1 + \frac{\alpha+\beta}{1-\beta-\alpha})$ и не менее чем 3-конкурентный.*

Экспериментальная проверка используя искусственно-порожденные данные (см. Раздел VII в [34]) показала, что покуда стоимость выделения α остается низкой, а размер буфера — большим, предложенные алгоритмы побеждают простые подходы, основанные на обучении. С ростом же стоимости выделения ρ-AAP начинает заметно проигрывать из-за чрезмерной осторожности, не используя полностью все доступные ресурсы.

4.4. Оптимизация задач вычисления-агрегирования

Одним из важных классов приложений, анализирующих большие данные, являются системы вычисления-агрегирования, в которых несколько распределенных кусков данных должны быть агрегированы в заданном месте. Подобные системы оптимизируются под уменьшение задержки и увеличение пропускной способности, но процесс оптимизации редко учитывает особенности и ограничения компьютерной сети, порождая такие проблемы как TCP-incast [25] и перегрузка сетевых каналов. К сожалению, совместная оптимизация, учитывающая подобные ограничения имеет слишком много степеней свободы, как следствие, мы вводим две *независимые* фазы: (1) поиск наиболее дешевого плана агрегирования; и (2) выполнение агрегаций и передач, оптимизируя в процессе желаемые целевые функции. В статье под названием «Formalizing Compute-Aggregate Problems in Cloud Computing» упор сделан на первую фазу, и, в частности, на поиск таких свойств задач вычисления-агрегирования, которые позволят построить эффективный план агрегаций, — последний результат Раздела 2.

Сеть моделируется как неориентированный граф $G = (V, E)$, где V — это множество вычислительных узлов, а E — множество соединяющих их каналов. Задача вычисления-агрегирования задается целевой вершиной $t \in V$, в которой должен быть агрегирован конечный результат, и множеством начальных «кусочков» (англ. *chunks*) данных $C = \{\boxed{x_1}, \ldots, \boxed{x_n}\}$. У каждого кусочка x есть *текущее местоположение*, обозначаемое как $\mathrm{v}(\boxed{x}) \in V$, и *размер*, обозначаемый как $\mathrm{size}(\boxed{x}) \in \mathbb{R}_{\geq 0}$. Для того, чтобы гибко объединить разнообразие целевых функций (время задержки, пропускную способность или уменьшение уровня загрузки каналов), вводится функция *стоимости* передачи данных через канал $c : E \to \mathbb{R}_{\geq 0}$, т. е. передача $\boxed{x}$ через $e \in E$ будет стоить $c(e) \cdot \mathrm{size}(\boxed{x})$.

«Передавать в корень» не всегда оптимально. Способ обработки, который на данный момент доминирует, состоит в том, чтобы отправить все кусочки в корень t и произвести всю агрегацию там. Такой способ может быть неоптимальным или нереализуемым по следующим причинам: (1) для агрегаций может не хватить имеющейся памяти, если вследствие высоких требований к задержке она производится полностью в RAM; (2) все данные отправляются в корень одновременно, приводя к TCP-incast [25]; (3) чрезмерная стоимость передачи (как она была определена ранее). Ситуацию с первыми двумя пунктами можно частично улучшить, отправляя данные в t по очереди и агрегируя постепенно по мере появления новых данных. Для того, чтобы сделать порядок агрегации более гибким, операции агрегации должны быть *ассоциативными*: $\mathbf{aggr}(\boxed{x}, \mathbf{aggr}(\boxed{y}, \boxed{z})) = \mathbf{aggr}(\mathbf{aggr}(\boxed{x}, \boxed{y}), \boxed{z})$, и *коммутативными*: $\mathbf{aggr}(\boxed{x}, \boxed{y}) = \mathbf{aggr}(\boxed{y}, \boxed{x})$.

Промежуточные агрегации. Принцип, согласно которому *вычисления должны быть ближе к данным*, широко используется при обработке больших данных, чтобы уменьшить трафик в сети. В данной работе этот принцип расширяется до *агрегация должна быть ближе к данным* путем агрегации данных на *промежуточных узлах*. Это расширение формализуется следующим образом: *план агрегации* P — это последовательность операций $(o_1, o_2, \ldots, o_m)$, где каждая из o_i может быть либо $\mathbf{move}(\boxed{x}, v)$, перемещающая кусочек $\boxed{x}$ в вершину v, либо $\mathbf{aggr}(\boxed{x}, \boxed{y})$, сливающая два кусочка $\boxed{x}$ и $\boxed{y}$ расположенные в одной вершине, создавая новый кусочек $\boxed{xy}$. После того, как все операции из P обработаны, должен оставаться только один кусочек $\boxed{z}$ такой, что $\mathrm{v}(\boxed{z}) = t$.

Каждый план агрегации P имеет некоторую соответствующую стоимость передачи $\mathrm{cost}(P)$, которая складывается из стоимости каждой из операций в P, определенных следующим образом: $\mathrm{cost}(\mathbf{move}(\boxed{x}, v)) = \mathrm{size}(\boxed{x}) \cdot d(\mathrm{v}(\boxed{x}), v)$, где $d(u, v)$ — это наиболее дешевый $(u \rightsquigarrow v)$-путь согласно стоимости c; и $\mathrm{cost}(\mathbf{aggr}(\boxed{x}, \boxed{y})) = 0$, т.к. при этом не происходит никакой передачи.

Сравнивая со стратегией, перемещающей все в корень, агрегация в промежуточных узлах имеет несколько преимуществ: (1) уменьшается входящий трафик отдельных узлов, улучшая поведение транспортных протоколов (избегая TCP-incast); (2) уменьшается объем данных, агрегируемый на отдельных узлах, облегчая требования к памяти; а также (3) уменьшается общий объем передаваемых данных и, как результат, стоимость передачи. Кроме того, заметим, что планирование агрегации полностью отделено как от транспортной функции сети, ответственной за распределение передач во времени, так и от приложения, осуществляющего сами агрегации.

Функция размера агрегации. Для того, чтобы завершить формальную постановку задачи, необходимо определить $\mathrm{size}(\boxed{xy})$ для произвольной операции **aggr**($\boxed{x}$, $\boxed{y}$). Узнать точное его значение может быть невозможно в фазу планирования, т.к., во-первых, оно может зависеть от конкретных данных в $\boxed{x}$ и $\boxed{y}$, а, во-вторых, это может потребовать произвести саму агрегацию. В качестве баланса между точностью и практической реализуемостью от каждого приложения требуется предоставить приближенное значение в виде *функции размера агрегации* $\mu : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, такой, что следующее верно (в реальности лишь приближенно): $\mathrm{size}(\boxed{xy}) = \mu(\mathrm{size}(\boxed{x}), \mathrm{size}(\boxed{y}))$. Функция μ должна удовлетворять тем же ограничениям коммутативности и ассоциативности, что и **aggr**. Вот некоторые примеры функции μ: $\mu(a, b) = \mathrm{const}$ для поиска k лучших элементов; $\mu(a, b) = \min(a, b)$ или $\mu(a, b) = \max(a, b)$ при выборе «наилучшего» кусочка; $\mu(a, b) = a + b$ для конкатенации или сортировки; $\max(a, b) \leq \mu(a, b) \leq a + b$ для объединения множеств.

Задача (CAM)**.** *Даны связный неориентированный граф $G = (V, E)$, функция стоимости c, целевая вершина t, множество начальных кусочков данных C, и функция размера агрегации μ, задача* $\mathrm{CAM}[\mu]$ *состоит в нахождении плана агрегации P минимальной стоимости* $\mathrm{cost}(P)$.

Существует два важных частных случая CAM: CCAM и TCAM. В первом предполагается, что из соображений безопасности запрещено агрегировать кусочки в промежуточных узлах. В результате, можно считать, что множество вершин, в которых расположены изначальные кусочки, совпадает с V. Во втором случае, разнообразие топологии сети G ограничивается деревом.

Сложность и связь с деревом Штейнера минимального веса. Без ограничения на ассоциативность μ, полиномиального алгоритма с константным фактором аппроксимации не существует, если только $\mathrm{P} \neq \mathrm{NP}$. Доказательство следующего факта основано на сведении от задачи о рюкзаке.

Теорема. *При условии* $\mathrm{P} \neq \mathrm{NP}$ *не существует полиномиального аппроксимационного алгоритма для задачи* CAM *без ограничения на ассоциативность μ даже если G имеет всего две вершины.*

Для лучшего понимания связи между CAM и *задачей о дереве Штейнера минимального веса (англ. minimum Steiner tree problem* – MStT*)* полезно рассмотреть частный случай, в котором все изначальные кусочки имеют одинаковый размер и $\mu(x, x) = x$. При таком выборе μ всегда выгодно сливать расположенные в одной вершине кусочки, поэтому оптимальная агрегация будет происходить строго вдоль дерева. Заметим, стоимость плана агрегации равна суммарной стоимости дерева умножить на x, т. е. CAM в таком случае эквивалентна MStT. В общем же случае, нам придется заплатить множителем, равным отношению между максимальным и минимальным размером любого кусочка, который мы можем встретить в процессе работы. Воспользовавшись ассоциативностью и коммутативностью, мы можем выразить обе величины следующим образом: $W_C[\mu] = \max_{C' \subseteq C}\{\mu(C')\}$ и $w_c[\mu] = \min_{C' \subseteq C}\{\mu(C')\}$.

Теорема. *Если существует полиномиальный α-аппроксимирующий алгоритм для* MStT*, то существует и полиномиальный алгоритм решающий* $\mathrm{CAM}[\mu]$ *с коэффициентом аппроксимации* $\alpha \frac{W_C[\mu]}{w_C[\mu]}$.

Существует простой 2-аппроксимирующий алгоритм для MStT, основанный на минимальном остовном дереве в графе кратчайших путей графа G, а также намного более изощренный алгоритм, имеющий коэффициент аппроксимации $\alpha = \ln 4 + \epsilon \leq 1.39$ [44]. Кроме того, рассмотрев с

позиции дерева Штейнера два частных случая CAM, — CCAM и TCAM, можно заметить, что они соответствуют минимальном остовному дереву и единственном дереву Штейнера в дереве, соответственно. Оба можно найти за полиномиальное время, поэтому в теореме выше $\alpha = 1$ для CCAM и TCAM.

Следствие. *Существует полиномиальный алгоритм, решающий* $\mathrm{CCAM}[\mu]$ *и* $\mathrm{TCAM}[\mu]$ *на множестве кусочков C с коэффициентом аппроксимации* $\frac{W_C[\mu]}{w_C[\mu]}$.

Если нам известно, что $\min\{a,b\} \le \mu(a,b) \le \max\{a,b\}$, тогда $W_C[\mu] = W_C = \max_{c \in C}\{\mathrm{size}(c)\}$ и $w[\mu] = w = \min_{c \in C}\{\mathrm{size}(c)\}$, т. е. в таком случае коэффициент аппроксимации не зависит от конкретных значений μ:

Теорема. *Если* $\min\{a,b\} \le \mu(a,b) \le \max\{a,b\}$ *для любых* a, b *и существует полиномиальный* α*-аппроксимирующий алгоритм для* MStT*, то существует и полиномиальный алгоритм, решающий* $\mathrm{CAM}[\mu]$*, с коэффициентом аппроксимации* $\alpha\frac{W_C}{w_C}$.

Следствие. *Если* $\min\{a,b\} \le \mu(a,b) \le \max\{a,b\}$*, то существует полиномиальный алгоритм, решающий* $\mathrm{CCAM}[\mu]$ *и* $\mathrm{TCAM}[\mu]$ *с коэффициентом аппроксимации* $\frac{W_C}{w_C}$.

Наконец, когда $\mu(a,b) \ge a + b$, то нет никакой выгоды сливать кусочки, и поэтому мы можем рассматривать каждый кусочек независимо от других, что позволяет нам свести CAM к задаче поиска кратчайших путей из t.

Теорема. *Если* $\mu(a,b) \ge a + b$ *для любых* a, b*, то существует оптимальный полиномиальный алгоритм для* $\mathrm{CAM}[\mu]$*,* $\mathrm{CCAM}[\mu]$*, и* $\mathrm{TCAM}[\mu]$*; при этом для* $\mathrm{TCAM}[\mu]$ *время работы составляет* $O(|C| + |G|)$.

5. Заключение

Данная работа направлена на решение фундаментальных ограничений вычислительных инфраструктур, следуя целям и задачам, введенным в Разделе 1.2.

Два важных новых шага было сделано в направлении *обработки данных внутри сети*. Во-первых, чтобы поддержать новые целевые функции и новые характеристики данных во время обработки нескольких потоков, была проанализирована с точки зрения худшего случая производительность нескольких естественных алгоритмов управления буфером, основанных на приоритетных очередях (Раздел 4.1). В этом анализе были впервые рассмотрены одновременно две характеристики — ценность пакета и объем требующейся работы, — что позволило сделать оптимальный выбор управляющего алгоритма в новой постановке. В частности, был показан контринтуитивный факт: алгоритм, ориентирующийся на отношение ценности к работе, не всегда является лучшим выбором.

Во-вторых, было показано, что сложные классификаторы можно представлять на имеющейся инфраструктуре, используя: (1) оригинальные алгоритмы, преобразующие тернарные классификаторы в LPM (Раздел 4.2.1), (2) простую, но эффективную стратегию разделения ресурсов сети (Раздел 4.2.2). В результате выразительность, которая требуется от сети для обработки современных объемов данных, может быть доступна без чрезмерных расходов.

Далее, для *эффективных бессерверных вычислений* был исследован баланс между задержкой запросов и максимизацией прибыли. Для этого была разработана и использована новая формализация экономической модели в бессерверном случае (Раздел 4.3). В рамках данного исследования были разработаны два алгоритма, планирующие объем выделенных ресурсов. Для них были показаны гарантии производительности в худшем случае и проведен анализ задержек, к которым они приводят. Главная польза от полученных результатов состоит в улучшении эффективности одной из наиболее эластичных (т. е. гибких) вычислительных инфраструктур — бессерверных вычислений.

Наконец, для того чтобы извлечь пользу из *промежуточных агрегаций данных*, был разработан двухэтапный подход (см. Раздел 4.4). На первом этапе происходит оптимизация под заданные ограничения бюджета, используя только информацию, необходимую для эффективного построения плана агрегации. Желаемые цели (к примеру, минимизация задержки или максимизация пропускной способность) учитываются косвенно во время первого этапа и оптимизируются во время второго уже компьютерной сетью. Разделение двух этапов позволяет сетевому транспорту быть менее «реактивным» упрощая его задачу до распределения агрегаций во времени согласно построенному плану. Использование промежуточных агрегаций в данном плане дает дополнительные преимущества, решая или снижая актуальность многих проблем, связанных с агрегацией, например проблемы TCP-incast или нехватки памяти.

Заметный прогресс был сделан в направлении обработки больших данных, раскрывая огромный потенциал современных компьютерных сетей, который ранее был удостоен лишь ограниченного внимания. Все вышеперечисленные результаты позволяют значительно улучшить обработку данных без радикальных изменений сетевой инфраструктуры.

Список литературы

[1] Hajirahimova Makrufa Sh., Aliyeva Aybeniz S. About Big Data Measurement Methodologies and Indicators // Intl. J. Modern Education and Computer Science. — 2017. — Vol. 9, no. 10. — P. 1–9.

[2] Amazon. AWS Lambda. — 2017. — https://aws.amazon.com/lambda/.

[3] Google. Cloud Functions. — 2017. — https://cloud.google.com/functions/.

[4] Microsoft. Azure Functions. — 2017. — https://azure.microsoft.com/en-us/services/functions/.

[5] Reserved, on demand or serverless: Model-based simulations for cloud budget planning / E. F. Boza [et al.] // 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM). — 2017. — Oct. — P. 1–6.

[6] Lloyd Wes [et al.]. Serverless Computing: An Investigation of Factors Influencing Microservice Performance // IC2E. — 2018. — P. 159–169.

[7] Dean Jeffrey, Ghemawat Sanjay. MapReduce: simplified data processing on large clusters // Commun. ACM. — 2008. — Vol. 51, no. 1. — P. 107–113.

[8] The Case for Evaluating MapReduce Performance Using Workload Suites / Yanpei Chen [et al.] // MASCOTS. — 2011. — P. 390–399.

[9] Providing Performance Guarantees in Multipass Network Processors / Isaac Keslassy [et al.] // IEEE/ACM Trans. Netw. — 2012. — Vol. 20, no. 6. — P. 1895–1909.

[10] Online Scheduling FIFO Policies with Admission and Push-Out / Kirill Kogan [et al.] // Theory of Computing Systems. — 2016. — Feb. — Vol. 58, no. 2. — P. 322–344. — URL: https://doi.org/10.1007/s00224-015-9626-4.

[11] Andelman Nir, Mansour Yishay, Zhu An. Competitive Queueing Policies for QoS Switches // Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. — SODA '03. — Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2003. — P. 761–770. — URL: http://dl.acm.org/citation.cfm?id=644108.644235.

[12] Kesselman Alex, Kogan Kirill, Segal Michael. Improved Competitive Performance Bounds for CIOQ Switches // Algorithmica. — 2012. — Jun. — Vol. 63, no. 1. — P. 411–424. — URL: https://doi.org/10.1007/s00453-011-9539-9.

[13] Kesselman Alex, Kogan Kirill, Segal Michael. Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing // Distributed Computing. — 2010. — Nov. — Vol. 23, no. 3. — P. 163–175. — URL: https://doi.org/10.1007/s00446-010-0114-4.

[14] Gupta P., McKeown N. Classifying packets with hierarchical intelligent cuttings // IEEE Micro. — 2000. — Jan. — Vol. 20, no. 1. — P. 34–41.

[15] Vamanan Balajee, Voskuilen Gwendolyn, Vijaykumar T. N. EffiCuts: Optimizing Packet Classification for Memory and Throughput // SIGCOMM Comput. Commun. Rev. — 2010. — Aug.. — Vol. 40, no. 4. — P. 207–218. — URL: http://doi.acm.org/10.1145/1851275.1851208.

[16] Fast packet classification using bloom filters / S. Dharmapurikar [et al.] // 2006 Symposium on Architecture For Networking And Communications Systems. — 2006. — Dec. — P. 61–70.

[17] Compressing Forwarding Tables for Datacenter Scalability / O. Rottenstreich [et al.] // IEEE Journal on Selected Areas in Communications. — 2014. — January. — Vol. 32, no. 1. — P. 138–151.

[18] SAX-PAC (Scalable And eXpressive PAcket Classification) / Kirill Kogan [et al.] // Proceedings of the 2014 ACM Conference on SIGCOMM. — SIGCOMM '14. — New York, NY, USA : ACM, 2014. — P. 15–26.

[19] Space and speed tradeoffs in TCAM hierarchical packet classification / Alexander Kesselman [et al.] // Journal of Computer and System Sciences. — 2013. — Vol. 79, no. 1. — P. 111 – 121. — URL: http://www.sciencedirect.com/science/article/pii/S0022000012001237.

[20] Bremler-Barr A., Hendler D. Space-Efficient TCAM-Based Classification Using Gray Coding // IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications. — 2007. — May. — P. 1388–1396.

[21] Kanizo Y., Hay D., Keslassy I. Palette: Distributing tables in software-defined networks // 2013 Proceedings IEEE INFOCOM. — 2013. — April. — P. 545–549.

[22] Optimizing the "One Big Switch" Abstraction in Software-defined Networks / Nanxi Kang [et al.] // Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies. — CoNEXT '13. — New York, NY, USA : ACM, 2013. — P. 13–24.

[23] Amazon AutoScaling. — http://aws.amazon.com/autoscaling/.

[24] Han R., et al. Lightweight Resource Scaling for Cloud Applications // CCGrid. — 2012. — P. 644–651.

[25] Understanding TCP Incast Throughput Collapse in Datacenter Networks / Yanpei Chen [et al.] // Proceedings of the 1st ACM Workshop on Research on Enterprise Networking. — WREN '09. — New York, NY, USA : ACM, 2009. — P. 73–82.

[26] Camdoop: Exploiting In-network Aggregation for Big Data Applications / Paolo Costa [et al.] // Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). — San Jose, CA : USENIX, 2012. — P. 29–42.

[27] Map-reduce-merge: Simplified Relational Data Processing on Large Clusters / Hung-chih Yang [et al.] // Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. — SIGMOD '07. — New York, NY, USA : ACM, 2007. — P. 1029–1040. — URL: http://doi.acm.org/10.1145/1247480.1247602.

[28] Optimal communication structures for big data aggregation / W. Culhane [et al.] // 2015 IEEE Conference on Computer Communications (INFOCOM). — 2015. — April. — P. 1643–1651.

[29] Priority Queueing for Packets With Two Characteristics / P. Chuprikov [et al.] // IEEE/ACM Transactions on Networking. — 2018. — Feb. — Vol. 26, no. 1. — P. 342–355.

[30] Borodin Allan, El-Yaniv Ran. Online Computation and Competitive Analysis. — New York, NY, USA : Cambridge University Press, 1998. — ISBN: 0-521-56392-5.

[31] Chuprikov P., Kogan K., Nikolenko S. General ternary bit strings on commodity longest-prefix-match infrastructures // 2017 IEEE 25th International Conference on Network Protocols (ICNP). — 2017. — Oct. — P. 1–10.

[32] Fulkerson D. R. Note on Dilworths decomposition theorem for partially ordered sets // Proc. Amer. Math. Soc. — 1956.

[33] Chuprikov Pavel, Kogan Kirill, Nikolenko Sergey. How to Implement Complex Policies on Existing Network Infrastructure // Proceedings of the Symposium on SDN Research. — SOSR '18. — New York, NY, USA : ACM, 2018. — P. 9:1–9:7. — URL: http://doi.acm.org/10.1145/3185467.3185477.

[34] Chuprikov P., Nikolenko S., Kogan K. On demand elastic capacity planning for service auto-scaling // IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications. — 2016. — April. — P. 1–9.

[35] Lorido-Botran Tania, Miguel-Alonso Jose, Lozano Jose A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments // Journal of Grid Computing. — 2014. — Dec. — Vol. 12, no. 4. — P. 559–592.

[36] Formalizing Compute-Aggregate Problems in Cloud Computing / Pavel Chuprikov [et al.] // Structural Information and Communication Complexity / Ed. by Zvi Lotker, Boaz Patt-Shamir. — Cham : Springer International Publishing, 2018. — P. 377–391.

[37] Taylor David E., Turner Jonathan S. ClassBench: A Packet Classification Benchmark // IEEE/ACM Trans. Netw. — 2007. — Jun.. — Vol. 15, no. 3. — P. 499–511.

[38] for Internet Data Analysis CAIDA The Cooperative Association. — [Online] http://www.caida.org/.

[39] Zukerman M., Neame T., Addie R. Internet traffic modeling and future technology implications // INFOCOM. — Vol. 1. — 2003. — March. — P. 587–596 vol.1.

[40] P4: Programming Protocol-independent Packet Processors / Pat Bosshart [et al.] // SIGCOMM Comput. Commun. Rev. — 2014. — Jul.. — Vol. 44, no. 3. — P. 87–95.

[41] Fast Regular Expression Matching Using Small TCAM / Chad R. Meiners [et al.] // IEEE/ACM Trans. Netw. — 2014. — Vol. 22, no. 1. — P. 94–109.

[42] Gupta P., McKeown N. Packet classification on multiple fields // SIGCOMM. — 1999. — P. 147–160.

[43] Minimizing Disjunctive Normal Form Formulas and AC^0 Circuits Given a Truth Table / Eric Allender [et al.] // SIAM J. Comput. — 2008. — Vol. 38, no. 1. — P. 63–84.

[44] An Improved LP-based Approximation for Steiner Tree / Jaroslaw Byrka [et al.] // Proceedings of the Forty-second ACM Symposium on Theory of Computing. — STOC '10. — New York, NY, USA : ACM, 2010. — P. 583–592. — URL: http://doi.acm.org/10.1145/1806689.1806769.