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

На правах рукописи

Демьянюк Виталий Юрьевич

НОВЫЕ МЕТОДЫ, АЛГОРИТМЫ И ТЕОРЕТИЧЕСКИЕ ОЦЕНКИ РАБОТЫ АЛГОРИТМОВ В ДИЗАЙНЕ СЕТЕВЫХ ЭЛЕМЕНТОВ

РЕЗЮМЕ
диссертации на соискание ученой степени
кандидата компьютерных наук

Санкт-Петербург — 2020

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, Ariel University, Senior Lecturer

1. Введение

В этом разделе мы описываем основные задачи дизайна сетевых элементов и вводим основные цели диссертационного исследования.

1.1. Тема диссертации и её актуальность

Широкий спектр услуг, работающих на экспоненциально растущем числе взаимосвязанных сетевых устройств, делает сетевые операции более сложными, чем когда-либо. Новое сложное поведение в масштабах сети, разнообразие желаемых целей, а также повышение уровней масштабируемости требуют, чтобы сетевая инфраструктура была более интеллектуальной, выразительной и надежной. Обычно эти требования приводят к значительной сложности эксплуатации и увеличению стоимости сетевой инфраструктуры. Уменьшение управляемого состояния сети за счет лучшего использования дорогостоящей сетевой инфраструктуры (эффективность) без ущерба для гибкости (выразительность) может преодолеть новые уровни операционной сложности и ограничений масштабируемости; поиск правильного баланса между ними и поиск способов эффективного представления управляемого состояния сети — это серьезные проблемы, требующие фундаментального понимания на основе аналитических наблюдений и теоретических исследований. Последние разработки в области программно-определяемых сетей (software-defined networking, SDN) частично улучшают выразительность, добавляя новые уровни программируемости, но, к сожалению, не дают окончательного ответа в области фундаментального компромисса между выразительностью и операционной сложностью. В итоге возникает потребность в новых подходах к проектированию сетей, которые требуют эффективных реализаций управляемого состояния в сетевых элементах. В этой диссертации мы рассматриваем эффективные представления управляемого состояния одного сетевого элемента и рассматриваем два фундаментальных вопроса:

(1) как эффективно представлять программы обработки пакетов, решая задачу поиска фундаментального компромисса между эффективностью и выразительностью;

(2) как использовать существующие сетевые ресурсы в условиях наличия локальных ограничений, экспоненциально растущего числа взаимосвязанных устройств и увеличения степени детализации операций.

Программы обработки пакетов. Основным компонентом программ обработки пакетов является классификатор пакетов. Классификаторы пакетов реализуют различные стандартные сервисы, такие как QoS (Quality of Service, качество обслуживания), списки контроля доступа, пересылка пакетов и другие. С принятием OpenFlow [1] и P4 [2] классификация пакетов стала еще более актуальной задачей. В частности, новый язык программирования P4 [2] предназначен для разработки конвейеров обработки пакетов; классификатор пакетов — это основной элемент конвейеров P4. Основная цель классификации пакетов — определить логику обработки (*действие*), которая должна применяться ко входящему пакету. Классификатор пакетов — это упорядоченный набор

Прав.	Фильтр		Действ.
#1	0	2	A_1
#2	0	3	A_1
#3	1	2	A_1
#4	1	3	A_1
#5	4	5	A_2
#6	4	6	A_2
#7	5	5	A_2
#8	5	6	A_2

(a)

Прав.	Фильтр		Действ.
#1	[0,1]	[2,3]	A_1
#2	[4,5]	[5,6]	A_2

(b)

Прав.	Фильтр		Действ.
#1	00*	01*	A_1
#2	10*	101	A_2
#3	10*	110	A_2

(c)

Рис. 1: Классификатор пакетов $\mathcal{K}$ для заголовков состоящих из двух трехбитных полей; поля в $\mathcal{K}$ представлены: (a) точными значениями; (b) интервалами; (c) троичными строками.

правил, в котором правило состоит из фильтра (которому должны соответствовать заголовки классифицируемых пакетов) и соответствующего действия, применяемого к пакетам с подходящими заголовками. Первое правило, к фильтру которого подходит заголовок входящего пакета, определяет, какое действие следует применить. Фильтр — это объединение представлений полей, участвующих в классификации.

В простейшем виде поля в фильтре представлены точными значениями (см. рис. 1a). В этом случае классификаторы пакетов могут быть реализованы с константным временем поиска, но число правил может быть настолько большим, что классификатор не удастся разместить в памяти. В другом крайнем случае, чтобы устранить ограничение масштабируемости, поля могут быть представлены диапазонами значений, или *интервалами* (см. рис. 1b). В этом случае число правил значительно уменьшается, но возрастает сложность поиска. В результате были введены некоторые промежуточные формы представлений полей в виде префиксов или более общих троичных строк, где каждый бит имеет три значения: 0, 1 или *безразлично*, что обычно обозначается через $*$ (см. рис. 1c). Для классификаторов с полями, представленными троичными строками, была введена специализированная память TCAM (ternary content-addressable memory) позволяющая классифицировать заголовки за константное время [3]. Однако память TCAM является энергоемким и дорогим ресурсом. Таким образом, задача состоит в том, чтобы минимизировать размер тернарных представлений классификаторов. В диссертационном исследовании мы представляем три основных направления исследований, касающихся представлений пакетных классификаторов:

(1) изучаем то, как кодирование интервалов (диапазонов значений) троичными строками влияет на размер тернарного представления классификатора, содержащего интервальные поля;

(2) предлагаем эффективные комбинированные представления нескольких классификаторов, которые используют идентичные шаблоны классификации;

(3) вводим понятие приближенных классификаторов, позволяющее ещё сильнее сокращать число правил классификации (и, следовательно, требующуюся память) за счёт ослабления требований к точности классификации.

Использование сетевых ресурсов. Мониторинг трафика требует значительного количества внутрисетевых ресурсов и имеет решающее значение для эффективных, надежных и безопасных сетевых операций. Понимание свойств трафика позволяет операторам сетей лучше планировать

пропускную способность, обеспечивать QoS, дифференцировать услуги, предотвращать атаки и многое другое. Масштабируемый мониторинг потоков трафика является сложной задачей из-за постоянного роста трафика, неоднородности устройств и неравномерности нагрузки. Во-первых, пока объем и число потоков трафика продолжает расти, растут также время обработки и размер памяти необходимые для мониторинга трафика в сетевых элементах. Во-вторых, сети состоят из неоднородных сетевых элементов, начиная от базовых устройств доступа IoT (интернет вещей) со значительно ограниченными возможностями до высокопроизводительных маршрутизаторов, которые могут пересылать миллионы потоков одновременно. В-третьих, нагрузка мониторинга трафика на разных элементах сети неравномерна, и один элемент может быть перегружен, даже если у других элементов есть свободные ресурсы.

Даже относительно простая задача подсчёта размеров всех потоков становится сложной, когда число потоков становится значительным. Большое число уже существующих публикаций предлагают приблизительные решения для учета трафика, которые жертвуют точностью вычислений, чтобы уменьшить объем памяти, необходимой для одного сетевого элемента. Существующие приближенные решения для расчета размера потока включают, например, оценки [4, 5, 6, 7] и эскизы (sketches): CM [8], CU [9], Pyramid Sketch [10], UnivMon [11] и Elastic Sketch [12]. Эвалюация CEDAR [7], SAC [6] и DISCO [5] на реальных данных показывает что их средние относительные ошибки превышают 12% в случае если счетчик каждого потока состоит из 8-бит [7]. Средняя относительная ошибка Elastic Sketch, одного из последних и лучших предложенных эскизов, может достигать 4 даже при использовании 0.2 МБ для представления счетчиков 110 тысяч потоков, т.е. около 15 бит на поток (см. рис. 9а в [12]). Такая точность может быть слишком низкой для многих практических целей.

Альтернативный подход к решению проблемы масштабируемости в одном сетевом элементе заключается в использовании ресурсов нескольких сетевых элементов. Если задача мониторинга трафика перегружает элемент, то выполнение задачи может быть перенесено с перегруженного элемента на другой элемент вдоль маршрута потока, на котором ресурсов достаточно. Это позволяет использовать ресурсы всей сети для того чтобы эффективно справляться с локальными перегрузками. Из-за сетевого шума, заключающегося в возможной потере пакетов и их переупорядочивании, обслуживание распределенных счетчиков пакетов само по себе является сложной задачей. В диссертации мы предлагаем решение, которое эффективно поддерживает распределенные счетчики пакетов в условиях сетевого шума без использования управляющих пакетов или какой-либо другой дополнительной связи между задействованными сетевыми элементами.

1.2. Цели исследования

В этом разделе мы определяем конкретные цели диссертационного исследования. Мы представляем три проекта по разработке эффективных представлений классификаторов пакетов и один проект по обслуживанию распределенных счетчиков пакетов.

Представления интервальных пакетных классификаторов. Чтобы поместить интервальный классификатор $\mathcal{K}$ в память TCAM, фильтры $\mathcal{K}$ нужно закодировать троичными строками. По-

скольку память TCAM — это дорогой ограниченный ресурс, такое кодирование должно минимизировать общий размер тернарного представления $\mathcal{K}$ в символах и в троичных строках.

Один из способов кодирования интервальных пакетных классификаторов в тернарном виде — это кодирование каждого диапазона в классификаторе отдельно одним из методов, представленных в работах [13, 14, 15, 16, 17]. Эти методы кодируют каждое поле-интервал с помощью нескольких префиксов или троичных строк, число которых не более чем линейно относительно битовой ширины поля. В этом случае, число троичных строк кодирующих фильтр завист экспоненциально от количества полей. Другие методы кодирования интервала, использующие структурные свойства, могут обеспечить более компактное представление [18, 19], но обычно они работают хорошо только тогда, когда число различных интервалов, которые требуется кодировать, относительно невелико. Обращаем внимание на то, что оба этих направления исследований сохраняют эквивалентность классификатора.

Представления пакетных классификаторов, предложенные в [20, 21, 22, 23], создают тернарные таблицы поиска по подмножеству полей, сохраняющие такие структурные свойства классификаторов, как непересекаемость правил (rule disjointness). Эти представления становятся эквивалентными изначально заданным классификаторам только тогда, когда построенные таблицы поиска дополняются ложноположительной проверкой по единственному совпадающему правилу (false positive check). Это позволяет разделить поля на те, которые требуются для реализации желаемых структурных свойств в таблице поиска, и те, которые участвуют только в ложноположительных проверках, где кодирование интервалов не требуется. В результате число закодированных интервалов в [20, 21, 22, 23] значительно уменьшается, что приводит к значительному сокращению общего числа троичных строк и символов в тернарном представлении классификатора.

В этой диссертации мы продолжаем направление, начатое в [20], и предлагаем представления RR (range reduction, уменьшение диапазона), реализующие структурные свойства классификаторов с побитовым разрешением (а не разрешением с точностью до поля, как в [20]). Представления, сужающие ширины используемых полей, могут значительно уменьшить требования к памяти для представлений классификатора.

Объединенные представления нескольких политик классификации. В современных сетевых элементах многие службы работают параллельно. Эти службы реализуются несколькими классификаторами пакетов, которые часто содержат общие шаблоны классификации (даже классификаторы пакетов, соответствующие различным службам, могут содержать общие шаблоны классификации). Чтобы не определять каждый фильтр для каждого классификатора каждый раз, была введена такая абстракция, как *класс*. Классы — это наборы фильтров классификатора, представляющие общие шаблоны классификации. В этом случае классификатор пакетов (политика) может быть определен как упорядоченный набор классов с соответствующими действиями. В результате классы позволяют определять объектно-ориентированные представления классификаторов пакетов. На практике различные поставщики сетевых элементов уже поддерживают понятие классов в объявлениях политик [24, 25], что позволяет им более эффективно абстрагировать и управлять шаблонами классификации. Например, Cisco IOS поддерживает до 256 различных политик QoS и до 4096 классов на сетевой элемент [26]. В реальных применениях, число классов в каждой поли-

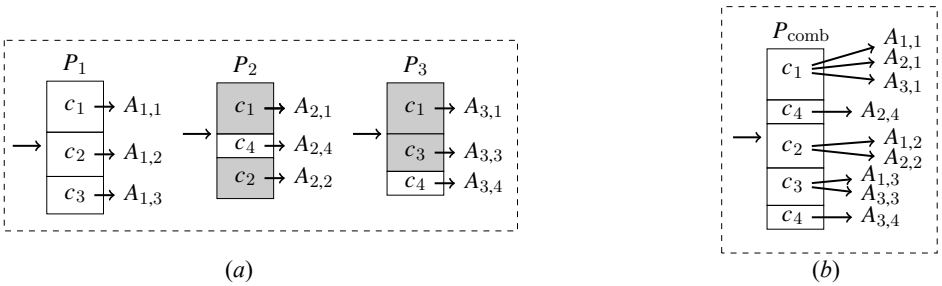


Рис. 2: (a) политики P_1, P_2, P_3; (b) объединённое представление P_{comb} эмулирующее поведение P_1, P_2, P_3. В (a) экземпляры классов отсутствующие в P_{comb} нарисованы серым цветом.

тике варьируется от десятков до сотен в зависимости от модели приложения [26]. Размер класса зависит от сложности представленного шаблона. Наши идеи основаны на том факте, что классы повторно используются в разных политиках. Традиционно для каждой политики, содержащей тот или иной класс, аллоцируется отдельный его экземпляр (см. политики P_1, P_2 и P_3 на рис. 2). Каждый аллоцированный экземпляр класса имеет соответствующее действие, указанное соответствующей политикой (на рис. 2 действие $A_{i,j}$ соответствует экземпляру класса c_j в политике P_i). Поскольку классы повторно используются в разных политиках, это позволяет нам рассматривать объединённые представления политик, где в идеале каждый класс появляется только один раз, обеспечивая значительную экономию в TCAM памяти. Неформально говоря, постороенные объединённые представления «эмулируют» поведение представляемых политик. В диссертационном исследовании мы рассматриваем объединённые представления политик, которые не увеличивают число запусков процедуры поиска подходящего правила.

Приближенные классификаторы с ограниченной ошибкой. Точные вычисления могут требовать чрезмерных вычислительных ресурсов. Приближенные вычисления имеют дело с потенциально неточными вычислениями в условиях ограниченных вычислительных ресурсов [27]. В диссертационном исследовании мы обобщаем классическую задачу классификации пакетов на приближенный случай. Существует целый ряд исследований, посвященных изучению различных методов оптимизации для поиска семантически эквивалентных классификаторов пакетов, где каждый заголовок соответствует одному и тому же действию как в изначальных, так и в оптимизированных классификаторах [28, 29, 30]. Мы рассматриваем случай, когда семантически эквивалентные классификаторы не могут достичь желаемых результатов оптимизации, и вводим приближенные представления классификаторов пакетов, позволяющие «мультиплексировать» несколько действий. Этот дополнительный уровень гибкости позволяет сократить требования к ресурсам, сохраняя при этом желаемый уровень точности. Большинство проприетарных эвристик, минимизирующих число правил в классификаторах обычных пакетов, можно свести к хорошо известным операторам, минимизирующим размер логических выражений [31, 32]. К сожалению, в приближенном случае оптимизационные возможности этих операторов ограничены, и использование этих операторов может привести к экспоненциальному увеличению соответствующих представлений. Чтобы избежать экспоненциального взрыва и улучшить результаты оптимизации, мы обобщаем основные операторы на приближенный случай и изучаем свойства последовательностей, состоящих из применяемых операций.

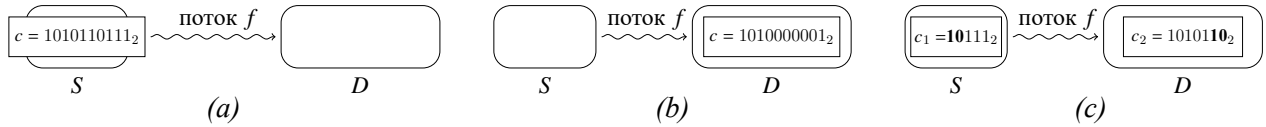


Рис. 3: Подсчет количества пакето в потоке f: (a) в источнике S, (b) в приёмнике D, and (c) распределенно на S и D. В (c) состояния счетчика перекрываются чтобы быть в согласованном состоянии несмотря на сетевой шум.

Распределённые счётчики пакетов. Чтобы поддерживать счетчики пакетов в потоке распределенно, можно было бы перемещать счетчик от источника потока к другому сетевому элементу на пути потока (см. рис. 3b). К сожалению, перенос всего счетчика пакетов в потоке от источника потока к другому сетевому элементу приведет к неточным результатам из-за потери пакетов, которая происходит на пути между этими двумя элементами. Заметим, что такие неточности возникают независимо от того, использует ли исходный элемент надежный или ненадежный транспортный протокол для передачи трафика. Поэтому, в сетевом элементе, являющемся источником потока, должно поддерживаться по крайней мере некоторое состояние счетчика пакетов в этом потоке (см. рис. 3c). В диссертации разработан надежный распределенный подход поддерживающий счетчик пакетов в нескольких сетевых элементах в корректном состоянии, несмотря на сетевой шум состоящий из переупорядочения и потерь пакетов при их передаче между сетевыми элементами. В отличии от приближенных методов жертвующих точностью результата, предлагаемый подход поддерживает точное восстановление количества пакетов в потоке. Разработанный распределенный подход не требует двунаправленного транспорта, не вводит никаких управляющих пакетов и передает состояние потока путем добавления нескольких (обычно, 2 или 4) управляющих битов.

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.

Представления интервальных пакетных классификаторов. В настоящей диссертации мы вводим понятие эквивалентных RR-представлений для интервальных пакетных классификаторов, которые сохраняют свойство непересекаемости правил (rule-disjointness) с побитовым разрешением [33]. Основным базовым элементом предложенных RR-представлений является понятие *подинтервала* на фиксированном множестве B битовых индексов интервала, которое позволяет реализовать структурные свойства классификатора на побитовом уровне без промежуточного кодирования интервалов. Мы доказываем, что каждый подинтервал на $|B|$-битовых индексах может быть представлен либо как $|B|$-битовый интервал, либо как объединение двух специальных интервалов на $|B|$ битах; предложенный в диссертации Алгоритм 1 строит такое подинтервальное представление. Число троичных строк в префиксном кодировании [13] и в SRGE [15] любого $|B|$-битового подинтервала не превосходит $2 \cdot |B| - 2$ (почти так же, как и для обычного $|B|$-битового интервала). Мы формулируем Задачу 1 минимизации количества строк и символов в тернарном кодировании RR-представлений, а также представляем Алгоритм 3, решающий Задачу 1.

Объединенные представления нескольких политик классификации. Для данного множества политик классификации $\mathcal{P}$ мы предлагаем объединенное представление, которое состоит из *объединенной политики* P_{comb}, симулирующей всё множество $\mathcal{P}$ [34, 35]. В диссертации исследованы два основных случая: (1) идеальные представления, в которых каждый класс встречается только однажды, и (2) неидеальные представления, в которых классы повторяются. В первом случае мы находим условия существования идеальных представлений и предлагаем методы построения P_{comb} когда эти условия выполнены. Во втором случае мы формулируем задачу PSP поиска последовательности классов $\mathbb{S}$ в P_{comb}, которая минимизирует число фильтров $W^+(\mathbb{S})$ в дополнительных копиях классов в $\mathbb{S}$. Предложенные в диссертации алгоритмы `AllOrOne` и `CliqueShare` строят такую последовательность $\mathbb{S}$; в диссертации доказаны теоретические верхние и нижние оценки этих алгоритмов на $W^+(\mathbb{S})$. В частности, доказано, что аппроксимационный фактор `CliqueShare` относительно $W^+(\mathbb{S})$ составляет не более $\alpha(G^{\mathsf{pair}}) \cdot \frac{|\mathcal{P}|^2}{4}$, где $|\mathcal{P}|$ — число политик в $\mathcal{P}$, G^{pair} — вспомогательный граф, используемый `CliqueShare`, а $\alpha(G^{\mathsf{pair}})$ — аппроксимационный фактор алгоритма, решающего классическую задачу поиска множества вершин минимального суммарного веса удаление которых сделает заданный граф ациклическим (Feedback Vertex Set, FVS) [36, 37, 38]. Доказана также нижняя оценка на аппроксимационный фактор алгоритма `CliqueShare`, равная $\frac{|\mathcal{P}|^2}{4}$. Кроме того, мы показываем, что не существует полиномиального алгоритма для задачи PSP с постоянным аппроксимационным фактором (если $\mathbf{P} \neq \mathbf{NP}$). Построенная в результате последовательность $\mathbb{S}$ может быть далее оптимизирована предложенными эффективными эвристиками `GreedyGluing` и `LocalDescend`. Чтобы поддерживать P_{comb} в валидном состоянии после модификаций политик из $\mathcal{P}$, мы предлагаем подход, корректирующий P_{comb} и $\mathbb{S}$ после каждой операции добавления или удаления классов за время $O(|\mathbb{S}| + \sum_{P \in \mathcal{P}} |P| + D(P))$, где $D(P)$ — число пар пересекающихся классов из P. В диссертации также исследованы объединенные представления, состоящие из нескольких объединенных политик. В частности, мы показываем, что использование нескольких P_{comb} не ослабляет условия существования идеальных объединенных представлений и предлагаем методы построения идеальных объединенных представлений с несколькими P_{comb}, которые минимизируют максимальное число политик, соответствующих одному P_{comb}.

Приближенные классификаторы с ограниченной ошибкой. В диссертации вводится понятие приближенного пакетного классификатора с ограниченной ошибкой [39]. Каждое правило R_i в приближенном классификаторе содержит *множество действий* (action set) $\mathcal{A}_i$, такое, что каждое действие из $\mathcal{A}_i$ представляет собой корректный результат классификации заголовка пакета классифицируемого правилом R_i. Понятие приближённой классификации позволяет дополнительно сократить количество правил в классификаторе, пожертвовав точностью классификации. В частности, в диссертации показано, что приближённые классификаторы могут улучшить масштабируемость QoS и таблиц пересылки пакетов (packet forwarding). Для минимизации приближённых классификаторов в диссертации обобщаются классические операторы оптимизации: прямое поглощение (forward subsumption), обратное поглощение (backward subsumption) и резолюция (resolution) [30]. Обобщённые операторы оптимизируют приближённые классификаторы строго лучше, чем исходные (точные) классификаторы.

Мы изучаем свойства *последовательностей операций* (operation sequences), которые состоят из

операций оптимизации, последовательно применяемых к тому или иному классификатору. Пусть **O** — множество, содержащее только те операции, которые применимы к исходному классификатору и не содержащее двух операций, использующих одно и то же правило. Такие последовательности определяют нижние оценки на длину максимальной последовательности операций и иллюстрируют комбинаторную сложность минимизации приближённых классификаторов. В диссертации доказано, что в случае точных классификаторов существует последовательность операций, содержащая все операции из **O**, а в случае приближённых классификаторов существует последовательность операций, содержащая все операции прямого поглощения и резолюции из **O**. Для иллюстрации сложности минимизации приближённых классификаторов мы приводим пример множества **O**, содержащего обратные поглощения и резолюции, которые не могут содержаться в одной и той же последовательности операций, и доказываем, что построение самой длинной последовательности операций из **O** является вычислительно трудной задачей.

Мы показываем, что всегда возможно преобразовать любую последовательность операций (в том числе последовательность максимальной длины) в другую той же длины, в которой все прямые поглощения и резолюции предшествуют обратным поглощениям. Это наблюдение помогает строить эвристические алгоритмы для построения последовательностей операций максимальной длины. Мы также показываем, что после предложенного преобразования последовательности операций множества действий в правилах оптимизированного классификатора остаются прежними или даже увеличиваются. Таким образом, это преобразование может использоваться как постоптимизация, позволяющая расширять уже построенные последовательности операций.

Распределённые счётчики пакетов. Чтобы сократить требования к памяти в каждом сетевом элементе (свитче), мы разделяем счётчик пакетов в потоке на две части, поддерживаемые источником и приемником потока (см. рис. 3c). В этом разделе мы разрабатываем методы для поддержания частей счётчика в согласованных состояниях в присутствии сетевого шума. Диссертация содержит два основных результата в этом направлении: (1) две модели сетевого шума, которые позволяют описывать устойчивость предложенных распределённых подходов к потере и переупорядочиванию пакетов; (2) Алгоритм 1 и Алгоритм 2, которые поддерживают части счётчиков пакетов в потоке в согласованных состояниях; устойчивость этих алгоритмов к сетевому шуму доказывается аналитически в соответствующих моделях сетевого шума. Показано, что устойчивость к сетевому шуму у Алгоритма 2 значительно выше, чем у Алгоритма 1. Мы также доказываем оценку на ошибку Алгоритма 2 в случаях, когда условия его корректности не выполнены.

3. Публикации и апробация работы

Каждый из результатов, представленных в предыдущем разделе, был ранее опубликован в одной из рецензируемых научных работ, представленных ниже. Во всех приведённых ниже работах автор диссертации является основным автором.

Публикации в конференциях высокого мирового уровня (журналы Q1 Web of Science / Scopus, конференции CORE ranking A/A^*):

- Demianiuk V., Nikolenko S., Chuprikov P., Kogan K. New Alternatives to Optimize Policy Classifiers // IEEE Transactions on Networking, vol. 28, no. 3, 2020, pp. 1088–1101. Конференционная версия ранее была опубликована в Proceedings of IEEE ICNP 2018.

  Вклад автора диссертации: идеальное представление в случае, когда все классы попарно не пересекаются, описанное в разделе 3.A; метод добавления префиксов к фильтрам из P_{comb}, содержащим пересекающиеся классы, описанный в разделе 3.B (кроме нижней оценки в Теореме 2); Теорема 3, определяющая условия существования идеальных представлений; Теорема 4 и Теорема 6, доказывающие свойства идеальных объединенных представлений, которые содержат несколько объединенных политик; определение задачи PSP (Задача 1); идея Алгоритма `AllOrOne` для решения задачи PSP; Теорема 11, представляющая нижнюю оценку на аппроксимационный фактор `AllOrOne`; Теорема 12, показывающая неприближаемость PSP с константным аппроксимационным фактором; Алгоритм `CliqueShare` для решения задачи PSP; Теорема 13, доказывающая корректность Алгоритма `CliqueShare`; Теоремы 14 и 15, устанавливающие верхнюю и нижнюю оценки на аппроксимационный фактор Алгоритма `CliqueShare`; эвристики локального спуска и жадного склеивания, описанные в разделе 5.E; Алгоритм 3 для эффективных динамических обновлений, представленный в разделе 6, и доказательство его корректности (Теорема 16).

- Demianiuk V., Kogan K., Nikolenko S. Approximate Classifiers with Controlled Accuracy // Proceedings of IEEE INFOCOM 2019.

  Вклад автора диссертации: понятие и мотивация приближённых классификаторов с контролируемой точностью, описанные в разделах 2,3; обобщения классических операций оптимизации, описанные в разделе 4; примеры в разделе 4.C, показывающие, что обобщенные операции строго более эффективны, чем исходные; Теорема 1, показывающая, что в случае обычных классификаторов всегда возможно применить все операции из **O**; Следствие 1, находящее максимальное множество **O**; Теорема 2, показывающая, что все прямые поглощения и все резолюции в **O** применимы вместе в случае приближённых классификаторов; Теорема 3, показывающая, что в приближённом случае построение максимальной последовательности операций из **O** — это вычислительно трудная задача; Лемма 1, предлагающая преобразование любой последовательности операций.

- Demianiuk V., Gorinsky S., Nikolenko S., Kogan K. Robust Distributed Monitoring of Traffic Flows // Proceedings of IEEE ICNP 2019.

  Вклад автора диссертации: алгоритмы `PL` и `PR`, поддерживающие распределенные счетчики пакетов в условиях частичного сетевого шума, который либо теряет, либо переупорядочивает пакеты; первая модель сетевого шума, описанная в разделе 4; Алгоритм 1 для распределенного подсчета пакетов в условиях общего сетевого шума, который и теряет, и переупорядочивает пакеты; Теорема 2, показывающая устойчивость Алгоритма 1 к сетевому шуму в первой модели; вторая модель сетевого шума, описанная в разделе 5; Алгоритм 2, являющийся расширением Алгоритма 1 для второй модели сетевого шума; Теорема 4, показывающая устойчивость Алгоритма 2 к сетевому шуму; Теорема 6, показывающая, что

условия корректности Алгоритма 2 шире, чем условия корректности Алгоритма 1; Теорема 7, устанавливающая оценки на ошибку Алгоритма 2 в случае, когда условия его корректности не выполняются.

Другие публикации:

- Demianiuk V., Kogan K. How to deal with range-based packet classifiers // Proceedings of ACM SOSR 2019.*

  Вклад автора диссертации: Наблюдение 1, показывающее, что любой интервал можно представить как объединение левого и правого граничных интервалов; определение понятия подинтервала, описанное в разделе 2; Лемма 3.2, показывающая, что подинтервал левого (правого) граничного интервала является левым (правым) граничным интервалом; Теорема 3.3, показывающая, что любой подинтервал можно представить как объединение двух граничных интервалов; Алгоритм 1 для построения таких представлений в виде подинтервалов; Лемма 3.4 и Теорема 3.5, доказывающие корректность Алгоритма 1; Теорема 3.6, доказывающая верхнюю оценку на число троичных строк в кодировке $|B|$-битового подинтервала; формализация эквивалентных представлений с побитовым разрешением, описанная в разделе 4; определение Задачи 1 минимизации размера тернарного кодирования таких эквивалентных представлений; Алгоритм 3, решающий Задачу 1.

Полученные результаты были апробированы двумя путями: во-первых, для каждого алгоритма оптимизации мы представляем строгое теоретическое исследование его свойств (доказательства корректности, оценки вычислительной сложности в худшем случае, оценки качества приближений в худшем случае, структурный анализ построенного и оптимального решений и т.д.). Во-вторых, для практических целей мы оцениваем предложенные методы на синтетических данных об сетевом трафике, показывая их качество работы в среднем случае. Алгоритмы для минимизации приближенных классификаторов (раздел 4.1.3) были оценены на классификаторах, порожденных системой *Classbench* [40], и на пяти IPv4 FIB классификаторах из реальных сетевых систем. Множества действий для правил в этих классификаторах были порождены искусственно в соответствии с практическими случаями пересылки пакетов и QoS. Методы для построения объединенных политик (раздел 4.1.2) оценивались на синтетических множествах политик $\mathcal{P}$. При порождении входов мы варьировали следующие основные характеристики $\mathcal{P}$, которые влияют на результаты оптимизации: число политик, число пересечений между классами и среднее число общих классов у двух политик. Алгоритмы построения эквивалентных представлений интервальных пакетных классификаторов (раздел 4.1.1) оценивались на случайно порожденных классификаторах, где варьировались число полей в фильтре и размеры полей в битах. Методы для поддержания распределенных счетчиков пакетов (раздел 4.2) оценивались в симуляторе YAPS [41], для которого был реализован однонаправленный ненадежный протокол передачи пакетов; логи сетевого трафика были получены из распределения размеров потоков в случае «data mining» [42, 43, 44].

*SOSR — это главная мировая конференция для научных публикаций по Software Defined Networking (SDN), основанная в 2015 г.

4. Содержание работы

В данном разделе мы даём обзор каждого из исследовательских проектов, в рамках которых были получены основные результаты диссертации, перечисленные в разделе 2, и описываем то, каким образом были выполнены поставленные в Разделе 1.2 задачи.

4.1. Представления пакетных классификаторов

Как уже упоминалось ранее, классификатор пакетов — это один из основных компонентов программ обработки пакетов. Основная цель классификации пакетов — определить действие, которое следует применить ко входящему пакету. В данном разделе мы предлагаем различные методы построения более эффективных представлений пакетных классификаторов.

Входящий пакет классифицируется по его *заголовку* H, состоящему из k полей $\{h_1, \ldots, h_k\}$, где каждое поле h_i — это целое число состоящее из w_i битов. Пакетный *классификатор* $\mathcal{K} = \{R_1, \ldots, R_N\}$ — это последовательность правил, где каждое *правило* (rule) $R_i = (F_i, A_i)$ состоит из *фильтра* (filter) F_i, на соответствие которому проверяются заголовки входящих пакетов, и соответствующего этому фильтру *действия* (action) A_i. Мы рассматриваем два типа представлений полей в классификаторе $\mathcal{K}$: (1) интервал значений (см. рис. 1b) и (2) троичная строка, где каждый символ может иметь одно из трех значений: 0, 1 или *безразлично*, что обычно обозначается через $*$ (см. рис. 1c). В первом случае фильтр F представляет собой последовательность из k интервалов $\{I_1, \ldots, I_k\}$; заголовок H удовлетворяет (matches) фильтру F, если h_i принадлежит I_i для каждого $1 \le i \le k$. Во втором случае фильтр F представляет собой троичную строку длины $l = \sum_{i=1}^{l} w_i$, полученную конкатенацией троичных строк, представляющих соответствующие поля; заголовок H — это двоичная строка длины l, полученная в результате объединения двоичных записей полей H; H удовлетворяет F, если для каждого битового индекса $1 \le i \le l$ либо $H[i] = F[i]$, либо $F[i] = *$. Классификация заголовка это поиск действия первого правила, такого что данный заголовок удовлетворяет фильтру этого правила; найденное правило называется *классифицирующим*. Если в $\mathcal{K}$ нет фильтра, которому удовлетворяет H, результатом классификации будет действие по умолчанию, которое отличается от всех действий в $\mathcal{K}$. Далее будем говорить, что два фильтра F_1 и F_2 *не пересекаются*, если ни один заголовок не удовлетворяет им обоим. В противном случае F_1 и F_2 *пересекаются*. Два правила (не) пересекаются, если их фильтры (не) пересекаются.

В разделе 4.1.1 мы даём обзор подхода, опубликованного в работе «How to deal with range-based packet classifiers» [33], соответствующего первому основному результату из раздела 2. В разделе 4.1.2 описаны результаты работы «New Alternatives to Optimize Policy Classifiers» [34, 35], раскрывающей второй основной результат из раздела 2. В разделе 4.1.3 описаны приближённые представления пакетных классификаторов, предложенные в работе «Approximate Classifiers with Controlled Accuracy» [39], соответствующие третьему основному результату из раздела 2.

4.1.1. Представления интервальных пакетных классификаторов

Чтобы хранить интервальный классификатор $\mathcal{K}$ в TCAM-памяти, фильтры $\mathcal{K}$ должны кодироваться троичными строками. Напомним, что число троичных строк (и число символов в этих строках)

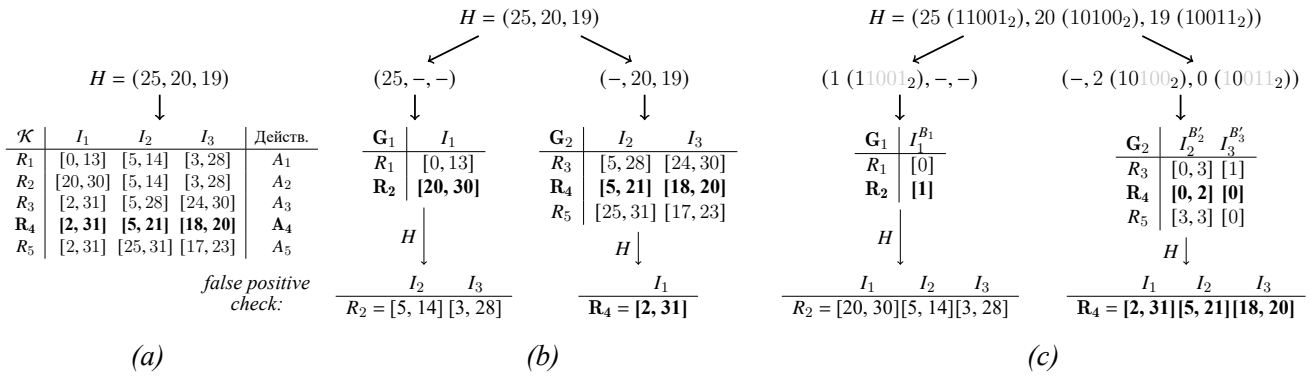


Рис. 4: RR против SAX-PAC [20]: (a) классификатор $\mathcal{K}$; (b) SAX-PAC представление $\mathcal{K}$ состоящее из двух групп таких что $\mathbf{F}_1 = \{I_1\}$ и $\mathbf{F}_2 = \{I_2, I_3\}$; (c) RR представление $\mathcal{K}$ состоящее из двух групп таких что $\mathcal{B}_1 = \{\{1\}, \emptyset, \emptyset\}$ и $\mathcal{B}_2 = \{\emptyset, \{1, 2\}, \{2\}\}$.

в кодировке $\mathcal{K}$ может расти экспоненциально с числом полей в $\mathcal{K}$. Чтобы уменьшить число интервальных полей, которые требуется кодировать, были предложены эквивалентные SAX-PAC [20] представления классификатора, в которых в классификационном запросе к TCAM-памяти участвуют только поля, сохраняющие свойство непересекаемости правил. Остальные поля правил проверяются уже после запроса к TCAM памяти отдельной ложноположительной проверкой. В этом разделе мы улучшаем результаты [20] и предлагаем метод сокращения интервалов (range reduction, RR), позволяющий реализовать непересекаемость правил с побитовым разрешением, что обеспечивает дополнительную экономию в размере кодирования. Мы сначала опишем SAX-PAC представления классификатора, затем введем понятие подинтервала, которое является основным элементом RR-представлений, а затем определим собственно RR-представления.

Представления SAX-PAC. Рассмотрим β непересекающихся групп правил классификатора $\mathcal{K}$ $\mathbf{G}_1, \mathbf{G}_2, \ldots \mathbf{G}_\beta$, таких, что правила в каждой группе $\mathbf{G}_i$ попарно не пересекаются на подмножестве полей $\mathbf{F}_i$. Представление SAX-PAC содержит отдельную таблицу поиска (lookup table) для каждой такой группы $\mathbf{G}_i$. Поскольку правила в каждой группе не пересекаются, каждый входящий заголовок H может удовлетворять только одному правилу из группы $\mathbf{G}_i$, даже если использовать для поиска только поля из $\mathbf{F}_i$. Следовательно, $\mathbf{G}_i$ представлена таблицей поиска содержащей только поля из $\mathbf{F}_i$ (т.е. кодируются только интервалы, соответствующие полям из $\mathbf{F}_i$), и ложноположительной проверкой, в которой оставшиеся поля заголовка сравниваются с соответствующими полями классифицирующего правила. Обозначим за C множество всех правил в $\mathcal{K}$, не принадлежащих ни одной из описанных выше β групп. Мы представляем C как обычный классификатор (кодируя все интервалы фильтров из C). Значение $\beta + 1$ соответствует числу допустимых «псевдопараллельных» обращений к TCAM-памяти (lookups). Процесс классификации в SAX-PAC устроен следующим образом: (1) найти классифицирующее правило в каждой группе (таблице поиска) и выполнить ложноположительную проверку для каждого найденного правила (не более β проверок в целом); (2) независимо от (1) найти классифицирующее правило в C; (3) из не более чем β совпадающих правил, прошедших ложноположительную проверку, и правила классификации, найденного в C, вернуть действие того правила, которое появляется первым в $\mathcal{K}$, или действие по умолчанию, если заголовок не удовлетворяет ни одному фильтру в $\mathcal{K}$ (см. рис. 4b). На рис. 4

число троичных строк в кодировке SAX-PAC представления в 8 раз меньше, чем число троичных строк в кодировке $\mathcal{K}$ в случае префиксного кодирования интервалов [13].

Подинтервалы. В данном разделе мы вводим понятие *подинтервала*, позволяющее реализовать свойство непересекаемости правил данного классификатора с побитовым разрешением, избегая промежуточного кодирования интервалов. Ясно, что если значения рассматриваются на подмножестве битовых индексов данного интервала, то рассматривать понадобится меньший набор значений, и в результате получатся потенциально более эффективные представления в троичных строках, чем у исходного интервала. Но этого недостаточно: для корректности такого сокращения нам нужно, чтобы значение, удовлетворяющее исходному интервалу, продолжало удовлетворять соответствующему подинтервалу на подмножестве битовых индексов. Предлагаемые подинтервалы удовлетворяют свойству корректности и позволяют строить эффективные представления.

Рассмотрим непустое подмножество битовых индексов $B \subseteq \{1, 2 \ldots, w\}$ интервала I на w битах. Для числа x обозначим через x^B число, полученное из x путём взятия значений битов на позициях из подмножества B. *Подинтервал* I^B интервала I — это множество значений на битовых индексах B, полученных из значений из I. Например, если I — это интервал 5-битных значений $[23, 25]$, и $B = \{1, 3, 5\}$, то $I^B = \{4, 5, 7\}$; таблица ниже показывает значения из I и I^B.

x	$23\ (10111_2)$	$24\ (11000_2)$	$25\ (11001_2)$
x^B	$7\ (111_2)$	$4\ (100_2)$	$5\ (101_2)$

Этот пример показывает, что подинтервал — это не обязательно интервал. Заметим, что если I состоит из одного значения x, то I^B тоже состоит из одного значения x^B для любого B. Далее мы рассматриваем интервалы, содержащие по крайней мере два различных значения.

Для интервала $I = [l, r]$, обозначим через $sim(I)$ первый индекс бита, значения которого различаются в бинарных представлениях l и r; например, для $I = [23, 25]$ мы получим $sim(I) = 2$. Будем говорить, что w-битовый интервал $I = [l, r]$ — это *левый граничный интервал* (left border interval), если $l \bmod 2^{w-sim(I)} = 0$; аналогично, будем говорить, что I — это *правый граничный интервал*, если $(r + 1) \bmod 2^{w-sim(I)} = 0$. Например, $I = [24(11000_2), 29(11101_2)]$ — это левый граничный интервал, а $I = [18(10010_2), 23(10111_2)]$ — это правый граничный интервал. Тогда интервал $I = [l, r]$ может быть представлен как объединение правого граничного интервала и левого граничного интервала. Например, $[9, 14]$ — это объединение правого граничного интервала $I_1 = [9, 11]$ и левого граничного интервала $I_2 = [12, 14]$. Следующие лемма и теорема показывают, что любой подинтервал также может быть представлен как объединение правого граничного интервала и левого граничного интервала.

Лемма. *Для любого подмножества битовых индексов $B \subseteq \{1, \ldots, w\}$ и w-битового интервала $I = [l, r]$, если I — это левый (правый) граничный интервал, то соответствующий подинтервал I^B — это $|B|$-битовый левый (правый) граничный интервал.*

Теорема. *Для любого подмножества битовых индексов $B \subseteq \{1, \ldots, w\}$ и w-битового интервала I множество I^B может быть представлено как объединение $|B|$-битового левого граничного интервала и $|B|$-битового правого граничного интервала.*

В рамках диссертации нам удалось построить алгоритм, который для заданного интервала I и подмножества битовых индексов B вычисляет два граничных интервала, объединение которых равно I^B, за время $O(w)$ (см. Алгоритм 1 в [33]). Корректность этого алгоритма неочевидна, и отдельно доказывается в [33] в Лемме 3.4 и Теореме 3.5. Отметим, что мы можем работать с подинтервалами как с обычными интервалами, поскольку (не)пересечение двух подинтервалов можно проверить за константное время.

Кодировка w-битового интервала в худшем случае состоит из $2 \cdot w - 2$ троичных строк для префиксного кодирования [13] и $2 \cdot w - 4$ строк для SRGE [15]. Следующая теорема показывает, что в худшем случае число тернарных строк в кодировке $|B|$-битового подинтервала почти такое же, как в кодировке обычного $|B|$-битового интервала.

Теорема. *Подинтервал I^B может быть закодирован не более чем $2 \cdot |B| - 2$ тернарными строками как при помощи префиксного кодирования, так и SRGE.*

RR-представления. В RR-представлениях мы следуем тому же процессу классификации, что и в SAX-PAC, но теперь каждая группа $\mathbf{G}_i$ содержит правила, которые не пересекаются на подмножестве подинтервалов (не более одного в каждом поле), определяемом как $\mathcal{B}_i = \{B_1, B_2, \ldots, B_k\}$, где B_j — подмножество битовых индексов подинтервала j-го поля (см. рис. 4c). Заметим, что B_j может быть пустым, что означает, что соответствующее поле не участвует в таблице поиска. Ложноположительные проверки в RR-представлениях выполняются для всех полей фильтра. На рис. 4 число троичных строк в кодировке RR-представления в 60 раз меньше, чем число троичных строк в кодировке $\mathcal{K}$ в случае префиксного кодирования подинтервалов.

Задача (Построение RR-представлений, RR-C)**.** *Для данного метода кодирования интервалов и интервального классификатора $\mathcal{K}$, найти RR-представление $\mathcal{R}$ классификатора $\mathcal{K}$, минимизирующее размер кодировки $\mathcal{R}$ в символах или в троичных строках.*

Даже в частном случае $\beta = 1$ задача RR-C вычислительно трудна, что может быть показано сведением к ней задачи SetCover [45]. Поскольку представления SAX-PAC принадлежат множеству решений RR-C, кодировка любого SAX-PAC представления будет не меньше, чем кодировка оптимального RR-представления. Для решения задачи RR-C нами была разработана жадная эвристика, которая начинается с разбиения правил $\mathcal{K}$ на несколько групп используя методы из SAX-PAC; затем эвристика находит подинтервалы для построенных групп, которые минимизируют общий размер кодировки и сохраняют свойство непересекаемости правил.

Экспериментальная валидация. В экспериментальной части этого исследования мы сравниваем общий размер кодировки в символах и троичных строках для SAX-PAC и RR-представлений синтетически порождённых классификаторов. В экспериментах мы варьируем число полей в правилах классификации и ширину интервала в битах. Результаты экспериментов показывают, что разница между размером кодировки RR-представления и размером кодировки представления SAX-PAC быстро увеличивается с ростом числа полей в правиле и ширины интервала. В частности, для классификатора на шести 32-битных интервалах общее число троичных строк при кодировании RR-представления может быть в 137 раз меньше, чем в представлении, полученном методом

SAX-PAC. Более подробно результаты экспериментальной валидации приводятся и обсуждаются в разделе 5 работы [33].

4.1.2. Объединенные представления нескольких политик классификации

Как уже упоминалось выше, классы позволяют строить объектно-ориентированные представления пакетных классификаторов (политик), что упрощает их объявление. Классы повторно используются разными политиками, что позволяет строить объединенные представления, поддерживающие одни и те же экземпляры классов для разных политик. Совместное использование экземпляров классов может существенно уменьшить общий размер представления множества политик. В данном разделе мы предлагаем методы построения объединённых представлений для заданного набора политик $\mathcal{P}$. Эти представления не увеличивают число обращений к TCAM-памяти.

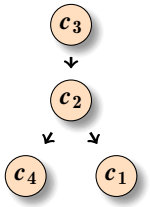
Определение представлений, основанных на политиках. Классы представляют собой промежуточный уровень абстракции: *класс* (class) c — это набор фильтров, представленных троичными строками. Заголовок H *удовлетворяет* классу c, если H удовлетворяет хотя бы одному фильтру из c. Чтобы определить *политику* (policy) P, нужно указать последовательность $\mathbb{S}(P)$ классов в P и назначить каждому классу из $\mathbb{S}(P)$ соответствующее действие. Для входящего заголовка H возвращается действие первого класса из $\mathbb{S}(P)$, которому он удовлетворяет (или действие политики по умолчанию, если таких классов нет). Политика P в $\mathcal{P}$, в которой должна выполняться классификация H, извлекается из внутренних структур данных коммутатора; в этом случае мы говорим, что H входит в *контекст* политики P. В данном исследовании, ми предложим методы построения представления состоящего из одной *объединенной политики* (combined policy) P_{comb} с минимальным числом фильтров, классифицирующую каждый входящий заголовок H в контексте $P \in \mathcal{P}$ на то же действие, что и P. Также, мы покажем что представления с несколькими комбинированными политиками не дают дополнительной экономии на количестве фильтров.

Дизъюнктность классов. Два класса c и c' *не пересекаются*, или *дизъюнктны* ($c \perp c'$), если не существует заголовка, удовлетворяющего и c, и c'; в противном случае классы *пересекаются*. Благодаря наличию дизъюнктных классов, перестановка классов в $\mathbb{S}(P)$ может привести к *эквивалентной* политике, которая выдаёт то же действие, что и P, для каждого заголовка. Например, рис. 5a показывает, что если поменять местами c_4 и c_1 в $\mathbb{S}(P)$, получится эквивалентная политика. Чтобы определить перестановки $\mathbb{S}(P)$, приводящие к политикам, эквивалентным P, мы вводим следующий частичный порядок $\prec_P$ на классах из $\mathbb{S}(P)$ (см. рис. 5b). Будем говорить, что $c_i \prec_P c_j$, если по крайней мере одно из следующих условий выполнено: (1) c_i пересекается с c_j, и c_i стоит перед c_j в $\mathbb{S}(P)$; (2) существует класс $c_k \in P$, для которого $c_i \prec_P c_k$ и $c_k \prec_P c_j$ (транзитивность частичного порядка). Рассмотрим политику P', полученную из P перестановкой классов в $\mathbb{S}(P)$. Если $\prec_{P'}$ совпадает с $\prec_P$, то P и P' эквивалентны. Предлагаемые ниже методы строят P_{comb} по частичными порядками политик в $\mathcal{P}$.

Идеальные представления. Пусть C — это набор всех классов, встречающихся в политиках из $\mathcal{P}$. В оптимальном случае объединенная политика P_{comb} содержит только один экземпляр каждого

Класс	Фильтр	Действ.
c_3	01** *1*0	A_1
c_2	**00	A_2
c_1	10**	A_3
c_4	00**	A_4

(a)



(b)

Рис. 5: (a) политика P с $\mathbb{S}(P) = c_3, c_2, c_1, c_4$; (b) частичный порядок $\prec_P$: $c_3 \prec_P c_1, c_3 \prec_P c_2, c_3 \prec_P c_4$, $c_2 \prec_P c_1, c_2 \prec_P c_4$.

класса из C, и мы будем называть такое представление P_{comb} *идеальным*. Далее мы для данного $\mathcal{P}$ определяем условия, гарантирующие существование идеальных представлений, и предлагаем методы построения идеального P_{comb}.

Если никакой заголовок не удовлетворяет двум разным классам из C, то можно построить идеальный P_{comb} содержащий все классы из C в произвольном порядке. Заголовок H можно классифицировать в P_{comb} вместо исходной политики P_i: если первый класс c (он же единственный), которому удовлетворяет H, принадлежит политике P_i, то результатом классификации будет действие ассоциированное с c в P_i; в противном случае результатом классификации будет действие по умолчанию в P_i. Отметим, что попарная дизъюнктность всех классов в C не является обязательной для существования идеального P_{comb}. Но при этом, в более общем случае, мы должны гарантировать, что для заголовка H, входящего в контекст P_i, первый совпавший класс c в идеальном P_{comb} принадлежит P_i. Чтобы выполнить это требование, мы добавляем к каждому фильтру класса c в P_{comb} троичный *префикс политики* (policy prefix) $\text{pp}(c)$, а к каждому заголовку H, поступающему на вход в контексте P_i, добавляем двоичный *префикс заголовка* (header prefix) pp_i, удовлетворяющий следующему свойству: pp_i удовлетворяет $\text{pp}(c)$ тогда и только тогда, когда $c \in P_i$. Один из вариантов $\text{pp}(c)$ и pp_i, удовлетворяющих определенному выше свойству, может быть следующим: $\text{pp}(c)$ — это троичная строка длины $|\mathcal{P}|$, для которой $\text{pp}(c)_i = *$, если $c \in P_i$, и $\text{pp}(c)_i = 0$ если $c \notin P_i$; pp_i — это битовая строка $0 \ldots 010 \ldots 0$ длины $|\mathcal{P}|$, в которой 1 встречается только в позиции i (см. рис. 6 a).

Теперь определим условия, которые по-прежнему гарантируют существование идеальных представлений. Для этого введем понятие *совместного графа* (joint graph) G^{jnt} для набора политик $\mathcal{P}$ над классами C; это ориентированный граф $G^{\text{jnt}}(\mathcal{P}) = (C, E^{\text{jnt}})$, где E^{jnt} содержит ребро от c_i к c_j для $c_i, c_j \in C$ тогда и только тогда, когда $c_i \prec_P c_j$ хотя бы для одной политики $P \in \mathcal{P}$ (см. рис. 6b).

Теорема. *Для данного набора политик $\mathcal{P}$ идеальное представление P_{comb} существует, если соответствующий совместный граф G^{jnt} ациклично.*

Из этой теоремы следует алгоритм построения идеального P_{comb}: берём топологический порядок классов в G^{jnt} и добавляем к каждому фильтру класса префикс политики, как описано выше.

Неидеальные представления. Для случаев, когда G^{jnt} не ациклично, можно построить объединенную политику, содержащую повторяющиеся экземпляры класса. Далее будем говорить, что последовательность $\mathbb{S}$, определяющая порядок экземпляров классов в P_{comb}, *совместима* с политикой $P_i \in \mathcal{P}$, если существует подпоследовательность $\mathbb{S}'_i$ из $\mathbb{S}$, состоящая из одного экземпляра

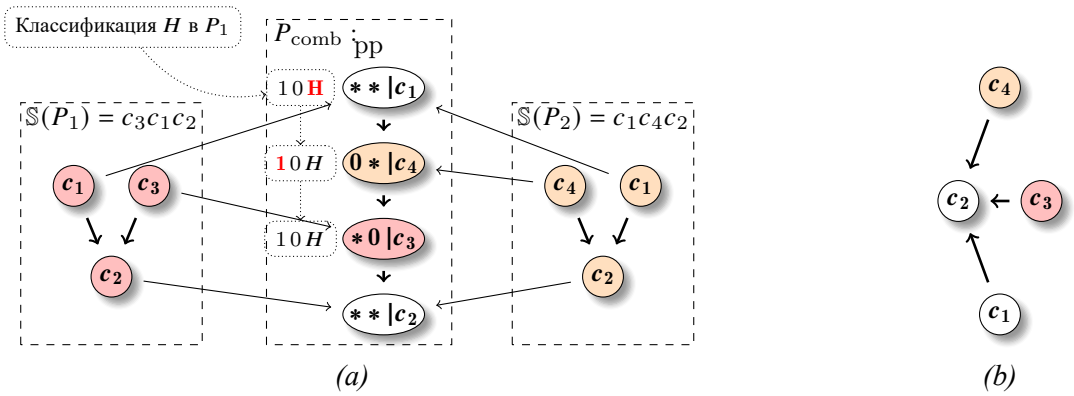


Рис. 6: (a) $\mathcal{P} = \{P_1, P_2\}$, идеальный P_{comb} и префиксы политик; (b) $G^{\text{jnt}}(\mathcal{P})$.

каждого класса в P_i и такая, что для любых двух классов $c_j, c_t \in P_i$, если $c_j \prec_{P_i} c_t$, то c_j появляется перед c_t в $\mathbb{S}'_i$. Последовательность $\mathbb{S}$ построенного представления P_{comb} должна быть совместима со всеми политиками в $\mathcal{P}$. Только экземпляры классов из $\mathbb{S}'_i$ участвуют в классификации по политике P_i, т.е. в соответствующем P_{comb} только для них i-й бит префикса политики установлен в $*$, в то время как для всех других экземпляров бит i префикса политики установлен в ноль. Ясно, что число фильтров в классах нужно учитывать при дублировании классов. Обозначим через $\mathrm{W}^+(\mathbb{S})$ общее число фильтров в дублированных экземплярах классов в $\mathbb{S}$.

Задача (Policy Sequence Packing, PSP)**.** *По данному набору политик $\mathcal{P}$ найти последовательность классов $\mathbb{S}$, которая совместима со всеми политиками из $\mathcal{P}$ и при этом минимизирует* $\mathrm{W}^+(\mathbb{S})$.

Теорема. *Если* $\mathrm{P} \neq \mathrm{NP}$*, то не существует полиномиального приближённого алгоритма, решающего задачу PSP с константным аппроксимационным фактором на* $\mathrm{W}^+(\mathbb{S})$.

Мы предлагаем алгоритмы `AllOrOne` и `CliqueShare` для решения PSP. Эти алгоритмы используют решение задачи поиска взвешенного разрезающего циклы множества вершин (Weighted Feedback Vertex Set, WFVS) [36], которая является NP-полной. *Разрезающее циклы множество вершин* — это набор вершин в ориентированном вершинно-взвешенном графе $G = (V, E)$, при удалении которых образуется ациклический граф, а задача WFVS состоит в том, чтобы найти разрезающее циклы множество вершин с минимальным общим весом. Например, в работе [38] предлагается алгоритм для решения WFVS с аппроксимационным фактором $O(\log |V| \log\log |V|)$, но есть и другие алгоритмы [37]. Обозначим через $\alpha(G)$ аппроксимационный фактор алгоритма решающего WFVS на графе G. Заметим, что задача WFVS не сложнее, чем PSP.

AllOrOne. Напомним, что основной причиной дублирования классов являются циклы в совместном графе. На первом этапе алгоритм `AllOrOne` строит G^{jnt}; на втором шаге он находит разрезающее циклы множество вершин V^{wfvs} в графе G^{jnt} с минимальным общим весом, где вес вершины равен числу фильтров в соответствующем классе; а на третьем строит $\mathbb{S}$ с помощью V^{wfvs}. Индуцированный подграф на вершинах, не лежащих в V^{wfvs}, является ациклическим, поэтому соответствующие классы появляются в $\mathbb{S}$ только один раз. Если $c \in V^{\text{wfvs}}$, то в построенной $\mathbb{S}$ будет отдельный экземпляр c для каждой политики, содержащей c. Алгоритм `AllOrOne` корректно решает задачу PSP, и его аппроксимационный фактор не превосходит $\alpha(G^{\text{jnt}}) \cdot (|\mathcal{P}| - 1)$ (см. Теорему 9 и Теорему 10 в работе [35]).

Теорема. *Аппроксимационный фактор алгоритма* `AllOrOne` *составляет не менее* $|\mathcal{P}| - 1$.

CliqueShare. Эффективность алгоритма, основанного на WFVS, сильно зависит от информации о $\mathbb{S}$, предоставляемой FVS. В алгоритме `AllOrOne` эта информация очень ограничена: FVS предоставляет только набор классов, появляющихся в $\mathbb{S}$ более одного раза. Чтобы преодолеть это ограничение, мы предлагаем другой алгоритм `CliqueShare`: для каждого класса c FVS в `CliqueShare` предоставляет пары политик, у которых экземпляры c в $\mathbb{S}$ различные. Алгоритм `CliqueShare` строит ещё один граф G^{pair}, позволяющий работать с более высоким разрешением. Для каждого класса c и каждой пары политик A, содержащих c, G^{pair} содержит вершину c^A. Для каждого P_i и любых двух классов $c_1 \prec_{P_i} c_2$ граф G^{pair} содержит ребро $(c_1^A, c_2^{A'})$ если $P_i \in A \cap A'$. На первом шаге алгоритм `CliqueShare` находит разрезающее циклы множество вершин V^{wfvs} в графе G^{pair} с минимальным общим весом. Если c^A содержится в V^{wfvs}, то результирующий $\mathbb{S}$ будет содержать различные экземпляры c для политик $P_i, P_j \in A$. Набор политик может совместно использовать один и тот же экземпляр класса c, если для любых двух политик из этого набора соответствующая вершина для класса c в G^{pair} не содержится в V^{wfvs}, мы называем такие подмножества *допустимыми*. Для каждого класса c алгоритм `CliqueShare` находит разбиение множества политик, содержащих c, на минимальное количество допустимых подмножеств.

Теорема. *Алгоритм* `CliqueShare` *корректно решает задачу PSP.*

Теорема. *Алгоритм* `CliqueShare` *имеет аппроксимационный фактор не более* $\alpha(G^{pair})\lfloor\frac{|\mathcal{P}|^2}{4}\rfloor$.

Теорема. *Аппроксимационный фактор алгоритма* `CliqueShare` *составляет не менее* $\lfloor\frac{|\mathcal{P}|^2}{4}\rfloor$.

Фактор аппроксимации алгоритма `CliqueShare` квадратичен по $|\mathcal{P}|$ и хуже, чем для алгоритма `AllOrOne` для всех $|\mathcal{P}| > 3$. Но на практике `CliqueShare` значительно превосходит `AllOrOne`, так как работает с более высоким разрешением.

Локальные оптимизации. Оба алгоритма, — и `AllOrOne`, и `CliqueShare` — можно улучшить за счет двух дополнительных оптимизаций. Эти алгоритмы строят вспомогательный ациклический граф G^*, топологический порядок вершин которого составляет $\mathbb{S}$. Первая предлагаемая оптимизация `GreedyGluing` итеративно объединяет пары вершин в G^* соответствующих одному и тому же классу сохраняя ацикличность G^*. На каждой итерации `GreedyGluing` сливает вершины максимального веса. Временная сложность `GreedyGluing` составляет $O(n^3)$, где n — число вершин в G^*.

Вторая оптимизация исходит из того факта, что предлагаемые алгоритмы обычно не гарантируют, что $\mathbb{S}$ будет *локально минимальным* решением, т.е. может случиться так, что можно удалить несколько экземпляров классов из полученного $\mathbb{S}$ и все равно сохранить корректную последовательность для P_{comb}. В связи с этим мы предлагаем метод пост-оптимизации `LocalDescent`, который определяется следующим образом: для данного $\mathbb{S}$ удаляем экземпляры классов в нем один за другим, до тех пор, пока $\mathbb{S}$ остаётся допустимым решением PSP.

Динамические обновления. Хотя экономические модели меняются редко, поддержка динамических обновлений в представляемых политиках может стать важной в некоторых сценариях. Мы

поддерживаем две основные операции с политиками в $\mathcal{P}$: (1) $\mathrm{delete}(P, c)$, удалить класс c из политики P; (2) $\mathrm{insert}(P, c, c_{succ})$, добавить в политику P класс c, так что бы он встречался в $\mathbb{S}(P)$ сразу перед c_{succ}. Заметим, что после каждой операции нужно корректно обновить P_{comb}. Чтобы реализовать быстрые и эффективные динамические обновления, мы предлагаем метод, который модифицирует P_{comb} после каждой операции за время $O(|\mathbb{S}| + \sum_{P \in \mathcal{P}} (|P| + D(P)))$, где $|\mathbb{S}|$ — число экземпляров классов в $\mathbb{S}$, $|P|$ — число классов в P, а $D(P)$ — число пар пересекающихся классов из P (см. раздел 6 в работе [35]).

Объединённые представление, состоящие из нескольких P_{comb}. Здесь мы рассматриваем представления, состоящие из нескольких P_{comb}. В этом случае из-за ограничения на одно обращение к TCAM памяти каждая политика соответствует только одной объединенной политике в итоговом представлении. Следовательно, представления, состоящие из нескольких P_{comb}, не позволяют уменьшить число фильтров по сравнению с представлениями, состоящими из одного P_{comb}. В частности, использование нескольких объединённых политик не позволяет построить идеальное представление, хранящее только один экземпляр каждого класса, если не существует одного идеального P_{comb}, представляющего данный набор политик.

Теорема. *Для данного набора политик, если существует идеальное представление, состоящее из нескольких объединённых политик, то существует идеальное представление, состоящее из одного P_{comb}.*

Однако использование нескольких объединённых политик может позволить уменьшить длины дополнительных префиксов в построенных объединённых политиках. В общем случае, максимальное количество политик соответствующих одному и тому же P_{comb} влияет на компромисс между максимальной шириной дополнительного префикса и максимально возможным сокращением фильтров в дублированных экземпляров классов. Теорема 6 в [35] определяет метод позволяющий построить идеальное представления состоящее из нескольких P_{comb} минимизируя число политик соответствующих одному и тому же P_{comb}.

Экспериментальная валидация. В экспериментальном исследовании мы сравниваем общее число фильтров во всех политиках в $\mathcal{P}$ с числом фильтров в P_{comb}, где $\mathbb{S}$ может быть построенно с помощью алгоритмов `CliqueShare`, `AllOrOne`, алгоритма `WSCS`, который строит кратчайшую взвешенную общую надпоследовательность [46] из всех последовательностей классов, определяющих политики в $\mathcal{P}$, и алгоритм `MajorityMerge`, предложенный в [46] с (3, 1)-lookahead расширениями [47, 48]. Для каждого рассматриваемого алгоритма мы также оцениваем его версию, расширенную за счет дополнительных оптимизаций. В частности, мы расширяем все алгоритмы с помощью `LocalDescent`, а также дополнительно расширяем алгоритмы `AllOrOne` и `CliqueShare`, работающие на основе графов, с помощью `GreedyGluing`.

Мы порождаем входы с разными значениями следующих основных характеристик: (1) число пересекающихся классов, (2) число политик, содержащих класс, (3) общее число политик. Экспериментальное исследование подтверждает полезность абстракций «класс» для оптимизации представлений политик как на основе последовательностей, так и на основе частичных по-

рядков классов, но вторые работают намного лучше. Алгоритм `CliqueShare` с `LocalDescend` и `GreedyGluing` значительно превосходит все другие оцениваемые алгоритмы. Более того, алгоритм `CliqueShare` без локальной оптимизации превосходит `WSCS` и `MajorityMerge`, даже если за ними следует оптимизация `LocalDescend`, и строит последовательности классов с почти таким же общим числом фильтров в дублированных экземплярах классов, что и `AllOrOne` с `LocalDescend` и `GreedyGluing`. Более подробно проведённое экспериментальное исследование описано в разделе 7 работы [35].

4.1.3. Приближенные классификаторы с контролируемой ошибкой

В этом проекте мы рассматриваем приблизительную классификацию, позволяющую дополнительно минимизировать число правил в данном классификаторе, жертвуя точностью классификации. Ошибки классификации контролируются следующим образом: каждому правилу классификации R_i ставится в соответствие множество действий $\mathcal{A}_i$ так, что любое действие в $\mathcal{A}_i$ может быть результатом классификации заголовка, классифицированного по R_i. В данной работе мы рассматриваем классификаторы, фильтры которых представляют собой троичные строки. Далее будем говорить, что $\mathcal{K}$ является *точным* классификатором, если для каждого правила $R_i \in \mathcal{K}$ множество $\mathcal{A}_i$ состоит из одного элемента (т.е. $\mathcal{K}$ является обычным классификатором пакетов); в противном случае $\mathcal{K}$ называется *приближенным* классификатором.

Будем говорить, что приближенный классификатор $\mathcal{K}'$, полученный после оптимизации данного приближенного классификатора $\mathcal{K}$, *приближенно эквивалентен* $\mathcal{K}$, если для каждого заголовка H множество действий правила классифицирующего H в $\mathcal{K}'$, является подмножеством множества действий правила классифицирующего H в $\mathcal{K}$. В данном разделе диссертации мы предлагаем методы, которые строят приближенно эквивалентные классификаторы, минимизируя число правил классификации.

Примеры использования. Начнем с двух мотивирующих примеров, которые вводят идею приближенных представлений. Во-первых, рассмотрим общую ситуацию, когда правила классификатора сопоставляют заголовки с некоторой количественной характеристикой, например с желаемой задержкой, которая далее агрегируется в классы обслуживания, на которые классифицируются пакеты. Например, на рис. 7 значения 1–3 сопоставлены с классом *Gold*, а 4–6 с классом *Silver*. Классификатор на рис. 7 является неприводимым, т.е. ни один меньший классификатор не может отобразить те же самые заголовки в те же самые классы. Однако давайте рассмотрим «пограничные» правила между классами обслуживания. Предположим, что мы можем позволить пакетам, подпадающим под правило R_4, которое в настоящее время отображается в *Silver*, но имеет значение 4, очень близкое к нижней границе класса *Gold*, классифицироваться как в *Gold*, так и в *Silver*. В этом случае мы можем оптимизировать размер классификатора ещё сильнее после того, как правило R_4 будет ассоциировано с действием *Gold* или *Silver*; на рис. 7b показано, как R_4 можно объединить с R_2 и заменить R_2 на R_{24}, поставив ему в соответствие действие *Gold*. Полученный классификатор можно оптимизировать таким способом и дальше: поскольку R_3 имеет значение 3, которое очень близко к верхней границе класса *Silver*, мы можем разрешить R_3 выдавать действие

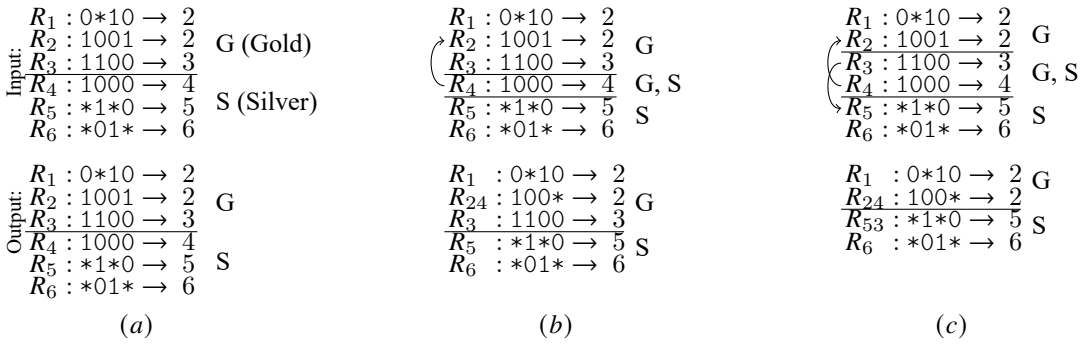


Рис. 7: Компромисс между точностью и размером аппроксимационного классификатора: (a) несжимаемый классификатор; (b) R_4 может классифицировать в Gold или Silver; (b) R_3 и R_4 могут классифицировать в Gold или Silver.

Silver (рис. 7c), что позволит объединить R_3 и R_5 в R_{53}. Отметим, что даже в этом конкретном примере не только пограничные правила между разными классами могут быть расширены с помощью дополнительных действий. Хотя R_6 отображается в значение 6, что очень далеко от нижней границы класса *Gold*, в некоторых случаях мы всё же можем разрешить расширить R_6 с помощью действия *Gold* вследствие тех или иных дополнительных соображений (например, если мы априори знаем, что требуемая полоса пропускания для трафика, классифицированного по правилу R_6, относительно мала по сравнению с общим объемом трафика *Gold*).

Второй мотивирующий пример имеет иную природу; он показывает, что приближённые представления классификаторов являются гораздо более общим инструментом. Проблема масштабируемости таблиц пересылки (FIB) — это на сегодняшний день в значительной степени нерешенная проблема [49]. Можно запустить сторонний процесс, который оценивает «качество» различных путей пересылки пакетов для пространства заголовков, покрытого данной матрицей трафика. Оказывается, что, используя эту оценку «качества» для расширения множеств действий в уже вычисленных FIB приемлемыми альтернативами, мы часто можем уменьшить размер классификатора (см. рис. 2 в [39]).

Обобщённые операции оптимизации. Методы оптимизации, минимизирующие размер классификаторов [28, 29, 30], обычно можно свести к методам, минимизирующим размер булевых выражений. Существующие эвристики включают набор основных сокращающих операторов, применяемых к классификатору, пока это возможно. Они состоят из двух основных блоков: *базовые операторы* и *процесс оптимизации*, который строит последовательность операций для сокращения классификатора. Обычно для классификаторов пакетов [50] рассматриваются три основных оператора, которые обеспечивают правильный баланс между своей вычислительной сложностью и применимостью. Эти операторы решают, когда правило можно удалить или заменить другим правилом. В данном разделе мы обобщаем их на приближенный случай. Далее будем говорить, что правило R_i *покрывает* R_j, если все заголовки, соответствующие R_j, также соответствуют R_i.

Прямое поглощение $\mathcal{F}(R_i)$. Эта операция удаляет все недостижимые правила и не зависит от структуры множеств действий. Формально говоря, правило $R_i \in \mathcal{K}$ можно удалить, если существует правило, которое покрывает R_i и появляется в $\mathcal{K}$ раньше R_i. Поскольку прямые поглощения не зависят от действий правил, получающийся в результате классификатор будет приближённо эк-

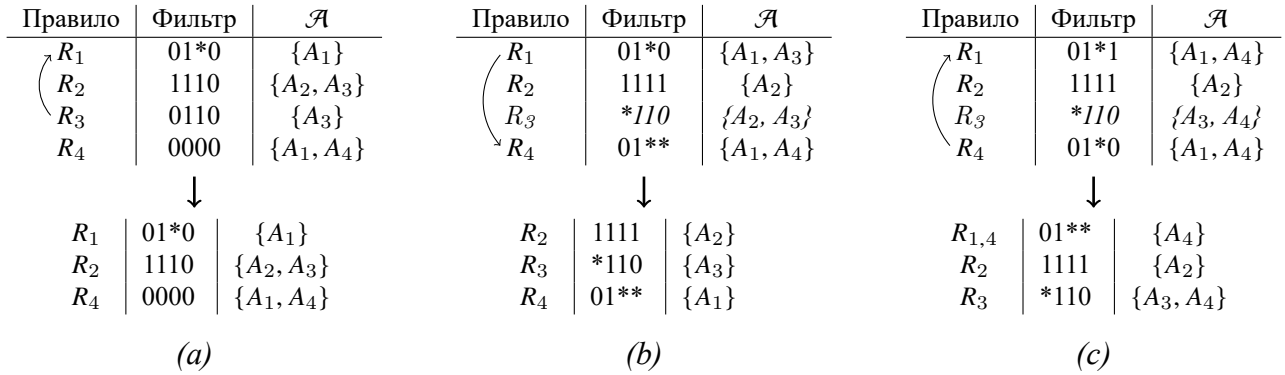


Рис. 8: Обобщенные операции оптимизации: (a) прямое поглощение; (b) обратное поглощение; (c) резолюция.

вивалентным исходному. В примере на рис. 8a, правило R_3 можно удалить, потому что правило R_1 его покрывает.

Обратное поглощение $\mathcal{B}(R_i)$. Иногда правило все же можно удалить, если оно покрыто правилом, появляющимся *после* R_i в $\mathcal{K}$. Формально говоря, правило $R_i \in \mathcal{K}$ можно удалить, если выполняются следующие условия: (1) существует такое правило R_j, что R_j покрывает R_i, R_j появляется в $\mathcal{K}$ после правила R_i, и $\mathcal{A}_j \cap \mathcal{A}_i \neq \emptyset$; (2) $\mathcal{A}_t \cap \mathcal{A}_i \neq \emptyset$ для каждого правила $R_t \in \mathcal{K}$, которое пересекается с R_i и расположено в $\mathcal{K}$ между R_i и R_j; после применения обратного поглощения мы положим $\mathcal{A}_t := \mathcal{A}_t \cap \mathcal{A}_i$. Если несколько R_j удовлетворяют первому условию, мы выбираем R_j, появляющееся в $\mathcal{K}$ раньше других. В примере на рис. 8b, правило R_1 может быть удалено путем обратного поглощения правилом R_4 даже несмотря на то, что R_1 пересекается с R_3, так как $\mathcal{A}_1 \cap \mathcal{A}_3 \neq \emptyset$; после применения этой операции обратного поглощения множества действий $\mathcal{A}_3$ и $\mathcal{A}_4$ изменяются соответствующим образом.

Резолюция $\mathcal{R}(R_i, R_j)$. Этот оператор происходит из пропозициональной теории доказательств: в логике высказываний выражение $(x \wedge C) \vee (\bar{x} \wedge C)$ эквивалентно C. Два правила R_i, R_j (пусть $i < j$) можно объединить и заменить новым правилом $R_{i,j}$ (вместо R_i), если: (1) фильтры F_i и F_j совпадают во всех тернарных символах, кроме k, и k-й символ не равен $*$ ни в F_i, ни в F_j; (2) множество $\mathcal{A}'_i = \mathcal{A}_i \cap \mathcal{A}_j \cap \bigcap_{t:i<t<j \text{ и } R_t \text{ пересекается с } R_j} \mathcal{A}_t$ (пересечение множеств действий у всех пересекающихся правил) непусто; у правила $R_{i,j}$ множество действий будет равным $\mathcal{A}'_i$. В примере на рис. 8b, фильтры R_1 и R_4 отличаются только в последнем символе; мы можем применить резолюцию к R_1 и R_4, поскольку $\mathcal{A}_1 \cap \mathcal{A}_3 \cap \mathcal{A}_4 \neq \emptyset$.

В общем случае, в приближенном классификаторе мы можем выбрать одно действие для каждого правила, и каждая такая комбинация станет точным классификатором. Таким образом, мы могли бы использовать базовые операции, предназначенные для минимизации точных классификаторов. Однако проблема заключается не только в экспоненциальном числе различных точных классификаторов, но и в том, что обобщенные версии основных операций расширяют их применимость. Рассмотрим в качестве примера следующие два приближенных классификатора, в которых обобщенные операторы могут достичь лучших результатов, чем даже минимизация всех точных версий приближенных классификаторов:

Правило	Фильтр	$\mathcal{A}$
R_1	010*	$\{A_1, A_2\}$
R_2	0**0	$\{A_1\}$
R_3	**0*	$\{A_2\}$
R_4	01**	$\{A_1, A_2\}$

Правило	Фильтр	$\mathcal{A}$
R_1	00*1	$\{A_1\}$
R_2	11*0	$\{A_2\}$
R_3	**0*	$\{A_1, A_2\}$
R_4	10*0	$\{A_2\}$
R_5	01*1	$\{A_1\}$

В первом классификаторе R_1 можно удалить при помощи обратного поглощения: R_4 покрывает R_1, и все необходимые условия выполняются. С другой стороны, ни в одной из четырех возможных точных специализаций первого классификатора мы не можем удалить ни R_1, ни какое-то другое правило. Во втором классификаторе резолюцию можно применить два раза: $\mathcal{R}(R_1, R_5)$ и $\mathcal{R}(R_2, R_4)$, но в любой точной специализации остается только одна из них.

Свойства последовательностей операций. *Процесс оптимизации* состоит из набора введенных выше основных операторов и *эвристики*, которая выбирает *последовательность операций*, состоящую из применений операций. В диссертационном исследовании мы изучаем свойства последовательностей операций. По определению, каждая примененная операция удаляет ровно одно правило, поэтому после применения последовательности операций $\mathcal{S}$ к классификатору $\mathcal{K}$ в оптимизированном классификаторе останется $|\mathcal{K}| - |\mathcal{S}|$ правил. Следовательно, рассматриваемые эвристики должны максимизировать длину $\mathcal{S}$. Комбинаторный поиск наиболее длинных последовательностей операций представляет собой вычислительно сложную задачу даже в случае точного классификатора. Мы показываем, что эта проблема становится еще более сложной в случае приближенных классификаторов.

Для начала, рассмотрим набор операций $\mathbf{O}$, применимых к исходному классификатору, который не содержит *конфликтующих* операций с использованием того же правила. Самая длинная последовательность операций, содержащая только операции в $\mathbf{O}$, дает нам нижнюю границу длины оптимальной последовательности операций. Операции в $\mathbf{O}$ неявно зависят друг от друга; например, операция резолюции может создать новое правило, которое будет пересекаться с другим правилом в применении обратного поглощения. Следующая теорема показывает, что в случае точного классификатора мы можем применять все операции в $\mathbf{O}$ без оглядки на эти неявные зависимоси. Несмотря на популярность булевой минимизации, нам не известно ни одной работы предшественников, в которой был бы доказан этот результат.

Теорема. *В случае точных классификаторов существует последовательность операций, содержащая все операции в* $\mathbf{O}$.

Следствие. *Длина оптимальной последовательности операций в случае точных классификаторов не меньше размера наибольшего* $\mathbf{O}$*; кроме того, эта нижняя граница может быть вычислена за время* $O(N^2 \cdot l + C_{\mathcal{R}}^{1.5})$*, где* $C_{\mathcal{R}}$ *— число резолюций, применимых в исходном классификаторе.*

В приближенном случае $\mathbf{O}$ может увеличиваться по сравнению с точным, поскольку обобщенные операции применимы чаще. Но, в отличие от точного случая, не гарантируется существование последовательности операций длины $|\mathbf{O}|$. Рассмотрим следующий приближенный классификатор:

Правило	Фильтр	$\mathcal{A}$
R_1	011*	$\{A_1\}$
R_2	11*1	$\{A_1, A_2\}$
R_3	*11*	$\{A_1, A_2\}$
R_4	01*1	$\{A_2\}$

В этом классификаторе можно применить резолюцию $\mathcal{R}(R_2, R_4)$ или обратное поглощение $\mathcal{B}(R_1)$, но нельзя применить оба. В общем случае, сложность построения самой длинной последовательности операций над **O** возникает из-за обратных поглощений.

Теорема. *В приближенном случае существует последовательность операций, содержащая все прямые поглощения и резолюции из* **O**.

Если в **O** есть правила обратного поглощения, то поиск самой длинной последовательности операций из **O** превращается в вычислительно трудную задачу.

Теорема. *В приближенном случае, поиск максимальной последовательности применимых обратных поглощений из* **O** *является NP-полной задачей, и, если* **P≠NP***, у неё нет полиномиальных приближённых алгоритмов с константным аппроксимационным фактором.*

Теперь рассмотрим общие последовательности операций, содержащие операции, которые не обязательно находятся в **O**. Любая последовательность операций (в том числе самая длинная) может быть изменена согласно следующей лемме.

Лемма. *Для последовательности операций $\mathcal{S}$ существует другая последовательность операций $\mathcal{S}'$, удовлетворяющая следующим условиям: (1) $\mathcal{S}'$ состоит из тех же операций что и $\mathcal{S}$, но обратное поглощение $\mathcal{B}(R)$ может быть заменено на прямое поглощение $\mathcal{F}(R)$; (2) резолюции и прямые поглощения применяются раньше, чем обратные поглощения; (3) для каждой пары обратных поглощений $\mathcal{B}(R_i)$,$\mathcal{B}(R_j)$ $(i < j)$, $\mathcal{B}(R_j)$ применяется раньше, чем $\mathcal{B}(R_i)$; (4) для каждого правила R_i, его множество действий $\mathcal{A}_i$, полученное после применения $\mathcal{S}$, является подмножеством множества действий $\mathcal{A}'_i$, полученного после применения $\mathcal{S}'$.*

Эту лемму можно использовать для разработки эвристик максимизирующих длину последовательности операций. Кроме того, после предложенного преобразования последовательности операций $\mathcal{S}$ можно применять дополнительные операции, даже если $\mathcal{S}$ до преобразования не могла быть расширена какой-либо другой операцией. Следовательно, это преобразование можно также использовать как пост-оптимизацию (см. пример 6 в работе [39]).

Экспериментальная валидация. В экспериментальной части этого раздела мы оцениваем эвристику `aBM`, применяющую введенные выше операторы минимизации. Сначала, `aBM` применяет прямые поглощения и резолюции, пока это возможно, а затем обратные поглощения. Алгоритм `aBM` был оценен на пяти классификаторах IPv4 FIB, используемых в реальных системах, и классификаторах, созданных средой симуляций *Classbench* [40]. Множества действий правил в рассматриваемых классификаторах были искусственно порождены в соответствии с случаями использования пересылки пакетов и QoS. Результаты экспериментов полностью подтверждают наши основные идеи и теоретические результаты: допуская небольшую ошибку в действиях классификатора, мы можем получить значительную экономию в требующейся для классификатора памяти,

часто сопоставимую с экономией в лучшем случае, когда всем правилам искусственно присваивается одно и то же действие. Более подробно результаты экспериментов описаны в разделе 7 работы [39].

4.2. Распределённые счётчики пакетов, устойчивые к шуму

В этом разделе мы описываем подход, предложенный нами в работе «Robust Distributed Monitoring of Traffic Flows» [51]. Чтобы снизить требования к памяти в отдельном сетевом элементе, в этой работе предлагаются методы, поддерживающие распределенные счетчики пакетов в потоках.

Поток f входит в сеть в коммутаторе-источнике S и покидает сеть в коммутаторе-приёмнике D. В коммутаторе S поток состоит из $|f|$ пакетов p_0, p_1,..., $p_{|f|-2}$, $p_{|f|-1}$. Соответствующий счетчик пакетов c вычисляет $|f|$. В предлагаемом распределенном подходе c делится на две части c_1 и c_2, поддерживаемые S и D соответственно; c_1 состоит из n битов. Сетевой шум может удалять или переупорядочивать пакеты в потоке f до того, как они прибудут в D. Далее мы предлагаем методы, которые поддерживают c_1 и c_2 в согласованных состояниях при наличии сетевого шума. После завершения потока f предложенные алгоритмы восстанавливают $|f|$ по c_1 и c_2; пока f находится в процессе передачи, c_2 позволяет получить в реальном времени нижнюю границу на число пакетов в f, которые источник S уже отправил. Связь между частями счетчика осуществляется внутри полосы передачи через пакеты отслеживаемого однонаправленного потока. Источник S добавляет не более t управляющих битов к каждому пакету.

Частичный сетевой шум. Сначала рассмотрим два крайних случая сетевого шума: (1) переупорядочивание пакетов без их потерь; (2) потери пакетов без переупорядочивания. В первом случае, следующее простое решение надежно вычисляет $|f|$: коммутатор-источник считает пакеты в f, а когда часть счетчика c_1 переполняется, отмечает контрольный бит в отправленном пакете; коммутатор-адресат увеличивает свою часть счетчика c_2 только после получения пакета с отмеченным контрольным битом. Это решение полностью устойчиво к переупорядочиванию пакетов и очень уязвимо к потере пакетов: оно может очень значительно недооценить $|f|$ при потере пакета с отмеченным контрольным битом. В случае потерь пакетов без переупорядочивания коммутатор-источник может установить управляющий бит в каждом отправленном пакете на значение самого старшего бита c_1, а затем увеличить c_1. Коммутатор назначения при этом увеличивает c_2 только после приема пакета, в котором бит управления установлен иначе, чем в ранее полученном пакете. Это решение устойчиво к потере до $2^{n-1} - 1$ последовательных пакетов, но очень уязвимо к переупорядочиванию пакетов. Например, если пакеты $2^{n-1} - 1$ и 2^{n-1} из потока прибывают в D в обратном порядке, D добавит единицу к c_2 три раза вместо одного: при получении пакетов 2^{n-1}, $2^{n-1} - 1$ и $2^{n-1} + 1$.

Общий случай сетевого шума. В работе [51] мы предложили Алгоритм 1, поддерживающий устойчивые вычисления в случае, когда сетевой шум может включать как переупорядочивание, так и потерю пакетов. Этот алгоритм использует *биты синхронизации*, т.е. общую t-битовую часть фрагментов счетчика в S и D, чтобы их синхронизировать. Биты синхронизации соответствуют

старшим битам t в c_1 и младшим битам t в c_2, что означает, что они соответствуют средним битам в результирующем объединенном счетчике c. Когда коммутатор S получает пакет p, коммутатор записывает биты синхронизации из c_1 в заголовок $h[p]$ пакета p и увеличивает c_1. Когда пакет p прибывает в D, коммутатор D вычисляет разницу между $h[p]$ и t битами синхронизации в c_2 по модулю 2^t. Если эта разница лежит между 1 и 2^{t-1}, коммутатор D добавляет ее к c_2. Чтобы восстановить $|f|$ после завершения передачи потока f, Алгоритм 1 устанавливает n старших разрядов результирующего счетчика c в c_2, а затем добавляет к c разницу между c_1 и n младшими битами c, вычисленную по модулю 2^n.

Устойчивость любого распределенного решения зависит от ограничений на сетевой шум; например, ни одно распределенное решение не способно справиться с потерей всех пакетов. Мы используем два параметра, чтобы описать ограничения на переупорядочивание пакетов и их потерю: *параметр переупорядочивания* (reordering parameter) R задаёт максимальное расстояние переупорядочивания пакетов, т.е. коммутатор-приёмник может получить пакет p_j перед пакетом p_i, только если $j \leq i+R$; *параметр потерь* (loss parameter) L — это предел *последовательных* потерь пакетов, т.е. коммутатор-приёмник получает хотя бы один пакет из любого интервала $p_i, \ldots, p_{i+L}$. Следующая теорема показывает устойчивость Алгоритма 1 к сетевому шуму.

Теорема. *Алгоритм 1 вычисляет $|f|$ корректно, если $L+R < 2^{n-1}$ пакетов и $R \leq 2^{n-1}-2^{n-t}$ пакетов.*

Параметр t контролирует компромисс между устойчивостью к переупорядочиванию и накладными расходами на передачу данных в каждом пакете. Установка t в n обеспечивает максимальную устойчивость к переупорядочиванию, которая всего на 14% больше, чем при $t = 4$ битах. Установка t в 1 бит сводит Алгоритм 1 к решению, предложенному для сетевого шума без переупорядочивания пакетов. Следовательно, мы рекомендуем на практике установить для t значение 2, 3 или 4 бита. Алгоритм 1 очень устойчив к сетевому шуму в практических условиях. Например, выделяя только $n = 8$ бит для фрагмента счетчика в коммутаторе S и используя $t = 2$ бита синхронизации, Алгоритм 1 гарантирует правильное вычисление $|f|$ при любых переупорядочиваниях пакетов $R < 64$ и потерях пакетов $L + R < 128$, что соответствует очень значительному уровню сетевого шума. Более того, удвоение значения t с 2 до 4 битов увеличивает допустимое значение R до 112 пакетов.

Альтернативные представления сетевого шума. В этом разделе мы предлагаем альтернативное представление, которое ограничивает не только абсолютные значения, но и частоту возникновения сетевого шума: мы разбиваем поток на коммутаторе S на группы по 2^k последовательных пакетов и вводим $\mathrm{span}(\gamma, k)$ как последовательность пакетов, в которой за каждым пакетом следует пакет из той же группы или γ последующих групп ($\gamma > 0$). Формально $\mathrm{span}(\gamma, k)$ определяется как последовательность пакетов p_{z_i} из потока f со следующими свойствами: (1) индексы z_i образуют возрастающую последовательность; (2) коммутатор D получает все пакеты p_{z_i}; (3) D получает p_{z_i} перед p_{z_j}, если $i < j$; (4) $\left\lfloor \frac{z_{i+1}}{2^k} \right\rfloor - \left\lfloor \frac{z_i}{2^k} \right\rfloor \leq \gamma$ для каждого i, т.е. $p_{z_{i+1}}$ отстоит не более чем на γ групп из 2^k пакетов от p_{z_i} в коммутаторе S; (5) $\left\lfloor \frac{z_0}{2^k} \right\rfloor \leq \gamma$ и $\left\lfloor \frac{|f|-1}{2^k} \right\rfloor - \left\lfloor \frac{z_{r-1}}{2^k} \right\rfloor \leq \gamma$, что соответствует граничным условиям на первый и последний пакеты в последовательности. Если $\mathrm{span}(\gamma, k)$ существует, условие 4 гарантирует, что переупорядочивание и потеря пакетов не могут повлиять на

все пакеты из γ последовательных групп. При таком представлении сетевого шума максимальная степень переупорядочивания пакетов по-прежнему составляет R, а $\mathrm{span}(\gamma, k)$ ограничивает потерю пакетов, так что в результате $L \leq (2^\gamma + 1) \cdot 2^k - 1$.

Это представление сетевого шума позволяет построить Алгоритм 2 (см. [51]), который корректно работает в более мягких условиях по сравнению с Алгоритмом 1. Алгоритм 2 устанавливает k в $n - t$ и отличается от Алгоритма 1 только процедурой обновления c_2; теперь коммутатор D обновляет c_2 только тогда, когда разница между $h[p]$ и битами синхронизации t в блоке счетчика c_2 не превышает γ по модулю 2^t.

Теорема. `Algorithm 2` *вычисляет* $|f|$ *корректно, если*

$$R \leq 2^n - (\gamma + 1) \cdot 2^{n-t} \text{ пакетов и } \exists\, \mathrm{span}(\gamma, n - t). \tag{1}$$

Эта теорема показывает множество различных компромиссов. Как и в случае с нашим исходным представлением сетевого шума, большие значения t увеличивают не только максимальную степень устойчивости к переупорядочиванию, но также перекрытие между c_1 и c_2. Более того, Алгоритм 2 может работать с более мягкими паттернами трафика при том же R для большего значения t.

Увеличение γ сужает максимальную степень переупорядочивания пакетов и ослабляет условия существования $\mathrm{span}(\gamma, k)$. Следовательно, параметр γ реализует компромисс между ограничениями на переупорядочивание пакетов и их потерю. Если установить значение γ в 1, мы получим максимальное допустимое переупорядочивание $R = 2^n - 2^{n-t+1}$ пакетов. При $\gamma = 2^{t-1}$ Алгоритм 1 идентичен Алгоритму 2, и следующая теорема показывает, что наше первое представление сетевого шума является частным случаем второго.

Теорема. *В случае, когда* $\gamma = 2^{t-1}$*, из условий корректности Алгоритма 1 следуют условия корректности Алгоритма 2.*

Настройки с $\gamma \geq 2^{t-1}$ полезны, когда потери происходят чаще, чем переупорядочивания. Когда $\mathrm{span}(\gamma, n - t)$ не существует, Алгоритм 2 недооценивает $|f|$, и мы можем получить аналитические гарантии этой недооценки.

Теорема. *Когда* $\mathrm{span}(\gamma, n - t)$ *не существует, и потеряно общее число пакетов* X*, Алгоритм 2 вычисляет* $|f|$ *с недооценкой не более* $\frac{2^t X}{2\gamma + 1 - 2^t}$ *пакетов, если*

$$R \leq 2^n - (\gamma + 1) \cdot 2^{n-t} \text{ пакетов и } \gamma \geq 2^{t-1} \text{ групп.} \tag{2}$$

Когда ограничение на R выполняется и не существует $\mathrm{span}(\gamma, n - t)$, алгоритм 2 с $t = 2$ битами синхронизации и $\gamma = 2$ группами недооценивает размер потока не более чем на $4X$ пакетов. С 3 битами синхронизации и $\gamma = 6$ группами недооценка составляет не более $1{,}6X$ пакетов. В случае $\gamma = 2^t - 1$ групп Алгоритм 2 не устойчив к переупорядочиваниям и увеличивает устойчивость к потерям до $L < 2^n - 2^{n-t}$ пакетов. Когда потери неограничены и нет переупорядочивания, Алгоритм 2 с такими значениями γ занижает размер потока не более чем на $\frac{X}{1 - 2^{-t}}$ пакетов.

С практической точки зрения, условия 2 делают Алгоритм 2 более устойчивым к переупоря-

дочиванию и потерям, чем при условиях 1. Например, когда c_1 содержит 8 битов, настройки с 2 и 3 битами синхронизации и $\gamma = 1$ поддерживают $R \leq 128$ пакетов и $R \leq 224$ пакетов при условии существования $\mathrm{span}(1, 64)$ и $\mathrm{span}(1, 32)$ соответственно. Для $n = 8$ битов, $t = 3$ битов и $\gamma = 6$ групп алгоритм 2 правильно вычисляет размер потока, когда существуют $\mathrm{span}(1, 192)$ и $R \leq 32$ пакетов.

Экспериментальная валидация. В экспериментальном исследовании используется тот же подход, что и в работах, посвящённых VL2 [42], pFabric [43] и pHost [44]. В частности, мы проводим имитационное моделирование на основе реалистичных паттернов трафика, порождённых из распределения данных размеров потоков «data-mining», где число потоков составляет 10^6, а максимальное число пакетов в потоке равно $\frac{2}{3} \cdot 10^6$ пакетов; таким образом, для подсчёта требуется 20-битный счетчик. В рассмотренных сценариях, 7-битная (а иногда даже 6-битная) часть счетчика на источнике потока оказывается достаточной для корректного поддержания распределенного счетчика пакетов для каждого потока. Более того, даже в случае 5-битных частей счётчика на источнике потока Алгоритм 2 и Алгоритм 1 правильно вычисляют число пакетов в 99.3% и 95.8% потоков, содержащих не менее 2^5 пакетов. Кроме того, даже в случаях, когда Алгоритм 2 вычисляет $|f|$ неточно, вычисленное значение почти всегда лежит в интервале $[|f| - X, 1.02 \cdot |f|]$, где X — число потерянных пакетов. Более подробно результаты экспериментальной валидации описаны в разделе 7 работы [51].

5. Заключение

В предложенной диссертации исследуются фундаментальные проблемы проектирования сетевых элементов в соответствии с конкретными целями и задачами, представленными в разделе 1.2. Диссертация вносит важный вклад в эффективное представление программ обработки пакетов и устойчивый распределенный мониторинг трафика.

Во-первых, чтобы минимизировать размеры тернарных представлений интервальных классификаторов, мы ввели понятие подинтервала, позволяющее работать с интервалами с побитовым разрешением, что в свою очередь, позволяет преодолеть ограничения представлений сохраняющих структурные свойства классификатора на уровне отдельных полей. Подинтервал — это базовый элемент предложенных эквивалентных RR-представлений, сохраняющих непересекаемость правил на подмножестве битов. Мы показали, что RR-представления обеспечивают существенную экономию требуемой памяти TCAM.

Во-вторых, мы предложили новые объединенные представления для нескольких классификаторов, использующих одни и те же паттерны классификации. Мы изучили случай, когда каждый паттерн классификации (класс) может размещаться в памяти только один раз, независимо от числа классификаторов, содержащих его. В общем случае предложены алгоритмы, минимизирующие количество дублированных фильтров в построенных комбинированных представлениях. Эффективность этих алгоритмов имеет аналитические гарантии и подтверждается экспериментально.

В-третьих, мы обобщили классическую задачу классификации пакетов, представив новую абстракцию приближенных классификаторов, в которой пользователи контролируют точность, за-

ранее выбирая, какие действия могут быть назначены каким фильтрам. Мы разработали методы оптимизации, использующие эту дополнительную гибкость для того, чтобы еще больше оптимизировать размер классификатора. Приблизительные классификаторы используют интересный и ранее не изученный компромисс между требующимися ресурсами и точностью результатов.

В-четвертых, чтобы снизить требования к памяти в одном коммутаторе, мы предлагаем распределенный подход, улучшающий масштабируемость мониторинга потокового трафика за счет использования ресурсов всей сети. Вместо того чтобы жертвовать точностью мониторинга трафика, предлагаемый подход позволяет восстановить точное число пакетов в потоке используя сетевые элементы, у которых есть свободные ресурсы. Чтобы сделать распределенное выполнение задачи мониторинга устойчивым к сетевому шуму, мы придерживались парадигмы однонаправленного транспортного протокола, в которой не вводится дополнительных пакетов, а состояние потока вдоль его маршрута отслеживается при помощи нескольких управляющих битов, добавляемых к пакетам отслеживаемых потоков. В диссертации были разработаны алгоритмы точного устойчивого распределенного вычисления числа пакетов в потоке с аналитически доказанными условиями их корректности. Предлагаемый подход может быть дополнительно расширен для вычисление телеметрических функций в реальном времени (например, подсчет количества утерянных пакетов).

В целом, в предложенной диссертации мы разрабатываем методы повышения эффективности программ обработки пакетов и снижения требований к памяти в одном коммутаторе. Наши результаты могут значительно улучшить вычислительные возможности всей сети и каждого задействованного сетевого элемента. Более сложные сервисы требуют реализации более выразительных и эффективных механизмов обработки пакетов. Эта диссертация показывает возможность такой реализации даже в существующей сетевой инфраструктуре.

Список литературы

[1] OpenFlow 1.3 specification, 2012. — http://www.openflow.org/ wk/index.php/OpenFlow_1_3_proposal. — 2012.

[2] $P4_{16}$ Language Specification. — https://p4.org/p4-spec/docs/P4-16-v1.2.0.html. — 2019.

[3] Synopsys. DesignWare Ternary and Binary Content-Addressable Memory Compilers. — https://www.synopsys.com/dw/ipdir.php?ds=dwc_tcam_memory_compilers.

[4] Cvetkovski A. An Algorithm for Approximate Counting Using Limited Memory Resources // SIGMETRICS. — 2007. — P. 181–190.

[5] Discount Counting for Fast Flow Statistics on Flow Size and Flow Volume / Chengchen Hu [et al.] // IEEE/ACM Trans. Netw. — 2014. — Jun.. — Vol. 22, no. 3. — P. 970–981.

[6] Stanojevic R. Small Active Counters // INFOCOM. — 2007. — P. 2153–2161.

[7] Tsidon E., Hanniel I., Keslassy I. Estimators Also Need Shared Values to Grow Together. // INFOCOM. — 2012. — P. 1889–1897.

[8] Cormode G., Muthukrishnan S. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications // J. Algorithms. — 2005. — Apr.. — Vol. 55, no. 1. — P. 58–75.

[9] Estan C., Varghese G. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice // ACM Trans. Comput. Syst. — 2003. — Aug.. — Vol. 21, no. 3. — P. 270–313.

[10] Pyramid Sketch: A Sketch Framework for Frequency Estimation of Data Streams / Y. Tong [et al.] // Proc. VLDB Endow. — 2017. — Aug.. — Vol. 10, no. 11. — P. 1442–1453.

[11] One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon / Z. Liu [et al.] // SIGCOMM. — 2016. — P. 101–114.

[12] Elastic Sketch: Adaptive and Fast Network-Wide Measurements / T. Yang [et al.] // SIGCOMM. — 2018. — P. 561–575.

[13] Fast and Scalable Layer Four Switching / Venkatachary Srinivasan [et al.] // SIGCOMM 1992. — 1998. — P. 191–202.

[14] Lakshminarayanan Karthik, Rangarajan Anand, Venkatachary Srinivasan. Algorithms for advanced packet classification with ternary CAMs // SIGCOMM. — 2005. — P. 193–204.

[15] Bremler-Barr Anat, Hendler Danny. Space-Efficient TCAM-Based Classification Using Gray Coding // Trans. Computers. — 2012. — Vol. 61, no. 1. — P. 18–30.

[16] Exact Worst Case TCAM Rule Expansion / Ori Rottenstreich [et al.] // IEEE Trans. Computers. — 2013. — Vol. 62, no. 6. — P. 1127–1140.

[17] Chang Yeim-Kuan, Su Cheng-Chien. Efficient TCAM Encoding Schemes for Packet Classification Using Gray Code // GLOBECOM. — 2007. — P. 1834–1839.

[18] DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors / Hao Che [et al.] // Trans. Computers. — 2008. — Vol. 57, no. 7. — P. 902–915.

[19] Bremler-Barr Anat, Hay David, Hendler Danny. Layered interval codes for TCAM-based classification // Computer Networks. — 2012. — Vol. 56, no. 13. — P. 3023–3039.

[20] SAX-PAC (Scalable And eXpressive PAcket Classification) / Kirill Kogan [et al.] // SIGCOMM. — 2014. — P. 15–26.

[21] FIB efficiency in distributed platforms / Kirill Kogan [et al.] // ICNP. — 2016. — P. 1–10.

[22] How to represent IPv6 forwarding tables on IPv4 or MPLS dataplanes / Sergey I. Nikolenko [et al.] // INFOCOM Workshops. — 2016. — P. 521–526.

[23] Chuprikov Pavel, Kogan Kirill, Nikolenko Sergey I. General ternary bit strings on commodity longest-prefix-match infrastructures // ICNP. — 2017. — P. 1–10.

[24] QoS: Modular QoS Command-Line Interface Configuration Guide, Cisco IOS. — https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_mqc/configuration/15-mt/qos-mqc-15-mt-book/qos-mqc.html.

[25] Class of Service Configuration Guide. — https://www.juniper.net/documentation/en_US/junos11.1/information-products/topic-collections/security/software-all/class-of-service/index.html.

[26] Configuring Modular QoS Packet Classification. — http://www.cisco.com/c/en/us/td/docs/routers/xr12000/software/xr12k_r4-2/qos/configuration/guide/qc42clas.html.

[27] Mittal S. A Survey of Techniques for Approximate Computing // ACM Comput. Surv. — 2016. — Vol. 48, no. 4. — P. 62:1–62:33.

[28] Liu A., Meiners C., Torng E. TCAM Razor: a systematic approach towards minimizing packet classifiers in TCAMs // Trans. Networking. — 2010. — Vol. 18, no. 2. — P. 490–500.

[29] Compressing rectilinear pictures and minimizing access control lists / David A. Applegate [et al.] // SODA. — 2007. — P. 1066–1075.

[30] Packet classifiers in ternary CAMs can be smaller / Qunfeng Dong [et al.] // SIGMETRICS. — 2006. — P. 311–322.

[31] Umans C. The Minimum Equivalent DNF Problem and Shortest Implicants // J. Comput. Syst. Sci. — 2001. — Vol. 63, no. 4. — P. 597–611.

[32] Khot S., Saket R. Hardness of Minimizing and Learning DNF Expressions // FOCS. — 2008. — P. 231–240.

[33] Demianiuk Vitalii, Kogan Kirill. How to Deal with Range-Based Packet Classifiers // SOSR. — 2019. — P. 29–35.

[34] New Alternatives to Optimize Policy Classifiers / Vitalii Demianiuk [et al.] // ICNP. — 2018. — Sep. — P. 121–131.

[35] New Alternatives to Optimize Policy Classifiers / V. Demianiuk [et al.] // IEEE/ACM Transactions on Networking. — 2020. — Vol. 28, no. 3. — P. 1088–1101.

[36] Garey Michael R., Johnson David S. Computers and Intractability: A Guide to the Theory of NP-Completeness. — NY : W. H. Freeman & Co., 1979. — ISBN: 0716710447.

[37] Demetrescu Camil, Finocchi Irene. Combinatorial Algorithms for Feedback Problems in Directed Graphs // Inf. Process. Lett. — 2003. — Vol. 86, no. 3. — P. 129–136.

[38] Approximating Minimum Feedback Sets and Multicuts in Directed Graphs / G. Even [et al.] // Algorithmica. — 1998. — Feb. — Vol. 20, no. 2. — P. 151–174.

[39] Demianiuk V., Kogan K., Nikolenko S. I. Approximate Classifiers with Controlled Accuracy // INFOCOM. — 2019. — P. 2044–2052.

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

[41] YAPS: Yet Another Packet Simulator. — `http://wiki.github.com/NetSys/simulator/`.

[42] VL2: A Scalable and Flexible Data Center Network / A. Greenberg [et al.] // SIGCOMM. — 2009. — P. 51–62.

[43] pFabric: Minimal Near-optimal Datacenter Transport / Mohammad Alizadeh [et al.] // SIGCOMM. — 2013. — P. 435–446.

[44] pHost: Distributed Near-Optimal Datacenter Transport over Commodity Network Fabric / Peter X. Gao [et al.] // CoNEXT. — 2015. — P. 1:1–1:12.

[45] Karp Richard M. Reducibility among Combinatorial Problems. — 1972. — P. 85–103.

[46] Foulser David E., Li Ming, Yang Qiang. Theory and Algorithms for Plan Merging // Artif. Intell. — 1992. — Vol. 57, no. 2-3. — P. 143–181.

[47] A computational framework for optimal masking in the synthesis of oligonucleotide microarrays / Simon Kasif [et al.] // Nucleic Acids Research. — 2002. — Vol. 30, no. 20. — P. e106.

[48] Ning Kang, Leong Hon. Towards a better solution to the shortest common supersequence problem: the deposition and reduction algorithm // BMC Bioinformatics. — 2006. — Vol. 7, no. Suppl 4. — P. S12.

[49] Elmokashfi A., Dhamdhere A. Revisiting BGP churn growth // CCR. — 2014. — Vol. 44, no. 1. — P. 5–12.

[50] Strategies for Mitigating TCAM Space Bottlenecks / Kirill Kogan [et al.] // HOTI. — 2014. — P. 25–32.

[51] Robust Distributed Monitoring of Traffic Flows / V. Demianiuk [et al.] // 2019 IEEE 27th International Conference on Network Protocols (ICNP). — 2019. — P. 1–11.