Моделирование и анализ информационных систем. Т. 26, №1 (2019), с. 122–133
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 122–133

DOI: 10.18255/1818-1015-2019-1-122-133

УДК 004.7

Алгоритм минимизации количества правил маршрутизации в ПКС

Петров И. С.

Поступила в редакцию 10 января 2019
После доработки 15 февраля 2019
Принята к публикации 17 февраля 2019

Аннотация. Архитектура ПКС (программно-конфигурируемые сети) предоставляет новые возможности по управлению сетью при помощи физического разделения уровня передачи данных (Data-Plane) от уровня управления данными (Control-Plane). Такое разделение достигается при помощи передачи функций управления сетью на отдельный сетевой элемент — контроллер. Архитектура ПКС позволяет устанавливать на контроллер сетевые приложения, которые могут использовать протокол OpenFlow для реализации множества различных сетевых функций, например для маршрутизации или анализа сетевой статистики. Анализ сетевой статистики производится при помощи счетчиков, установленных на правилах маршрутизации. Чтобы собирать информацию о числе пакетов в различных потоках, ПКС приложения могут устанавливать дополнительные правила маршрутизации, единственной целью которых является подсчет пакетов с определенными заголовками. Для полноценного анализа сетевой статистики приложения должны устанавливать в сеть большое количество дополнительных правил, что может привести к снижению производительности сети. В силу ограниченного размера таблиц маршрутизации, большое число дополнительных правил может мешать другим приложениям устанавливать свои правила. Таким образом, необходимо разработать алгоритм, который будет минимизировать число дополнительных правил. В данной работе рассмотрена задача минимизации числа дополнительных правил, устанавливаемых на ПКС коммутаторы приложениями для анализа сетевой статистики. Был разработан эвристический алгоритм минимизации количества правил маршрутизации, основанный на алгоритме Блейка нахождения сокращенной дизъюнктивной нормальной формы (ДНФ). Экспериментальные исследования показали, что алгоритм уменьшает число правил более чем в 2.2 раза на равномерно распределенных входных данных.

Ключевые слова: ПКС, сетевая статистика, счетчики правил маршрутизации

Для цитирования: Петров И. С., "Алгоритм минимизации количества правил маршрутизации в ПКС", *Моделирование и анализ информационных систем*, **26**:1 (2019), 122–133.

Об авторах:
Петров Иван Сергеевич, orcid.org/0000-0002-7180-6373, аспирант,
Московский государственный университет имени М.В. Ломоносова,
ул. Ленинские горы, 1, строение 52, г. Москва, 119992, Россия, e-mail: ipetrov@cs.msu.ru

Введение

Архитектура ПКС (программно-конфигурируемые сети) предоставляет новые возможности по управлению сетью при помощи физического разделения уровня передачи данных (Data-Plane) от уровня управления данными (Control-Plane). Такое разделение достигается при помощи передачи функций управления сетью на отдельный сетевой элемент — контроллер.

Одним из наиболее популярных протоколов, реализующих логику ПКС, является протокол OpenFlow [4]. Протокол OpenFlow позволяет контроллеру управлять коммутаторами при помощи установки на них правил маршрутизации. Такие правила описывают логику обработки коммутатором пакетов с различными заголовками, поступающих на порты коммутатора.

Правила маршрутизации на коммутаторах хранятся в таблицах маршрутизации. Каждое правило представляет собой кортеж $<match,priority,action>$. Поле *match* описывает заголовки пакетов, которые должно обрабатывать правило маршрутизации. Поле *priority* представляет собой приоритет правила маршрутизации. Данное поле необходимо для разрешения неопределенностей в обработке пакетов. Например, в случае когда два различных правила могут обрабатывать один и тот же пакет, будет выбрано правило с наивысшим приоритетом. В поле *action* записан тип действия, совершаемого с пакетом после обработки пакета данным правилом.

Архитектура ПКС позволяет устанавливать на контроллер сетевые приложения, которые могут использовать протокол OpenFlow для реализации множества различных сетевых функций, например для маршрутизации или анализа сетевой статистики.

Протокол OpenFlow предоставляет разработчикам сетевых приложений механизм анализа сетевой статистики при помощи счетчиков, установленных на правилах маршрутизации. Такие счетчики описывают количество пакетов, обработанных некоторым правилом маршрутизации, с момента его установки в сеть. Так как правила маршрутизации определяют заголовки пакетов, такие счетчики отражают статистику по потокам в сети.

Одним из методов анализа сетевой статистики, используемых такими приложениями, является установка дополнительных правил маршрутизации на границе сети [2]. Такие правила предназначены для того, чтобы считать пакеты для различных потоков в сети.

Для полноценного анализа сетевой статистики приложения должны устанавливать в сеть большое количество дополнительных правил, что может привести к снижению производительности сети. Также могут возникнуть ситуации, когда таблица маршрутизации будет переполнена, что приведет к отказу в обслуживании для вновь подключенных клиентов. Таким образом, возникает задача минимизации количества OpenFlow правил, устанавливаемых в сеть приложениями для анализа сетевой статистики.

В данной работе рассматривается задача минимизации количества OpenFlow правил, устанавливаемых на коммутаторы приложениями, анализирующими сетевую статистику. Модель Header Space используется для описания наборов правил. Основная идея данной модели заключается в абстракции понятия заголовка пакета от конкретных протоколов, составляющих данный заголовок. Таким образом,

заголовки пакетов представляются битовыми векторами длины L, где L — длина максимального заголовка.

В данной работе был разработан эвристический алгоритм минимизации количества OpenFlow правил, основанный на алгоритме Блейка нахождения сокращенной дизъюнктивной нормальной формы (ДНФ). Идея минимизации количества правил маршрутизации основана на установке правилам различных приоритетов.

Работа организована следующим образом. В разделе 1. представлена постановка задачи. Раздел 2. описывает модель и введенные для нее операции. В разделе 3. приводится описание разработанного эвристического алгоритма минимизации количества правил маршрутизации. В разделе 4. описана реализация разработанного алгоритма. В разделе 5. приведены результаты экспериментального исследования.

1. Анализ сетевой статистики

Каждое правило в ПКС имеет свой счетчик, который отражает число пакетов, обработанных правилом со времени его установки в сеть. Каждое правило обрабатывает пакеты с определенными заголовками, и счетчики, таким образом, содержат информацию о числе пакетов в каждом отдельном потоке в сети. Значение счетчика может понадобиться контроллеру для анализа распределения пакетов по потокам в сети. Таким образом, ПКС представляет централизованный способ анализа сетевой статистики.

Анализ сетевой статистики может быть использован для балансировки нагрузки, поиска узких мест в сети и тяжеловесных потоков [7]. Также такой анализ позволяет находить аномалии в сетевом трафике, например, если оборудование было скомпрометировано [8–12].

Для того чтобы собрать полное представление о сетевой статистике, контроллер может устанавливать дополнительные правила в сеть, которые не влияют на поведение сети. Единственная цель этих правил — подсчет числа пакетов в потоках.

Одной из проблем анализа статистики является то, что при наличии в сети большого количества потоков, приложение может устанавливать большое количество дополнительных правил [1], что будет ухудшать работу сети (например, переполнять таблицы маршрутизации). Таким образом, нужен алгоритм, который сможет минимизировать число дополнительных правил, устанавливаемых в сеть для сбора статистики.

2. Представление заголовков пакетов

2.1. Header Space

Для разработки алгоритма минимизации требуется модель, которая будет описывать множества заголовков пакетов и позволит производить операции над этими заголовками. За основу была взята модель Header Space Analysis [5]. Она представляет заголовки пакетов как точки в геометрическом пространстве и изначально использовалась авторами для анализа достижимости узлов в сети.

Пространство заголовков, H: практические значения битов заголовков пакетов, заданные различными протоколами, игнорируются, и заголовок рассматривается как набор 0 и 1 длины L. Формально заголовок становится областью в L-мерном пространстве $\{0,1\}^L$, которое и называется пространством заголовков. Базовый способ определения объектов в H — *wildcard*-выражение (маска), где каждый бит может быть представлен как 0, 1, x. Каждое такое выражение соответствует гиперкубу в пространстве H, а любая область пространства может быть представлена как объединение таких выражений.

Пространство сети, N: модель сети представляется совокупностью коммутаторов с внешними интерфейсами (портами), каждый из которых имеет собственный идентификатор в модели. $N = \{0,1\}^L \times \{1,...,P\}$, где $\{1,...,P\}$ — все порты в сети.

2.2. Представление заголовков

Важная особенность данной модели — не все множества заголовков можно представить в виде одного wildcard-выражения. Например,

$$\{000, 001, 010, 011, 100, 101, 110\}$$

не может быть описан менее, чем суммой трех масок $0xx + 10x + 110$. Сумма масок соответствует объединению задаваемых ими областей. Другой способ представления множества заголовков — разность wildcard-выражений. Например, описанное выше множество может быть представлено следующим образом: $xxx - 111$.

Итак, любое множество заголовков можно представить посредством применения множественных операций к маскам:

$$X = \sum_{i \in [1,n]} \left(x_i - \sum_{j \in [1,m_i]} y_{ij}\right)$$

Такие суммы будем называть *структурами*. Элементы x_i таких структур будем называть "положительными", а элементы y_{ij} — отрицательными. Каждое отдельное подвыражение первой суммы будем называть *срезом*. Пример такого представления множества выглядит следующим образом:

1xxx0	0xxxx	1xx11
- 11x10	- 0xx1x	- 10x11
- 101x0	- 0xxx0	

Это представление соответствует набору заголовков:

$$A = \{10001, 10010, 11000, 11100, 00001, 00101, 01001, 01101, 11011, 11111\}$$

Также оно может быть представлено как:

$$A = (1xxx0 - (11x10 + 101x0)) + (0xxxx - (0xx1x + 0xxx0)) + (1xx11 - (10x11))$$

2.3. Операции с заголовками

Операции, которые можно применять к wildcard-выражениям, определяются следующим образом.

Операция объединения двух структур A и B: срезы B добавляются в A.

$$\begin{aligned} A \cup (1xx00 - (11000)) &= (1xxx0 - (11x10 + 101x0)) + \\ &+ (0xxxx - (0xx1x + 0xxx0)) + \\ &+ (1xx11 - (10x11)) + (1xx00 - (11000))) \end{aligned}$$

Операция пересечения двух областей A и B: для всех пар положительных элементов A и B считается их пересечение. Если пересечение c_{ij} для каких-то a_i и b_j не пусто, то оно становится новым положительным элементом результирующей структуры. c_{ij} также содержит отрицательные элементы, которые получаются при пересечении всех отрицательных элементов a_i и b_j с c_{ij}.

$$A \cap (1xx00 - (11000)) = (1xx00 - (101x0 + 11000))$$

Дополнение к области, соответствующей структуре A, – это пересечение дополнений к каждому положительному элементу a_i со всеми отрицательными элементами всех срезов A.

Разностью между структурами A и B является пересечение A с дополнением B.

$$\begin{aligned} A \setminus (1xx00 - 11000) &= A \cap (0xxxx + xxx1x + xxxx1 + +11000) \\ &= ((1xx10 - (11010 + 11110 + 10110)) + (11000) + \\ &+ (1xxx0 - (11x10 + 101x0)) + (0xxxx - (0xx1x + 0xxx0)) + \\ &+ (1xx11 - (10x11)) + (1xx00 - (11000))) \end{aligned}$$

2.4. Минимизация

Набор правил, используемый для анализа сетевой статистики, может быть представлен в виде *структур* модели Header Space. Для минимизации числа правил маршрутизации нужно минимизировать число слагаемых в структуре. Возможность использовать в базовом представлении операцию вычитания дает следующий вариант представления множества заголовков правилами. Рассмотрим множество заголовков, задающееся объединением трех масок $M = 1xx \cup x1x \cup xx1$. Такое множество требует установку трех правил, соответствующих всем трем маскам. Но при использовании вычитания это множество может быть представлено как: $M = xxx - 000$. Структуры с использованием операции разности могут быть представлены как набор правил OpenFlow с разными приоритетами.

Приоритет правила OpenFlow — это значение, используемое для разрешения конфликтов при обработке пакетов. Такие конфликты появляются, когда заголовки различных OpenFlow правил имеют непустое пересечение. В таком случае входящие пакеты с заголовками из этого пересечения будут обработаны правилом с наибольшим приоритетом.

Приоритеты могут использоваться для минимизации количества правил следующим образом. Например, имеется некоторый набор правил A с различными заголовками. Эти правила собирают статистику с нескольких потоков, выбранных

приложением. Когда создается минимальное представление B для заголовков из A, оно содержит положительные элементы b_i и отрицательные элементы $\overline{b}_j$. Правилам, соответствующим $\overline{b}_j$, поставим более высокий приоритет, чем правилам, соответствующим b_i. Это новое представление далее будет установлено в сеть.

После проведения минимизации в сеть будет установлен набор правил OpenFlow с различными приоритетами.

В алгоритме также не рассматриваются счетчики правил, соответствующих отрицательным элементам структуры. И так как структура с операциями отрицания соответствует изначальному множеству правил, сумма значений счетчиков не изменится.

3. Алгоритм минимизации

Идея алгоритма минимизации основана на модификации метода Блейка [6] для создания дизъюнктивной нормальной формы.

3.1. Метод Блейка

Дизъюнктивная нормальная форма (ДНФ) — дизъюнкция n различных элементарных конъюнкций E_i. Элементарная конъюнкция E_i представляется следующим выражением:

$$E_i = \bigwedge_{j \in [1, m_i]} B_{ij},$$

где $X(n) = \{x_1, x_2, ..., x_n\}$ — множество булевых переменных.

Совершенная ДНФ — это ДНФ, которая не может быть уменьшена с использованием операции импликации для элементарных конъюнкций.

Можно установить взаимнооднозначное соответствие между ДНФ и wildcard-выражениями из Header Space. Пусть маска W представляется как $w_1 w_2 \dots w_k$. Элементарная конъюнкция E, которая соответствует W, создается по следующим правилам:

$$\begin{aligned} w_i = 1 \quad &\Rightarrow \quad B_i \in E \\ w_i = 0 \quad &\Rightarrow \quad \overline{B}_i \in E \\ w_i = x \quad &\Rightarrow \quad B_i, \overline{B}_i \notin E \end{aligned}$$

для всех $i \in [1, k]$.

Так как элементарная конъюнкция соответствует одной маске, ДНФ можно представить в виде суммы масок. То есть ДНФ представляет собой выражение с *wildcard*-масками без использования операции разности.

Метод Блейка состоит из двух операций: *расширения* и *склеивания*. Операция расширения применяется к паре масок следующим образом:

$$w_1 w_2 x + w_1 x w_3 = w_1 w_2 x + w_1 x w_3 + x w_2 w_3,$$

где w_i — это 0 или 1 и x представляет произвольный бит.

Операция склеивания применяется к паре масок W_1 и W_2. Эта операция уменьшает число масок в выражении следующим образом: если $W_1 \subseteq W_2$, то W_1 будет удалено. Операция "$\subseteq$" обозначает, что множество заголовков, представленное W_1, является подмножеством множества, представленного W_2.

Метод Блейка работает следующим образом. Операция расширения используется для создания новых масок до тех пор, пока не останется возможности построить новую маску. После этого уменьшается число масок в выражении при помощи операции склеивания.

3.2. Минимизация масок

В данной работе метод Блейка используется для создания эвристического алгоритма минимазации числа масок в выражениях. Основа эвристики — это уменьшение числа заголовков путем присваивания каждому новому элементу W, созданному в методе Блейка, собственного счетчика. Счетчик элемента W подсчитывает число успешных операций склейки, произведенных с W в течение работы алгоритма. Под успешной операцией склейки понимается операция, которая привела к удалению одного из аргументов. Эти счетчики используются в конце работы алгоритма для удаления элементов, которые не содержат уникальных заголовков. Значения счетчиков таких элементов будут равны нулю, так как они не были объединены с другими элементами *структуры*.

Такая эвристика гарантирует, что число элементов в *структуре* не увеличится.

Также необходимо учитывать особенности представления данных, а именно наличие отрицательных масок. То есть нужно модифицировать операции *расширения* и *склеивания* в методе Блейка.

Операция *расширения* изменяется следующим образом. При проведении операции расширения $W_1 \cup W_2 = W_1 \cup W_2 \cup W_3$ необходимо проверять, что отрицательные элементы W_1 и W_2 пересекаются с элементом W_3. Если пересечение I не пусто, то элементы из I становятся отрицательными элементами W_3.

Операция *поглощения* изменяется следующим образом. При проведении операции поглощения с аргументами W_1 и W_2 (W_2 поглощается W_1) необходимо проверять, что все отрицательные элементы W_1, пересекающиеся с W_2, содержались в отрицательных элементах W_2. Таким образом, W_2 не содержит уникальных заголовков и может быть удален.

В результате модификации операций новое представление множества заголовков не больше, чем изначальное представление.

4. Реализация

В контексте имеющегося представления данных и операций над ними был реализован эвристический алгоритм. Алгоритм основан на модифицированной версии метода Блейка, в котором учитываются отрицательные маски и подсчет количества операций поглощения.

Разработанный алгоритм работает следующим образом. К каждой паре положительных масок W_1 и W_2 применяется операция расширения. Если полученное

Algorithm 1 Минимизация *wildcard*-выражений

Input: *Wildcard*-выражение W и размер L пространства заголовков
Output: Минимизированное *wildcard*-выражение W

```
1: k ← |W|
2: count ← new array(|W|)
3: for i from 1 to |W| do
4:    count[i] ← 1
5: for i from 1 to |W| do
6:    for j from 1 to |W| do
7:       for s from 1 to L do
8:          if W[i].elem[s] ≠ W[j].elem[s] ≠ x then
9:             w ← extend(W[i], W[j], s)
10:            if w ≠ ∅ then
11:               W ← W ∪ w
12:               k ← k + 1
13:               for all w̄ ∈ W[i].neg ∪ W[j].neg do
14:                  x ← w ∩ w̄
15:                  if x ≠ ∅ then
16:                     W[k].neg ← W[k].neg ∪ x
17:               count[k] ← 0
18:               for r from 1 to |W| do
19:                  if W[r] ⊆ W[k] then
20:                     delete W[r]
21:                     count[k] ← count[k] + 1
22:               if count[k] = 0 then
23:                  delete W[k]
24: for i from 1 to |W| do
25:    for j from i to |W| do
26:       if W[i] ⊆ W[j] then
27:          delete W[i]
28:          break
29:       if W[j] ⊆ W[i] then
30:          delete W[i]
31: for i from 1 to |W| do
32:    if count[i] = 0 then
33:       delete W_i
```

расширение W_3 не пусто, то оно добавляется в *wildcard*-выражение как новый положительный элемент. Отрицательные элементы, соответствующие новому положительному элементу W_3, создаются из отрицательных элементов W_1 и W_2.

На следующем шаге каждой пары положительных элементов W_1 и W_2 проверяется, что $W_1 \subseteq W_2$. Если это утверждение верно, то элемент W_1 удаляется.

В алгоритме также учитываются индивидуальные счетчики для каждого элемента W. Эти счетчики описывают количество успешных операций склейки с элементом W и используются для удаления элементов, которые не содержат уникальных заголовков.

Разработанный алгоритм представлен в Algorithm 1. Положительные элементы структуры A обозначаются $A[i].elem$, отрицательные — $A[i].neg[j]$. Операция $extend(W_1, W_2, s)$ описывает *расширение* двух масок W_1 и W_2 в поцизии s.

5. Экспериментальное исследование

5.1. Эксперименты

Экспериментальное исследование проводилось с использованием случайных *wildcard*-выражений и вычислением *коэффициента уменьшения структуры*. Этот коэффициент определяется как размер оригинального представления множества заголовков, деленный на размер минимизированного представления.

Случайные *wildcard*-выражения создаются следующим образом. Выбирается случайное число N — число положительных масок в *wildcard*-выражении. Для каждого положительного элемента *wildcard*-выражения выбирается случайное число отрицательных элементов $n_i, i = 1, 2, \ldots, N$. Каждый положительный элемент генерируется побитово. Для каждого отрицательного элемента биты генерируются случайно только на тех позициях, на которых в положительной маске стоит x.

Для оценки зависимости эффективности работы алгоритма от входных множеств заголовков измерялся *коэффициент уменьшения структуры* в двух различных сценариях:

1. В первом сценарии измерялся *коэффициент уменьшения структуры* в зависимости от размера заголовков.

2. Во втором сценарии измерялся *коэффициент уменьшения структуры* в зависимости от количества отрицательных элементов.

5.2. Результаты

Для каждой экспериментальной ситуации производилось 100 запусков, полученные значения усреднялись.

На рисунке 1 показано отношение степени уменьшения размера структуры к размеру заголовка. Этот эксперимент показывает нелинейный рост степени уменьшения структуры при росте размера заголовка.

На рисунке 2 показана зависимость коэффициента уменьшения структуры от отношения среднего числа отрицательных масок к числу положительных.

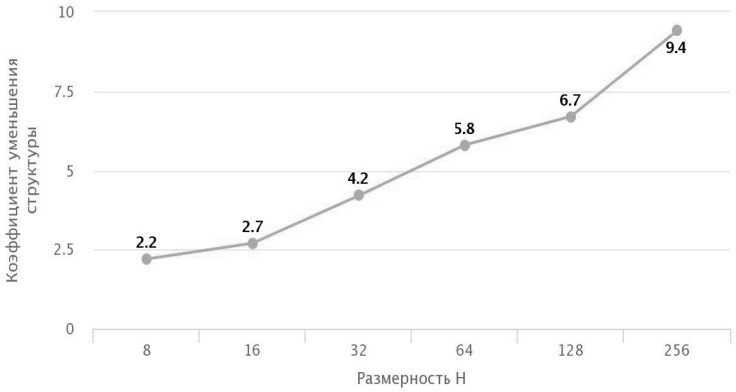

Рис. 1. Зависимость коэффициента уменьшения структуры от размера заголовка

Fig. 1. Reduction rate depending on the header size

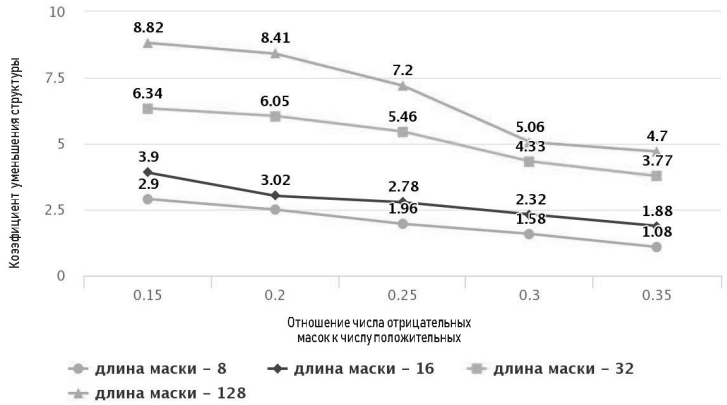

Рис. 2. Зависимость коэффициента уменьшения структуры от относительного числа отрицательных элементов

Fig. 2. Reduction rate depending on the negative element percentage

Этот эксперимент показывает, что “квадратная“ форма структуры негативно влияет на эффективность работы алгоритма. “Квадратной“ считается форма, где число положительных масок равно среднему числу отрицательных масок. Таким образом, алгоритм показывает лучший результат, когда количество отрицательных элементов меньше количества положительных.

На рисунке 3 показана зависимость времени работы алгоритма от размера заголовка. Этот эксперимент показывает линейный рост времени с ростом размера заголовка.

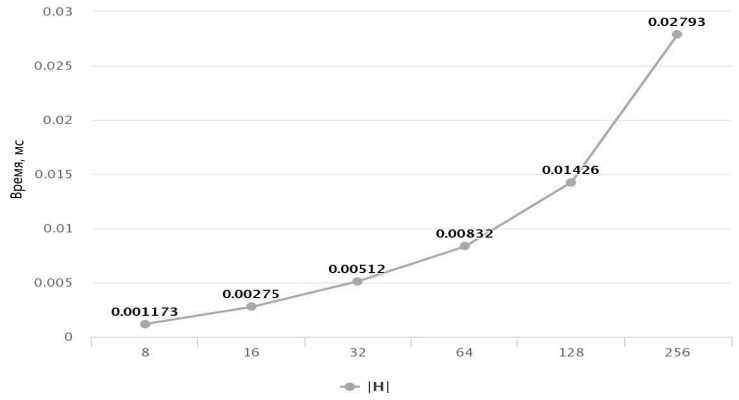

Рис. 3. Зависимость времени (в мс) от размера заголовка

Fig. 3. Time (in ms) depending on the header size

Заключение

Архитектура ПКС представляет разработчикам сетевых приложений механизм анализа сетевой статистики при помощи счетчиков, установленных на правилах маршрутизации. Но для полноценного анализа сетевой статистики приложениям необходимо устанавливать в сеть большое количество дополнительных правил, что может привести к снижению производительности сети.

В данной работе рассматривается задача минимизации количества OpenFlow правил, устанавливаемых на коммутаторы приложениями, анализирующими сетевую статистику. Был разработан эвристический алгоритм минимизации количества OpenFlow правил, основанный на алгоритме Блейка нахождения сокращенной ДНФ. Минимизация количества OpenFlow правил производится при помощи настройки полей *match* и приоритетов правил.

Разработанный алгоритм был реализован в виде прототипов экспериментальных средств. Также было произведено экспериментальное исследование разработанного алгоритма, которое показало, что количество правил уменьшается более чем в 2.2 раза и зависит от длины заголовка и от соотношения числа положительных и отрицательных элементов в представлении множества заголовков.

Список литературы / References

[1] Петров И. С., Смелянский Р. Л., “Минимизация группового трафика и обеспечение его отказоустойчивости в программно-конфигурируемых сетях”, *Известия Российской академии наук. Теория и системы управления*, 2018, № 3, 64–75; in English: Petrov I.S., Smeliansky R.L., “Minimization of Multicast Traffic and Ensuring Its Fault Tolerance in Software-Defined Networks”, *Journal of Computer and Systems Sciences International*, **57**:3 (2018), 407–419.

[2] Petrov I. S., “Mathematical model for predicting forwarding rule counter values in SDN”, Young Researchers in Electrical and Electronic Engineering (2018 IEEE Conference of Russian), 1313–1317.

[3] Смелянский Р. Л., "Программно-конфигурируемые сети", *Открытые системы. СУБД*, **9** (2012), 15–26; [Smeliansky R.L., "Software Defined Network", *Open Systems. DBMS*, **9** (2012), 15–26, (in Russian).]

[4] *OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06)*, Open Networkig Foundation, 2015.

[5] Kazemian P., Varghese G., McKeown N., "Header Space Analysis: Static Checking for Networks", *Proceedings of NSDI'12*, 2012, 1–14.

[6] Ложкин С. А., *Лекции по основам кибернетики*, М., 2017; [Lozhkin S. A., *Lektsii po osnovam kibernetiki*, Moscow, 2017, (in Russian).]

[7] Akyildiz I.F., et al., "A roadmap for traffic engineering in SDN-OpenFlow networks", *Computer Networks*, **71** (2014), 1–30.

[8] Pang Chunhui, Yong Jiang, and Qi Li, "FADE: Detecting forwarding anomaly in software-defined networks", *2016 IEEE International Conference on Communications (ICC)*, IEEE, 2016, 1–6.

[9] Kamisinski A., Carol F., "Flowmon: Detecting malicious switches in software-defined networks", *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*, ACM, 2015, 39–45.

[10] Chao Tzu-Wei, et al, "Securing data planes in software-defined networks", *NetSoft Conference and Workshops (NetSoft)*, IEEE, 2016, 465–470.

[11] Гайворонская С. А., Петров И. С., "О применимости методов обнаружения шеллкодов платформы x86 для платформы ARM", *Проблемы информационной безопасности. Компьютерные системы*, 2014, №3, 115–122; [Gaivoronskaya S.A., Petrov I.S., "Towards Applicability of Shellcode Detection Methods Based on x86 Platform for Arm", *Information Security Problems. Computer Systems*, 2014, №3, 115–122, (in Russian).]

[12] Dhawan M., et al., "SPHINX: Detecting Security Attacks in Software-Defined Networks", *NDSS*, 2015, 8–11.

Petrov I. S., "Algorithm for Reducing the Number of Forwarding Rules Created by SDN Applications", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 122–133.

DOI: 10.18255/1818-1015-2019-1-122-133

Abstract. Software-Defined Networking (SDN) is a network architecture that introduces a physical separation of data-plane from control-plane. It implements a new way of analyzing network statistics through counters installed on forwarding rules. These counters measure the number of packets processed by these rules and represent per-flow network statistics. In order to get information about the number of packets from different flows SDN applications can install additional forwarding rules, sole purpose of which is to count packets with specific headers. But in order to produce a full network statistics analysis these applications may install a large amount of forwarding rules thus limiting the space in the forwarding table for other applications. So we need algorithms to minimize the number of such rules. In this paper, we consider the problem of minimizing the number of forwarding rules installed on SDN switches by applications that analyze network statistics. We introduce a heuristic algorithm that creates a reduced representation for sets of rules installed in the network. The experimental results show that this algorithm reduces the number of rules by at least 2.2 times on uniformly distributed random input.

Keywords: SDN; network statistics; forwarding rule counters

On the authors:
Ivan S. Petrov, orcid.org/0000-0002-7180-6373, PhD student
Lomonosov Moscow State University,
1, bd. 52 Leninskie gory, Moscow, 119992, Russia, e-mail: ipetrov@cs.msu.ru