Модел. и анализ информ. систем. Т. **20**, № **6** (2013) 36–51

УДК 519.681

Формальная модель и задачи верификации программно-конфигурируемых сетей

Захаров В.А.[1], Смелянский Р.Л.[2], Чемерицкий Е.В.[2]

Московский государственный университет им. М.В. Ломоносова
119991, Москва, ГСП-1, Ленинские горы, д. 1, стр. 52
Центр прикладных исследований компьютерных сетей

e-mail: zakh@cs.msu.su, rsmeliansky@arccn.ru, echemeritskiy@arccn.ru

получена 10 ноября 2013

Ключевые слова: программно-конфигурируемая сеть, коммутатор, контроллер, правило коммутации, пакет, формальная модель, спецификация, верификация моделей

Программно-коммутируемые сети (ПКС) — это класс компьютерных телекоммуникационных сетей, появившийся несколько лет назад в стремлении упростить проектирование и повысить гибкость управления сетями за счет разделения потоков данных (пакетов) и потоков управления (сообщений и команд), циркулирующих в сетях. ПКС представляет собой распределенную систему, в которой один или несколько контроллеров управляют множеством сетевых коммутаторов, обеспечивающих продвижение пакетов по каналам сети. Функциональные возможности и порядок взаимодействия коммутаторов и контроллеров ПКС определяются протоколом OpenFlow. На основе аппарата булевых функций и дискретных преобразователей нами предложена формальная модель ПКС, введен прототип формального языка спецификаций, поставлены задачи верификации моделей ПКС и получены оценки их сложности. Для одной из задач верификации моделей ПКС описан метод ее решения, на основе которого разработано программно-инструментальное средство верификации ПКС.

1. Программно-конфигурируемые сети и протокол OpenFlow

С самого начала развития сетевых технологий компьютерные телекоммуникационные сети состояли из разнородных специализированных компонентов (маршрутизаторов, коммутаторов, межсетевых экранов, шлюзов). Каждое из вычислительных

[1]Работа поддержана грантом РФФИ 12-01-00706.
[2]Работа поддержана фондом «Сколково», грант N 79, июль, 2012.

устройств в составе сети выполняет довольно изощренные алгоритмы, обеспечивающие решение таких задач, как раскрытие топологии сети, маршрутизация пакетов, отслеживание и балансировка нагрузки, контроль доступа и др. Обычная крупномасштабная сеть может включать сотни или тысячи подобных устройств, причём аппаратура и программное обеспечение большинства из них закрыты для постороннего доступа. Управление такими сетями осуществляется путём конфигурирования их отдельных устройств через специализированные интерфейсы. По мере того как размер сетей увеличивается, а сетевые протоколы становятся замысловатыми, правильное и оптимальное конфигурирование обычных сетей становится очень сложной задачей, сопряженной с большим числом трудно выявляемых ошибок. Сложность настройки традиционных сетей, связанная с согласованием работы большого числа разнообразных независимо работающих устройств, является одним из наиболее серьезных препятствий в развитии новых сетевых технологий, таких как центры обработки данных и облачные вычисления.

В последнее время активно развивается новый вид архитектуры сети — программно-конфигурируемые сети (ПКС, Software Defined Networks, SDN), — широкое внедрение которого способно существенно упростить решение указанной проблемы. Отличительные особенности ПКС состоят в том, что: 1) пространство потоков данных (data plane, data flows) и пространство потоков команд управления этими данными (control plane, control flows) физически разделены, и 2) несколько устройств, регулирующих распространение потоков данных в сети, могут находиться под контролем одной и той же управляющей программы. За счет этого значительно расширяются возможности управления функционированием телекоммуникационной сети, поведение ее компонентов становится более согласованным. Как и в компьютерных сетях традиционного вида, потоки данных циркулируют по каналам связи между коммутаторами. Однако в ПКС потоки управления циркулируют по специально выделенным каналам (в частности, виртуальным, в том случае, если канал передачи данных и канал управления разделяют одну и ту же физическую линию связи), соединяющим коммутаторы и контроллеры. Контроллеры – это вычислительные устройства (серверы), на которых выполняются прикладные программы, управляющие коммутацией и маршрутизацией пакетов в сети. Взаимодействие коммутаторов и контроллеров обеспечивают специальные сетевые протоколы. Одним из таких протоколов является протокол Open Flow [1]. Основные положения этого протокола перечислены ниже.

ПКС — это распределенная система реального времени, компоненты которой разделены на два класса — коммутаторы и контроллеры.

Коммутатор — это сетевое устройство, снабженное несколькими портами, каждый из которых имеет входной и выходной буфер. Порты коммутатора соединены с портами других коммутаторов физическими каналами связи (каналами передачи данных). По этим каналам циркулируют пакеты. Каждый порт коммутатора имеет уникальный номер, который выступает в роли имени порта. Кроме того, один из портов коммутатора соединен каналом управления с контроллером. По этому каналу коммутатор пересылает контроллеру пакеты и статистические данные и принимает от контроллера сообщения. Коммутатор снабжен таблицей коммутации. Пакет, поступивший во входной буфер одного из портов, подключенных к каналу передачи данных, передается в таблицу коммутации, обрабатывается этой таблицей

(операция коммутации) и затем либо поступает в выходной буфер одного из портов передачи данных или управления, либо сбрасывается. Пакеты, поступившие в выходной буфер канала передачи данных, пересылаются по этому каналу во входной буфер того порта, который находится на другом конце этого канала (операция пересылки). Пакет, поступивший в выходной буфер канала управления, пересылается контроллеру. Сообщения, поступившие во входной буфер порта, подключенного к каналу управления, могут содержать:

- запросы на предоставление статистических данных о применении определенных правил коммутации,
- управляющие команды, вносящие изменения в таблицы коммутации,
- команды, требующие обработки прилагающихся к этим командам пакетов согласно заданным инструкциям.

В свою очередь, в выходной буфер порта, подключенного к каналу управления, коммутатор может отправить:

- пакет, решение о коммутации которого должен принять контроллер,
- предупреждение об удалении из таблиц коммутации тех правил, срок активности которых истек, или тех правил, изъятия которых потребовал контроллер.

Пакеты — это элементарные структуры данных, автономно циркулирующие в сети под воздействием операций коммутации и пересылки. Каждый пакет состоит из заголовка и нагрузки. При выполнении операций коммутации и пересылки, а также при обработке контроллером сообщения с вложенным пакетом, нагрузка пакета во внимание не принимается и не изменяется. Каждый заголовок пакета состоит из нескольких полей. В этих полях указываются физические и виртуальные адреса абонентов сети (как правило, адреса отправителя и получателя пакета), информация о сетевом протоколе, которым должен обрабатываться пакет, и пр. При пересылке пакета его заголовок не изменяется.

Правило коммутации состоит из шаблона, списка действий, приоритета, сроков активности, счетчика.

Шаблон — это список пар вида $(field, pattern)$, которые мы будем называть масками, где $field$ — наименование некоторого поля заголовка пакета, $pattern$ — строка, состоящая из символов 0,1 (двоичные символы) и * (символ неопределенности). Заголовок пакета совместим с маской, если все двоичные символы строки $pattern$ совпадают с соответствующими битами поля $field$ в заголовке пакета. Правило коммутации применимо к пакету, если его заголовок совместим со всеми масками шаблона.

Действие — это элементарная операция обработки заголовка пакета в процессе его коммутации. Существуют два типа действий: действие модификации заголовка пакета и действие коммутации пакета. Действие модификации заголовка пакета вносит изменение в одно из полей заголовка пакета. Действие коммутации пакета направляет пакет в выходной буфер порта с заданным именем. Действия применяются к обрабатываемому пакету в порядке их расположения в списке. Если среди действий этого списка есть действия коммутации пакета в выходные буфера,

то копия пакета с тем заголовком, который был преобразован предшествующими действиями списка, направляется в соответствующий выходной порт. Если список действий не оканчивается действием коммутации пакета, то пакет сбрасывается.

Приоритет — это натуральное число, указывающее степень значимости правила и использующееся для их избирательного применения: из нескольких правил таблицы, применимых к одному и тому же пакету, для его обработки выбирается правило с максимальным приоритетом.

Срок активности — это максимальная продолжительность времени пребывания правила в таблице правил коммутатора. Два срока активности определяют время жизни правила: максимальный срок жизни и максимальный срок простоя. Если хотя бы один из сроков активности правила истек, то правило удаляется из таблицы, и коммутатор оповещает об этом событии контроллер.

Счетчики служат для учета количества пакетов, при обработке которых приходилось обращаться к правилу.

Таблица коммутации пакетов — это список правил коммутации пакетов, снабженный алгоритмом (процедурой) выбора подходящих правил для обработки пакетов, поступающих на вход таблицы. Обработка очередного пакета проводится на основании следующих положений. В таблице выделяются все правила, применимые к данному пакету, и из них выбирается правило с наибольшим приоритетом. Протокол OpenFlow не указывает, как должна быть разрешена коллизия, при которой несколько правил с одинаковым максимальным приоритетом применимы к одному и тому же пакету. Если такое правило единственно, то список его инструкций применяется к обрабатываемому пакету, и этот пакет покидает таблицу. При этом показатели счетчиков, приписанных данному правилу, соответствующим образом изменяются. В таблице коммутации также зарезервировано специальное правило для пропущенных пакетов, которое применяется ко всем пакетам, не подпадающим под шаблон какого-либо другого правила.

Состав таблицы коммутации изменяется под воздействием следующих факторов:

1. Истечение одного из сроков активности какого-либо правила. В этом случае правило, срок активности которого истек, удаляется из таблицы. Об этом событии оповещается контроллер.

2. Коммутатор получает по каналу управления команду, требующую добавить в таблицу новое правило. В этом случае правило, являющееся параметром такой команды, добавляется в таблицу коммутации. В том случае, если добавляемое правило уже присутствует в таблице, то выполнение указанной команды приводит к сбросу показаний счетчика и переустановки таймеров этого правила.

3. Коммутатор получает по каналу управления команду, требующую изъять из таблицы все правила с определенными шаблонами. В этом случае все указанные правила удаляются из таблицы.

Контроллер преобразует таблицы тех коммутаторов, которые находятся под его управлением. Преобразование таблиц выполняется прикладной программой на основании информации, поступающей от коммутаторов, при помощи команд преобразования таблиц. В описании протокола OpenFlow допустимы следующие основные

типы сообщений и команд, которыми могут обмениваться контроллер и коммутатор по каналу управления (здесь перечислены лишь те типы сообщений, которые существенно влияют на поведение коммуникационной системы).

1. Команды изменения содержимого таблиц коммутации. Команда *add* предписывает коммутатору внести в таблицу новое правило коммутации с заданными параметрами. Команда *delete* предписывает удалить из таблицы все правила с заданным приоритетом, шаблоны которых вкладываются в некоторый заданный шаблон. Команда $modify$ предписывает во всех правилах с заданным приоритетом, шаблоны которых вкладываются в некоторый заданный шаблон, заменить существующий в этих правилах список действий на новый заданный список действий.

2. Команды обработки индивидуальных пакетов предписывают заданному коммутатору применить к заданному пакету, который прилагается к этой команде, последовательно все действия, указанные в заданном списке действий.

3. Команды запроса статистики предписывают заданному коммутатору отправить контроллеру статистическую информацию (показания счетчиков) всех правил таблицы коммутации, шаблоны которых вкладываются в некоторый заданный шаблон. В ответ на эту команду коммутатор посылает контроллеру сообщение с запрашиваемыми статистическими данными.

4. Сообщения о необработанном пакете. Они отправляются контроллеру всякий раз, когда одно из правил некоторой таблицы коммутатора отправляет пакет в выходной буфер порта, подключенного к каналу управления. Это сообщение предупреждает контроллер о том, что задача коммутации пакета возлагается на прикладные программы контроллера.

5. Сообщения об удалении правила. Они предупреждают контроллер о том, что правило с заданными параметрами было удалено из таблицы заданного коммутатора и при этом в момент удаления счетчики этого правила имели определенные показания.

ПКС могут значительно упростить существующие сетевые приложения и послужить удобной платформой для разработки новых применений сетевых технологий (см. [2]). Главное преимущество ПКС-технологии состоит в том, что программист может сравнительно просто управлять поведением всей сети, изменяя должным образом правила коммутации пакетов в таблицах коммутаторов. Тем не менее, задача проверки корректности остается очень острой. Кроме того, ПКС допускает возможность для нескольких контроллеров управлять одной и той же сетью коммутаторов; возникающие при этом конфликты между управляющими программами могут нарушить политику коммутации всей сети. Решить эти проблемы могло бы программно-инструментальное средство, позволяющее проверять 1) корректность отдельных прикладных программ, управляющих контроллерами ПКС, относительно индивидуальных политик коммутации (все изменения, которые вносятся в сеть, удовлетворяют требованиям политики коммутации), 2) проверять непротиворечивость политик коммутации, реализуемых этими прикладными программами (хотя

бы одна конфигурация сети удовлетворяет требованиям всех политик коммутации), и 3) отслеживать и проверять безопасность и корректность поведения всей ПКС в целом.

Существует совсем немного работ, авторы которых попытались применить формальные методы для проверки поведения ПКС. В статье [3] введена реляционная модель коммуникационной сети и описано средство верификации свойств достижимости в маршрутизации пакетов, в котором модель сети представляется посредством двоичных разрешающих диаграмм (BDD). Аналогичные модели были рассмотрены в статьях [4, 5, 6, 7]; их авторы использовали иные методы (проверка выполнимости булевых формул, преобразования ДНФ) для верификации того же самого класса свойств достижимости. Недостаток моделей ПКС, предложенных в этих работах, состоит в том, что в них моделируется лишь сеть коммутаторов без контроллера; таким образом, эти модели позволяют описывать и верифицировать лишь моментальные конфигурации сети. В моделях, описнных в статье [8, 9, 10], ПКС рассматривается как конечный автомат (система переходов), изменяющая свое состояние в тех случаях, когда коммутатор отправляет пакет или сообщение по каналу связи или модифицирует таблицу правил. Для верификации автоматных моделей применялись методы статического анализа программ. Однако в автоматные модели плохо внедряются методы символьных вычислений, обращение к которым неизбежно для сетей большого размера. Что касается формальной спецификации поведения ПКС, то авторы всех упомянутых работ использовали для этой цели темпоральные логики (CTL или LTL) как средство описания множеств маршрутов, прокладываемых в сети коммутаторами.

В данной статье также исследуется задача верификации ПКС. Наш вклад в исследование этой задачи состоит в том, что

- введена комбинированная реляционно-автоматная формальная модель, которая охватывает некоторые существенные аспекты поведения ПКС;
- предложен вариант формального языка спецификаций политик коммутации пакетов, в котором для описания свойств достижимости используется оператор транзитивного замыкания отношения коммутации пакетов, а для описания свойств поведения ПКС в целом используются темпоральные операторы;
- в предложенной модели сформулирована задача верификации ПКС относительно спецификаций политик коммутации пакетов.

2. Формальная модель программно коммутируемых сетей

В этом разделе мы опишем введенную нами комбинированную реляционно-автоматную формальную модель ПКС. В отличие от ранее исследованных моделей ПКС, предложенная нами модель позволяет описывать и анализировать как отношения коммутации пакетов между узлами сети (реляционная составляющая модели), так и поведение контроллера, взаимодействующего с коммутаторами сети (автоматная составляющая модели).

Введенная нами модель ПКС — это конечная модель дискретного времени; на этом уровне абстракции не отражены некоторые особенности поведения ПКС, которые определяются таймерами и счетчиками. Вследствие этого, представленная модель не учитывает такие возможности коммутаторов, как удаление правил коммутации пакетов из таблиц коммутации по истечении срока активности правил, подсчет числа срабатываний правил коммутации пакетов. В отличие от моделей ПКС, представленных и исследованных в статьях [8, 9, 10], наша модель предназначена для анализа маршрутов передачи пакетов, прокладываемых правилами коммутации, а не для анализа состояний отдельных пакетов в процессе их перемещения в сети. Семантика предложенной модели ПКС определяется *отношением коммутации пакетов*, которое задается на множестве *состояний пакетов*. Состояние пакета определяется заголовком пакета и его местоположением в сети. Отношение коммутации пакетов описывает, как изменяются состояния пакетов при перемещении их в сети. Правило коммутации, будучи примененным к пакету, изменяет его состояние за счет модификации заголовка пакета и переноса пакета из одного узла (буфера) коммутатора в другой узел (буфер). Состояние пакета изменяется и в том случае, когда пакет пересылается по каналу передачи данных от одного коммутатора сети к другому коммутатору или по каналу управления от коммутатора сети к контроллеру сети. В последнем случае состояние пакета служит сообщением, которое сеть коммутаторов отправляет контроллеру. В ответ на такое сообщение контроллер может выработать последовательность команд, вносящих изменения в таблицы коммутации некоторых коммутаторов сети. Таким образом, контроллер может мыслиться как дискретный преобразователь, имеющий множество состояний пакетов в качестве входного алфавита и множество команд модификации таблиц коммутации в качестве выходного алфавита. При получении сообщения от коммутатора контроллер переходит в новое состояние управления и вырабатывает конечную последовательность команд, адресованную коммутаторам сети. Эти команды модифицируют таблицы коммутации и, таким образом, изменяют отношение коммутации пакетов. Для стороннего наблюдателя поведение ПКС может представляться в виде чередующейся последовательности сообщений, отправляемых контроллеру, и отношений коммутации пакетов, которые реализуются сетью коммутаторов. Формальное определение указанной модели таково.

Далее мы будем использовать термин «сеть» для множества коммутаторов, соединенных каналами передачи данных, а термин «ПКС» — для распределенной системы, состоящей из сети и контроллера, взаимодействующих по каналу управления.

Заголовки пакетов представляются двоичными векторами $\mathbf{h} = (h_1, h_2, \ldots, h_N)$; для компонент заголовков будем использовать записи вида $\mathbf{h}[i]$, $\mathbf{h}[i] = h_i$, $1 \le i \le N$. Множество всех заголовков обозначим символом $\mathcal{H}$, $\mathcal{H} = \{0,1\}^N$.

Порт коммутатора представляется вектором $\mathbf{p} = (p_0, p_1, p_2, \ldots, p_k)$, для его компонент будем использовать записи вида $\mathbf{p}[i]$, $\mathbf{p}[i] = p_i$, $0 \le i \le k$. Если $\mathbf{p}[0] = 1$ ($\mathbf{p}[0] = 0$), то порт $\mathbf{p}$ считается входным (выходным). Будем полагать, что все коммутаторы имеют одинаковое число портов. Множество всех (входных, выходных) портов коммутатора обозначим записью $\mathcal{P}$ (соответственно $\mathcal{IP}, \mathcal{OP}$). Выходной порт $\mathbf{p} = (0, 0, \ldots, 0)$ будем считать портом сброса пакетов и обозначать записью *drop*. Выходной порт $\mathbf{p} = \langle 0, 1, 1, \ldots, 1\rangle$ будем считать портом канала управления и обо-

значать записью *octr*; через этот порт коммутатор отправляет сообщения контроллеру. Входной порт $\mathbf{p} = \langle 1, 1, 1, \ldots, 1\rangle$ — это входной порт канала управления; он обозначается записью *ictr*, и через этот порт коммутатор получает команды от контроллера.

Индивидуальным именем каждого коммутатора служит двоичный вектор $\mathbf{w} = (w_1, w_2, \ldots, w_m)$. Множество всех имен коммутаторов обозначим записью $\mathcal{W}$.

Пары $\langle \mathbf{h}, \mathbf{p}\rangle,\ \mathbf{h} \in \mathcal{H},\ \mathbf{p} \in \mathcal{P}$, назовем *локальными состояниями пакетов*, а пары $\langle \mathbf{p}, \mathbf{w}\rangle,\ \mathbf{p} \in \mathcal{P},\ \mathbf{w} \in \mathcal{W}$, — *узлами сети*. Тройки $\langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle,\ \mathbf{h} \in \mathcal{H},\ \mathbf{p} \in \mathcal{P}, \mathbf{w} \in \mathcal{W}$, назовем *состояниями пакетов*. Множество состояний пакетов обозначим буквой $\mathcal{S}$.

Шаблоном заголовка назовем троичный вектор $\mathbf{z} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$, где $\sigma_i \in \{0, 1, *\},\ 1 \le i \le N$, а *шаблоном порта* — подобного же рода троичный вектор $\mathbf{y} = (\delta_1, \delta_2, \ldots, \delta_k)$. Шаблоны используются как для выбора правил коммутации пакетов, так и для модификации заголовков пакетов.

Мы рассматриваем два типа действий: *действие коммутации* $OUTPUT(\mathbf{y})$, где $\mathbf{y} \in \mathcal{OP}$, и *действие модификации заголовка* $SET\_FIELD(\mathbf{z})$, где $\mathbf{z}$ — шаблон заголовка. Конечная последовательность действий называется *инструкцией*.

Правило коммутации пакетов определяется тройкой $\mathbf{r} = \langle(\mathbf{z}, \mathbf{y}), \alpha\rangle$, где $\mathbf{z}, \mathbf{y}$ — это шаблоны заголовка и порта, а α — инструкция. *Таблица коммутации* — это конечное множество правил коммутации.

Топология сети и функциональность коммутаторов описываются бинарными отношениями на множествах узлов и локальных состояний пакетов; эти отношения задаются посредством квантифицированных булевых формул. В этих формулах мы будем использовать две параметризованные вспомогательные функции $U_\sigma(u, v)$ и $E_\sigma(u)$, где $\sigma \in \{0, 1, *\}$, а u, v — двоичные векторы:

- если $\sigma = *$, то $U_\sigma(u, v) = u \equiv v$ и $E_\sigma(u) = 1$,
- если $\sigma \in \{0, 1\}$, то $U_\sigma(u, v) = u \equiv \sigma$ и $E_\sigma(u) = u \equiv \sigma$.

Действие $a = OUTPUT(\mathbf{y})$ отправляет пакеты без изменения их заголовков во все выходные порты, имена которых подпадают под шаблон $\mathbf{y} = (\delta_1, \delta_2, \ldots, \delta_k)$. Действие $b = SET\_FIELD(\mathbf{z})$ изменяет заголовки пакетов в соответствии с шаблоном $\mathbf{z} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$: бит заголовка $\mathbf{h}[i]$ остается неизменным, если $\mathbf{z}[i] = *$; в противном случае $\mathbf{h}[i]$ изменяет свое значение на $\mathbf{z}[i]$. Семантика обоих действий описывается бинарными отношениями

$$\begin{aligned} R_a(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) &= \bigwedge_{i=1}^{N} (\mathbf{h}[i] \equiv \mathbf{h}'[i]) \ \wedge \ \bigwedge_{i=1}^{k} U_{\delta_i}(\mathbf{p}'[i], \mathbf{p}[i]) \\ R_b(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) &= \bigwedge_{i=1}^{N} U_{\sigma_i}(\mathbf{h}'[i], \mathbf{h}[i]) \ \wedge \ \bigwedge_{i=1}^{k} (\mathbf{p}[i] \equiv \mathbf{p}'[i]) \ . \end{aligned}$$

на множестве локальных состояний пакетов $\mathcal{H} \times \mathcal{P}$.

Инструкция α вычисляет последовательную композицию составляющих ее действий. Если α — пустая последовательность, то пакет сбрасывается, т.е. коммутируется в порт *drop*. Поэтому будем полагать по умолчанию, что всякая инструкция завершается действием коммутации пакета. Семантика инструкции α описывается бинарным отношением R_α, которое определяется так:

1. если инструкция α пустая, то $R_\alpha = \mathbf{false}$;

2. если $\alpha = a, \beta$, то отношение R_α задается следующими формулами в зависимости от действия a:

   (a) если a — это действие коммутации пакета, то

   $$R_\alpha(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \;=\; R_a(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \vee R_\beta(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle)\ ,$$

   (b) если a — это действие модификации заголовка пакета, то

   $$R_\alpha(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \;=\; \exists \mathbf{h}''\ (R_a(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}'', \mathbf{p}\rangle) \wedge R_\beta(\langle \mathbf{h}'', \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle))\ .$$

Правило коммутации пакетов $r = (\mathbf{z}, \mathbf{y}, \alpha)$ применяет инструкцию α ко всем тем пакетам, локальные состояния которых подпадают под соответствующие шаблоны заголовка $\mathbf{y}$ и порта $\mathbf{z}$. Семантика правила определяется следующим бинарным отношением R_r на множестве локальных состояний пакетов $\mathcal{H} \times \mathcal{P}$:

$$R_r(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \;=\; precond_r(\langle \mathbf{h}, \mathbf{p}\rangle) \;\wedge\; R_\alpha(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle)\ ,$$

где предикат

$$precond_r(\langle \mathbf{h}, \mathbf{p}\rangle) \;=\; \bigwedge_{i=1}^{\ell} E_{\delta_i}(\mathbf{p}[i]) \;\wedge\; \bigwedge_{j=1}^{N} E_{\sigma_j}(\mathbf{h}[j])$$

играет роль *предохранителя* правила r.

Таблица коммутации tab представляет собой пару (D, β), где $D = \{r_1, r_2, \ldots, r_n\}$ — множество правил коммутации, а β — инструкция умолчания. Коммутатор применяет правила таблицы ко всем пакетам, поступающим на его входные порты. Если ни одно из правил множества D нельзя применить к пакету, то этот пакет обрабатывается инструкцией β. Семантика таблицы коммутации tab задается следующим бинарным отношением:

$$\begin{aligned} R_{tab}(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \;=\; & \bigvee_{i=1}^{n} R_{r_i}(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \vee \\ & \vee\ (\neg(\bigwedge_{i=1}^{n} precond_{r_i}(\langle \mathbf{h}, \mathbf{p}\rangle)) \;\wedge\; R_\beta(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle))\ . \end{aligned}$$

на множестве локальных состояний пакетов $\mathcal{H} \times \mathcal{P}$. Множество всевозможных таблиц коммутации для заданной сети обозначим записью Tab.

Топология сети полностью определяется *отношением пересылки пакетов* $T \subseteq (\mathcal{OP} \times \mathcal{W}) \times (\mathcal{IP} \times \mathcal{W})$. Хотя в рамках нашей модели допустимы любые отношения указанного выше типа, для реальных сетей отношение T должно быть инъективной функцией. Узлы, вовлеченные в отношение T, называются *внутренними узлами* сети; все прочие узлы сети считаются *внешними узлами*. Будем использовать записи In и Out для обозначения множеств внешних входных и внешних выходных узлов сети. Предполагается, что внешние узлы сети подключены к внешним устройствам (контроллерам, серверам, сетевым шлюзам и др.), которые находятся вне сферы влияния контроллера ПКС. Пакеты поступают в сеть через внешние входные узлы и покидают сеть через внешние выходные узлы.

Для заданного множества коммутаторов $\mathcal{H}$ и топологии T *конфигурацией сети* называется всюду определенная функция $Net: \mathcal{W} \to Tab$, ассоциирующая с каждым коммутатором сети некоторую таблицу коммутации пакетов. Семантика сети с заданной конфигурацией Net определяется отношением

$$\begin{aligned} R_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}'\rangle) \;\; = \;\; & (C_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}'\rangle) \wedge Out(\mathbf{p}', \mathbf{w}')) \vee \\ & \vee \exists \mathbf{p}''(C_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle, \langle \mathbf{h}', \mathbf{p}'', \mathbf{w}\rangle) \wedge \\ & \qquad \wedge T(\langle \mathbf{p}'', \mathbf{w}\rangle, \langle \mathbf{p}', \mathbf{w}'\rangle)) \end{aligned}$$

на множестве состояний пакетов $S = \mathcal{H} \times \mathcal{P} \times \mathcal{W}$, где

$$C_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}'\rangle) \;=\; \Big(\bigvee_{\mathbf{w} \in \mathcal{W}} R_{Net(\mathbf{w})}(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \Big) \;\wedge\; \bigwedge_{j=1}^{k} (w_j \equiv w_j') \; .$$

Если отношение $R_{Net}(\mathbf{s}, \mathbf{s}')$ выполняется для пары состояний пакетов $s = \langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle$ и $s' = \langle \mathbf{h}', \mathbf{p}', \mathbf{w}'\rangle$, то пакет с заголовком $\mathbf{h}$, поступающий на порт $\mathbf{p}$ коммутатора $\mathbf{w}$, может быть за один скачок отправлен либо на входной порт $\mathbf{h}'$ коммутатора $\mathbf{w}'$, либо на устройство, подключенное к внешнему выходному порту $\mathbf{h}'$ коммутатора $\mathbf{w}$.

Контроллер — это реагирующая программа, которая получает сообщения от коммутаторов по каналам управления и вырабатывает отклики, которые изменяют содержимое таблиц коммутации. Отправляя сообщение контроллеру, коммутатор выражает требование модифицировать его таблицу коммутации: это сообщение означает, что коммутатор не имеет подходящего правила для обработки поступившего пакета. Само сообщение представляет собой локальное состояние данного пакета. В нашей модели контроллер может вырабатывать два типа команд для добавления и удаления правила. Команда $add(\mathbf{w}, r)$, где $\mathbf{w} \in \mathcal{W}$ и r — это правило коммутации, требует вставить правило r в таблицу коммутатора $\mathbf{w}$. Команда $del(\mathbf{w}, \mathbf{z}, \mathbf{y})$, где $\mathbf{w} \in \mathcal{W}$, и $\mathbf{z}, \mathbf{y}$ — шаблоны порта и пакета, требует удалить из таблицы коммутатора $\mathbf{w}$ все правила $\mathbf{r} = \langle (\mathbf{z}', \mathbf{y}'), \alpha\rangle$ в тех случаях, когда шаблоны этих правил $\mathbf{z}', \mathbf{y}'$ покрываются шаблонами $\mathbf{z}, \mathbf{y}$ соответственно. Обозначим буквой $\mathcal{C}$ множество всех возможных команд. Команды обоих типов изменяют конфигурацию сети; мы будем использовать запись $Net' = update(cmd, Net)$ для обозначения того, что команда cmd преобразовала конфигурацию Net в конфигурацию Net'. Если $\omega = cmd_1, cmd_2, \ldots, cmd_n$ — это конечная последовательность команд, то выражение $update(\omega, Net)$ будет считаться сокращенной записью композиции $update(cmd_n, update(\ldots, update(cmd_2, update(cmd_1, Net))))$.

Формальной моделью контроллера служит дискретный преобразователь $A = (\mathcal{H}, \mathcal{C}, Q, q_0, \Delta)$, в котором

- $\mathcal{H}$ и $\mathcal{C}$ — входной и выходной алфавиты соответственно,
- Q — множество внутренних состояний управления контроллера,
- $q_0,\ q_0 \subseteq Q$ — начальное состояние управления, и
- $\Delta,\ \Delta \subseteq Q \times \mathcal{H} \times C^* \times Q$ — отношение переходов.

Четверка $(q, \mathbf{s}, \omega, q')$ из Δ означает, что контроллер A после получения сообщения $\mathbf{s}$ в состоянии управления q может выработать конечную последовательность команд ω и перейти в состояние управления q'.

Для каждой конфигурации сети Net контроллер A может получать только такие сообщения, которые были индуцированы пакетами, поступившими на внешние узлы сети. Эти сообщения включают модифицированный заголовок пакета и имя порта того коммутатора, который отправил пакет контроллеру. Для описания множества сообщений $Event(Net)$, допустимых в конфигурации Net, рассмотрим рефлексивно-транзитивное замыкание R^*_{Net} отношения одношаговой коммутации R_{Net}. Тогда

$$Event(Net) = \{\langle \mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0\rangle \ : \ \exists\, \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}' \quad (\langle \mathbf{y}, \mathbf{z}\rangle \in In \wedge \\ \wedge\, R^*_{Net}(\langle \mathbf{x}, \mathbf{y}, \mathbf{z}\rangle, \langle \mathbf{x}', \mathbf{y}_0, \mathbf{z}_0\rangle) \wedge \\ \wedge\, R_{Net}(\langle \mathbf{x}', \mathbf{y}_0, \mathbf{z}_0\rangle, \langle \mathbf{x}_0, octr, \mathbf{z}_0\rangle))\}.$$

Формальная модель ПКС определяется множествами $\mathcal{W}, \mathcal{P}, \mathcal{H}$ коммутаторов, портов и заголовков пакетов, отношением пересылки пакетов T и контроллером A. *Частичный прогон* ПКС $M = (\mathcal{W}, \mathcal{P}, \mathcal{H}, T, A)$ представляет собой последовательность (конечную или бесконечную) вида

$$run = (Net_0, q_0) \overset{\mathbf{s_1}}{\to} (Net_1, q_1) \overset{\mathbf{s_2}}{\to} \cdots \overset{\mathbf{s_i}}{\to} (Net_i, q_i) \overset{\mathbf{s_{i+1}}}{\to} (Net_{i+1}, q_{i+1}) \overset{\mathbf{s_{i+2}}}{\to} \cdots \ (*)$$

где для каждого $i,\ 0 \le i$,

1. Net_i — сетевая конфигурация, q_i — состояние управления контроллера A, и s_i — состояние пакета,
2. $\mathbf{s}_{i+1} \in Event(Net_i)$,
3. отношение переходов контроллера A содержит такую четверку $(q_i, \mathbf{s_{i+1}}, \omega_i, q_{i+1})$, что $Net_{i+1} = update(\omega_i, Net_i)$.

Пары (Net_i, q_i) мыслятся как состояния ПКС, а состояния пакетов $\mathbf{s}_{i+1}$ играют роль сообщений, отправленных контроллеру. *Полный прогон* — это частичный прогон, который либо является бесконечным, либо завершается в таком состоянии ПКС (Net_i, q_i), что $Event(Net_i) = \varnothing$. Для заданной ПКС M и конфигурации сети Net_0 запись $Run(M, Net_0)$ будет обозначать множество всех полных прогонов M, начинающихся парой (Net_0, q_0).

3. Спецификация политик коммутации пакетов

Обычно на коммуникационную систему налагаются разнообразные требования, касающиеся правильности, надежности и безопасности соединений. Ограничимся рассмотрением лишь тех из них, которые предъявляются к отношению коммутации пакетов. Некоторые пакеты должны обязательно попасть в заданный узел, а какие-то пакеты должны быть обязательно сброшены. Для некоторых пакетов доступ к определенным коммутаторам запрещен, но через иные коммутаторы маршруты должны пролагаться в обязательном порядке. Циклы в маршрутах недопустимы. Эти и подобные им требования составляют *политику коммутации*. Одна из задач сетевого администрирования состоит в том, чтобы обеспечить такое наполнение таблиц коммутации правилами, чтобы соблюсти политику коммутации. Коль скоро в ПКС таблицы коммутации модифицируются контроллером, возникают следующие две основные проблемы сетевого программирования:

1. *Верификация ПКС относительно политик коммутации*: для заданной ПКС M и множества начальных сетевых конфигураций $\mathcal{N}$ проверить, что для всякой конфигурации $Net,\ Net \in \mathcal{N}$ любой прогон из множества $Run(M, Net)$ удовлетворяет заданной политике коммутации;

2. *Реализация политик коммутации*: для заданной политики коммутации и множества начальных сетевых конфигураций $\mathcal{N}$ построить такой контроллер A, чтобы для всякой конфигурации $Net,\ Net \in \mathcal{N}$ любой прогон из множества $Run(M, Net)$ удовлетворял заданной политике коммутации.

Для разработки формальных методов решения указанных задач необходим формальный язык спецификации политик коммутации.

В этом разделе мы приведем описание пробного варианта языка спецификаций политик коммутации для ПКС. Так как поведение сети изменяется со временем и все промежуточные конфигурации значимы для политик коммутации, имеет смысл воспользоваться темпоральными логиками для описания поведения ПКС. К тому же политики коммутации налагают ограничения на сетевые конфигурации. Эти ограничения касаются маршрутов, которые прокладываются в сетях правилами коммутации пакетов. Устройство этих маршрутов можно выразить посредством отношения одношаговой коммутации R_{Net}. Для этой цели мы выбрали логику FO[TC] — логику предикатов первого порядка с оператором транзитивного замыкания. Выбор был обусловлен тем, что

1. это расширение языка логики предикатов является более выразительным формализмом, нежели темпоральные логики или регулярные выражения, ранее использованные для этих целей в статьях [3, 4, 5, 6, 7];

2. в отличие от темпоральных логик наш язык позволяет оперировать отношением коммутации в явном виде, и это облегчает понимание смысла спецификаций;

3. задача верификации формул используемого нами языка на конечных моделях имеет эффективные алгоритмы решения.

Однако свойства маршрутов выражаются в терминах отношений между состояниями пакетов. Поэтому нам нужен простой язык для формулировки таких отношений. Поскольку состояние пакета — это двоичный вектор, нам достаточно взять для этих целей булевы формулы. Так мы пришли к концепции трехуровневого языка спецификаций политик маршрутизации, который далее опишем более подробно.

Рассмотрим множество переменных $Var = \{X_1, X_2, \dots\}$, принимающих значения из множества состояний пакетов $\mathcal{S} = \mathcal{H} \times \mathcal{P} \times \mathcal{W} = \{0,1\}^{N+k+m}$. *Спецификацией состояния пакета* назовем всякую булеву формулу φ над множеством булевых переменных $\{X_i[j] \ :\ X_i \in Var,\ 1 \le j \le N+k+m\}$ и связок $\neg,\ \wedge$. Множество всех таких формул обозначим записью $\mathcal{L}_0$.

В языке спецификации сетевых конфигураций $\mathcal{L}_1$ используются двухместный предикатный символ R и одноместные предикатные символы $I,\ O$. Это наименьший язык, удовлетворяющий следующим требованиям:

1. если $\varphi \in \mathcal{L}_0$, то $\varphi \in \mathcal{L}_1$;

2. если $X, Y \in Var$, то атомы $R(X,Y),\ I(X),\ O(Y)$ принадлежат $\mathcal{L}_1$;

3. если $\psi(X,Y)$ — формула из $\mathcal{L}_1$, содержащая не более двух свободных переменных, то $TC(\varphi(X,Y)) \in \mathcal{L}_1$;

4. если ψ_1 and ψ_2 — формулы из $\mathcal{L}_1$, и $X \in Var$, то формулы $(\neg\psi_1)$, $(\psi_1 \wedge \psi_2)$, $(\exists X\ \psi_1)$ принадлежат $\mathcal{L}_1$.

Семантика языка $\mathcal{L}_1$ определяется так. Пусть Net — сетевая конфигурация, и $\mathbf{s} = \langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle$ и $\mathbf{s}' = \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle$ — пара состояний пакетов. Тогда

1. $Net \models R(X,Y)[\mathbf{s}, \mathbf{s}']$ тогда и только тогда, когда $(\mathbf{s}, \mathbf{s}') \in R_{Net}$;

2. $Net \models I(X)[\mathbf{s}]$ тогда и только тогда, когда $\langle \mathbf{p}, \mathbf{w} \rangle \in In$;

3. $Net \models O(X)[\mathbf{s}]$ тогда и только тогда, когда $\langle \mathbf{p}, \mathbf{w} \rangle \in Out$;

Отношение выполнимости для прочих формул языка $\mathcal{L}_1$ определяется очевидным образом.

Здесь следует отметить ряд известных фактов о сложности задачи проверки выполнимости формул логики FO[TC]. Как следует из результатов работ [11], задача верификации моделей программ относительно формул FO[TC] является NLOG-полной. Кроме того, как было установлено в статьях [12, 13], как μ-исчисление, так и пропозициональная динамическая логика транслируются в FO[TC] (хотя при этом размер формулы может возрастать экспоненциально). Нами была установлена

Теорема. Задача верификации модели сети $Net \models \psi$ относительно замкнутых формул языка $\mathcal{L}_1$ является PSPACE-полной.

Доказательство. Принадлежность указанной задачи классу сложности PSPACE следует из упомянутого выше результата работы [11]. Обоснование PSPACE-трудности этой задачи основывается на том, что заголовок пакета может мыслиться как ограниченная лента машины Тьюринга. Команды машины Тьюринга, оперирующей на ограниченной ленте, моделируются правилами коммутации пакетов. Для моделирования самой машины Тьюринга достаточно одного коммутатора с двумя портами, соединенными друг с другом каналом связи. Тогда вопрос о завершаемости вычисления машины Тьюринга с ограниченной лентой, являющийся PSPACE-полной задачей (см. [14]), оказывается равносильным вопросу о сбрасываемости пакета, поступившего на входной порт такого коммутатора.

Политика коммутации, описывающая желаемое поведение ПКС, может быть формально задана формулами пропозициональной динамической логики, в которой замкнутые формулы языка $\mathcal{L}_1$ выступают в роли атомарных высказываний. Пусть $\mathcal{L}_0(X)$ и $\mathcal{L}_1(X)$ — множества всех тех формул языков $\mathcal{L}_0$ и $\mathcal{L}_1$ соответственно, в которых единственная свободная переменная X. Тогда $LTL(\mathcal{L}_1)$ — это наименьший язык, удовлетворяющий следующим требованиям:

1. если $\psi(X) \in \mathcal{L}_1(X)$, то $\psi(X) \in LTL(\mathcal{L}_1)$;

2. если $\Phi(X),\ \Psi(X) \in LTL(\mathcal{L}_1)$, то формулы $(\neg\Phi(X))$, $(\Phi(X) \wedge \Psi(X))$, $(\mathbf{X}\ \Phi(X))$ и $(\Phi(X)\ \mathbf{U}\ \Psi(X))$ принадлежат $LTL(\mathcal{L}_1)$.

Оценка истинности формул из $LTL(\mathcal{L}_1)$ проводится на бесконечных последовательностях сетевых конфигураций $\{Net_i\}_{i=1}^{\infty}$ для заданного состояния пакета $\mathbf{s}$ следующим образом:

- если $\psi(X) \in \mathcal{L}_1(X)$, то $\{Net_i\}_{i=1}^{\infty} \models \psi(\mathbf{s})$ тогда и только тогда, когда $Net_1 \models \psi(\mathbf{s})$,
- $\{Net_i\}_{i=1}^{\infty} \models \mathbf{X}\ \Phi(\mathbf{s})$ тогда и только тогда, когда $\{Net_i\}_{i=2}^{\infty} \models \Phi(\mathbf{s})$,
- $\{Net_i\}_{i=1}^{\infty} \models \Phi(\mathbf{s})\mathbf{U}\Psi(\mathbf{s})$ тогда и только тогда, когда $\{Net_i\}_{i=k}^{\infty} \models \Psi(\mathbf{s})$ for some $k,\ 1 \leq k$, and $\{Net_i\}_{i=j}^{\infty} \models \Psi(\mathbf{s})$ для каждого $j,\ 1 \leq j < k$,
- семантика связок $\neg$ и $\wedge$ определяется обычным образом.

Язык спецификации политик коммутации $\mathcal{L}_2$ состоит из множества выражений вида $\varphi(X) \Rightarrow \Phi(X)$, где $\varphi(X) \in \mathcal{L}_0(X)$, а $\Phi(X)$ — темпоральная формула языка $LTL(\mathcal{L}_1)$. Семантика этих выражений определяется посредством отношения выполнимости на прогонах формальных моделей ПКС. Для заданного прогона $(*)$ модели ПКС M и выражения $\varphi(X) \Rightarrow \Phi(X)$ языка $\mathcal{L}_2$ отношение $run \models \varphi(X) \Rightarrow \Phi(X)$ имеет место тогда и только тогда, когда $Net_{i\,i=n}^{\infty} \models \Phi(\mathbf{s_n})$ для каждого $n,\ 1 \leq n,$, удовлетворяющего условию $\varphi(\mathbf{s_n}) = 1$.

Политика коммутации FP описывается ограничением ψ на множество допустимых начальных конфигураций сети, представляющим собой замкнутую формулу языка $\mathcal{L}_1$, и конечным множеством $\{\varphi_1(X) \Rightarrow \Phi_1(X), \ldots, \varphi_n(X) \Rightarrow \Phi_n(X)\}$ выражений языка $\mathcal{L}_2$. Считается, что ПКС M *реализует политику коммутации* FP в том случае, когда для любой сетевой конфигурации Net_0, удовлетворяющей условию $Net_0 \models \psi$, каждый прогон $(*)$ из множества $Run(M, Net_0)$ удовлетворяет соотношению $\varphi_i(X) \Rightarrow \Phi_i(X)$, $1 \leq i \leq n$. Таким образом, задача верификации моделей ПКС состоит в том, чтобы проверить для заданной формальной модели ПКС M, удовлетворяет ли она спецификации политики коммутации FP.

4. Заключение

Можно заметить, что в том случае, когда контроллер ПКС представляет собой конечный автомат, сформулированная задача верификации моделей ПКС разрешима. Это объясняется тем, что конечная сеть может иметь лишь конечное число конфигураций, и поэтому все прогоны конечной модели ПКС можно представить в конечной размеченной системе переходов. Таким образом, указанная задача сводится к хорошо известной проблеме верификации конечных моделей программ для темпоральной логики PLTL. Однако использовать на практике такой подход затруднительно, поскольку размер пространства состояний этой системы переходов оценивается двойной экспонентой от размера описания модели ПКС. К моменту написания этой статьи авторы не смогли найти более эффективный способ преодоления этого препятствия. Тем не менее, нам удалось построить программно-инструментальное средство верификации сетевых конфигураций относительно их спецификаций на языке $\mathcal{L}_1$. При помощи этого средства можно осуществлять в оперативном режиме верификацию формальных спецификаций политик коммутации, состоящих из инвариантов вида $true \Rightarrow \mathbf{G}\ \psi$.

Список литературы

1. OpenFlow Switch Specification. Version 1.4.0, August 5, 2013. www.opennetworking.org.

2. Kim H., Feamster N. Improving network management with software defined networking // Communications Magazine. IEEE, 2013. P. 114—119.

3. Al-Shaer E., Marrero W., El-Atawy A., El Badawi K. Network Configuration in a Box: Toward End-to-End Verification of Network Reachability and Security // 17th IEEE International Conference on Network Protocols (ICNP'09). Princeton, New Jersey, USA, 2009. P. 123—132.

4. Mai H., Khurshid A., Agarwal R., Caesar M., Godfrey R.B., King S.T. Debugging of the Data Plane with Anteater // Proceedings of the ACM SIGCOMM conference. 2011. P. 290—301.

5. Kazemian P., Varghese G., McKeown N. Header space analysis: Static checking for networks // Proceedings of 9-th USENIX Symposium on Networked Systems Design and Implementation. 2012.

6. Khurshid A., Zhou W., Caesar M., Godfrey P.B. VeriFlow: Verifying Network-Wide Invariants in Real Time // Proceedings of International Conference "Hot Topics in Software Defined Networking"(HotSDN). 2012. P. 49—54.

7. Gutz S., Story A., Schlesinger C., Foster N. Splendid isolation: A Slice Abstraction for Software Defined Networks // Proceedings of International Conference "Hot Topics in Software Defined Networking"(HotSDN). 2012. P. 79—84.

8. Reitblatt M., Foster N., Rexford J.,Walker D. Consistent updates for software-defined networks: change you can believe in! // HotNets. 2011. V. 7.

9. Reitblatt M., Foster N., Rexford J., Schlesinger C., Walker D. Abstractions for Network Update // Proceedings of ACM SIGCOMM conference. 2012. P. 323—334.

10. Canini M., Venzano D., Peresini P., Kostic D., Rexford J. A NICE way to Test OpenFlow Applications // Proceedings of Networked Systems Design and Implementation, April 2012.

11. Immerman N. Languages that capture complexity classes // SIAM Journal of Computing. 1987. V. 16, No 4. P. 760—778.

12. Immerman N., Vardi M. Model checking and transitive closure logic // Lecture Notes in Computer Science. 1997. P. 291—302.

13. Alechina N., Immerman N. Reachability logic: efficient fragment of transitive closure logic // Logic Journal of IGPL. 2000. V. 8, No 3. P. 325—337.

14. Galil Z., Hierarchies of Complete Problems // Acta Informatica. 1976. No 6. P. 77—88.

A Formal Model and Verification Problems for Software Defined Networks

Zakharov V.A., Smelyansky R.L., Chemeritsky E.V.

Lomonosov Moscow State University
Leninskiye Gory, 1-52, Moscow, GSP-1, 119991, Russia
Applied Research Center for Computer Networks

Keywords: software defined network, switch, controller, forwarding rule, packet, formal model, specification, model checking

Software-defined networking (SDN) is an approach to building computer networks that separate and abstract data planes and control planes of these systems. In a SDN a centralized controller manages a distributed set of switches. A set of open commands for packet forwarding and flow-table updating was defined in the form of a protocol known as OpenFlow. In this paper we describe an abstract formal model of SDN, introduce a tentative language for specification of SDN forwarding policies, and set up formally model-checking problems for SDN.

Сведения об авторах:
Захаров Владимир Анатольевич,
Московский государственный университет им. М.В. Ломоносова,
доцент;
Смелянский Руслан Леонидович,
Московский государственный университет им. М.В. Ломоносова,
профессор;
Чемерицкий Евгений Викторович,
Московский государственный университет им. М.В. Ломоносова,
аспирант.