УДК 519.681 DOI 10.5862/JCTCS.212.7

В.А. Захаров, В.С. Алтухов, В.В. Подымов, Е.В. Чемерицкий

VERMONT – СРЕДСТВО ВЕРИФИКАЦИИ ПРОГРАММНО-КОНФИГУРИРУЕМЫХ СЕТЕЙ

V.A. Zakharov, V.S. Altukhov, V.V. Podymov, E.V. Chemeritskiy

VERMONT – A TOOLSET FOR VERIFICATION OF SOFTWARE DEFINED NETWORKS

Представлено программно-инструментальное средство VERMONT (VERifying MONiTor) для верификации в оперативном режиме программно-конфигурируемых сетей (ПКС) относительно формально специфицированных политик маршрутизации пакетов (ПМП). VERMONT может быть установлен в сети между контроллером и коммутаторами для наблюдения за сетью путем перехвата сообщений и команд, которыми обмениваются контроллер и коммутаторы, построения модели сети и проверки того, в какой мере изменения, происходящие в сети в результате выполнения команд реконфигурирования, подключения и отключения каналов связи и коммутационных устройств, согласуются с заданными требованиями ПМП. Перед тем как отправить команду реконфигурирования таблиц коммутации, VERMONT моделирует результат ее выполнения и проверяет выполнимость требований ПМП для модифицированной модели ПКС. Если VERMONT обнаруживает нарушение какого-либо требования ПМП, он блокирует пересылку команды и оповещает об этом системного администратора.

Описана математическая модель ПКС и формальный язык спецификации ПМП, используемые в нашей системе верификации. Рассказано об устройстве и алгоритмических принципах функционирования системы VERMONT. Представлены результаты экспериментов по применению разработанной системы для верификации некоторых ПКС, а также проведен сравнительный анализ системы VERMONT и других систем верификации ПКС.

ВЕРИФИКАЦИЯ В ОПЕРАТИВНОМ РЕЖИМЕ; ФОРМАЛЬНАЯ СПЕЦИФИКАЦИЯ; ВЕРИФИКАЦИЯ МОДЕЛЕЙ ПРОГРАММ; ПРОГРАММНО-КОНФИГУРИРУЕМЫЕ СЕТИ; КОНТРОЛЛЕР; КОММУТАТОР; ПОЛИТИКА МАРШРУТИЗАЦИИ ПАКЕТОВ; РЕКОНФИГУРИРОВАНИЕ СЕТЕЙ.

In this paper we present the software toolset VERMONT (VERifying MONiTor) for runtime checking the consistency of Software Defined Network (SDN) configurations with formally specified invariants of Packet Forwarding Policies (PFPs). VERMONT can be installed in line with the control plane to observe state changes of a network by intercepting the exchange of messages and commands between network switches and SDN controller, to build an adequate formal model of the whole network, and to check every event, such as installation, deletion, or modification of rules, port and switch up and down events, against a set of PFP invariants. Before retransmitting a network updating command to the switch, VERMONT simulates the result of its execution and checks PFP requirements. If a violation of some PFP invariant is detected, VERMONT blocks the change, alerts a network administrator, and gives some additional information to elucidate a possible source of the error. We define a SDN mathematical model used in our toolset, discuss some algorithmic and engineering issues of our toolset. After introducing a formal model of SDN and a formal language for PFP specification, we outline the main algorithms used in VERMONT for SDN model building, model checking, and model modification, and describe the structure of VERMONT and the functionality of its components. Finally, we demonstrate the results of our experiments on the application of VERMONT to a real-life network.

RUNTIME VERIFICATION; FORMAL SPECIFICATION; MODEL CHECKING; SOFTWARE DEFINED NETWORK; CONTROLLER; SWITCH; PACKET FORWARDING RELATION; NETWORK UPDATE.

Верификация в оперативном режиме (мониторинг, runtime verification, online verification) – это способ анализа и верификации информационных систем, при котором извлечение информации о поведении системы, проверка выполнимости заданных свойств поведения и необходимое вмешательство в работу системы в случае нарушения определенных требований ее поведения осуществляется непосредственно по ходу функционирования системы. Он занимает промежуточное положение между полной формальной верификацией программ и тестированием.

При верификации в оперативном режиме в отличие от верификации моделей программ или дедуктивной верификации программ анализируется лишь одна вычислительная трасса программы. За счет этого удается избежать трудностей, связанных с эффектом комбинаторного взрыва числа состояний, присущих полной формальной верификации программ. Достоверность результатов верификации в оперативном режиме существенно выше, поскольку анализу подвергается сама информационная система, а не ее модель. Такую верификацию можно проводить даже в том случае, когда код проверяемой программы недоступен. Наконец, верификация в оперативном режиме позволяет предотвращать недопустимые действия программы в процессе ее выполнения.

С другой стороны, применение строгих математических методов проверки выполнимости формальных спецификаций поведения программ дают более информативные и содержательные результаты, нежели сведения, полученные при простом тестировании программ.

В статье описано программно-инструментальное средство VERMONT (VERifying MONiTor), предназначенное для верификации в оперативном режиме программно-конфигурируемых сетей (ПКС) и предотвращения появления сетевых конфигураций, нарушающих заданные требования политики маршрутизации пакетов (ПМП).

Программно-конфигурируемые сети

ПКС – это новая архитектура телекоммуникационных сетей, предложенная недавно для преодоления принципиальных трудностей администрирования сетей традиционного вида [1]. Отличительная особенность ПКС состоит в том, что пространство потоков данных (data plane) и пространство потоков команд управления этими данными (control plane) физически разделены, и при этом несколько коммутирующих устройств могут находиться под контролем одной и той же управляющей программы. Наиболее развитым стандартом ПКС является протокол OpenFlow [2].

Коммутаторы ПКС снабжены *портами*, которые соединены друг с другом *каналами передачи данных*; по ним передаются *пакеты данных*. Пакеты, поступающие в порты коммутатора, обрабатываются *таблицей коммутации*, содержащей *правила коммутации пакетов*. Правило коммутации включает в себя *шаблон*, *инструкцию*, *приоритет*, *счетчик* и *срок активности*. Пакеты состоят из двух частей: *заголовка* и *нагрузки*. Для коммутации пакетов используются только их заголовки: коммутатор выбирает из таблицы то правило коммутации, шаблон которого подходит к заголовку поступившего в его порт пакета. Если подходят несколько правил, то выбирается правило с наибольшим приоритетом. Если в таблице коммутации нет подходящих правил, то применяется специальное правило для пропущенных пакетов.

Как только правило коммутации выбрано, его инструкция применяется к пакету. Инструкция – это последовательность *действий*, к числу которых относятся действия коммутации копии пакета в заданный порт коммутатора, сброс пакета, отправление пакета контроллеру, переписывание некоторых полей заголовка пакета. Счетчик служит для учета числа срабатываний правила. По истечении установленного срока активности правило удаляется из таблицы.

Коммутаторы ПКС находятся под управлением *контроллера*, который связан *каналом управления* с каждым коммутатором сети; по этому каналу коммутаторы отправляют *сообщения* контроллеру и получают от него *команды*. Сообщения могут содержать пакеты, отправленные контроллеру для бо-

лее глубокого анализа, статистические данные об использовании правил коммутации, оповещения о добавлении и удалении правил в таблицах коммутации. При помощи команд контроллер может изменять содержимое таблиц коммутации, запрашивать данные о состоянии сети и таблиц коммутации, отправлять заданный пакет из того или иного порта выбранного коммутатора. Более подробно с этими возможностями можно ознакомиться в описании стандарта протокола OpenFlow [2].

Главное преимущество ПКС состоит в том, что администратору предоставляется возможность управлять поведением всей сети за счет установления, изъятия или модификации правил коммутации пакетов в таблице любого коммутатора. Поэтому ПКС служит удобной парадигмой для разработки и совершенствования сетевых приложений (см. [3]). С ее возникновением получили развитие многочисленные проекты, направленные на создание языков и инструментальных средств сетевого программирования: Frenetic [4], Maestro [5], Procera [6], Nettle [7]. Как и для обычных программ, вопрос о корректности сетевых приложений является одной самых острых проблем. Поведение ПКС должно подчиняться политике маршрутизации пакетов – набору требований корректности и безопасности коммуникаций между компонентами сети. Сетевые приложения, работающие на ПКС-контроллере, должны обеспечивать только такую загрузку таблиц коммутации, которая удовлетворяет заданной ПМП.

Разработка универсального транслятора, преобразующего ПМП в программу управления ПКС-контроллером, представляется делом отдаленного будущего. Пока эволюция сетевого программирования следует проторенным путем развития системного программирования: создаются языки сетевого программирования высокого уровня, снабженные инструментальными и библиотечными средствами решения стандартных задач разработки, компиляции, отладки, моделирования, верификации и поддержания корректных и эффективных прикладных программ управления ПКС.

Чтобы гарантировать правильность работы программы, управляющей ПКС-контроллером, нужно иметь технологию и средства верификации ПКС относительно заданной ПМП: для заданных формальной спецификации ПМП Ф, формальной модели ПКС M и начальной сетевой конфигурации N проверить, что все вычисления модели M из конфигурации N удовлетворяют требованию Ф. Для решения этой задачи нужно создать систему верификации, способную проверять непротиворечивость требований ПМП, корректность отдельных приложений ПКС контроллера относительно заданных ПМП, корректность и безопасность работы всей ПКС.

Формальные методы применялись уже ранее для верификации традиционных телекоммуникационных сетей, но с появлением концепции ПКС интерес к этим методам возрос. После первой работы в этом направлении исследований [8] было предложено несколько методов решения проблемы верификации ПКС. Их можно разделить на два класса: верификацию потоков управления и верификацию потоков данных.

Верификация потоков управления занимается проверкой правильности ПКС-контроллера и прикладных программ управления контроллером. Для этой цели применяются методы статического и дедуктивного анализа, верификация моделей программ и пр. В статье [9] описана система верификации моделей программ NICE; она призвана обнаруживать ошибки путем проверки при помощи символьных методов всех трасс в модели прикладной программы управления ПКС, представленной размеченной системой переходов. Однако исследования показали, что этот подход пригоден только для сетей небольшого размера. Поэтому к системе NICE было добавлено средство организации и проведения тестирования [10]. В статье [11] для проверки контроллеров использован метод верификации моделей программ на основе абстрактных моделей состояний данных и состояний сети. Наиболее весомый вклад в этом направлении сделали авторы работы [12]. В ней представлена система верификации программ управления ПКС, способная

проверять их поведение на всевозможных топологиях и для всевозможных последовательностей событий. Допустимые топологии сети описываются формулами первого порядка, а сама процедура верификации базируется на классическом дедуктивном подходе Флойда–Дейкстры при поддержке системы автоматического доказательства теорем Z3.

Для верификации потоков данных в ПКС применялись разные подходы. Вначале внимание ограничивалось только средствами статического анализа. Отдельные конфигурации ПКС представлялись конечными размеченными системами переходов и для них обеспечивалась проверка в режиме off-line свойств достижимости узлов сети, отсутствия циклических маршрутов и др. при помощи различных алгоритмов верификации моделей программ: вычислений на основе BDD (FlowChecker [13]) и ДНФ (Hassel [14]), процедур решения проблемы выполнимости SAT (Anteater [15]). Затем были созданы динамические системы, позволяющие проводить верификацию ПКС в оперативном режиме: VeriFlow [16], NetPlumber [17], AP-Verifier [18]. Система VeriFlow предназначена для проверки сетевых инвариантов – отсутствия циклических маршрутов, потери пакетов и пр. – в режиме реального времени. В системе NetPlumber для проверки ПМП используется метод анализа заголовков пакетов. Авторы системы верификации AP-Verifier предлагают оригинальный метод описания совокупности всех шаблонов правил коммутации пакетов при помощи ограниченного множества атомарных формул, позволяющих существенно сократить размер данных и тем самым значительно ускорить проверку требований достижимости.

Системы верификации ПКС нового поколения – VeriFlow, NetPlumber, AP-Verifier – позволяют проводить проверку ПМП в оперативном режиме, т. е. они обладают возможностью модифицировать модели ПКС в зависимости от тех сообщений и команд, которыми обмениваются коммутаторы и контроллер. Модификация моделей должна выполняться очень быстро, поэтому модели ПКС в этих системах имеют явное представление в виде графов. При таком представлении моделей можно проводить эффективную проверку лишь простых требований ПМП. Кроме того, явное представление моделей ПКС допустимо лишь для сетей небольшого размера.

Почти во всех известных системах верификации ПКС для формального описания требований ПМП используются формулы темпоральных логик CTL и LTL. Исключение составляет система NetPlumber; ее формальный язык спецификаций ПМП основан на регулярных выражениях. Мы полагаем, что такие способы описания ПМП не вполне соответствуют задаче верификации потоков данных в ПКС, поскольку темпоральные логики и регулярные выражения предназначены для описания свойств вычислений, т. е. процессов, протекающих во времени, тогда как конфигурации ПКС являются статическими объектами, семантика которых определяется не в терминах вычислений, а посредством отношения коммутации пакетов.

Представленная в этой статье система верификации ПКС VERMONT также способна работать в оперативном режиме. Ее отличительные особенности таковы.

1. В отличие от систем верификации VeriFlow, NetPlumber и AP-Verifier, в ней для представления моделей ПКС и проверки выполнимости требований ПМП используется аппарат двоичных решающих диаграмм (BDDs). Благодаря символьному представлению моделей и организации символьных вычислений наша система значительно менее чувствительна к размеру анализируемой ПКС.

2. В отличие от системы верификации FlowChecker, наша система снабжена процедурами быстрой модификации моделей ПКС, поэтому может проводить верификацию в оперативном режиме.

3. В основу формального языка спецификаций системы VERMONT положен фрагмент логики второго порядка, поэтому наш язык обладает большими выразительными возможностями для описания требований ПМП.

Среди существующих средств верификации ПКС VERMONT занимает про-

межуточное положение. Наша система превосходит по скорости работы такие статические системы верификации ПКС, как FlowChecker, Anteater и Hassel, и лишь немногим уступает по эффективности таким динамическим системам верификации, как VeriFlow и NetPlumber. При этом выразительные возможности языка спецификаций ПМП, используемые в системе VERMONT, позволяют формально описывать и верифицировать все требования ПМП, которые могут быть проверены во всех других системах верификации ПКС.

Модель программно-конфигурируемой сети

Опишем формальную модель конфигураций ПКС, которая используется в системе верификации VERMONT (см. [19, 20]).

Заголовки пакетов, наименования портов и коммутаторов сети представляются в этой модели двоичными векторами. Длины этих векторов варьируются в зависимости от размера сети, типов коммутаторов, используемых сетевых протоколов и др. Для заголовков пакетов будем использовать обозначение $\boldsymbol{h}$; компоненты заголовков будем обозначать записью $\boldsymbol{h}[i]$, $1 \le I \le |\boldsymbol{h}|$; множество всех заголовков пакетов обозначим буквой H. Для имен портов коммутатора будем использовать обозначение $\boldsymbol{p}$, компоненты каждого такого вектора будем обозначать записью $\boldsymbol{p}[j]$, $1 \le j \le |\boldsymbol{p}|$.

Среди портов каждого коммутатора особо выделяются два порта: *drop* и *ctrl*. При попадании пакета в порт *drop* осуществляется его сброс. При попадании пакета в порт *ctrl* осуществляется его пересылка контроллеру ПКС. Множество всех имен остальных портов коммутатора обозначим буквой P. Для имен коммутаторов будем использовать обозначение $\boldsymbol{w}$; множество всех имен коммутаторов обозначим буквой W. Пары векторов $(\boldsymbol{p}, \boldsymbol{w})$ будем называть *точками сети,* пары векторов $(\boldsymbol{h}, \boldsymbol{p})$ – *локальными состояниями пакетов*, а тройки векторов $(\boldsymbol{h}, \boldsymbol{p}, \boldsymbol{w})$ – *глобальными состояниями пакетов*. Множество всех точек сети обозначим буквой V, множество локальных состояний пакетов – L, множество всех глобальных состояний пакетов – S.

Шаблоном заголовка пакета (порта) является троичный вектор соответствующей длины, состоящий из двоичных разрядов 0,1 и группового символа *. В модели ПКС шаблоны играют двоякую роль. Будем говорить, что двоичный вектор $\boldsymbol{x}$ *сочетается с шаблоном* $\boldsymbol{X}$ той же длины, если каждый двоичный разряд $\boldsymbol{X}[i]$ шаблона равен соответствующему разряду $\boldsymbol{x}[i]$ вектора $\boldsymbol{x}$. Двоичный вектор $\boldsymbol{x}$ *преобразуется шаблоном* $\boldsymbol{X}$ в двоичный вектор $\boldsymbol{y}$ той же длины, у которого каждый разряд $\boldsymbol{y}[i]$ либо равен $\boldsymbol{x}[i]$, если $\boldsymbol{X}[i] = *$, либо равен $\boldsymbol{X}[i]$, если $\boldsymbol{X}[i]$ – это двоичный разряд.

При помощи шаблонов определяются *действия* коммутаторов. В рассматриваемой модели ПКС допускаются действия двух видов. Действие коммутации пакетов OUTPUT($\boldsymbol{Y}$), где $\boldsymbol{Y}$ – шаблон порта, для каждого пакета с заголовком $\boldsymbol{h}$ направляет копию этого пакета с тем же заголовком в каждый порт $\boldsymbol{p}$, который сочетается с шаблоном $\boldsymbol{Y}$. Действие модификации заголовка пакета SET_FIELD($\boldsymbol{Z}$), где $\boldsymbol{Z}$ – шаблон заголовка пакета, для каждого пакета с заголовком $\boldsymbol{h}$ преобразует заголовок этого пакета шаблоном $\boldsymbol{Z}$.

Инструкцией является любая последовательность действий α, оканчивающаяся действием коммутации пакетов. Действия инструкции выполняются поочередно. Таким образом, каждая инструкция α определяет отношение R_α, $R_\alpha \subseteq H \times L$: для всякого пакета с заголовком $\boldsymbol{h}$ вычисляется множество локальных состояний пакетов $(\boldsymbol{h}', \boldsymbol{p})$, в которые действия этой инструкции преобразуют заголовок $\boldsymbol{h}$.

Правило коммутации задается четверкой $r = (\boldsymbol{Y}, \boldsymbol{Z}, \alpha, n)$, где $\boldsymbol{Y}$ и $\boldsymbol{Z}$ – шаблоны порта и заголовка, α – инструкция, n – приоритет правила. Правило r применимо к локальному состоянию пакетов $(\boldsymbol{h}, \boldsymbol{p})$, если векторы $\boldsymbol{h}$ и $\boldsymbol{p}$ сочетаются с шаблонами $\boldsymbol{Y}$ и $\boldsymbol{Z}$. Оно определяет отношение R_r, $R_r \subseteq L \times L$: для каждого локального состояния пакетов $(\boldsymbol{h}, \boldsymbol{p})$, к которому применимо правило, вычисляется множество локальных состояний $\{(\boldsymbol{h}', \boldsymbol{p}'): (\boldsymbol{h}, (\boldsymbol{h}', \boldsymbol{p}')) \in R_\alpha\}$.

Таблица коммутации $tab = (D, \beta)$ состоит из множества правил $D = \{r_1, r_2, \ldots, r_m\}$ и инструкции β для пропущенных пакетов. Если пакет с заголовком $\boldsymbol{h}$ поступает в порт

$\boldsymbol{p}$, то в из множества D выбирается правило $r = (\boldsymbol{Y}, \boldsymbol{Z}, \alpha, n)$ с наибольшим приоритетом, применимое к локальному состоянию пакетов $(\boldsymbol{h}, \boldsymbol{p})$, и инструкция α этого правила применяется к пакетам с заголовком $\boldsymbol{h}$. Если ни одно из правил множества D неприменимо к локальному состоянию $(\boldsymbol{h}, \boldsymbol{p})$, то к пакетам с заголовком $\boldsymbol{h}$ применяется инструкция для пропущенных правил β. Таким образом, таблица коммутации определяет отношение R_{tab}, $R_{tab} \subseteq L \times L$ на множестве локальных состояний пакетов. Множество всех таблиц коммутации обозначим записью *Tab*.

Топология сети определяется множеством T, $T \subseteq V \times V$, *каналов связи*, каждый из которых задается парой точек сети, соединенных этим каналом. Точка сети, соединенная каналом связи с какой-либо другой точкой, называется *внутренней точкой сети*. Остальные точки сети называются *внешними точками*. К внешним точкам сети подключены периферийные устройства (серверы, межсетевые шлюзы и др.); из этих точек пакеты поступают в сеть и покидают ее. Множество внешних точек сети обозначим записью *EXT*.

Пусть задано множество коммутаторов W и топологии сети T. Тогда *конфигурация сети* – это функция Net: $W \rightarrow Tab$, указывающая распределение таблиц коммутации по сетевым коммутаторам. Каждая конфигурация сети Net определяет отношение одношаговой маршрутизации пакетов R_{Net} : $R_{Net} \subseteq S \times S$ на множестве глобальных состояний пакетов. Два состояния пакетов $(\boldsymbol{h}, \boldsymbol{p}, \boldsymbol{w})$ и $(\boldsymbol{h}', \boldsymbol{p}', \boldsymbol{w}')$ находятся в отношении R_{Net} тогда и только тогда, когда соблюдено одно из следующих условий:

1) для некоторого порта $\boldsymbol{p}_0$ выполняются отношения $((\boldsymbol{h}, \boldsymbol{p}), (\boldsymbol{h}', \boldsymbol{p}_0)) \in R_{Net(\boldsymbol{w})}$ и $((\boldsymbol{w}, \boldsymbol{p}_0), (\boldsymbol{w}', \boldsymbol{p}')) \in T$, то есть пакет с заголовком $\boldsymbol{h}$, поступивший в порт $\boldsymbol{p}$ коммутатора $\boldsymbol{w}$, вначале коммутируется с измененным заголовком $\boldsymbol{h}'$ в порт $\boldsymbol{p}_0$ того же коммутатора, а затем пересылается по каналу связи в порт $\boldsymbol{p}'$ коммутатора $\boldsymbol{w}'$;

2) $\boldsymbol{w} = \boldsymbol{w}'$ и $\boldsymbol{p}' \in EXT$, при этом $((\boldsymbol{h}, \boldsymbol{p}), (\boldsymbol{h}', \boldsymbol{p}')) \in R_{Net(\boldsymbol{w})}$, то есть пакет с заголовком $\boldsymbol{h}$, поступивший в порт $\boldsymbol{p}$ коммутатора $\boldsymbol{w}$, коммутируется с измененным заголовком $\boldsymbol{h}'$ во внешний порт $\boldsymbol{p}'$ и выходит за пределы сети.

Реляционная модель конфигурации ПКС задается парой $M_{Net} = (R_{Net}, EXT)$. Для изменения сетевых конфигураций контроллеры используют команды реконфигурирования. Стандартом OpenFlow [2] предусмотрены следующие основные виды команд реконфигурирования:

1) $add(\boldsymbol{w}, r)$ добавляет в таблицу коммутатора $\boldsymbol{w}$ правило коммутации r;

2) $del(\boldsymbol{w}, \boldsymbol{Y}, \boldsymbol{Z}, n)$ удаляет из таблицы коммутатора $\boldsymbol{w}$ все правила коммутации приоритета n, шаблоны которых сочетаются с шаблонами $\boldsymbol{Y}$, $\boldsymbol{Z}$;

3) $\mathrm{mod}(\boldsymbol{w}, \boldsymbol{Y}, \boldsymbol{Z}, \alpha, n)$ изменяет в таблице коммутатора $\boldsymbol{w}$ во всех правилах коммутации приоритета n, шаблоны которых сочетаются с шаблонами $\boldsymbol{Y}$, $\boldsymbol{Z}$, имеющуюся у этих правил инструкцию на новую инструкцию α.

Команды реконфигурирования поступают коммутаторам сети от контроллера по каналу управления. Получив команду *com*, коммутатор исполняет ее и изменяет соответствующим образом содержание своей таблицы. В результате образуется новая конфигурация $com(Net)$.

Формальное описание всех перечисленных выше отношений, определяющих семантику ПКС, представлено в статье [19].

Язык спецификаций политик маршрутизации пакетов

По существу, ПМП – это описание требований, которым должны удовлетворять конфигурации ПКС, образующиеся при выполнении команд реконфигурации по ходу функционирования ПКС. Значительная часть этих требований касается свойств маршрутов, прокладываемых в сети правилами коммутации пакетов. Для формального описания требований такого рода можно воспользоваться фрагментом **L** языка логики предикатов первого порядка с оператором транзитивного замыкания $FO^2[TC]$. Как установлено в статье [20], эта логическая система охватывает многие формальные языки спецификаций, такие как CTL, LTL, PDL, μ-исчисление.

Для построения формул используемого фрагмента **L** логики $FO^2[TC]$ достаточно двух переменных X, Y, которые принимают в качестве значений двоичные векторы из множества S, и двух предикатов R и E, обозначающих отношение одношаговой маршрутизации пакетов и множество глобальных состояний пакетов во внешних точках сети. *Спецификацией состояния пакетов* называется всякая булева формула $\varphi(X, Y)$, которая строится из булевых переменных $X[j]$, $Y[j]$, $1 \le j \le N$, где N – длина двоичных векторов из множества S, при помощи связок $\neg$, $\wedge$ и $\vee$. Эти формулы используются для описания свойств состояний пакетов и отношений между парами состояний пакетов.

Формулы языка **L** строятся по следующим правилам:

1) всякая спецификация состояний пакетов $\varphi(X, Y)$ является формулой языка **L**;

2) выражения $R(X', X'')$, $E(X')$, где $X', X'' \in \{X, Y\}$, являются формулами языка **L**;

3) если $\Phi(X, Y)$ – формула языка **L**, то выражение $TC[\Phi(X, Y)]$ является формулой языка **L**;

4) если Φ_1 и Φ_2 – формулы языка **L**, то выражения $\neg\Phi_1$, $\Phi_1 \wedge \Phi_2$ и $\Phi_1 \vee \Phi_2$, а также выражения $\forall X\, \Phi_1$ и $\exists X\, \Phi_1$ также являются формулами языка **L**.

Спецификацией ПМП называется всякая замкнутая формула языка **L**.

Отношение выполнимости формул языка **L** для заданной модели ПКС $M_{Net}=(R_{Net}, EXT)$ на заданной паре глобальных состояний пакетов **s** и $\boldsymbol{s}'$ определяется следующим образом:

1) спецификация состояний пакетов $\varphi(X, Y)$ выполняется, если $\varphi(\boldsymbol{s}, \boldsymbol{s}') = \text{true}$;

2) формула $R(X, Y)$ выполняется, если $(\boldsymbol{s}, \boldsymbol{s}') \in R_{Net}$;

3) формула $E(X)$ выполняется, если $\boldsymbol{s} = (\boldsymbol{h}, \boldsymbol{p}, \boldsymbol{w})$ и при этом $(\boldsymbol{p}, \boldsymbol{w}) \in EXT$;

4) формула $TC[\Phi(X, Y)]$ выполняется, если существует такая конечная последовательность глобальных состояний пакетов $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$, которая удовлетворяет равенствам $\boldsymbol{s}_0 = \boldsymbol{s}$, $\boldsymbol{s}_m = \boldsymbol{s}'$ и при этом $\Phi(X, Y)$ выполняется для каждой пары состояний $\boldsymbol{s}_i, \boldsymbol{s}_{i+1}$, где $0 \le i \le m-1$;

5) выполнимость формул с булевыми связками и кванторами определяется так же, как и в классической логике.

Примером спецификации ПМП может служить формула, выражающая требование отсутствия циклических маршрутов движения пакетов в сети:

$$\neg\exists X(E(X) \wedge \exists Y(TC[R(X,Y)] \wedge \wedge TC[R(Y,Y)]).$$

Построение, проверка и модификация моделей ПКС

Верификация в оперативном режиме предназначена для проверки правильности поведения программы по ходу выполнения самой программы. В случае ПКС эту задачу можно сформулировать так: для заданной начальной модели $M_{Net} = (R_{Net}, EXT)$, последовательности команд реконфигурирования (вычисления ПКС) $com_1, \ldots, com_i, \ldots$ и списка формальных спецификаций ПМП $\Psi_1, \ldots, \Psi_k$ проверить, что для каждой модели ПКС M_i, полученной в результате применения конечной последовательности команд $com_1, \ldots, com_i$ к начальной конфигурации *Net*, выполняются все спецификации $\Psi_1, \ldots, \Psi_k$, то есть формулы этого списка являются инвариантами данного вычисления ПКС.

Чтобы проводить верификацию ПКС в оперативном режиме, нужно решить следующие три задачи:

1) построения модели ПКС: для заданной топологии сети T и заданной конфигурации ПКС *Net* построить формальную модель M_{Net};

2) верификации модели ПКС: проверить выполнимость заданной спецификации Ψ для заданной формальной модели ПКС M_{Net};

3) модификации модели ПКС: для заданной формальной модели ПКС M_{Net} и заданной команды реконфигурирования *com* построить формальную модель ПКС $M_{com(Net)}$.

Далее мы вкратце опишем наш подход к решению перечисленных задач.

Формальная модель ПКС M_{Net} полностью определяется отношением одношаговой маршрутизации R_{Net} на множестве глобальных состояний пакетов и конечным множеством (свойством) узлов сети *EXT*.

Поскольку и состояния пакетов, и узлы сети представляются двоичными векторами, можно воспользоваться двоичными разрешающими диаграммами (BDDs) для представления указанных отношений (см. [21]) и одним из инструментальных пакетов для построения и преобразования BDDs. В нашей системе для этой цели задействован пакет BuDDy, имеющий простое устройство и удобный интерфейс. С его помощью шаг за шагом можно построить BDDs, представляющие все те определенные выше отношения R_α, R_r, R_{tab}, которые необходимы для построения R_{Net}.

Что касается второй задачи, то ее также можно решить при помощи средств построения BDDs. Каждую формулу **Ф** языка спецификаций **L** можно представить в виде абстрактного синтаксического дерева T_Φ. Его листьями служат переменные X и Y, а внутренние вершины помечены базовыми предикатами R и E, а также булевыми связками, кванторами и оператором транзитивного замыкания TC. Для проверки выполнимости заданной формулы **Ф** в заданной модели ПКС M_{Net} достаточно вычислить истинностное значение подформул формулы **Ф** во всех вершинах дерева T_Φ на модели M_{Net}. Для вершин, помеченных базовыми предикатами, достаточно просто обратиться к BDDs, представляющим соответствующие отношения на двоичных векторах, проведя в случае необходимости переименование переменных. Если вершина дерева T_Φ помечена булевой операцией или квантором, то вызывается соответствующая процедура из пакета, которая по ранее вычисленным BDDs для вершин-операндов вычисляет BDD, представляющую отношение или истинностную оценку для той формулы, которая отвечает указанной вершине дерева. Особого внимания заслуживают вершины, помеченные оператором транзитивного замыкания TC.

Для построения BDDs, представляющей отношение $TC(R_0)$, на основании BDDs, представляющей бинарное отношение R_0, применяется итеративная процедура вычисления последовательности бинарных отношений R_i по следующей схеме:

$$R_{i+1}(X,Y) = \exists Z(R_i(X,Z) \wedge R_i(Z,Y)).$$

Вычисление проводится до тех пор, пока не будет достигнуто равенство $R_{i+1} = R_i$. Эта процедура является наиболее затратной по времени. Так как каждая спецификация представлена замкнутой формулой, в корне дерева T_Φ будет вычислена булева константа, дающая оценку выполнимости формулы **Ф** для заданной модели ПКС M_{Net}.

Эффективность решения задачи модификации моделей ПКС чрезвычайно важна для практического применения системы верификации, поскольку производительность процедуры модификации модели должна соответствовать тому темпу изменений конфигураций ПКС, который возможен при функционировании реальных сетей. Поэтому применение для этих целей процедуры построения моделей ПКС приводит к неоправданно большим задержкам. На самом деле изменение базовых отношений в модели ПКС можно провести сравнительно быстро. Ввиду того, что многие требования ПМП, наподобие отсутствия циклов, достижимости заданных точек сети и пр., опираются только на транзитивное замыкание отношения одношаговой маршрутизации R^+_{Net}, важно уметь быстро модифицировать именно это отношение. Для этой цели нами были предложены специальные приемы модификации отношения R^+_{Net}, не требующие итеративных вычислений. Подробное описание этих приемов представлено в статье [22].

Система верификации VERMONT: устройство и функциональность

Система VERMONT верификации ПКС в оперативном режиме состоит из четырех основных компонентов, как показано на рисунке:

прокси-сервера, осуществляющего перехват OpenFlow команд и сообщений, которыми обмениваются коммутаторы и контроллер ПКС;

верификатора, осуществляющего построение, верификацию и модификацию моделей ПКС;

загрузочного модуля, поставляющего данные для начального построения модели ПКС;

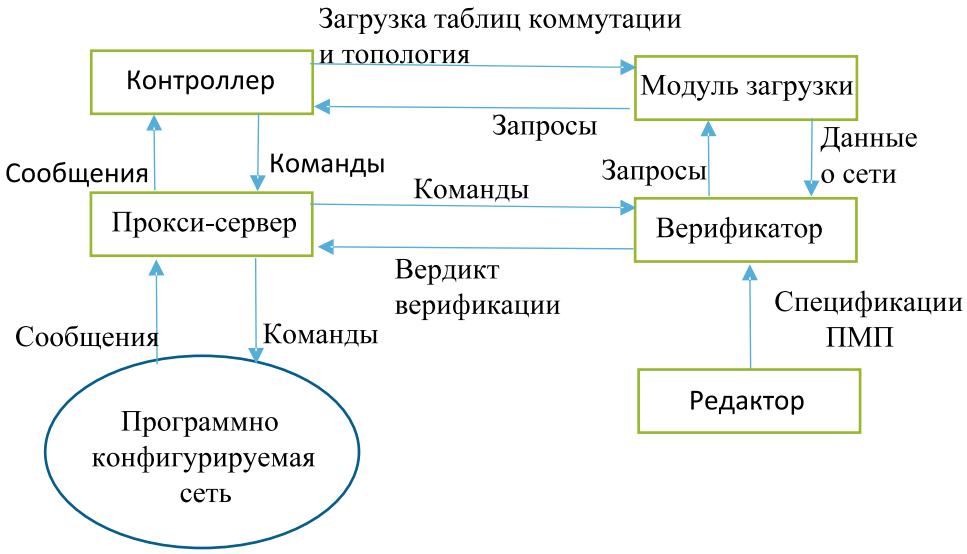

Схема устройства системы верификации ПКС VERMONT

редактора, позволяющего создавать ПМП спецификаций и транслировать их в абстрактные синтаксические деревья.

Одна из главных задач сетевого программирования – поддерживать такую загрузку таблиц коммутации пакетов, чтобы были выполнены требования ПМП. VERMONT позволяет до некоторой степени автоматизировать ее решение. Это инструментальное средство можно установить в канале передачи управления сети с тем, чтобы оно могло отслеживать те изменения конфигураций сети, которые обусловлены командами реконфигурирования или подтверждаются сообщениями от коммутаторов сети. На основании перехваченной информации VERMONT строит и уточняет адекватную модель сети, а затем проверяет, в какой мере те или иные изменения конфигураций ПКС соответствуют заданным требованиям ПМП. Перед тем как отправить команду реконфигурирования нужному коммутатору, VERMONT моделирует результат ее выполнения и проверяет, удовлетворяет ли полученная модель ПКС всем требованиям ПМП. Если ни одно из этих требований не нарушено, то команда отправляется по назначению. В противном случае ее пересылка блокируется и формируется соответствующее предупреждение для сетевого администратора. Основные функциональные возможности VERMONT таковы.

Прокси-сервер поддерживает связь с контроллером, коммутаторами сети и верификатором. Он перехватывает все команды и сообщения, которыми обмениваются контроллер и коммутаторы. Прокси-сервер находится под управлением пользователя VERMONT, который наделен правом включения и отключения модуля, выбора режима работы (SEAMLESS, MIRROR, INTERRUPT), настройки рабочих параметров. В зависимости от выбранного режима прокси-сервер может задерживать для проверки некоторые команды, отправлять данные об изменениях в сети верификатору и блокировать команды реконфигурирования.

Верификатор поддерживает связь с прокси-сервером, инициализатором и редактором. Этот модуль выполняет три процедуры:

инициализация модели: для заданного описания конфигурации ПКС вычисляется BDD-представление модели ПКС M_{Net};

верификация модели ПКС: для заданного списка спецификаций ПМП Ψ_1, ..., Ψ_k проверяется их выполнимость на заданной модели ПКС M_{Net};

модификация модели: для заданных BDD-представления модели ПКС M_{Net} и команды реконфигурирования *com* вычисляется BDD-представление модели ПКС $M_{com(Net)}$.

Верификатор получает от прокси-сервера данные об изменении сетевой конфигурации и возвращает ему результат верификации очередной конфигурации относительно имеющихся спецификаций ПМП. Верификатор также получает от загрузочного модуля данные о топологии сети и загрузке таблиц коммутации, а от редактора – спецификации ПМП.

Модуль загрузки по запросу от верификатора обращается к контроллеру, забирает у него данные об устройстве сети и пересылает их верификатору. Использование этого модуля делает нашу систему независимой от типа используемого контроллера.

При помощи редактора пользователь системы (сетевой администратор) может составлять списки спецификаций ПКС и вносить в них изменения.

Система верификации VERMONT имеет три режима работы.

1. В режиме SEAMLESS система верификации работает как обычный канал передачи управления между контроллером и коммутаторами сети: прокси-сервер не обращается к верификатору и пропускает все команды и сообщения без задержки.

2. В режиме MIRROR прокси-сервер отправляет верификатору данные об изменении конфигурации ПКС, но при этом пропускает все команды и сообщения без задержки. Верификатор предупреждает пользователя об обнаруженных нарушениях требований ПМП.

3. В режиме INTERRUPT проводится полная верификация ПКС: пересылка каждой команды, перехваченной прокси-сервером, задерживается до тех пор, пока не будет проверена допустимость ее выполнения в рамках требований ПМП.

Сравнительный анализ систем верификации ПКС

Для оценки производительности системы VERMONT были проведены некоторые эксперименты. В первой серии экспериментов VERMONT был применен к ПКС, имеющей топологию трехъярусного «толстого» дерева FT(H), состоящего из 27 коммутаторов: два коммутатора располагаются на верхнем уровне, пять пар коммутаторов на среднем уровне и пять пар коммутаторов на нижнем уровне, которые подключены к конечным хостам в количестве H от 50 до 100. Длина заголовка пакета равна 40. Каждое правило коммутации имеет некоторый срок активности, по истечении которого оно удаляется. Контроллер оповещается об этом событии и пытается восстановить конфигурацию при помощи команд добавления правил. VERMONT отслеживает изменения конфигураций и проверяет следующие два требования: $\mathbf{\Phi}_1$ – в сети отсутствуют топологические циклы; $\mathbf{\Phi}_2$ – в сети есть маршруты длины пять, но нет маршрутов длины шесть.

Во второй серии экспериментов проводилась проверка конфигураций сети Стэнфордского университета (SUN); она состоит из 16 коммутаторов, в каждом из которых есть три таблицы коммутации. Эта сеть используется в качестве общепринятого примера, на котором сравниваются разные средства анализа сетей. Длина заголовка пакета равна 128. VERMONT применялся для проверки требования $\mathbf{\Phi}_1$, а также двух требований: $\mathbf{\Phi}_3$ – в сети нет маршрутов длины более трех, и $\mathbf{\Phi}_4$ – в сети нет маршрутов длины более четырех.

Результаты проведенных экспериментов представлены в табл. 1.

Конфигурации SUN используются в работах [13–18] в качестве универсального тестового примера. Результаты сравнительного анализа всех средств верификации конфигураций ПКС представлены в табл. 2.

Как видно из этой таблицы, VERMONT имеет наиболее выразительный язык спецификаций ПМП и довольно быстро строит модель ПКС. Однако некоторые системы верификации (VeriFlow [16], AP-Verifier [18]) превосходят VERMONT по эффективности модификации модели. Такой быстротой решения задачи модификации модели ПКС эти средства верификации обязаны в первую очередь особым формам представления пространства состояний пакетов. Множество заголовков пакетов $\boldsymbol{H}$ разбивается на классы эквивалентности: заголовки $\boldsymbol{h'}$ и $\boldsymbol{h''}$ считаются эквивалентными на конфигурации *Net*, если каждый коммутатор применяет одинаковые правила коммутации пакетов

Таблица 1

Результаты экспериментальной проверки системы верификации ПКС VERMONT

	FT(60)	FT(80)	FT(100)	SUN (mod)
Количество правил коммутации пакетов	36900	49300	61500	15484
Построение модели	2756 ms	3689 ms	4574 ms	4714 ms
Проверка Φ_1	0,4 ms	0,4 ms	0,4 ms	51 ms
Проверка Φ_2	30 ms	36 ms	32 ms	–
Проверка Φ_3	–	–	–	222 ms
Проверка Φ_4	–	–	–	316 ms
Модификация модели, Команда add, max	9 ms	168 ms	172 ms	426 ms
Модификация модели, Команда add, average	6 ms	6 ms	6 ms	67 ms
Модификация модели, Команда del, max	178 ms	174 ms	176 ms	307 ms
Модификация модели, Команда del, avarege	8 ms	9 ms	10 ms	99 ms

Таблица 2

Сравнительный анализ системы верификации ПКС VERMONT

Средство верификации	Построение модели, ms	Модификация и верификация модели, ms	Язык спецификаций
VERMONT	4700	700	FO^2[TC]
NetPlumber	37000	1000	CTL
VeriFlow	4000	100	Фиксированные свойства
AP Verifier	1000	1	Фиксированные свойства
FlowChecker	120000	350–67 000	CTL
Anteater	400000	???	Фиксированные свойства

с этими заголовками при поступлении их в один и тот же порт. Этот метод позволяет конструировать явные теоретико-графовые представления отношений одношаговой маршрутизации пакетов R_{Net}, но его можно использовать только в том случае, когда правила коммутации не изменяют заголовки пакетов. Кроме того, такое «сжатое» представление отношения R_{Net} позволяет проводить проверку лишь ограниченного набора простых требований ПМП. Система верификации VERMONT, напротив, способна работать с произвольными конфигурациями и проверять требования ПМП значительно более широкого диапазона.

Работа выполнена при финансовой поддержке РФФИ, проекты № 13-07-00669 и 15-01-05742.

СПИСОК ЛИТЕРАТУРЫ

1. **McKeown N., Anderson T., Balakrishnan H., et al.** Openflow: Enabling Innovation in Campus Networks // SIGCOMM Computer Communication Review. 2008. Vol. 38. No. 2. Pp. 69–74.

2. OpenFlow Switch Specification. Version 1.4.0. [электронный ресурс] / URL: https://www.opennetworking.org (дата обращения 14.10.2013).

3. **Kim H., Feamster N.** Improving Network Management with Software Defined Networking // Communications Magazine. 2013. Pp. 114–119.

4. **Foster N., Harrison M., Freedman M.J., et al.** Frenetic: A Network Programming Language // Proc. of the 16th ACM SIGPLAN Internat. Conf. on Functional Programming. 2011. Pp. 279–291.

5. **Zheng Cai T., Cox A.L.** Maestro: A System for Scalable OpenFlow Control // Technical Report, TR10-08. Rice University, 2010.

6. **Voellmy A., Kim H., Feamster N.** Procera: A Language for High-Level Reactive Network Control // Proc. of the 1st Workshop on Hot Topics in Software Defined Networks. 2012. Pp. 43–48.

7. **Voellmy A., Hudak P.** Nettle – a Language for Configuring Routing Networks // Proc. of the IFIP TC 2 Working Conference on Domain-Specific Languages. 2009. Pp. 211–235.

8. **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 // Proc. of the 17th IEEE Internat. Conf. on Network Protocols. Princeton, New Jersey, USA, 2009.

9. **Canini M., Venzano D., et al.** A Nice Way to Test Openflow Applications // Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation. 2012.

10. **Kuzniar M., Perecini P., Canini M., et al.** A SOFT Way for OpenFlow Switch Interoperability Testing // Proc. of the 8th Internat. Conf. on Emerging Networking Experiments and Technologies. 2012. Pp. 265–276.

11. **Sethi D., Narayana S., Malik S.** Abstractions for Model Checking SDN Controllers. Formal Methods in Computer Aided Design. 2013.

12. **Ball T., Bjorner N., et al.** Defined Networks. VeriCon: Towards Verifying Controller Programs in Software Defined Networks // Proc. of the 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2014. Pp. 282–293.

13. **Al-Shaer E., Al-Haj S.** Flowchecker: Configuration Analysis and Verification of Federated Openflow Infrastructures // Proc. of the 3rd ACM Workshop on Assurable and Usable Security Configuration. 2010. Pp. 37–44.

14. **Kazemian P., Varghese G., McKeown N.** Header Space Analysis: Static Checking for Networks // Proc. of 9th USENIX Symp. on Networked Systems Design and Implementation. 2012.

15. **Mai H., Khurshid A., Agarwal R., et al.** Debugging of the Data Plane with Anteater // Proc. of the ACM SIGCOMM Conf. 2011, Pp. 290–301.

16. **Khurshid A., Zhou W., Caesar M., Godfrey P.B.** VeriFlow: Verifying Network-Wide Invariants in Real Time // Proc. of 1st Internat. Conf. Hot Topics in Software Defined Networking. 2012.

17. **Kazemian P., Chang M., Zeng H., et al.** Real Time Network Policy Checking Using Header Space Analysis // Proc. of USENIX Symp. on Networked Systems Design and Implementation. 2013.

18. **Yang H., Lam S.S.** Real-time Terification of Network Properties Using Atomic Predicates // Proc. of IEEE Internat. Conf. on Network Protocols. 2013.

19. **Захаров В.А., Смелянский Р.Л., Чемерицкий Е.В.** Формальная модель и задачи верификации программно-конфигурируемых сетей // Моделирование и анализ информационных систем. 2013. Т. 20. № 6. С. 33–48.

20. **Alechina N., Immerman N.** Reachability Logic: Efficient Fragment of Transitive Closure Logic // Logic Journal of IGPL. 2000. Vol. 8. No. 3. Pp. 325–337.

21. **Bryant R.E.** Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. 1986. Vol. C-35(8). Pp. 677–691.

22. **Altukhov V.S., Chemeritskiy E.V., Podymov V.V., Zakharov V.A.** A Runtime Verification System for Software Defined Networks // Proc. of the 2nd Internat. Conf. Tools & Methods of Program Analysis. Kostroma: KGTU, 2014. Pp. 19–28.

REFERENCES

1. **McKeown N., Anderson T., Balakrishnan H., et al.** Openflow: Enabling Innovation in Campus Networks, *SIGCOMM Computer Communication Review*, 2008, Vol. 38, No. 2, Pp. 69–74.

2. *OpenFlow Switch Specification. Version 1.4.0.* Available: https://www.opennetworking.org, (Accessed October 14, 2013).

3. **Kim H., Feamster N.** Improving Network Management with Software Defined Networking, *Communications Magazine*, 2013, Pp. 114–119.

4. **Foster N., Harrison M., Freedman M.J., et al.** Frenetic: A Network Programming Language, *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, 2011,

Pp. 279–291.

5. **Zheng Cai T., Cox A. L.** Maestro: A System for Scalable OpenFlow Control. *Technical Report, TR10-08*, Rice University, 2010.

6. **Voellmy A., Kim H., Feamster N.** Procera: A Language for High-Level Reactive Network Control, *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks*, 2012, Pp. 43–48.

7. **Voellmy A., Hudak P.** Nettle – a Language for Configuring Routing Networks, *Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*, 2009, Pp. 211–235.

8. **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, *Proceedings of the 17th IEEE International Conference on Network Protocols,* Princeton, New Jersey, USA, 2009.

9. **Canini M., Venzano D., et al.** A Nice Way to Test Openflow Applications, *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

10. **Kuzniar M., Perecini P., Canini M., et al.** A SOFT Way for OpenFlow Switch Interoperability Testing, *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012, Pp. 265–276.

11. **Sethi D., Narayana S., Malik S.** *Abstractions for Model Checking SDN Controllers. Formal Methods in Computer Aided Design*, 2013.

12. **Ball T., Bjorner N., et al.** Defined Networks. VeriCon: Towards Verifying Controller Programs in Software Defined Networks, *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, Pp. 282–293.

13. **Al-Shaer E., Al-Haj S.** Flowchecker: Configuration Analysis and Verification of Federated Openflow Infrastructures, *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*, 2010, Pp. 37–44.

14. **Kazemian P., Varghese G., McKeown N.** Header Space Analysis: Static Checking for Networks, *Proceedings of 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

15. **Mai H., Khurshid A., Agarwal R., et al.** Debugging of the Data Plane with Anteater, *Proceedings of the ACM SIGCOMM Conference*, 2011, Pp. 290–301.

16. **Khurshid A., Zhou W., Caesar M., Godfrey P.B.** VeriFlow: Verifying Network-Wide Invariants in Real Time, *Proceedings of 1st International Conference Hot Topics in Software Defined Networking*, 2012.

17. **Kazemian P., Chang M., Zeng H., et al.** Real time network policy checking using header space analysis, *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, 2013.

18. **Yang H., Lam S.S.** Real-time Verification of Network Properties Using Atomic Predicates, *Proceedings of IEEE International Conference on Network Protocols*, 2013.

19. **Zakharov V.A., Smelyanskiy R.L., Chemeritskiy Ye.V.** Formalnaya model i zadachi verifikatsii programmno-konfiguriruyemykh setey [Formal model and verification tasks in software-configurable network], *Modelirovaniye i analiz informatsionnykh sistem*, 2013, Vol. 20, No. 6. Pp. 33–48. (rus)

20. **Alechina N., Immerman N.** Reachability logic: efficient fragment of transitive closure logic, *Logic Journal of IGPL*, 2000, Vol. 8, No. 3, Pp. 325–337.

21. **Bryant R.E.** Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, 1986, Vol. C-35(8), Pp. 677–691.

22. **Altukhov V.S., Chemeritskiy E.V., Podymov V.V., Zakharov V.A.** A Runtime Verification System for Software Defined Networks, *Proceedings of the 2nd International Conference: Tools & Methods of Program Analysis,* Kostroma: KGTU, 2014. Pp. 19–28.

ЗАХАРОВ Владимир Анатольевич – *профессор кафедры математической кибернетики Московского государственного университета имени М.В. Ломоносова, доктор физико-математических наук.*
119991, Россия, Москва, ГСП-1, ул. Ленинские Горы, д. 1.
E-mail: zakh@cs.msu.su

ZAKHAROV, Vladimir A. *Lomonosov Moscow State University.*
119991, GSP-1, Leninskie Gory, Moscow, Russia.
E-mail: zakh@cs.msu.su

АЛТУХОВ Виктор Сергеевич – *студент лаборатории вычислительных комплексов Московского государственного университета имени М.В. Ломоносова.*
119991, Россия, Москва, ГСП-1, ул. Ленинские Горы, д. 1.
E-mail: victoralt@lvk.cs.msu.su

ALTUKHOV, Viktor S. *Lomonosov Moscow State University.*
119991, GSP-1, Leninskie Gory, Moscow, Russia.
E-mail: victoralt@lvk.cs.msu.su

ПОДЫМОВ Владислав Васильевич – *младший научный сотрудник кафедры математической кибернетики Московского государственного университета имени М.В. Ломоносова.*
119991, Россия, Москва, ГСП-1, ул. Ленинские Горы, д. 1.
E-mail: valdus@yandex.ru

PODYMOV, Vladislav V. *Lomonosov Moscow State University.*
119991, GSP-1, Leninskie Gory, Moscow, Russia.
E-mail: valdus@yandex.ru

ЧЕМЕРИЦКИЙ Евгений Викторович – *ведущий программист Центра прикладных исследований компьютерных сетей.*
E-mail: tyz@lvk.cs.msu.su

CHEMERITSKIY, Eugene V. *Applied Research Center for Computer Networks.*
E-mail: tyz@lvk.cs.msu.su