Тестирование правил настройки сетевого коммутатора программно конфигурируемой сети

И.Б. Бурдонов <igor@ispras.ru>[1]
Н.В. Евтушенко <yevtushenko@ispras.ru>
А.С. Косачев <kos@ispras.ru>[2]
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В настоящее время SDN-технология активно используется в виртуальных сетях для реализации различных служебных сервисных функций. В основе сети лежит связный неориентированных граф физических связей (англ. Resource Network Connectivity Topology, RNCT), вершинами которого являются сетевые коммутаторы (switches) и хосты (hosts). В настоящей работе рассматривается топология, когда каждый хост соединен ровно с одним коммутатором. Коммутаторы работают по таблицам правил, которые настраиваются централизованно с помощью контроллера, работающего независимо от сетевого оборудования. Настройка коммутаторов сети предназначена для обеспечения передачи пакетов из начальных хостов в конечные хосты в зависимости от значений параметров пакетов. В статье обсуждается связь настроек коммутаторов и множества реализуемых ими путей передачи пакетов в зависимости от свойств графа физических связей. Исследуется задача тестирования настройки коммутаторов. Под целью тестирования понимается подача пакетов, позволяющих «пройти» по каждому правилу каждого коммутатора хотя бы один раз, подав пакет с необходимыми параметрами. Показывается, что в общем случае не любая настройка любого коммутатора проверяема. Возможности тестирования зависят от принимаемых гипотез о работе коммутатора. В статье рассматриваются две гипотезы: гипотеза о коммутаторе предполагает, что работа коммутатора не зависит от настроек других коммутаторов; более сильная гипотеза о правиле, кроме этого, предполагает, что работы коммутатора по данному правилу не зависит от других правил в настройке этого коммутатора. После введения, в разд. 2 вводятся необходимые определения и обозначения. Раздел 3 посвящен взаимосвязи путей в сети и правил в коммутаторах. В разд. 4 рассматривается тестирование на основе гипотезы о правиле, доказываются необходимые и достаточные условия возможности проверки заданного правила заданного коммутатора. В разд. 5 рассматривается тестирование на основе гипотезы о коммутаторе и доказывается необходимое (но не достаточное) условие и достаточное (но не необходимое) условие проверяемости любой настройки коммутатора. В заключении обсуждаются проблемы, возникающие при установлении необходимого и достаточного условия проверяемости любой настройки коммутатора, а также ставится задача определения таких условий для заданной настройки коммутатора.

Ключевые слова: виртуальная сеть; SDN-технология; хосты и коммутаторы; настройка сетевых коммутаторов; передача пакетов; реберно-простые пути; тестирование сетевых коммутаторов.

DOI: 10.15514/ISPRAS-2018-30(6)-4

Для цитирования: Бурдонов И.Б., Евтушенко Н.В., Косачев А.С. Тестирование правил настройки сетевого коммутатора программно конфигурируемой сети. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 69-88. DOI: 10.15514/ISPRAS-2018-30(6)-4

[1] Работа частично поддержана проектом РФФИ № 17-07-00682 А.
[2] Работа частично поддержана проектом РФФИ № 16-07-01106 А.

1. Введение

В настоящее время большое внимание уделяется развитию виртуальных сетей, одной из технологий предля которых является SDN-технология (Software-Defined Networking), позволяющая эффективно выполнять реализацию различных служебных сервисных функций с использованием общих ресурсов и принципов управления ими. В SDN-технологии логическое централизованное управление сетевыми устройствами осуществляется через контроллер, работающий независимо от сетевого оборудования, который может рассматриваться как определенная операционная система [1]. Как результат, SDN обеспечивает гибкую управляемость и программируемость путем разделения уровней управления и данных (уровень пересылки) через введение открытого интерфейса между этими уровнями. Несмотря на большое количество публикаций по верификации SDN-сетей, по-прежнему актуальны исследования по верификации и тестированию сетевых устройств на различных уровнях, чтобы устранить ошибки в неправильных конфигурациях или программном обеспечении, которые могут привести к сбоям сети [2]. Кроме того, даже если каждый компонент хорошо отлажен в изоляции, их композиция может столкнуться с проблемами взаимодействия, и одним из решений в данном случае является разработка методов для тестирования системы в целом.

Существующие подходы к тестированию для программируемых инфраструктур можно разделить на три группы. Подходы первой группы направлены на создание «критических» ситуаций для сети или ее компонента. Вводимые данные являются «неожидаемыми» запросами; результаты тестирования оцениваются в зависимости от способности контроллера (или коммутатора) правильно обработать запрос или отклонить его [3, 4]. Вторая группа методов относится к тестированию конформности или соответствия, когда имеется формальная спецификация тестируемого компонента и набор тестов генерируется на основе этой спецификации [5, 6]. Такие методы на основе формальных моделей в настоящее время активно используются для проверки правильности, например, критических модулей платформ виртуализации, таких как, например, модуль размещения [7]. Подходы последней третьей группы представляют собой комбинацию формальных

методов верификации и тестирования, а именно, контрпримеры, полученные при верификации, используются в качестве тестов [8, 9].

В данной работе исследуется задача тестирования работы коммутатора по его настройке. Под целью тестирования понимается подача пакетов, позволяющих «пройти» по данному правилу настройки данного коммутатора хотя бы один раз, подав пакет с необходимыми параметрами. Это возможно не для всякой настройки коммутаторов. В частности, может возникать «зацикливание», когда пакет бесконечно двигается по циклу графа связей. Достаточно очевидно, что возможности тестирования зависят от принимаемой гипотезы о работе коммутаторов. В данной работе рассматриваются две такие гипотезы и, соответственно, два способа тестирования.

Сначала исследуется тестирование отдельного правила в настройке отдельного коммутатора. Для полноты тестирования принимается (сильная) гипотеза о правиле: каковы бы ни были таблицы настройки данного коммутатора, содержащие данное правило, программа коммутатора, обрабатывающая эти таблицы, при пересылке пакетов выполняет данное правило так, как если бы других правил в таблицах не было, т.е. независимо от других правил. Иными словами, предполагается, что в программе коммутатора не может быть ошибок, сводящихся к такой зависимости и означающих, что коммутатор неправильно срабатывает по данному правилу только при наличии некоторых других правил в таблицах коммутатора, в то время как при наличии/отсутствии других правил в своих таблицах работает правильно. Если гипотеза о правиле не верна, то тестирование отдельного правила может не выявить такого рода ошибки.

Поэтому необходимо более «тонкое» тестирование, не опирающееся на гипотезу о правиле. Гораздо более реалистичной представляется более слабая гипотеза о коммутаторе: работа данного коммутатора по его правилам не зависит от правил других коммутаторов. Соответственно, тестировать нужно не отдельное правило данного коммутатора, а всю его настройку целиком как заданную совокупность его правил. Однако не любая настройка любого коммутатора проверяема, т.е. не всегда возможно при фиксированной настройке данного коммутатора так настроить остальные коммутаторы, чтобы настройка сети оказалась правильной (пакеты пересылаются, откуда надо и куда надо без зацикливаний) и давала возможность проверить каждое правило фиксированной настройки данного коммутатора. В данной статье исследует вопрос о том, при каких условиях любая настройка данного коммутатора может быть проверяема. Эти условия, естественно, налагаются на граф физических связей сети, разбиение узлов сети на хосты и коммутаторы, выделение начальных и конечных хостов, а также место в сети данного коммутатора.

Структура статьи следующая. В разд. 2 вводятся необходимые определения и обозначения. Разд. 3 посвящен взаимосвязи путей в сети и правил в коммутаторах. В разд. 4 рассматривается тестирование отдельного правила настроки коммутатора, здесь доказывается ряд утверждений, в частности, необходимые и достаточные условия возможности проверки заданного правила

заданного коммутатора. В разд. 5 рассматривается тестирование всей сосокупности правил настройки коммутатора, доказывается необходимое (но не достаточное) и достаточное (но не необходимое) условия проверяемости любой настройки данного коммутатора. В заключении обсуждаются проблемы, возникающие при установлении необходимого и достаточного условия проверяемости любой настройки коммутатора, а также ставится задача определения таких условий для заданной настройки коммутатора.

2. Определения и обозначения

В работе рассматривается сеть, состоящая из коммутаторов, хостов и физических связей между ними, которая задается с помощью конечного неориентированного связного графа без кратных ребер и петель $G = (V, E)$ где $E \subseteq \{ \{a, b\} \mid a \in V \,\&\, b \in V \,\&\, a \neq b \}$, называемого далее *графом физических связей* (*Resource Network Connectivity Topology, RNCT*). Вершины в графе G разделяются на хосты и коммутаторы: $V = H \cup S$, $H \cap S = \varnothing$. Каждый хост соединен только с одним коммутатором: $\forall\, h \in H\ \boldsymbol{deg}(h) = 1\ \&\ \exists\, s \in S\ \{h, s\} \in E$, где $\boldsymbol{deg}(x)$ – степень вершины x. Предполагается, что каждое ребро может передавать пакеты в двух направлениях. Заметим, что если граф G не связен, то это означает только, что каждую компоненту связности можно рассматривать как самостоятельную сеть, которая тестируется отдельно. В рамках данной статьи будем считать, что G, H и S фиксированы и не будем указывать их в используемых ниже обозначениях функций, зависящих от них. Для вершины $a \in V$ обозначим её открытое множество соседей (*open neighborhood*) через $N(a) = \{ b \in V \mid \{a, b\} \in E \}$.

В настоящей работе мы будем исходить из следующего основного предположения: выходные порты, по которым коммутатор посылает принятый им пакет, зависят от многих параметров, и в частности, могут зависеть от порта коммутатора, по которому принимается пакет.

Мы предполагаем, что тестер может:

1) настраивать коммутаторы через контроллер,
2) посылать пакеты из хостов,
3) наблюдать передачу пакетов между любыми двумя соседними узлами сети.

Таким образом, примером тестируемой системы является система, обведенная пунктирной линией на рис. 1 [10]. Входными символами для системы являются множества путей, которые необходимо реализовать, а выходными символами – пути, реализованные на графе физических связей, которые наблюдаются при мониторинге.

Хосты и коммутаторы. Вершины в графе разделяются на хосты и коммутаторы. Хост может генерировать передаваемые пакеты и передавать их в единственный коммутатор, который связан с этим хостом [10]. Коммутаторы только передают принятые пакеты; более того, мы предполагаем, что

коммутатор не изменяет заголовок принятого пакета. Иными словами, заголовок пакета не изменяется при передаче по сети.

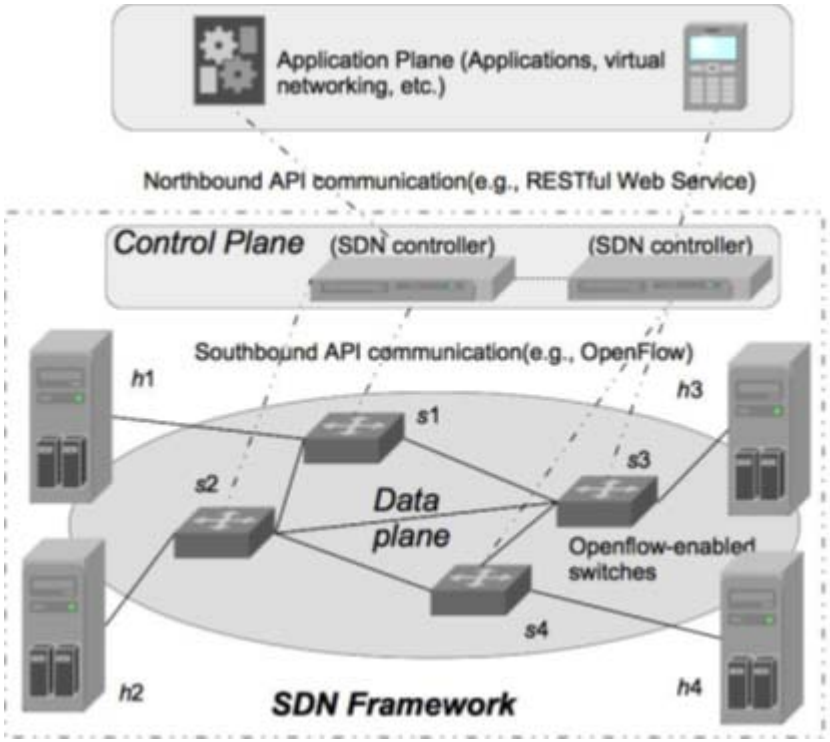

Рис.1. Пример SDN-структуры [10]
Fig. 1. SDN structure example [10]

Пути. Поскольку в графе G нет кратных ребер, путь p будет пониматься как последовательность соседних вершин графа G. Формально: путь – это последовательность вершин такая, что вершины, соседние в последовательности, являются соседними в графе (соединены ребром). Будем говорить, что путь $p = x_1, ..., x_n$ начинается в вершине x_1, заканчивается в вершине x_n, ведет от x_1 до x_n, имеет длину n-1, и проходит по дуге (a, b), если $a = x_i$ и $b = x_{i+1}$ для некоторого $i = 1 .. n$-1. Если путь проходит по дуге (a, b) или дуге (b, a), то будем говорить, что путь проходит по ребру $\{a, b\}$. Если путь $p = x_1, ..., x_n$ заканчивается в той вершине, в которой начинается путь $q = y_1, ..., y_k$, то есть $x_n = y_1$, то конкатенацией этих путей будем называть путь $p{\cdot}q = x_1, ..., x_n, y_2, ..., y_k$ для $k > 1$ или путь $p{\cdot}q = p$ для $k = 1$. Путь называется *реберно-простым*, если он проходит по каждой дуге (т.е. по каждому ребру в каждом направлении) не более одного раза: $(x_i, x_{i+1}) = (x_j, x_{j+1}) \Rightarrow i = j$. Путь

называется вершинно-простым, если все его вершины разные: $x_i = x_j \Rightarrow i = j$. Путь будем называть *правильным*, если вершина-хост является только началом и/или концом пути. Правильный путь будем называть *полным*, если он начинается и заканчивается в хосте.

На произвольном множестве P путей вводится отношение частичного порядка «является отрезком»: путь q является отрезком пути $p{\cdot}q{\cdot}r$. Обозначим через $P^{\#}$ множество отрезков длиной не меньше 2 путей из множества P: $P^{\#} = \{ q \mid \exists p, r \, (p{\cdot}q{\cdot}r \in P \,\&\, |q| > 2) \}$. Множество максимальных путей во множестве путей P обозначим $P^{max} = \{ q \mid \forall p, r \, (p{\cdot}q{\cdot}r \in P \Rightarrow |p| = |q| = 1) \}$. Если множество P состоит из путей длиной не менее 2, то $P^{\# max} = P^{max}$.

Вектор значений параметров. По каждому пути передаются пакеты с заголовками, согласно вектору значений параметров [11]; поэтому можно считать, что каждому пути поставлен в соответствие некоторый вектор значений параметров. Для заданного вектора значений параметров определены два множества $H_0 \subseteq H$ и $H_1 \subseteq H$ *начальных* и *конечных хостов*, из которых можно передавать и в которые должен прийти пакет, имеющий заголовок с данным вектором значений параметров: множество H_0 содержит все хосты, в которых могут генерироваться пакеты; множество H_1 содержит все хосты, в которые должны приходить пакеты.

В настоящей работе мы предполагаем, что реализация двух множеств путей, помеченных различными векторами значений параметров, осуществляется независимо. Поэтому в дальнейших рассуждениях мы рассматриваем свойства множества путей и правил для заданного вектора значений параметров, который не будем указывать в используемых ниже обозначениях функций, зависящих от него.

Порты. Будем считать, что с каждой вершиной a графа ассоциированы множество входных портов и множество выходных портов, причём каждый входной (выходной) порт соответствует некоторому одному ребру, инцидентному вершине a, и, наоборот, каждому ребру, инцидентному вершине a, в вершине a соответствуют её входной и её выходной порты. Тем самым, между множеством инцидентных вершине рёбер, множеством её входных портов и множеством её выходных портов существует взаимно-однозначное соответствие. Поскольку в графе G нет кратных ребер и петель, каждому ребру, инцидентному вершине a, взаимно-однозначно соответствует соседняя с a вершина. Поэтому без ограничения общности мы можем вместо идентификатора порта вершины a использовать идентификатор соседа вершины a.

Таким образом, в настоящей работе мы предполагаем, что через контроллер таблицы коммутаторов настраиваются так, чтобы (для данного вектора значений параметров) каждое правило в таблице определяло выходные порты, по которым посылается пакет, только в зависимости от порта, по которому принят пакет. Поскольку в графе G нет кратных ребер, то это равносильно тому, что в коммутаторе определяется соседний коммутатор, которому нужно

передать пакет, принятый от предыдущего коммутатора. Мы также предполагаем, что коммутаторы *«чистые»*, т.е. реализуют в своих таблицах только то, что мы через контроллер попросили сделать.

Правила. Мы предполагаем, что коммутатор работает по правилам, отражаемым в его таблицах. Опираясь на основное предположение модели, *примитивным правилом* (для данного вектора значений параметров) будем считать тройку $(a,s,b) \in V \times S \times V$, где a и b – соседи коммутатора s. Такое правило говорит, что коммутатор s, получая пакет (с данным вектором значений параметров) от соседа a, должен отправить его соседу b. Если есть несколько правил, отличающихся только соседом b, то это значит, что коммутатор s выполняет *клонирование*, т.е. принятый пакет посылается сразу нескольким соседям. Правило в таблице может быть не примитивным, т.е. описывать множество примитивных правил. Например, правило (all,s,b) означает, что пакет (с данным вектором значений параметров), принятых коммутатором s от любого соседа, посылается соседу b. При выполнении правил в коммутаторах в общем случае присутствуют приоритеты. Однако приоритетами можно пренебречь, если использовать достаточно простую редукцию. Пусть есть два правила, выражаемые двумя множествами примитивных правил: A и B, и первое правило приоритетнее второго. При таких условиях остаются примитивные правила $A \cup \{ (a,s,b) \in B \mid \forall\, b'\, (a,s,b') \notin A \}$, т.е. в настройке коммутатора остается только множество равно приоритетных примитивных правил.

Множество правил коммутатора s называется *настройкой коммутатора* s и обозначается $T_s \subseteq N(s) \times \{s\} \times N(s)$. Объединение $T = \cup \{ T_s \mid s \in S \}$ настроек всех коммутаторов сети называется *настройкой сети*, при этом через $T(s) = \{ (a,s,b) \in T \}$ обозначим настройку коммутатора s как часть настройки сети T.

3. Взаимосвязь путей и правил

В этом разделе мы обсуждаем (для пакетов с данным вектором значений параметров) два вопроса: 1) какие пути будут проходить пакеты при заданной настройке сети, 2) при какой настройке сети пакеты будут проходить заданное множество полных путей.

Правила порождают пути. Правило (a,b,c) порождает путь a,b,c длиной 2. Если правилами порождены два пути вида $p{\cdot}(x,y){\cdot}q$ и $p`{\cdot}(x,y){\cdot}q`$, то эти правила порождают также пути $p{\cdot}(x,y){\cdot}q`$ и $p`{\cdot}(x,y){\cdot}q$, отличные от пути (x,y), по которым также пройдут пакеты. Таким образом, настройка сети T порождает следующее множество путей, которое будем обозначать $T{\uparrow}$: $\forall\, a,b,c,x,y,p,q,p`,q`$

(1) $(a,b,c) \in T \quad \vdash (a,b,c) \in T{\uparrow}$,

(2) $(|p|>1 \vee |q`|>1)\ \&\ p{\cdot}(x,y){\cdot}q \in T{\uparrow}\ \&\ p`{\cdot}(x,y){\cdot}q` \in T{\uparrow} \quad \vdash p{\cdot}(x,y){\cdot}q` \in T{\uparrow}$.

Правила порождают только правильные пути длиной не менее 2, и вместе с каждым порожденным путем порождают все его отрезки длиной не менее 2: $T{\uparrow} = T{\uparrow}^{\#}$. Если правила порождают полный путь, то он, очевидно, является максимальным во множестве порожденных путей. Обратное, однако, не всегда верно. Если есть правило (a,b,c), но в коммутаторе c нет правила вида (b,c,e), то путь a,b,c может быть только конечным отрезком порождаемого пути, а такой путь не является полным, поскольку c коммутатор. В этом случае даже если пакет пройдет путь a,b,c, после этого он будет потерян. Если есть правило (a,b,c), но в коммутаторе a нет правила вида (e,a,b), то путь a,b,c может быть только началом порождаемого пути, а такой путь не является полным, поскольку a коммутатор. В этом случае для прохождения пути a,b,c пакет должен был бы генерироваться в коммутаторе a, что противоречит тому, что пакеты генерируются только в хостах. Правила могут порождать «зацикливание» пакетов, т.е. бесконечные пути: например, три правила (a,b,c), (b,c,a) и (c,a,b) порождают цикл a,b,c,a,b с повтором дуги (a,b) и, следовательно, порождают бесконечный путь $a,b,c,a,b,c,...$

Правильные пути порождают правила. Для того чтобы пакет мог пройти правильный путь $p{\cdot}(a,b,c){\cdot}q$, необходимо правило (a,b,c). Формально множество P правильных путей порождает следующее множество правил, которое будем обозначать $P{\downarrow}$: $\forall\, a,b,c,p,q$

(3) $p{\cdot}(a,b,c){\cdot}q \in P \quad \vdash (a,b,c) \in P{\downarrow}$.

Согласно правилам вывода 1-3, заданное множество P правильных путей порождает множество путей $P{\downarrow}{\uparrow}$ следующим образом:

$\forall\, a,b,c,x,y,p,q,p`,q`$

(4) $p{\cdot}(a,b,c){\cdot}q \in P \quad \vdash (a,b,c) \in P{\downarrow}{\uparrow}$,

(5) $(|p|>1 \vee |q`|>1)\ \&\ p{\cdot}(x,y){\cdot}q \in P{\downarrow}{\uparrow}\ \&\ p`{\cdot}(x,y){\cdot}q` \in P{\downarrow}{\uparrow} \quad \vdash p{\cdot}(x,y){\cdot}q` \in P{\downarrow}{\uparrow}$.

Как отмечено выше, мы предполагаем, что пакеты должны генерироваться и приниматься без дальнейшей пересылки только в хостах [10]; поэтому нас будут интересовать полные пути, т.е. правильные пути, которые начинаются и заканчиваются в хостах. Определим понятие правильной настройки, при которой, во-первых, все максимальные пути в $T{\uparrow}$ полные (полный путь всегда максимален среди правильных путей) и ведут из начальных хостов во все конечные хосты, а, во-вторых, любой порождаемый путь является отрезком максимального пути, т.е. не происходит «зацикливания». Такое «зацикливание» возникает тогда, когда пакет дважды проходит дугу. Поэтому для того, чтобы не было «зацикливаний», необходимо и достаточно, чтобы все пути в $T{\uparrow}$ были реберно-простые. Для заданных множеств H_0 начальных и H_1 конечных хостов настройка сети T называется *правильной* (относительно данного вектора значений параметров), если выполняются следующие условия.

1) Множество $T{\uparrow}$ содержит только реберно-простые (правильные) пути (*условие отсутствия циклов*).

2) Каждый максимальный путь из $T\uparrow$ начинается в начальном хосте (*условие начала путей*).
3) Каждый максимальный путь из $T\uparrow$ заканчивается в конечном хосте, и для любого начального хоста h_0: если хотя бы один путь из $T\uparrow$ начинается в h_0, то для любого конечного хоста h_1, в $T\uparrow$ существует путь из h_0 в h_1 (*условие конца путей*).

Заметим, что условия правильности настройки требует, чтобы пакеты генерировались в начальных хостах, но не требуют, чтобы это можно было делать в каждом начальном хосте. Если в $T\uparrow$ все пути реберно-простые, то в $T\uparrow$ существуют пути, максимальные по длине среди всех путей, и любой путь является отрезком некоторого максимального пути из $T\uparrow^{max}$. Очевидно, что для правильной настройки T во множестве $T\uparrow$ полные пути совпадают с максимальными путями. Заметим также, что T не обязано содержать настройки всех коммутаторов. Если в T нет ни одного правила для коммутатора s, т.е. правила вида (a,s,b), то такой коммутатор «глотает» все поступающие к нему пакеты. Однако это не создает никаких проблем: если порождаемое множество $T\uparrow^{max}$ не содержит путей, проходящих через коммутатор s, то таких «глотаемых» пакетов не будет.

Множество P полных путей будем называть *реализуемым*, если настройка $P\downarrow$ правильная. Очевидно, что для множества P полных путей $P \subseteq P\downarrow\uparrow^{max}$. Реализуемое множество P полных путей будем называть *строго реализуемым*, если выполнено дополнительное условие:

4) $P = P\downarrow\uparrow^{max}$.

4. Тестирование правила

Для правильной настройки сети T для каждого правила каждого коммутатора существует, по крайней мере, один путь, при передаче пакетов по которому используется (проверяется) это правило, т.е. имеет место следующее утверждение.

Утверждение 4.1. Пусть T правильная настройка. Тогда для любого правила (a,s,b) любого коммутатора s во множестве $T\uparrow$ существует путь $p{\cdot}(a,s,b){\cdot}q$, который начинается в некотором хосте из H_0 и заканчивается в хосте из H_1.

Доказательство. По правилу вывода (1) каждое правило настройки (a,s,b) порождает, по крайней мере, путь a,s,b. Если все пути, порождаемые настройкой сети, реберно-простые (условие отсутствия «зацикливания»), то путь a,s,b является отрезком некоторого максимального порождаемого пути $p{\cdot}(a,s,b){\cdot}q$. Но если настройка T правильная, то этот путь начинается в начальном хосте (условие начала путей) и заканчивается в конечном хосте (условие конца путей). □

Заметим, что для некоторых правил таких полных путей, «проверяющих» это правило, может быть несколько.

Из утверждения 4.1 следует, что тестирование правильной настройки T сводится к посылке пакета (с данным вектором значений параметров) из каждого хоста h, для которого $T\uparrow^{max}$ содержит путь, начинающийся в h. Наблюдая пересылки пакетов между соседними узлами, можно обнаружить любую ошибку или в настройке коммутатора, и/или в работе коммутатора по правильной настройке.

Будем говорить, что правило (a,s,b) *проверяемо*, если существует правильная настройка сети T такая, что правило $(a,s,b) \in T$. Если верна гипотеза о правиле, то работа коммутатора s по правилу (a,s,b) не зависит от других правил в настройке коммутатора s и от настроек других коммутаторов сети. Эта гипотеза дает возможность проверять правила настройки «по одному»: если при некоторой настройке сети коммутатор правильно работает по данному правилу, то он будет правильно работать по этому правилу при любой настройке сети, в которой есть это правило. Поэтому для полной проверки работы коммутаторов сети достаточно отдельно проверить работу каждого коммутатора по каждому правилу его настройки, которое проверяемо. Для каждого проверяемого правила выполняется настройка сети так, чтобы по ней можно было проверить это правило: послать «пакет», проходящий по этому правилу хотя бы один раз. Такое тестирование можно также понимать как первоначальное «грубое» тестирование настроек, если гипотеза о правиле не верна. В этом разделе мы сформулируем и докажем необходимое и достаточное условие проверяемости правила.

Пусть s – коммутатор, и граф G содержит ребра $\{s,a\}$ и $\{b,s\}$. Рассмотрим граф, получающийся из графа G удалением обоих ребер $\{s,a\}$ и $\{b,s\}$. Обозначим через V_a, V_s, V_b множества вершин тех компонент связности этого графа, которым принадлежат вершины a, s, b, соответственно.

Утверждение 4.2. Необходимым и достаточным условием существования реберно-простого пути, начинающегося в хосте из H_0, заканчивающегося в хосте из H_1 и содержащего отрезок (a,s,b), где $a \neq b$, является условие $H_0 \cap V_a \neq \varnothing \vee H_1 \cap V_b \neq \varnothing \vee H_0 \cap V_s \neq \varnothing \;\&\; H_1 \cap V_s \neq \varnothing$.

Доказательство. Необходимость (см. рис. 2 сверху). Поскольку $H_0 \subseteq V, H_1 \subseteq V$, $H_0 \neq \varnothing$, $H_1 \neq \varnothing$, и $V = V_a \cup V_b \cup V_s$, из равенства $V_a = V_b$ следует $H_0 \cap V_a \neq \varnothing \vee H_0 \cap V_s \neq \varnothing$ и $H_1 \cap V_b \neq \varnothing \vee H_1 \cap V_s \neq \varnothing$, что влечет истинность условия. Поэтому, если условие нарушено, то $V_a \neq V_b$, т.е. ребро $\{s,a\}$ или ребро $\{b,s\}$ является мостом. Рассмотрим произвольный путь p_{0a} от $h_0 \in H_0$ до a, и произвольный путь q_{b1} от b до некоторого хоста $h_1 \in H_1$. Такие пути существуют, если существует путь, начинающийся в хосте из H_0, заканчивающийся в хосте из H_1 и содержащий отрезок (a,s,b).

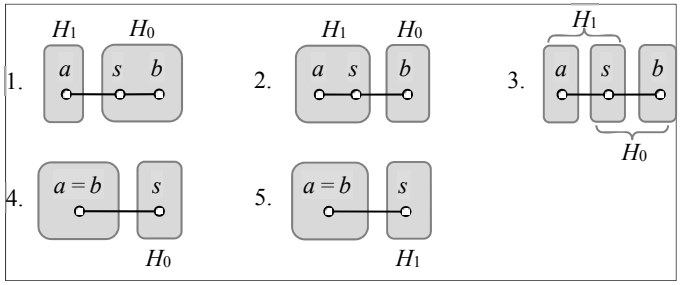

Рис. 2. Случаи, когда правило (a, s, b) проверить нельзя
Fig. 2. Cases where the rule (a, s, b) cannot be checked

Если ребро $\{s, a\}$ является мостом, а ребро $\{b, s\}$ не является мостом (см. рис. 2, случай 1), то из нарушения условия следует, что $H_0 \cap V_a = \varnothing$ (и, следовательно, $H_0 \cap V_s \neq \varnothing$), $H_1 \cap V_b = \varnothing$ и $H_1 \cap V_s = \varnothing$. А тогда пути p_{0a} и q_{b1} проходят по дуге (s, a). Если ребро $\{b, s\}$ является мостом, а ребро $\{s, a\}$ не является мостом (см. рис. 2, случай 2), то из нарушения условия следует, что $H_1 \cap V_b = \varnothing$ (и, следовательно, $H_1 \cap V_s \neq \varnothing$), $H_0 \cap V_a = \varnothing$ и $H_0 \cap V_s = \varnothing$. А тогда пути p_{0a} и q_{b1} проходят по дуге (b, s). Если оба ребра $\{s, a\}$ и $\{b, s\}$ являются мостами (см. рис. 2, случай 3), то из нарушения условия следует, что $H_0 \cap V_a = \varnothing$, $H_1 \cap V_b = \varnothing$ и $H_0 \cap V_s = \varnothing \vee H_1 \cap V_s = \varnothing$. А тогда пути p_{0a} и q_{b1} проходят либо по дуге (b, s), если $H_0 \cap V_s = \varnothing$, либо по дуге (s, a), если $H_1 \cap V_s = \varnothing$. Поэтому любой путь вида p_{0a}▪(a, s, b)▪q_{b1} не является реберно-простым, поскольку дважды проходит по дуге (s, a) или по дуге (s, b).

Достаточность. Условие утверждения представляет собой дизъюнкцию трех выражений. Рассмотрим три случая, соответствующие этим трем выражениям. Если $H_0 \cap V_a \neq \varnothing$ (см. рис. 3, случай 1), то существует вершинно-простой путь p_{0a} от некоторой вершины $h_0 \in H_0$ до a, не проходящий по ребрам $\{s, a\}$ и $\{b, s\}$. Поскольку граф G связен, существует вершинно-простой путь q_{b1} от b до некоторого хоста $h_1 \in H_1$. Пути p_{0a}▪(a, s, b) и q_{b1} имеют общую вершину b, поэтому их можно представить в виде p_{0a}▪$(a, s, b) = p_{0c}$▪p_{cb} и $q_{b1} = q_{bc}$▪q_{c1}, где p_{0c} и q_{bc} заканчиваются в одной вершине c – последней на пути q_{b1} общей вершине путей p_{0a}▪(a, s, b) и q_{b1}. Обозначим через p_{bc} путь p_{cb}, проходимый в обратном порядке от b к c. Тогда путь p_{0a}▪(a, s, b)▪p_{bc}▪q_{c1} является искомым реберно-простым путем.

Если $H_1 \cap V_b \neq \varnothing$ (см. рис. 3, случай 2), то существует вершинно-простой путь q_{b1} от b до некоторого хоста $h_1 \in H_1$, не проходящий по ребрам $\{s, a\}$ и $\{b, s\}$. Поскольку граф G связен, существует вершинно-простой путь p_{0a} от h_0 до a. Пути (a, s, b)▪q_{b1} и p_{0a} имеют общую вершину a, поэтому их можно представить в виде (a, s, b)▪$q_{b1} = q_{ac}$▪q_{c1} и $p_{0a} = p_{0c}$▪p_{ca}, где q_{ac} и p_{0c} заканчиваются в одной вершине c – первой на пути p_{0a} общей вершине путей (a, s, b)▪q_{b1} и p_{0a}.

Обозначим через q_{ca} путь q_{ac}, проходимый в обратном порядке от c к a. Тогда путь p_{0c}▪q_{ca}▪(a, s, b)▪q_{b1} является искомым реберно-простым путем.

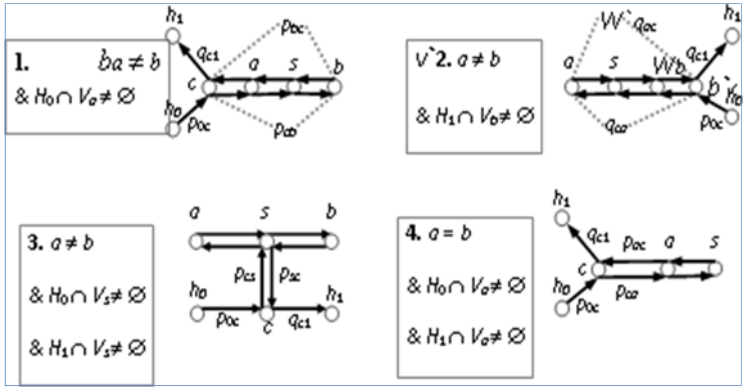

Рис. 3. Случаи, когда правило (a, s, b) проверить можно
Fig. 3. Cases where the rule (a, s, b) can be checked

Если $H_0 \cap V_s \neq \varnothing$ & $H_1 \cap V_s \neq \varnothing$ (см. рис. 3, случай 3), то существуют вершинно-простые пути p_{0s} от h_0 до s и q_{s1} от s до некоторого хоста $h_1 \in H_1$, не проходящие по ребрам $\{s, a\}$ и $\{b, s\}$. Эти пути имеют общую вершину s, поэтому их можно представить в виде $p_{0s} = p_{0c}$▪p_{cs} и $q_{s1} = q_{sc}$▪q_{c1}, где p_{0c} и q_{sc} заканчиваются в одной вершине c – первой на пути p_{0s} общей вершине путей p_{0s} и q_{s1}. Обозначим через p_{sc} путь p_{cs}, проходимый в обратном порядке от s к c. Тогда, поскольку $a \neq b$, путь p_{0s}▪(s, a, s, b, s)▪p_{sc}▪q_{c1} является искомым реберно-простым путем. □

Аналогично доказывается подобное утверждение, когда $a = b$.

Утверждение 4.3. Необходимым и достаточным условием существования реберно-простого пути, начинающегося в хосте из H_0, заканчивающегося в хосте из H_1 и содержащего отрезок (a, s, a), является следующее условие: $H_0 \cap V_a \neq \varnothing$ & $H_1 \cap V_a \neq \varnothing$.

Доказательство. Необходимость (см. рис. 2 снизу). Рассмотрим произвольный путь p_{0a} от некоторой вершины $h_0 \in H_0$ до a, и произвольный путь q_{a1} от a до некоторого хоста $h_1 \in H_1$. Если $H_0 \cap V_a = \varnothing$ (см. рис. 2, случай 4), то путь p_{0a} проходит по дуге (s, a). Если $H_1 \cap V_a = \varnothing$ (см. рис. 2, случай 5), то путь q_{a1} проходит по дуге (s, a). В обоих случаях любой путь вида p_{0a}▪(a, s, a)▪q_{a1} не является реберно-простым, поскольку дважды проходит по дуге (s, a).

Достаточность (см. рис. 3, случай 4). Если $H_0 \cap V_a \neq \varnothing$, то существует вершинно-простой путь p_{0a} от h_0 до a, не проходящий по ребру $\{s, a\}$. Если $H_1 \cap V_a \neq \varnothing$, то существует вершинно-простой путь q_{a1} от a до некоторого хоста $h_1 \in H_1$, не проходящий по ребру $\{s, a\}$. Пути p_{0a} и q_{a1} имеют общую вершину a, поэтому их можно представить в виде $p_{0a} = p_{0c}$▪p_{ca} и $q_{a1} = q_{ac}$▪q_{c1}, где p_{0c} и q_{ac} заканчиваются в одной вершине c – последней на пути q_{a1} общей вершине путей

p_{0a} и q_{a1}. Обозначим через p_{ac} путь p_{ca}, проходимый в обратном порядке от a к c. Тогда путь p_{0a}▪(a,s,a)▪p_{bc}▪q_{c1} является искомым реберно-простым путем. □

Утверждение 4.4. Пусть P – множество путей, начинающихся в одном и том же хосте $h_0 \in H_0$, заканчивающихся в некоторых хостах из H_1, такое, что все пути из $P{\downarrow}{\uparrow}$ реберно-простые, и $P{\downarrow}{\uparrow}^{max} = P$. Тогда множество P можно расширить до множества путей, обладающего теми же свойствами, но содержащего пути до всех хостов из H_1.

Доказательство. Пусть в хосте $h \in H_1$ не заканчивается ни один путь из P. Тогда выберем в графе G вершинно-простой путь q из h_0 в h. Если нет ребра, которое путь q проходит в том же направлении, в каком его проходит хотя бы один путь из P, добавим путь q ко множеству P. В противном случае путь q можно представить в виде $q = q_1$▪(x,y)▪q_2, где ребро $\{x,y\}$ – последнее на пути q, которое какой-нибудь путь $p \in P$ проходит в том же направлении (x,y). Такой путь p можно представить в виде $p = p_1$▪(x,y)▪p_2. Тогда добавим путь p_1▪(x,y)▪q_2 ко множеству P. Сделаем это для каждого пути из P, который проходит по тому же ребру $\{x,y\}$ в том же направлении (x,y). Очевидно, полученное множество путей обладает теми же свойствами, что исходное множество P, но теперь в нём есть путь от h_0 до h. Будем повторять эту операцию добавления путей до тех пор, пока в H_1 есть хосты, в которых не заканчиваются пути из P. В конечном итоге получим искомое множество путей. □

Из утверждений 4.2, 4.3 и 4.4 непосредственно следует следующее утверждение.

Утверждение 4.5. Необходимым и достаточным условием того, что правило (a,s,b) проверяемо, является следующие условия: при $a \neq b$ – условие $H_0 \cap V_a \neq \varnothing \vee H_1 \cap V_b \neq \varnothing \vee H_0 \cap V_s \neq \varnothing$ & $H_1 \cap V_s \neq \varnothing$, а при $a = b$ – условие $H_0 \cap V_a \neq \varnothing$ & $H_1 \cap V_a$.

Из этого утверждения вытекают следующие достаточно простые следствия. Пусть s – коммутатор, и граф G содержит ребра $\{s,a\}$ и $\{b,s\}$.

Следствие 4.1. Правило (a,s,b) не проверяемо, если a – хост, не являющийся начальным хостом, т.е. $a \in H \setminus H_0$, или b – хост, не являющийся конечным хостом, т.е. $b \in H \setminus H_1$.

Следствие 4.2. Если ребра $\{s,a\}$ и $\{b,s\}$ не являются мостами в графе G, то правило (a,s,b) проверяемое.

Следствие 4.3. Если $H_0 = H_1$, то правило (a,s,b) проверяемое.

Следствие 4.4. Если $H_0 \cap H_1 \neq \varnothing$ и $a \neq b$, то правило (a,s,b) проверяемое.

5. Проверяемость любой настройки коммутатора

Будем говорить, что данная настройка T_s коммутатора s *проверяема*, если существует правильная настройка сети T такая, что $T(s) = T_s$. Если верна гипотеза о коммутаторе, то работа коммутатора s по правилам настройки T_s не зависит от настроек других коммутаторов сети. Эта гипотеза дает возможность

проверять настройки коммутаторов «по одному»: если коммутатор правильно работает по правилам настройки T_s при данной настройке сети T такой, что $T(s) = T_s$, то он будет правильно работать по правилам настройки T_s при любой настройке сети T, для которой $T(s) = T_s$. Поэтому для полной проверки работы сети достаточно отдельно проверить работу каждого коммутатора по каждой его настройке, которая проверяема. В этом разделе мы исследуем условия, при которых любая настройка заданного коммутатора проверяема. Мы сформулируем и докажем необходимое (но не достаточное) и достаточное (но не необходимое) условия проверяемости любой настройки коммутатора.

Утверждение 5.1. Пусть P – множество правильных путей длины не меньше 2. Тогда $P^{\#} \subseteq P{\downarrow}{\uparrow}$.

Доказательство. Пусть путь $x_1, ..., x_n$ принадлежит $P^{\#}$, и $n > 2$. Тогда этот путь является отрезком некоторого правильного пути q▪$(x_1, ..., x_n)$▪$r \in P$. Этот путь по правилу вывода (4) порождает в $P{\downarrow}{\uparrow}$ пути, среди которых есть следующие: x_i, x_{i+1}, x_{i+2}, где $i = 1..n\text{-}2$. Тогда по правилу вывода (5) в $P{\downarrow}{\uparrow}$ порождается путь $x_1, ..., x_n$. □

Если для любых двух путей p▪(x,y)▪q и p`▪$(x,y)$▪$q$`, принадлежащих множеству путей P и проходящих по одной дуге (x,y), пути p▪(x,y)▪q` и $p$`▪(x,y)▪q, отличные от пути (x,y), также принадлежат P, то множество P будем называть *замкнутым по дуге* (x,y). Множество P будем называть *замкнутым по дугам*, если оно замкнуто по каждой дуге графа. Заметим, что правило вывода (5), по сути, говорит о замкнутости по дугам множества $P{\downarrow}{\uparrow}$. Из определения замкнутости по дугам множества путей непосредственно следует следующее утверждение.

Утверждение 5.2. Если все пути из множества путей P, проходящие через данную дугу, имеют одинаковые префиксы до этой дуги или одинаковые постфиксы после этой дуги, то множество P замкнуто по этой дуге.

Утверждение 5.3. Пусть множество P состоит из правильных путей длины не меньше 2 и замкнуто по дугам. Тогда $P{\downarrow}{\uparrow} = P^{\#}$.

Доказательство. Согласно утверждению 5.10 нам достаточно показать, что $P{\downarrow}{\uparrow} \subseteq P^{\#}$. Пусть путь $p = x_1, ..., x_n$ принадлежит $P{\downarrow}{\uparrow}$. Поскольку все пути в $P{\downarrow}{\uparrow}$ имеют длину не меньше 2, $n > 2$. Тогда для $i = 1..n\text{-}2$ найдутся пути p_i▪(x_i, x_{i+1}, x_{i+2})▪$q_i \in P$. Из замкнутости по дугам множества P следует, что пути p_1▪$(x_1, ..., x_{i+2})$▪$q_i \in P$ для $i = 1..n\text{-}2$. В частности, p_1▪$(x_1, ..., x_n)$▪$q_{n-2} \in P$, т.е. p_1▪p▪$q_{n-2} \in P$. Следовательно, $p \in P^{\#}$. □

Утверждение 5.4. Пусть множество P реберно-простых полных путей замкнуто по дугам. Тогда $P{\downarrow}{\uparrow}$ состоит из реберно-простых путей, и $P{\downarrow}{\uparrow}^{max} = P$.

Доказательство. Поскольку каждый хост соединен ребром только с коммутатором, все полные пути имеют длину не меньше 2. Тогда по утверждению 5.3 $P{\downarrow}{\uparrow} = P^{\#}$. Поскольку отрезок реберно-простого пути является реберно-простым, а все пути в P реберно-простые, все пути в $P^{\#}$ и, следовательно, в $P{\downarrow}{\uparrow}$ тоже реберно-простые. Поскольку $P{\downarrow}{\uparrow} = P^{\#}$, имеем

$P{\downarrow}{\uparrow}^{max} = P^{\#\,max} = P^{max}$. Но все пути в P полные, следовательно, они максимальные. Поэтому $P^{max} = P$, что влечет $P{\downarrow}{\uparrow}^{max} = P$. □

Обозначим через G_s подграф графа G, получающийся удалением коммутатора s вместе со всеми инцидентными ему ребрами.

Утверждение 5.5 (необходимое условие проверяемости любой настройки коммутатора). Если коммутатор s является *шарниром*[3] в графе G, то не любая настройка коммутатора s проверяема.

Доказательство. Пусть коммутатор s – шарнир в графе G. Тогда подграф G_s содержит, по крайней мере, две компоненты связности, и хотя бы одна из этих компонент содержит вершину из H_1. Обозначим множество вершин этой компоненты через V_1: $H_1 \cap V_1 \neq \varnothing$. Тогда найдётся вершина $a \in N(s) \setminus V_1$. Рассмотрим настройку коммутатора s, состоящую из одного правила (a, s, a). Необходимым условием проверяемости такой настройки является наличие в графе G_s хоста из H_0, из которого была бы достижима как вершина a, так и вершины из $H_1 \cap V_1$. Однако в графе G_s такого хоста быть не может. Поэтому такая настройка не проверяема. □

Следствие 5.1. Если у коммутатора s среди соседей есть хосты, то не любая настройка коммутатора s проверяема на этой физической сети.

Доказательство. Поскольку хост имеет степень 1, коммутатор s является шарниром. Тогда по утверждению 5.5, не любая настройка коммутатора s проверяема на заданном графе физических связей. □

Если ребро $\{u, v\}$ графа G является мостом, то будем называть его s-мостом, если вершины u и v коммутаторы. Компонента связности графа, получающегося из исходного графа удалением всех его мостов, называется *компонентой рёберной двусвязности* исходного графа. Если удаляются только s-мосты, то будем говорить о *компоненте рёберной s-двусвязности* графа.

Утверждение 5.6 (достаточное условие проверяемости любой настройки коммутатора). Если коммутатор s не является шарниром в графе G, и у графа G_s есть компонента рёберной s-двусвязности, в которой есть начальный и конечный хосты, то любая настройка коммутатора s проверяема.

Доказательство. Пусть коммутатор s не является шарниром в графе G, и у графа G_s есть компонента рёберной s-двусвязности $X = (V_X, E_X)$, в которой есть начальный и конечный хосты. Рассмотрим произвольную настройку T_s коммутатора s. Для ее проверяемости достаточно существования такого строго реализуемого множества P полных путей, которое для каждого правила $(a, s, b) \in T_s$ содержит путь, в котором есть отрезок (a, s, b). Доказательство конструктивное: мы опишем алгоритм построения множества P и докажем, что построенное множество удовлетворяет указанным требованиям. Шаги алгоритма иллюстрируются на рис. 4.

[3] Шарнир (*cut point, articulation point*) – вершина, удаление которой вместе с инцидентными ей ребрами увеличивает число компонентов связности графа.

Алгоритм. Выберем какие-нибудь начальный хост $v \in V_X \cap H_0$ и конечный хост $w \in V_X \cap H_1$. Пусть $v`$ и $w`$ – коммутаторы, к которым подсоединены хосты v и w. Поскольку в компоненте X нет s-мостов, после удаления из этой компоненты хостов и инцидентных им ребер в ней нет мостов. Как известно, в графе без мостов для любых двух вершин существует реберно-простой цикл, проходящий через них. Поэтому для коммутаторов $v`$ и $w`$ в компоненте X существует реберно-простой цикл C, проходящий через них. Этот цикл порождает подграф $X_C = (V_C, E_C)$, состоящий из вершин и ребер цикла. Поскольку коммутатор s не является шарниром в графе G, граф G_s связный.

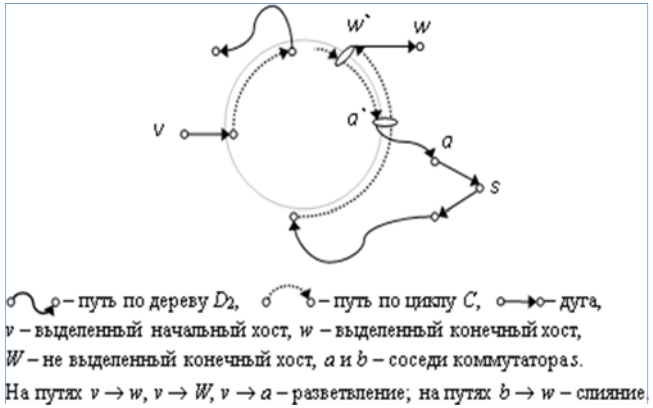

Рис. 4. Построение путей от одного начального хоста ко всем конечным хостам
Fig. 4. Building paths from one sender host to all destination hosts

Выберем в графе G_s произвольное остовное дерево $D = (V, E_D)$. Будем последовательно удалять ребра дерева D, не принадлежащие циклу C, т.е. ребра множества $E_D \setminus E_C$, если при этом не нарушается связность подграфа $(V, E_C \cup E_D)$. В результате получим лес D_1, в котором из каждой вершины x есть единственный вершинно-простой путь до цикла C – до вершины, которую мы обозначим через $x`$. Затем будем последовательно удалять ребра леса $D_1$, инцидентные его листовым вершинам, не принадлежащим $H_1 \cup N(s) \cup V_C$. В результате получим лес $D_2$, в котором из каждой вершины $x \in H_1 \cup N(s)$ есть единственный вершинно-простой путь до цикла $C$ – до вершины $x`$. Тогда для каждого правила $(a, s, b) \in T_s$ выбираем путь $p(a, s, b) = (v, v`)\cdot L_{v`a`}\cdot P_{a`a}\cdot(a, s, b)\cdot P_{bb`}\cdot R_{b`w`}\cdot(w`, w)$, где $L_{v`a`}$ – путь по циклу C по часовой стрелке от $v`$ до $a`$, $P_{a`a}$ – путь по лесу $D_2$ от $a`$ до a, $P_{bb`}$ – путь по лесу $D_2$ от $b$ до $b`$, $R_{b`w`}$ – путь по циклу C против часовой стрелки от $b`$ до $w`$. Также для каждого конечного хоста $W \in H_1$ выбираем путь $p(W) = (v, v`)\cdot L_{v`W`}\cdot P_{W`W}$, где $L_{v`W`}$ – путь по циклу C по часовой стрелке от $v`$ до $W`$, $P_{W`W}$ – путь по лесу $D_2$ от $W`$ до W. Множество выбранных путей обозначим P.

Докажем, что алгоритм строит требуемое множество путей P. По построению это множество для каждого правила $(a,s,b) \in T_s$ содержит путь $p(a,s,b)$, в котором есть отрезок (a,s,b). Каждый путь из P реберно-простой и полный: начинается в $v \in H_0$ и заканчивается в хосте из H_1, т.е. P – множество полных путей и выполнено условие 2 – условие начала путей в определении правильной настройки. Поскольку каждый путь из P заканчивается в хосте из H_1, а для каждого конечного хоста W множество P содержит путь $p(W)$, ведущий из $v \in H_0$ в W, т.е. выполнено условие 3 – условие конца путей в определении правильной настройки. Кроме того, множество P обладает тем свойством, что все пути, проходящие по одной дуге, имеют одинаковые префиксы до этой дуги (разветвление), если эта дуга лежит на пути вида $(v,v`)\blacksquare L_{v`a`}\blacksquare P_{a`a}\blacksquare(a,s)$ или $(v,v`)\blacksquare L_{v`W`}\blacksquare P_{W`W}$, или одинаковые постфиксы после этой дуги (слияние), если дуга лежит на пути вида $(s,b)\blacksquare P_{bb`}\blacksquare R_{b`w`}\blacksquare(w`,w)$. Следовательно, по утверждению 5.2 множество P замкнуто по дугам. А тогда по утверждению 5.4 множество $P{\downarrow}{\uparrow}$ состоит из реберно-простых путей, т.е. выполнено условие 1 – условие отсутствия циклов в определении правильности настройки, и $P{\downarrow}{\uparrow}^{max} = P$, т.е. выполнено условие 4 в определении строгой реализуемости. □

Следствие 5.2. Если в графе G_s нет s-мостов, то любая настройка коммутатора s проверяема.

6. Заключение

Утверждение 4.5 дает необходимое и достаточное условие проверямости заданного правила заданного коммутатора. Соответствующее тестирование всех проверяемых правил всех коммутаторов «по одному» будет полным, если верна (сильная) гипотеза о правиле: работа коммутатора по данному правилу не зависит от других правил настройки этого коммутатора и от настроек других коммутаторов.

Если гипотеза о правиле не верна, можно воспользоваться (слабой) гипотезой о коммутаторе, предполагающей независимость работы коммутатора только от настроек других коммутаторов. Утверждение 5.5 дает необходимое (но не достаточное) условие проверяемости любой настройки коммутатора: коммутатор не должен быть шарниром в графе G. Утверждение 5.6 дает достаточное (но не необходимое) условие проверяемости любой настройки коммутатора: у графа G_s есть компонента рёберной s-двусвязности, в которой есть начальный и конечный хосты.

Остается неисследованным случай, когда коммутатор s не является шарниром, но на дереве компонент реберной s-двусвязности графа G_s есть компоненты, которые содержат начальные хосты, и есть компоненты, которые содержат конечные хосты, но нет компоненты, которая содержала бы как начальный, так и конечный хосты.

Кроме того, для коммутаторов соседних с хостами было бы естественно рассматривать не любые настройки, а только те, которые не являются заведомо

не проверяемыми. Иными словами было бы правильно исключить из рассмотрения настройки, содержащие правило (a,s,b), если $a \in H \setminus H_0$, т.е. a – хост, но не начальный, и/или $b \in H \setminus H_1$, т.е. b – хост, но не конечный.

Все эти случаи, конечно, являются частными случаями более общей задачи: определить необходимые и достаточные условия проверяемости заданной настройки T_s данного коммутатора s. Условия, естественно, налагаются на совокупность (G,S,H,H_0,H_1,s,T_s). Для проверяемой настройки коммутатора требуется также найти эффективный алгоритм построения проверяющей ее правильной настройки сети.

Благодарности

Авторы выражают благодарность исследователям Хорхе Лопэзу, Наталье Кушик, Джамалю Зеглашу университета Телеком Южный Париж (Еври, Франция) за многочисленные плодотворные дискуссии при подготовке данной работы.

Список литературы

[1]. Sezer S., Scott-Hayward S., Chouhan P. K. et al. Are we ready for sdn? Implementation challenges for software-defined networks. IEEE Communications Magazine, vol. 51, no. 7, 2013, pp. 36–43.

[2]. Gill P., Jain N., and Nagappan N. Understanding network failures in data centers: Measurement, analysis, and implications. In Proc. of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, 2011, pp. 350–361.

[3]. Scott C., Wundsam A., Raghavan B. et al. Troubleshooting blackbox sdn control software with minimal causal sequences. ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, 2015, pp. 395–406.

[4]. Shalimov A., Zuikov D., Zimarina D., Pashkov V., and Smeliansky R. Advanced study of sdn/openflow controllers. In Proc. of the 9th Central & Eastern European Software Engineering Conference in Russia, 2013.

[5]. Yao J., Wang Z., Yin X., Shiyz X., and Wu J. Formal modeling and systematic black-box testing of sdn data plane. In The IEEE 22nd International Conference on Network Protocols, 2014, pp. 179–190.

[6]. Zhang Z., Yuan D., and Hu H. (2016). Multi-layer modeling of openflow based on efsm. In Proceedings of the 2016 4th International Conference on Machinery, Materials and Information Technology Applications, 2016, pp. 524-529.

[7]. J López, N. Kushik, D. Zeghlache. Quality Estimation of Virtual Machine Placement in Cloud Infrastructures. Lecture Notes in Computer Science, vol. 10533, 2017, pp. 213-229.

[8]. Canini M., Kostic D., Rexford J., and Venzano D. Automating the testing of openflow applications. In Proceedings of the 1st International Workshop on Rigorous Protocol Engineering, 2011.

[9]. Canini M., Venzano D., Peresini P. et al. A nice way to test openflow applications. In Proc. of the 9th USENIX conference on Networked Systems Design and Implementation, 2012, pages 127–140.

[10]. A. Berriri, J. López, N. Kushik, N. Yevtushenko, D. Zeghlache. Towards Model based Testing for Software Defined Networks. In Proc. of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, 2018, pp. 440-446.
[11]. ONOS Project. Key and field description. Режим доступа: https://wiki.onosproject.org/display/ONOS/Flow+Rules#FlowRules-Keyandfielddescription. Дата обращения 15.11.2018.

Testing switch rules in software defined networks

I.B. Burdonov <igor@ispras.ru>
N.V. Yevtushenko <yevtushenko@ispras.ru>
A.S. Kossatchev <kos@ispras.ru>

Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. SDN-technology is efficiently used for implementing service function chains can be efficiently implemented utilizing common resources and their management principles in virtual networks. The network is based on a connected undirected graph of physical links called usually referred to as resource network connectivity topology (RNCT); graph nodes are network switches and hosts and each host is connected exactly with one switch. Switches operate based on rule tables that are configured by a controller that operates independently of network equipment. The configuration of network switches provides the transmission of packets from the initial to final hosts depending on the values of the packet parameters. The paper discusses the relationship between switch configurations and paths which are created for trasmitting packets depending on RNCT properties. It is shown that, in general, not any configuration of any switch is verifiable. Testing abilities depend on the accepted hypotheses about the switch operating. Two hypotheses are discussed in the paper: the switch hypothesis assumes that the switch operation does not depend on the settings of other switches; a stronger hypothesis about the rule, besides this, assumes that the switch operation according to this rule does not depend on other rules in the configuration of this switch. Section 2 contains preliminaries while Section 3 is devoted to the relationship between switch rules and sets of paths to be implemented. In Section 4, the problem of testing the switch configuration is considered based on the rule hypothesis; a number of statements are estableshed, in particular, the necessary and sufficient conditions of the ability of testing a given rule of a given switch. Section 5 discusses and proves the necessary (but not sufficient) condition and sufficient (but not necessary) condition for checking any switch configuration based on the switch hypothesis. In conclusion, the problems of establishing the necessary and sufficient conditions for verifiability of any switch configuration are discussed.

Keywords: SDN-technology; hosts and switches; network switch configuration; packet transmission; edge simple paths; testing network switches.

DOI: -10.15514/ISPRAS-2018-30(6)-4

For citation: Burdonov I.B., Yevtushenko N.V., Kossatchev A.S. Testing switch rules in software defined networks. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 69-88 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-4

References

[1]. Sezer S., Scott-Hayward S., Chouhan P. K. et al. Are we ready for sdn? Implementation challenges for software-defined networks. IEEE Communications Magazine, vol. 51, no. 7, 2013, pp. 36–43.
[2]. Gill P., Jain N., and Nagappan N. Understanding network failures in data centers: Measurement, analysis, and implications. In Proc. of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, 2011, pp. 350–361.
[3]. Scott C., Wundsam A., Raghavan B. et al. Troubleshooting blackbox sdn control software with minimal causal sequences. ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, 2015, pp. 395–406.
[4]. Shalimov A., Zuikov D., Zimarina D., Pashkov V., and Smeliansky R. Advanced study of sdn/openflow controllers. In Proc. of the 9th Central & Eastern European Software Engineering Conference in Russia, 2013.
[5]. Yao J., Wang Z., Yin X., Shiyz X., and Wu J. Formal modeling and systematic black-box testing of sdn data plane. In The IEEE 22nd International Conference on Network Protocols, 2014, pp. 179–190.
[6]. Zhang Z., Yuan D., and Hu H. (2016). Multi-layer modeling of openflow based on efsm. In Proceedings of the 2016 4th International Conference on Machinery, Materials and Information Technology Applications, 2016, pp. 524-529.
[7]. J López, N. Kushik, D. Zeghlache. Quality Estimation of Virtual Machine Placement in Cloud Infrastructures. Lecture Notes in Computer Science, vol. 10533, 2017, pp. 213-229.
[8]. Canini M., Kostic D., Rexford J., and Venzano D. Automating the testing of openflow applications. In Proceedings of the 1st International Workshop on Rigorous Protocol Engineering, 2011.
[9]. Canini M., Venzano D., Peresini P. et al. A nice way to test openflow applications. In Proc. of the 9th USENIX conference on Networked Systems Design and Implementation, 2012, pages 127–140.
[10]. A. Berriri, J. López, N. Kushik, N. Yevtushenko, D. Zeghlache. Towards Model based Testing for Software Defined Networks. In Proc. of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, 2018, pp. 440-446.
[11]. ONOS Project. Key and field description. Available at: https://wiki.onosproject.org/display/ONOS/Flow+Rules#FlowRules-Keyandfielddescription. Accessed 15.11.2018.