ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

Министерство науки и высшего образования Российской Федерации
Ярославский государственный университет им. П. Г. Демидова

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

Том 26 № 1 (79) 2019

Основан в 1999 году
Выходит 4 раза в год

Адрес редакции: ЯрГУ, ул. Советская, 14, г. Ярославль, 150003, Россия
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru; телефон (4852) 79-77-73

Научные статьи в журнал принимаются по электронной почте. Статьи должны содержать УДК, аннотации на русском и английском языках и сопровождаться набором текста в редакторе LaTeX. Плата с аспирантов за публикацию рукописей не взимается.

12+

СОДЕРЖАНИЕ

Моделирование и анализ информационных систем. Т. 26, №1. 2019

Свидетельство о регистрации СМИ ПИ № ФС 77 – 66186 от 20.06.2016 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Учредитель – Федеральное государственное бюджетное образовательное учреждение высшего образования "Ярославский государственный университет им. П. Г. Демидова". Подписной индекс – 31907 в Объединенном каталоге "Пресса России". Редактор, корректор А.А. Аладьева. Редактор перевода Э.И. Соколова. Подписано в печать 07.03.2019. Дата выхода в свет 29.03.2019. Формат 60x84$^1/_8$. Усл. печ. л. 22.0. Уч.-изд. л. 19,5. Объем 190 с. Тираж 46 экз. Свободная цена. Заказ 003/019. Адрес типографии: ул. Советская, 14, оф. 109, г. Ярославль, 150003 Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия.

ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

P.G. Demidov Yaroslavl State University

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

Volume 26 No 1 (79) 2019

Founded in 1999
4 issues per year

Editorial Office Address: P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru

Contents

Modeling and Analysis of Information Systems. Vol. 26, No 1. 2019

Computer Networks and Communications

От главного редактора

Уважаемые читатели!

Этот номер журнала "Моделирование и анализ информационных систем" содержит расширенные версии избранных докладов, представленных на 2-й Международной научно-технической конференции "Modern Network Technologies 2018"(MoNeTec-2018), которая проходила 25–26 октября 2018 года в Москве, в Сколковском институте науки и технологий (Сколтех). В ней приняли участие представители международного научного сообщества, исследовательских подразделений и корпораций, стартапов, промышленности и бизнеса России, институтов развития и органов государственной власти в сфере компьютерных сетей, виртуализации сетевых ресурсов и облачных вычислений.

Организаторами и спонсорами научного ИТ-форума выступили Сколтех, МГУ им. М.В. Ломоносова, Центр прикладных исследований компьютерных сетей, университет Иннополис, ФГАУ ГНИИ ИТТ "Информика", АО "Концерн «Автоматика»", Объединенный институт ядерных исследований, Институт инженеров электротехники и электроники IEEE и другие.

Конференция была нацелена на совместное обсуждение актуальных вопросов исследования, разработки и внедрения современных телекоммуникационных технологий, основанных на технологии построения современных компьютерных сетей и информационных инфраструктур SDN и NFV. Программно-конфигурируемая сеть – SDN (Software Defined Networking) – сеть передачи данных, в которой уровень управления сетью отделён от устройств передачи данных и реализуется программно. Ключевые принципы программно-конфигурируемых сетей – разделение процессов передачи и управления данными, централизация управления сетью при помощи унифицированных программных средств, виртуализация физических сетевых ресурсов.

Тематика конференции отражала следующие направления:

– построение современных компьютерных сетей;

– виртуализация сервисов в SDN сетях;

– применение современных сетевых технологий;

– организация облачных вычислений;

– применение облачных технологий.

В состав Программного комитета конференции вошли ведущие учёные и разработчики из более чем 10 стран мира.

Руководство Программным комитетом осуществляли:

– Р.Л. Смелянский, чл.-корр. РАН, профессор МГУ им. М.В. Ломоносова (председатель),

– А.П. Кулешов, академик РАН, Президент Сколковского института науки и технологий (сопредседатель),

В настоящее время в мире начато применение на практике ключевых технологий SDN и NFV построения современных компьютерных сетей и информационных инфраструктур в целом, но остаётся ещё много проблем для исследования, разработки и внедрения. Вошедшие в данный номер журнала статьи в той или иной мере отражают эти проблемы.

В. А. Соколов

From the Editor-in-Chief

Dear readers!

This issue of the journal "Modeling and Analysis of Information Systems" contains extended versions of selected reports presented at the 2nd International Scientific and Technical Conference "Modern Network Technologies 2018" (MoNeTec-2018), which was held on October 25–26, 2018 in Moscow, in Skolkovo Institute of Science and Technology (Skoltech). It was attended by representatives of the international scientific community, research units and corporations, start-ups, industry and business in Russia, development institutions and government bodies in the field of computer networks, network resources virtualization and cloud computing.

The organizers and sponsors of the scientific IT forum were Skoltech, Lomonosov Moscow State University, Applied Research Center for Computer Networks, Innopolis University, State Institute of Information Technologies and Telecommunications (SIIT&T "Informika"), Concern "Avtomatika", Joint Institute for Nuclear Research, Institute of Electrical and Electronics Engineers (IEEE) and others.

The conference was aimed at joint discussion of topical issues of research, development and implementation of modern telecommunication technologies based on the technology of building modern computer networks and information infrastructures SDN and NFV. SDN (Software Defined Networking) is a data network in which the network management level is separated from the data transfer devices and implemented programmatically. The key principles of software-configured networks are separation of data transfer and data management, centralization of network management using unified software, virtualization of physical network resources.

Subjects of the conference reflected the following areas:

– construction of modern computer networks;

– service virtualization in SDN;

– applications of modern network technologies;

– organization of cloud computing;

– applications of cloud technologies.

The conference Program Committee includes leading scientists and developers from more than 10 countries of the world. The leadership of the Program Committee was carried out by:

– R.L. Smeliansky, Corresponding Member of the Russian Academy of Sciences, Professor of Lomonosov Moscow State University (Chairman);

– A.P. Kuleshov, Academician of the Russian Academy of Sciences, President of the Skolkovo Institute of Science and Technology (Co-Chair).

Currently, the world has begun to put into practice the key technologies of SDN and NFV for building modern computer networks and information infrastructures in general, but there are still many problems for study, development and implementation. The articles in this issue of the journal reflect to some extent these problems.

Valery A. Sokolov

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 7–22
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 7–22

Компьютерные сети и коммуникации
Computer Networks and Communications

DOI: 10.18255/1818-1015-2019-1-7-22

УДК 517.9

Оркестрация жизненного цикла многопользовательской виртуальной сетевой функции

Антоненко В. А., Смелянский Р. Л., Плакунов А. В., Михеев П. А.

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

Аннотация. Виртуализация сетевых функций (NFV) – перспективная технология предоставления качественного, гибкого и масштабируемого сервиса для клиентов телекоммуникационных компаний и операторов центров обработки данных. Одной из важных возможностей этой технологии является предоставление “сложного” (состоящего из нескольких виртуальных функций) сервиса. Есть два типа виртуальных функций: те, которые ориентированы на работу с конкретным пользователем (далее su-VF); и те, которые используют разные пользователи (далее mu-VF). Если выход mu-VF соединен с входами нескольких su-VF, то возникает необходимость в механизме идентификации и разделения трафика разных пользователей в NFV-инфраструктуре. В облачной среде идентификация пользователей традиционными способами через VLAN теги, IP и MAC-адреса не всегда возможна. В статье рассматривается описанная выше проблема идентификации трафика конкретного пользователя NFV-инфраструктуры, и представлено ее решение, реализованное на MANO-платформе C2.

Ключевые слова: NFV, MANO, цепочки сетевых сервисов, SDN

Для цитирования: Антоненко В. А., Смелянский Р. Л., Плакунов А. В., Михеев П. А., "Оркестрация жизненного цикла многопользовательской виртуальной сетевой функции", *Моделирование и анализ информационных систем*, **26**:1 (2019), 7–22.

Об авторах:
Антоненко Виталий Александрович, канд. физ.-мат. наук, orcid.org/0000-0002-5245-4763
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: anvial@lvk.cs.msu.su

Смелянский Руслан Леонидович, чл.-кор. РАН, д-р физ.-мат. наук, проф., orcid.org/0000-0003-2311-4513
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: smel@cs.msu.su

Плакунов Артем Владимирович, ведущий программист-разработчик, orcid.org/0000-0003-1448-2074
Центр прикладных исследований компьютерных сетей,
Ленинские горы, 1, стр. 77, г. Москва, 119992 Россия, e-mail: aplakunov@arccn.ru

Михеев Павел Алексеевич, программист-разработчик, orcid.org/0000-0002-0934-6935
Центр прикладных исследований компьютерных сетей,
Ленинские горы, 1, стр. 77, г. Москва, 119992 Россия, e-mail: pmikheev@arccn.ru

Благодарности:
Работа выполнена при финансовой поддержке гранта РФФИ № 18-07-01245.

Введение

Для управления облачной инфраструктурой, согласно концепции технологии network function virtualization (NFV), используют специализированные платформы оркестрации виртуальных сервисов, называемые MANO-платформы. MANO-платформа предоставляет возможность управления сервисом индивидуально для каждого пользователя, при помощи порталов самообслуживания. MANO-платформа оперирует понятием сервиса, который представляет собой взаимосвязанное множество виртуальных функций. Структура взаимосвязей обычно представляет собой цепочку, но может быть и более сложной, например, графом.

В данной работе использовано расширенное определение виртуальной функции, включающее в себя не только классические сетевые виртуальные функции (VNF), например, коммутации, маршрутизации, межсетевого экранирования и т.д., но и облачные приложения для ЦОД (VAF). Далее будем использовать общий термин для VNF и VAF — виртуальная функция (VF) [1].

Очевидно, что разные VF предъявляют разные требования к пользовательскому трафику. Например, некоторые VF могут работать только с почтовым трафиком, некоторые только с web-трафиком и т.д. VF могут настраиваться специально под конкретного пользователя, например, позволяя выбирать анализаторы и сигнатуры при конфигурировании IDS, либо иметь единую конфигурацию для трафика всех пользователей.

Во многих случаях нет необходимости создавать отдельный экземпляр VF под каждого пользователя. Это может быть обусловлено экономическими причинами (излишняя трата вычислительных, сетевых и ресурсов хранения) либо технологическими (сервис не является мультитенантным). Например, некоторые сервисы обеспечения информационной безопасности (Anti-DDoS и контроллеры доставки приложений [2], [3]) для корректной работы требуют проверки на едином экземпляре всего входящего в ЦОД трафика. Подобные VF мы будет называть многопользовательскими (mu-VF), так как один и тот же экземпляр VF может быть компонентом сервисов нескольких пользователей одновременно. Соответственно функции, ориентированные на работу с конкретным пользователем, будем называть однопользовательскими (su-VF).

Не следует путать введенные термины (mu-VF, su-VF) с терминами Chain-aware, Chain-unaware [4], так как они используются в контексте принадлежности экземпляра VF к какому-либо сервису и не являются аналогами su-VF и mu-VF.

Актуальным является вопрос управления и оркестрации mu-VF, а также сочетания внутри одного сервиса как mu-VF, так и su-VF. Напомним, что сервисная архитектура NFV является высокоуровневой архитектурой. Существуют подходы как с использованием специализированных протоколов (например, NSH [5]) для реализации сервисной цепочки, так и с использованием стандартного сетевого стека протоколов (например, SFC [6]).

Возможность идентифицировать экземпляры VF является необходимой для развертывания гибкого и динамически масштабируемого e2e-сервиса в сетях поколения 5G [7]. Подобная идентификация, которая не зависит от расположения экземпляра сервиса, требуется для поддержки балансировки нагрузки между элементами

инфраструктуры в течение всего жизненного цикла VF или же в случае непредвиденного сбоя в работе VF.

Процесс пересылки пакетов в сетях 5G имеет решающее значение для работы под нагрузкой многих подсистем телекоммуникационных сетей, таких как EPC [8] и RAN [9].

Трафик разных пользователей мы можем отличать по информации в сетевом заголовке. Однако проблема заключается в том, что мы не можем заранее знать, какой из уровней L2, L3, L4—L7, будет использован в VF. Для решения данной проблемы в статье предложено и рассмотрено решение на основе SDN подхода, названное Cube и реализованное на платформе C2 [1].

Структура статьи следующая: в разделе 1 рассматриваются существующие MANO-платформы и их видение относительно mu-VF, а также способы организации цепочек функций. В разделе 2 ставятся задачи, которые должен выполнять модуль Cube, и описывается его архитектура. В разделе 3 проводится функциональное и нагрузочное тестирование модуля Cube.

1. Схожие работы

Проблема построения цепочек функций в основном сосредоточена на подходе к построению цепочки VF и маршрутизации соответствующего трафика через данные VF в правильном порядке.

Эталонная архитектура MANO-платформы (ETSI [10]) определяет множество абстракций и интерфейсов их взаимодействия, например, *сервис*, *виртуальная функция (VF)*, *политика оркестрации*, *восстановление и масштабирование VF*, *шаблонное описание VF* и т.д. Однако реализация многих из этих абстракций и интерфейсов различается как в коммерческих, так и в открытых реализациях MANO-платформ.

Авторам не известна ни одна MANO-платформа, поддерживающая механизм формирования виртуального сервиса, который позволил бы идентифицировать трафик конкретного пользователя на каждом этапе его прохождения через компоненты виртуального сервиса, состоящие из комбинации mu-VF и su-VF. Одной из причин отсутствия механизма идентификации трафика пользователя у MANO-платформ является то, что они разрабатывались либо для операторов корпоративных ЦОД (например, Cloudify [11]), либо для телекоммуникационных компаний (например, Nokia Cloudband [12]). В первом случае [11] все VF являются однопользовательскими, то есть экземпляр VF создается на каждый пользовательский запрос на предоставление сервиса. Во втором случае [12] все VF являются многопользовательскими, то есть оператор MANO-платформы заранее определяет, какой сервис будет предоставлен каждому из пользователей, и в ручном режиме настраивает маршрутизацию между экземплярами VF.

Решение проблемы идентификации и разделения трафика пользователя в NFV-инфраструктуре, как правило, реализовано в функции маршрутизации трафика между VF, входящими в состав сервисной цепочки (Service Forwarding Function — SFF или Virtual Routing Function — VRF). Можно выделить несколько основных подходов к реализации этой функции.

Все эти подходы основаны на инкапсуляции метаданных в реализацию цепочки VF (далее просто *метаданные*). *Метаданные* предоставляют контекстную информацию о пакетах, которые проходят через сервисную функциональную цепочку (SFC). *Метаданные* могут использоваться для транспортировки контекстной информации, которая доступна в одном месте в сети, в другое место в сети, где эта информация недоступна [13].

Сетевые мосты (virtual bridges) должны иметь подробную информацию о каждом потоке трафика, такую как ID абонента и связанные классификаторы (то есть наборы правил, которые должны применяться к потоку трафика).

Существует два основных метода для управления трафиком в SFC: на основе заголовка и на основе тегов. Эти методы позволяют пакетам переносить информацию о маршруте и функциях, необходимых для пользователя. Основанные на заголовке подходы обычно определяют формат заголовка, в то время как основанные на тегах кодируют теги в доступные поля заголовков пакетов.

Примерами основанных на заголовке методов являются заголовок сетевой службы (NSH) [5], заголовок цепочки услуг (SCH) [14] и подход, основанный на сегментной маршрутизации [15]. Суть этого подхода заключается в построении цепочек VF путем маршрутизации от источника, где необходимые данные передаются с использованием специального протокола (SRH) внутри специального добавленного заголовка этого протокола, называемого service-header. Недостатком подходов на основе заголовков является дополнительная служебная нагрузка [16].

Перейдем к рассмотрению методов управления трафиком на основе тегов. Эти методы основаны либо на кодировании определенных тегов в доступных полях, которые обычно определяются большинством протоколов как поля *метаданных* (например, инкапсуляция VLAN / VXLAN), либо на использовании существующих полей пакета, таких как MAC-адрес. Например, в статье [17] предлагается способ повышения масштабируемости за счет использования MAC-адреса источника в качестве идентификатора цепочки услуг. Другим примером является проект FlowTag [18], который предлагает расширенную архитектуру SDN и включает в себя специализированные сетевые устройства с возможностью добавлять теги к исходящим пакетам. FlowTag соответствует идентификаторам VLAN.

В другой работе применяется метод абстракций на уровне приложений (L4–L7) [19]. Технически этот метод наиболее простой: идея состоит в том, что цепочки сервисов ставятся в соответствие потокам трафика. Например, поток пользовательского трафика, который должен пройти через цепочку VF, сопоставляется с номером TCP сессии, относящимся к конкретному приложению.

Также можно использовать протокол Multi-Protocol Label Switching (MPLS) [20]. MPLS позволяет сопоставить сетевому домену метку фиксированной длины. Как только пакет попадает в домен, ему назначается данная метка, которую можно использовать для маршрутизации.

Значительное преимущество подхода с использованием тегов — это отсутствие дополнительных требований для поддержки специальных протоколов (NSH, SCH, SMH и др.). Однако этот подход требует, чтобы MANO-платформа работала в SDN окружении: каждый физический и виртуальный коммутатор должен поддерживать концепцию SDN и работать под управлением SDN контроллера. Таким образом, MANO-платформа обязана включать в себя SDN контроллер.

В MANO-платформе C2 используется третий подход (L2 / L3) с модификациями, необходимыми для поддержки виртуального сервиса, состоящего из комбинации mu-VF и su-VF. Подробное описание представлено в разделе 2.

2. Предлагаемый подход

2.1. Цепочки функций в платформе C2

На рисунке 1 показано соответствие архитектуры платформы C2 эталонной модели ETSI MANO. В работе [21] подробно описана архитектура MANO-платформы C2 и задачи каждого её модуля. В MANO-платформе C2 роль менеджера виртуальной инфраструктуры (VIM) выполняет OpenStack [22]. Изоляция сетей в OpenStack реализуется с помощью VLAN и VXLAN тегов. Интерфейс для работы с OpenStack реализован в модуле C2-Core.

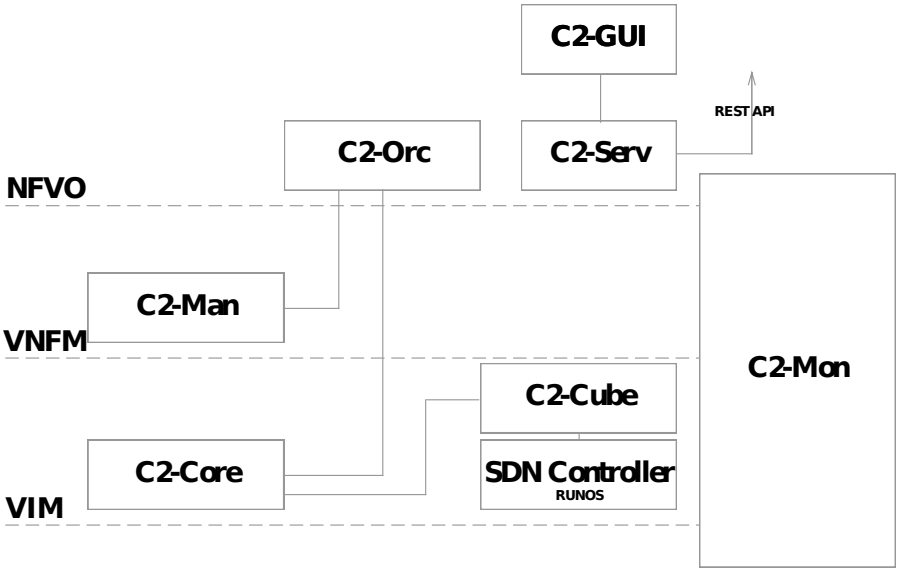


Рис. 1. Архитектура MANO-платформы C2
Fig. 1. Architecture of C2 MANO-platform

В платформе C2 для построения цепочек VF используется библиотека networking-SFC [6]) — это расширение OpenStack ML2 плагина для openvswitch (OVS) [23]. Согласно классификации, введенной в предыдущем разделе, принцип его работы основан на использовании MAC-адреса в качестве идентификатора цепочки услуг. В точке начала цепочки и на выходном порте каждой VF подставляется MAC-адрес следующей VF, а для сохранения принадлежности пакетов к цепочке есть возможность использовать MPLS метки или протокол NSH.

В ходе подготовки VF на основе приложений от сторонних разработчиков было обнаружено, что для VF, реализующих любое из рассмотренных нами приложений, верны следующие тезисы:

1. VF являются chain-unaware;

2. VF работают с фиксированным числом сетевых портов (обычно не более 3). Создание новых сетевых портов у VF для обработки трафика в ходе её работы нежелательно или невозможно;

3. VF могут иметь или не иметь свойство *L2-transparent*. Это означает, что VF работает как обычный сетевой bridge: её интерфейсы не имеют IP-адреса и не меняют заголовки протоколов уровня 2 и выше.

Далее по тексту, если не указано иное, под VF будем понимать экземпляр VF. Экземпляр VF — это конкретная реализация VF по запросу пользователя. Термин mu-VF означает, что один экземпляр используется несколькими пользователями.

Проблема возникает, когда услуга представляет собой смесь su-VF и mu-VF. Networking-SFC классифицирует пользовательский трафик на выходном порте VF, однако невозможно различить потоки трафика, принадлежащие разным пользователям, после потоков, прошедших через mu-VF. На рисунке 2 представлена схема подсоединения mu-VF.

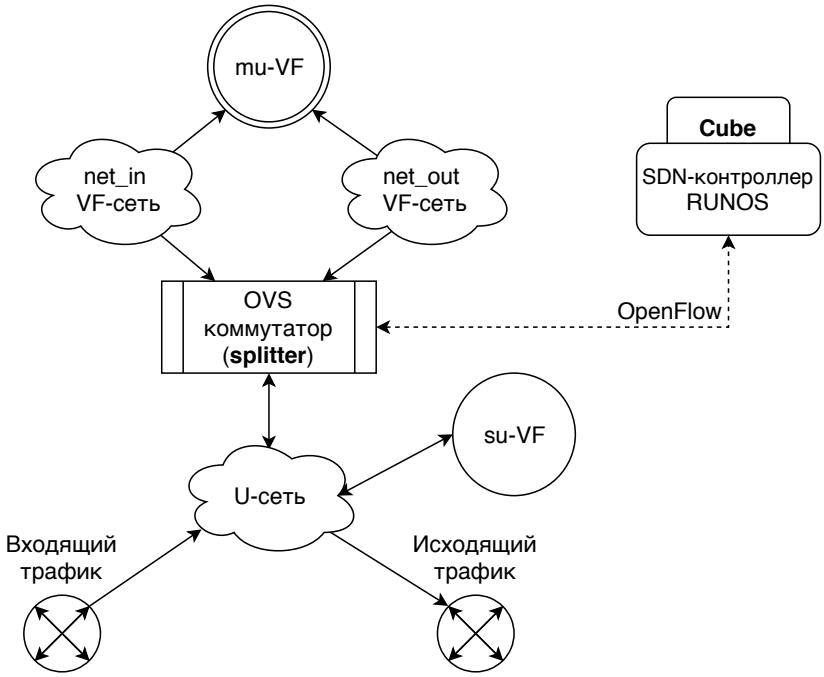


Рис. 2. Схема подсоединения mu-VF
Fig. 2 The mu-VF network connection scheme

MANO-платформа C2 работает с двумя типами виртуальных сетей: пользовательская сеть (U-сеть) и VF-сеть. U-сеть — это точка обслуживания (PoP). Каждый su-VF в сервисной цепочке подключается напрямую к U-сети, тогда как mu-VF (см. 2) требует дополнительного специализированного сетевого моста, чтобы различать потоки трафика от разных пользователей. Это означает, что mu-VF должен иметь свои собственные сети (VF-сети), где будут накапливаться потоки трафика входящих и выходящих пользователей (см. рисунок 2). В итоге проблема в том, как реализовать виртуальный мост между VF-сетью и U-сетью.

В данной статье предлагается решение данной проблемы при помощи специализированного виртуального сетевого моста. Такой виртуальный мост должен передавать трафик между двумя сетями при адаптации к свойствам VF и распределять поток трафика между несколькими пользователями. Эти задачи выполняются модулем C2-Cube (далее Cube).

Для корректной работы модуля Cube потребуется дополнительное ограничение: в точке разделения трафика между пользователями каждый пользователь должен идентифицироваться уникальным IP-адресом. Достичь этого можно, либо выделяя уникальную подсеть для каждого виртуального сегмента сети, либо добавляя в цепочку перед mu-VF функцию, выполняющую трансляцию адресов (NAT).

2.2. Архитектура модуля Cube

Cube был разработан как приложение для SDN контроллера RUNOS [24]. В платформе данный модуль занимается управлением устройств openvswitch по протоколу OpenFlow [25]. Стоит отметить, что модуль Cube не берет на себя функцию полного управления виртуальными сетями всей MANO-платформы, а управляет только виртуальными устройствами, осуществляющими манипуляции с трафиком, отличные от задачи маршрутизации и организации цепочек. Устройство openvswitch, контролируемое Cube (рисунок 2), назовем splitter. Схема портов splitter представлена на рисунке 3.

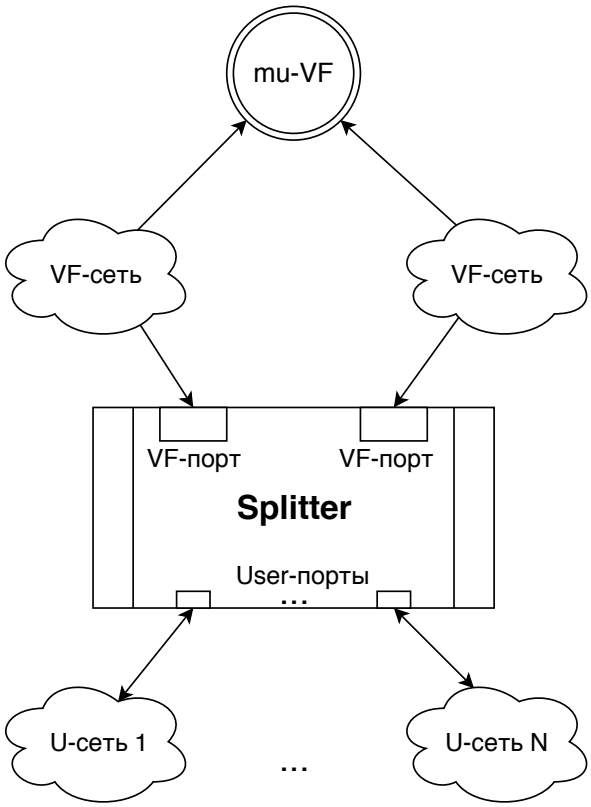


Рис. 3. Схема портов splitter
Fig. 3 Splitter port scheme

VF может обрабатывать трафик по-разному в зависимости от направления его передачи (вход трафика в VF или выход трафика из VF). Например, при использовании функции NAT, при передаче из внутренней сети подставляется IP-адрес функции NAT, а при передаче из внешней — восстанавливает IP-адрес получателя. Для реализации такой зависимости от направления в splitter используется тройка:

$$< Port_{ing}, TPP, Port_{eng} >,$$

где

- TPP (Traffic Processing Paradigm) описывает действие, которое должно быть применено к трафику, проходящему из $Port_{ing}$ в $Port_{eng}$. После прохождения трафика через VS и возвращения его на splitter, его необходимо перенаправить во второй порт пары. Максимально может быть $N = u * K$ таких пар, где u — количество пользователей, K — количество TPP.

- TPP должна явно задавать наличие или отсутствие свойства L2-transparent. Отсутствие свойства означает, что MAC адрес получателя в пакете должен совпадать с MAC-адресом интерфейса VF.

- В случае отсутствия свойства L2-transparent информация о MAC-адресах должна быть доступна в Cube. Свойство L2-transparent явно задается в шаблоне VF (в MANO-платформе C2 — это TOSCA-шаблон).

Основную задачу, которую выполняет устройство splitter — для каждой пары портов конфигурации ($Port_{ing}$, $Port_{eng}$) запоминать трафик, проходящий через данную пару, чтобы после получения из VF единого потока трафика суметь разделить его на несколько подпотоков, отправив каждый на соответствующий порт устройства splitter.

Простейший способ сохранить ассоциацию между портами и трафиком — это разметка трафика с помощью тегов: VLAN, VXLAN, MPLS. Однако это подразумевает, что VF при приеме размеченного трафика способна у каждого пакета снять метку на входе и поставить ее же заново при выходе из VF. Однако это требование весьма специфично и не реализуется ни в одной из рассмотренных нами VF. Поэтому было принято решение не использовать теги, а получать всю информацию о пользователе из заголовков пакетов, относящихся к каждому потоку трафика.

В данном подходе splitter сохраняет шаблон каждой TCP, UDP или ICMP сессии, проходящей через порт. Для этого используются следующие поля:

$$< Ip_{src}, Ip_{dst}, [tcp|udp]Port_{src}, [tcp|udp]Port_{dst} > \qquad (1)$$

Алгоритм работы splitter следующий:

- Пакет приходит на один из User-портов splitter. Splitter ищет соответствие заголовку пакета, исходя из кортежа (1):

  - Если в таблице потоков splitter есть подходящее правило OpenFlow, пакет отправляется на VF с изменением MAC-адреса получателя (если VF не имеет свойства L2-transparent).

  - Если подходящего правила на splitter нет, то Cube добавляет его с временем жизни 60 секунд. После этого пакет отправляется в соответствии с добавленными правилами.

- Пакет приходит на VF-порт устройства splitter. Splitter ищет соответствие заголовку пакета, исходя из кортежа (1):

  - Если в таблице потоков splitter есть подходящее правило OpenFlow, оно уникально определяет, на какой порт следует отправить пакет.
  - Если подходящего правила на splitter нет, Cube попытается найти наиболее подходящий заголовок в списке сохраненных заголовков. Достаточно найти заголовок с совпадающим IP-адресом источника или получателя и одинаковым TCP/UDP портом источника или получателя. Найденный заголовок однозначно определяет, в какой User-порт следует отправить пакет. Если заголовок не найден, пакет сбрасывается.

3. Экспериментальное исследование

В этом разделе приводится описание результатов функционального тестирования и тестирования производительности. Цель этого эксперимента — продемонстрировать рабочий вариант использования предложенного подхода и оценить задержки потоков трафика пользователя, вызванные модулем Cube.

Все исследования в данном разделе проводились на стойке из четырех серверов, на которых был развернут OpenStack версии Mitaka на ОС Ubuntu 14.04.5 LTS. Один из серверов выполнял роль controller и network узла, остальные были compute узлами. Серверы соединены между собой каналами пропускной способности 10 Гбит через один L3 коммутатор. Характеристики серверов и коммутатора показаны в таблице 1.

Таблица 1. Характеристики оборудования стенда
Table 1. Testbed Hardware Characteristics

Тип оборудования	**Характеристики оборудования**
Серверы	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, 32 vCPU, 64GB RAM
Коммутатор	Dell Networking S4810 10/40GbE

3.1. Функциональное тестирование

В данном разделе будет показано, как Cube работает на практике. Пусть у пользователя А есть виртуальная машина в облаке. Выдадим этой машине доступ в интернет через VNF SNAT и защитим её с помощью VNF Anti-DDoS. Пользователь Б закажет VAF типа “apache веб-сервер”, которая также защищена функцией Anti-DDoS.

Пользователь В закажет ещё одну VAF типа “веб-сервер” и защитит его функциями Web Application Firewall (WAF) и Anti-DDoS. В данном примере SNAT и WAF — это su-VF, а Anti-DDoS — это mu-VF. Рассматриваемый локальный сегмент сети и используемые IP-адреса показаны на рисунке 4. В качестве Anti-DDoS использовалась VNF от компании БИФИТ [26], а в качестве WAF — VNF от компании Positive Technologies [27].

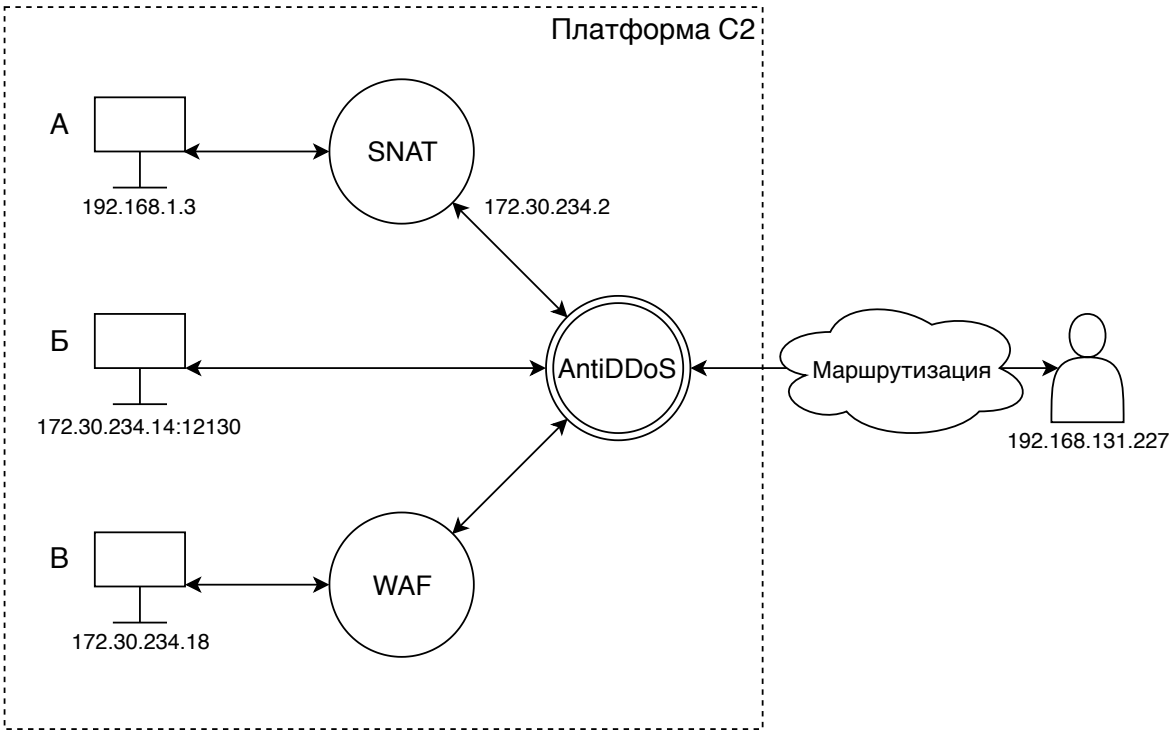


Рис. 4. Схема сетевого взаимодействия в эксперименте
Fig. 4. Network Interaction Scheme in the Experiment

Выполним следующие действия для создания сетевой активности для симуляции прохождения трафика через mu-VF:

1. на виртуальной машине пользователя А выполним команду “ping 8.8.8.8”;
2. на VAF пользователя Б обратимся с HTTP запросом;
3. с VAF пользователя В установим SSH соединение.

Рассмотрим, как работает *splitter* для функции Anti-DDoS. Изначально на *splitter* нет правил, подходящих для приходящих пакетов, поэтому первый пакет сессии будет отправлен в Cube. После того, как Cube добавит необходимые правила на *splitter*, последующие пакеты той же сессии смогут проходить без участия Cube.

Listing 1. OpenFlow правила для пользователя A

```
icmp,in_port=1,dl_dst=fa:16:3e:36:23:d0,nw_src=172.30.234.2,
        nw_dst=8.8.8.8 actions=output:2
icmp,in_port=4,nw_src=172.30.234.2,
        nw_dst=8.8.8.8 actions=output:3
icmp,in_port=3,dl_dst=fa:16:3e:c5:b1:ec,nw_src=8.8.8.8,
        nw_dst=172.30.234.2 actions=output:4
icmp,in_port=2,nw_src=8.8.8.8,
        nw_dst=172.30.234.2 actions=output:1
```

В данном примере порты с номерами 2 и 4 соединены с двумя портами VNF, остальные порты соединены с сетями разных пользователей: номера портов 1, 3 принадлежат пользователю A, номера портов 7, 8 принадлежат пользователю Б, номера портов 5, 6 принадлежат пользователю B. В листинге 1 показаны правила *splitter*, соответствующие сессии пользователя A, где *in_port* — номер порта *splitter*, куда приходит трафик, *dl_dst* — MAC-адрес получателя (так как используется networking-sfc, это MAC-адрес порта splitter), *nw_src* — IP-адрес отправителя, *nw_dst* — IP-адрес получателя, output: N — действие отправки пакета в порт с номером N.

Первые два правила соответствуют ICMP запросу от виртуальной машины. Так как пакет уже прошел SNAT, IP-адрес отправителя изменен. ICMP запрос попадает на порт 1 *splitter* и отправляется через порт 2, который соединен с VF. После прохождения VF пакет возвращается на порт 4 *splitter* и отправляется через порт 3. Так как в данном случае *splitter* — это конец цепочки, далее пакет следует по назначению.

Вторые два правила соответствуют ICMP ответу от удаленного хоста. Из правил видно, что путь пакета симметричен.

В листинге 2 показаны правила splitter, соответствующие сессиям пользователей Б и B.

Listing 2. OpenFlow правила для пользователей Б и B

```
tcp,in_port=7,dl_dst=fa:16:3e:a4:b0:88,nw_src=192.168.131.227,
        nw_dst=172.30.234.14,tp_src=33260,
        tp_dst=12130 actions=output:4
tcp,in_port=2,nw_src=192.168.131.227,nw_dst=172.30.234.14,
        tp_src=33260,tp_dst=12130 actions=output:8
tcp,in_port=8,dl_dst=fa:16:3e:f5:33:e2,nw_src=172.30.234.14,
        nw_dst=192.168.131.227,tp_src=12130,
        tp_dst=33260 actions=output:2
tcp,in_port=4,nw_src=172.30.234.14,nw_dst=192.168.131.227,
        tp_src=12130,tp_dst=33260 actions=output:7

tcp,in_port=6,dl_dst=fa:16:3e:35:ec:f1,nw_src=192.168.131.227,
        nw_dst=172.30.234.18,tp_src=34386,
        tp_dst=22 actions=output:4
tcp,in_port=2,nw_src=192.168.131.227,nw_dst=172.30.234.18,
        tp_src=34386,tp_dst=22 actions=output:5
tcp,in_port=5,dl_dst=fa:16:3e:13:8d:08,nw_src=172.30.234.18,
        nw_dst=192.168.131.227,tp_src=22,
        tp_dst=34386 actions=output:2
tcp,in_port=4,nw_src=172.30.234.18,nw_dst=192.168.131.227,
        tp_src=22,tp_dst=34386 actions=output:6
```

В листинге 2 *tp_src* — это TCP порт отправителя, *tp_dst* — TCP порт получателя. Видно, что TCP трафик на порты 22 (ssh) и 12130 (выбранный порт для веб-сервера) и предназначающийся виртуальным машинам пользователей Б и B отправляется на один и тот же экземпляр, что и трафик пользователя A. Отметим, что пользователям не требуется знать тип функции. Для них работа с mu-VF и su-VF выглядит полностью одинаково.

3.2. Временные задержки и пропускная способность

В данном разделе проводится исследование, как наличие *splitter* в цепочке VF влияет на задержку пакетов в сети и пропускную способность сети. В качестве mu-VF будет использоваться фиктивная виртуальная машина, не выполняющая никаких действий над трафиком, кроме пересылки на выходной порт.

Для исследования влияния Cube на задержку пакетов создадим две виртуальные машины. Одна из них будет отправлять по 100 ICMP пакетов размером от 44 до 1500 байт с шагом 10 байт другой машине. Задержка будет высчитываться как среднее время задержки для 100 пакетов.

График зависимости задержки пакетов от их размера для mu-VF (с использованием Cube) и для su-VF (без использования Cube) показан на рисунке 5.

Основное отличие заключается в том, что при использовании Cube наблюдается повышенное отклонение между минимальным средним и максимальным средним RTT: 0,25 мс по сравнению с 0,1 мс. Средняя разница между графиками для всех экспериментов составляет 0,03 мс в пользу Cube. Обратите внимание, что эксперименты проводились в локальной сети. Таким образом, можно сделать вывод, что влияние Cube на задержку пакетов в глобальном масштабе незначительно и не зависит от размера пакета.

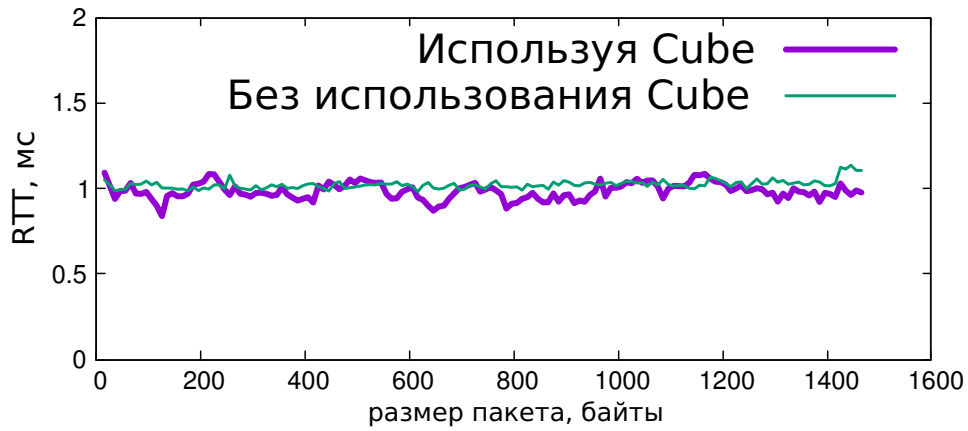


Рис. 5. Влияние Cube на задержку пакетов
Fig. 5. Cube impact on a packet delay

Перед исследованием влияния Cube на пропускную способность сети выясним, какую пропускную способность обеспечивает инфраструктура стенда. Этот шаг необходим, так как в стенде не используется технология Intel DPDK [28] и, как известно, в этом случае производительность может значительно отличаться от оптимальной [29], [30].

Тестирование проводилось с помощью утилиты *iperf* по протоколу TCP. Результаты показаны в таблице 2.

Предположительно дополнительное падение пропускной способности при использовании цепочек связано с тем, что текущая реализация использует MPLS теги.

В эксперименте создадим 32 тенанта, в каждом из которых будут находиться две виртуальные машины, одна из которых — это *iperf*-клиент, вторая — *iperf*-сервер.

Таблица 2. Максимальная пропускная способность инфраструктуры стенда
Table 2. Maximum Throughput of Testbed Infrastructure

Эксперимент	**Пропускная способность**
От физического сервера до физического сервера	9.39 Гбит/с
От виртуальной машины до виртуальной машины на разных серверах	4.05 Гбит/с
От виртуальной машины до виртуальной машины на разных серверах с использованием цепочек функций	1.46 Гбит/с

Для тестирования Cube направим весь трафик между каждой парой клиент—сервер через одну mu-VF. Расположение виртуальных машин по серверам следующее:

- все *iperf*-клиенты расположены на сервере 1;
- все *iperf*-серверы расположены на сервере 2;
- VF и OVS под управлением Cube расположены на узле 3.

В одно и то же время N *iperf*-клиентов открывали соединение к своему *iperf*-серверу, где N варьировалось от 1 до 32. Пропускная способность соединений ограничивалась 300 Мбит/с, что соответствует равному разделению всей полосы пропускания физического канала между 32 пользователями. Отметим, что результаты, приведенные в таблице 2, показывают, что суммарная пропускная способность всех соединений через VF ограничена 1.46 Гбит/с. Это означает, что нам не удастся достичь результата, при котором полоса пропускания для каждого соединения равна 300 Мбит/с. Вместо этого ожидаемый результат — равное разделение полосы пропускания в 1.46 Гбит/с между 32 соединениями. Для запуска использовалась утилита “parallel” [31].

На рисунке 6 представлены измеренные пропускные способности в зависимости от количества одновременно запущенных *iperf*-соединений. Разными линиями показаны максимальная, минимальная, средняя и суммарная измеренные пропускные способности. Отметим, что вертикальная ось показана в логарифмической шкале.

Из графиков видно, что результат эксперимента близок к ожидаемому: сумма всех пропускных способностей клиентов варьируется в пределах 1.3—1.35 Гбит/с. Это на 7—10% меньше, чем максимум. За неимением альтернатив для сравнения, мы находим такой результат приемлемым. На графике также показаны минимальная и максимальная пропускные способности для каждого количества соединений. В процентном соотношении разница между минимальной и максимальной пропускной способностью не меняется значительно после того, как суммарная пропускная способность достигает максимума. Минимальная пропускная способность составляет 80—90% от максимальной.

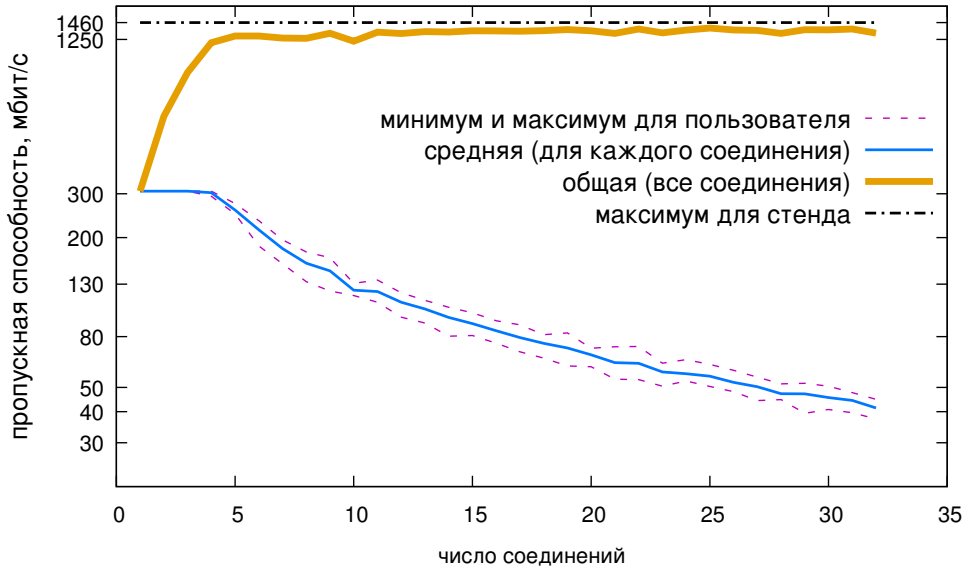


Рис. 6. Результаты эксперимента с пропускной способностью
Fig. 6. The Results of the throughput experiment

4. Заключение

В этой статье мы предлагаем способ работы (передача трафика) цепи VF, которая представляет собой смесь mu-VF и su-VF. Для правильного управления пользовательским трафиком в смешанной цепочке VF поток пользовательского трафика должен быть дифференцирован. Чтобы решить эту проблему дифференциации, мы предложили и рассмотрели решение на основе подхода SDN. Это решение было реализовано как модуль Cube в MANO-платформе C2 [1].

Это решение имеет существенное преимущество — повышение эффективности использования ресурсов, управляемых облачной платформой, путем совместного использования mu-VF и su-VF как части сервисной цепочки VF.

Эффективность использования сетевых ресурсов и накладные расходы были оценены экспериментально. Результаты экспериментов демонстрируют эффективное использование пропускной способности и низкие задержки дополнительных сетевых пакетов. Мы полагаем, что продемонстрированных результатов достаточно для построения реальных примеров использования VF. Например, можно создать веб-сервер, на котором размещаются сайты для нескольких пользователей, и создать отдельную цепочку услуг для каждого пользователя, состоящую из su-VF и mu-VF.

Список литературы / References

[1] Antonenko V., Smeliansky R., Ermilov A., Plakunov A., Pinaeva N., Romanov A., “C2: General purpose cloud platform with NFV life-cycle management”, *2017 IEEE 9th International Conference on Cloud Computing Technology and Science*, 2017, 353–356, https://doi.org/10.1109/CloudCom.2017.57.

[2] *Radware and Intel – Virtualizing Application Delivery Controllers in an NFV Environment*, 2015, https://networkbuilders.intel.com/docs/Radware_Solution_Brief_Final.pdf, lastaccessed Feb. 19, 2018.

[3] Alharbi T., et al., "Smart and Lightweight DDoS Detection Using NFV", *Proceedings of the International Conference on Compute and Data Analysis, ICCDA 2017* (Lakeland, FL, USA, May 19–23, 2017), 2017, 220–227, https://doi.org/10.1145/3093241.3093253.

[4] Halpern J., and Pignataro C., *RFC 7665 – Service Function Chaining (SFC) Architecture*, 2015, https://tools.ietf.org/html/rfc7665, lastaccessed Feb. 19, 2018.

[5] Quinn P., and Elzur U., and Pignataro C., *Network Service Header (NSH)*, 2017, https://tools.ietf.org/html/draft-ietf-sfc-nsh-28, lastaccessed Feb. 19, 2018.

[6] *OpenStack Docs: Service Function Chaining*, 2018, https://docs.openstack.org/newton/networking-guide/config-sfc.html, lastaccessed Feb. 19, 2018.

[7] *Network Functions Virtualisation – White Paper on NFV priorities for 5G*, 2017, https://portal.etsi.org/nfv/nfv_white_paper_5g.pdf, lastaccessed May 15, 2018.

[8] Skulysh M., and Klimovych O., "Approach to virtualization of evolved packet core network functions", *CADSM*, 2015, 193–195, https://doi.org/10.1109/CADSM.2015.7230833.

[9] Mijumbi R., et al., "Server placement and assignment in virtualized radio access networks", *CNSM*, IEEE Computer Society, 2015, 398–401, https://doi.org/10.1109/CNSM.2015.7367390.

[10] *Network Functions Virtualisation (NFV); Management and Orchestration*, 2014, http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf, lastaccessed Feb. 19, 2018.

[11] *Cloud & NFV Orchestration Based on TOSCA*, 2018, https://cloudify.co, lastaccessed Feb. 19, 2018.

[12] *CloudBand | Nokia*, 2018, https://networks.nokia.com/products/cloudband, lastaccessed Feb. 19, 2018.

[13] Quinn P., and Nadeau T., *Problem Statement for Service Function Chaining*, 2015, https://tools.ietf.org/html/rfc7498#section-3.3, lastaccessed May 15, 2018.

[14] Zhang H., et al., *Service Chain Header*, 2014, https://tools.ietf.org/pdf/draft-zhang-sfc-sch-00.pdf, lastaccessed May 15, 2018.

[15] Abdelsalam A., et al., "Implementation of virtual network function chaining through segment routing in a linux-based NFV infrastructure", *IEEE Conference on Network Softwarization, NetSoft 2017* (Bologna, Italy, July 3–7, 2017), 2017, 1–5, https://doi.org/10.1109/NETSOFT.2017.8004208.

[16] Song H., et al., *SFC Header Mapping for Legacy SF*, 2017, https://tools.ietf.org/pdf/draft-song-sfc-legacy-sf-mapping-08.pdf, lastaccessed May 15, 2018.

[17] Ding W., et al., "OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV", *IEEE Network*, **29**:3 (2015), 30–35.

[18] Fayazbakhsh S.K., et al., "FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions", *HotSDN*, 2013, 19–24.

[19] Zhang C., et al., *L4–L7 Service Function Chaining Solution Archetecture*, 2015, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/L4-L7_Service_Function_Chaining_Solution_Architecture.pdf, lastaccessed Feb. 19, 2018.

[20] Misra S., Goswami S., *Routing and MPLS Traffic Engineering, Network Routing: Fundamentals, Applications, and Emerging Technologies*, Wiley Telecom, 2014, https://doi.org/10.1002/9781119114864.ch5.

[21] Antonenko V., Smeliansky R., Ermilov A., Romanov A., Pinaeva N., Plakunov A., "Cloud Infrastructure For Researchers basing on NFV Management and Orchestration", *Proceedings of the XXVI International Symposium on Nuclear Electronics & Computing (NEC 2017)* (Becici, Budva, Montenegro, September 25 – 29, 2017), 2017.

[22] "OpenStack is open source software for creating private and public clouds", 2018, https://www.openstack.org, lastaccessed Feb. 19, 2018.

[23] *Neutron/ML2 – OpenStack*, 2018, https://wiki.openstack.org/wiki/Neutron/ML2, lastaccessed Feb. 19, 2018.

[24] Shalimov A., Nizovtsev S., Morkovnik D., Smeliansky R., "The Runos OpenFlow Controller", *2015 Fourth European Workshop on Software Defined Networks*, IEEE Computer Society, 2015, 103–104, https://ieeexplore.ieee.org/document/7313624.

[25] *Open Datapath Standardized Switch Protocol in Software Defined Network (SDN)*, 2018, https://www.opennetworking.org/projects/open-datapath, lastaccessed Feb. 19, 2018.

[26] *BIFIT mitigator*, 2018, http://www.mitigator.ru, lastaccessed May 15, 2018.

[27] *PT AF – Web Application Firewall (WAF) – Web App Security Solution*, 2018, https://www.ptsecurity.com/ww-en/products/af, lastaccessed Feb. 19, 2018.

[28] *Data Plane Development Kit (DPDK)*, 2018, https://dpdk.org, lastaccessed Feb. 19, 2018.

[29] Jardin V., *High Performance NFV Infrastructure (NFVI): DPDK Host Applications with Neuron/OpenStack and VNF Acceleration*, 2014, https://events.static.linuxfound.org/sites/events/files/slides/Openstack-v4_0.pdf, lastaccessed Feb. 19, 2018.

[30] Wu X., et al., *Understanding the Performance of DPDK as a Computer Architect*, 2016, https://dpdksummit.com/Archive/pdf/2016USA/Day02-Session03-PeilongLi-DPDKUSASummit2016.pdf, lastaccessed Feb. 19, 2018.

[31] Tange O., "GNU Parallel: The Command-Line Power Tool", *;login: The USENIX Magazine*, **36**:1 (2011), 42–47.

Antonenko V. A., Smeliansky R. L., Plakunov A. V., Mikheev P. A., "Shared Virtual Function Orchestration Technique", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 7–22.

DOI: 10.18255/1818-1015-2019-1-7-22

Abstract. Network function virtualization (NFV) is a promising technique of high quality, flexible and scalable service for telecommunication companies clients and for enterprise data center clients. One of the important capabilities of this technique is providing a virtual service as a combination of multiple virtual functions. There are two types of virtual functions: those intended for a single customer (su-VF) and those that can serve multiple users (mu-VF). In case when output of mu-VF is chained with inputs of several different su-VFs, there is a need for a mechanism of identification and separation of users network flows passing through mu-VF to allocate them correctly between inputs of su-VFs in the NFV infrastructure. In the cloud environment, it is not always possible to use VLAN tags, IP and MAC addresses for that. In this paper, we consider the problem of identification of network traffic coming from a certain user inside an NFV platform and present a solution implemented in C2 MANO-platform.

Keywords: NFV, MANO, service chaining, SDN

On the authors:
Vitaly A. Antonenko, PhD, orcid.org/0000-0002-5245-4763
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: anvial@lvk.cs.msu.su

Ruslan L. Smeliansky, orcid.org/0000-0003-2311-4513,
Corresponding Member of Russian Academy of Sciences, professor, doctor of sciences,
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: smel@cs.msu.su

Artem V. Plakunov, senior software developer, orcid.org/0000-0003-1448-2074
Applied Research Center for Computer Networks,
1, bd. 77 Leninskie Gory, Moscow, 119992 Russia, e-mail: aplakunov@arccn.ru

Pavel A. Mikheev, software developer, orcid.org/0000-0002-0934-6935
Applied Research Center for Computer Networks
1, bd. 77 Leninskie Gory, Moscow, 119992 Russia, e-mail: pmikheev@arccn.ru

Acknowledgments:
This work was supported by the Russian Foundation for Basic Research, Grant No 18-07-01245.

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 23–38
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 23–38

DOI: 10.18255/1818-1015-2019-1-23-38

УДК 517.9

Построение бортовых сетей реального времени на основе технологии ПКС

Балашов В. В., Костенко В. А., Ермакова Т. И.

Поступила в редакцию 10 января 2019
После доработки 12 февраля 2019
Принята к публикации 15 февраля 2019

Аннотация. В интегрированных модульных комплексах бортового оборудования (КБО) используются коммутируемые сети AFDX и FC-AE-ASM-RT, реализующие основанный на виртуальных каналах подход к передаче данных в реальном времени. Основным недостатком этих сетей являются ограниченные или отсутствующие возможности динамической реконфигурации виртуальных каналов, что приводит к невозможности динамического формирования режимов функционирования КБО, в частности при множественных отказах оборудования. Для снятия выявленных ограничений в данной работе предложен подход к использованию программно-конфигурируемых сетей (ПКС) для построения бортовых сетей реального времени. Предложенный подход основан на реализации в сети ПКС, поддерживающей протокол OpenFlow1.3, механизма виртуальных каналов, аналогичного используемому в сетях AFDX и FC-AE-ASM-RT. Подход реализован в виде функционального прототипа и экспериментально апробирован в виртуальной сетевой среде, основанной на программных ПКС-коммутаторах Ofsoftswitch13 и сетевом контроллере RUNOS. Эксперименты показали, что предложенная схема передачи данных позволяет передавать сообщения с соблюдением заданных ограничений на задержку и джиттер, а также не допускает превышения ограничения на пропускную способность виртуального канала. Эксперименты также подтвердили, что динамическая реконфигурация виртуальных каналов в ПКС не нарушает передачу данных по не изменяемым виртуальным каналам. Важным направлением дальнейших исследований является разработка алгоритмов динамического формирования новых маршрутов виртуальных каналов в процессе реконфигурации КБО. Конечной целью работ является создание на основе ПКС сетевой технологии, обеспечивающей как передачу данных в реальном времени, так и автоматическое переконфигурирование сети при смене режимов функционирования КБО, в том числе при парировании множественных отказов.

Ключевые слова: программно-конфигурируемые сети, системы реального времени, бортовые системы

Для цитирования: Балашов В. В., Костенко В. А., Ермакова Т. И., "Построение бортовых сетей реального времени на основе технологии ПКС", *Моделирование и анализ информационных систем*, **26**:1 (2019), 23–38.

Об авторах:
Балашов Василий Викторович, ст. науч. сотр., канд. физ.-мат. наук, orcid.org/0000-0001-5211-805X,
Московский государственный университет имени М.В. Ломоносова, факультет ВМК
Ленинские горы, 1, стр. 52, г. Москва, ГСП-1, 119991 Россия, e-mail: hbd@cs.msu.su

Костенко Валерий Алексеевич, канд. техн. наук, доцент, orcid.org/0000-0002-7895-2322,
Московский государственный университет имени М.В. Ломоносова, факультет ВМК, e-mail: kostmsu@gmail.com

Ермакова Татьяна Ивановна, студент магистратуры, orcid.org/0000-0003-0224-9136,
Московский государственный университет имени М.В. Ломоносова, факультет ВМК, e-mail: tanyaerm@mail.ru

Благодарности:
Работа выполнена при поддержке РФФИ (проект № 17-07-01566).

Введение

В современных комплексах бортового оборудования (КБО) авиационного, морского и космического назначения широко используются два вида архитектур — федеративная и интегрированная модульная [1]. В соответствии с интегрированной модульной архитектурой (ИМА), КБО состоит из набора стандартных вычислительных модулей, соединенных коммутируемой сетью обмена данными с поддержкой виртуальных каналов. Такая сеть может быть построена на основе одного из следующих стандартов:

- ARINC 664 (AFDX) [2], основанный на Ethernet;
- профиль Fibre Channel реального времени (FC-AE-ASM-RT) [3, 4].

Аппаратные ресурсы вычислительных модулей в интегрированном модульном КБО разделяются между различными прикладными программами (подсистемами), каждая из которых содержит в себе ряд прикладных задач.

В данной работе приведен обзор возможностей сетей AFDX и FC-AE-ASM-RT по передаче данных в реальном времени и выделены присущие им ограничения; предложен подход к организации передачи данных в реальном времени в коммутируемых сетях ПКС с использованием элементов протокола AFDX. Данный подход апробирован в виртуальной сетевой среде, использующей сетевой контроллер RUNOS [5].

Использование ПКС для передачи данных в реальном времени в бортовых сетях позволит избавиться от ограничений существующих сетей AFDX и FC-AE-ASM-RT, в значительной мере сужающих возможности реализации динамических режимов функционирования КБО, включая реконфигурацию в случае множественных отказов.

1. Требования к передаче данных в реальном времени в бортовых сетях

На этапе проектирования КБО определяются режимы работы комплекса. Для каждого режима определяется набор прикладных задач, которые должны в нем выполняться, и набор сообщений, которые должны быть переданы между задачами и/или бортовыми датчиками и исполнительными устройствами. При смене режима работы КБО набор прикладных задач и сообщений также меняется.

Для каждого сообщения задаются:

1. требования к передаче в реальном времени;
2. размер;
3. максимальный джиттер (флуктуация задержки) порождения сообщения внутри директивного интервала;
4. абонент-отправитель сообщения;

5. один или несколько абонентов-получателей сообщения.

В интегрированном модульном КБО сетевыми абонентами могут являться вычислительные модули или унаследованные устройства с федеративной архитектурой. Требования к передаче сообщений в реальном времени задаются, как правило, путем назначения сообщению частоты передачи F (или периода $1/F$) и фазовых сдвигов ($\phi 1$, $\phi 2$). Частота передачи сообщения определяет набор временных интервалов, длины которых равны периоду сообщения. При помощи фазовых сдвигов для каждого такого интервала определяется подынтервал (s_i, f_i), в рамках которого должно быть передано сообщение, называемый директивным интервалом (см. Рис. 1).

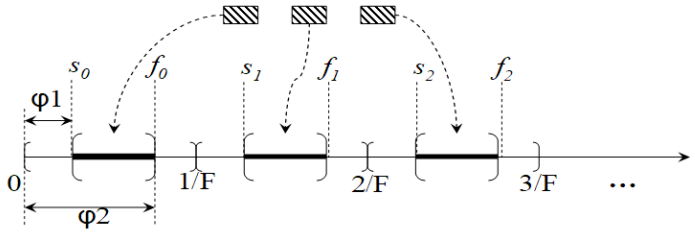


Рис. 1: Требования к передаче сообщения в реальном времени
Figure 1: Requirements to real-time transfer of messages

Периодические сообщения, описанные выше, определяют поток данных через сеть.

При передаче сообщений должны выполняться следующие ограничения:

1. Сообщение должно быть передано ровно один раз в период, в рамках директивного интервала. Если сообщение порождается чаще одного раза в период, то повторные экземпляры сообщения должны сбрасываться.

2. Максимальная длительность передачи сообщения с момента выдачи сообщения абонентом-отправителем до момента получения сообщения всеми абонентами-получателями не должна превышать заданного значения. Сообщение считается принятым, когда получен его последний кадр.

3. Максимальный джиттер передачи сообщения (разность между максимальной и минимальной длительностью передачи сообщения) не должен превышать заданного значения.

2. Передача данных в реальном времени через коммутируемые сети с виртуальными каналами

Используемые в интегрированных модульных КБО сети AFDX и FC-AE-ASM-RT имеют много общего в части организации передачи данных в реальном времени. В частности, в обоих видах сетей используется механизм виртуальных каналов для

разделения пропускной способности и контроля трафика. В данном разделе подробно описана схема передачи данных в сетях AFDX, а также приведены основные особенности передачи данных в сетях FC-AE-ASM-RT.

Сеть AFDX состоит из следующих элементов (Рис. 2):

- бортовые системы (абоненты), которые порождают и принимают сообщения;
- оконечные системы, которые предоставляют интерфейс между абонентами и сетью;
- пакетные коммутаторы;
- линии передачи данных.

Рис. 2: Типовая структура сети AFDX
Figure 2: Typical structure of an AFDX network

В данной работе не рассматривается детально принятое в AFDX разделение на абонентов и оконечные системы. Далее будем считать, что сеть состоит из узлов (принимающих и отправляющих сообщения), коммутаторов и линий передачи данных.

Сети AFDX используют стек протоколов, основанный на семействе протоколов TCP/IP. На канальном уровне используется модифицированный протокол Ethernet с маршрутизацией на основе виртуальных каналов (вместо маршрутизации на основе MAC-адресов). На сетевом уровне используется протокол IP, однако маршрутизация на этом уровне не производится, так как функция маршрутизации перекладывается на канальный уровень. На транспортном уровне используется протокол UDP.

Обмен данными между узлами осуществляется путем передачи сообщений по виртуальным каналам, маршруты которых задаются заранее. Для каждого виртуального канала определен один узел-отправитель и один или более узел-получатель.

Использование виртуальных каналов в сети AFDX обеспечивает разделение пропускной способности между различными потоками данных и соблюдение ограничений на время передачи сообщений через сеть.

Узел-отправитель разбивает сообщение на кадры, не превышающие определенного максимального размера, которые в свою очередь выдаются в физическую линию передачи данных. Номер виртуального канала записывается в поле MAC-адреса каждого кадра. После доставки кадра узлу-получателю, он буферизуется для последующей сборки сообщения. После прихода последнего кадра собранное сообщение направляется задачам, выполняемым на узле.

Надежность передачи данных в сетях AFDX обеспечивают за счет резервирования сетевого оборудования. Каждый узел соединен с двумя независимыми сетями AFDX, имеющими одинаковую структуру. Каждый кадр выдается в обе сети и следует в них по одинаковым маршрутам. На узле-получателе происходит проверка целостности кадров, и если в одной из сетей диагностируется ошибка передачи кадра (например, не совпадает контрольная сумма), то кадр берется из той сети, где не было ошибки. Если корректный кадр приходит из обеих сетей, используется тот кадр, который пришел раньше.

Таблицы маршрутизации AFDX-коммутаторов (содержащие информацию о виртуальных каналах) настраиваются заранее, до начала функционирования КБО. Помимо маршрутизации кадров, AFDX-коммутаторы выполняют контроль и фильтрацию трафика. В процессе фильтрации трафика для каждого виртуального канала проверяются целостность и порядок следования кадров. Контроль трафика обеспечивает виртуальному каналу требуемую пропускную способность и предотвращает ее превышение. Контроль пропускной способности виртуальных каналов в AFDX осуществляется при помощи алгоритма текущего ведра, в котором виртуальному каналу сопоставляется счетчик кредита. Если кредита недостаточно для поступившего на коммутатор кадра виртуального канала, коммутатор сбрасывает кадр. Скорость восстановления кредита определяется пропускной способностью виртуального канала.

Пропускная способность и маршрут виртуального канала устанавливаются заранее, до начала функционирования КБО. Таким образом, маршрутизация в сети AFDX задается статически, возможность динамически (во время функционирования КБО) изменять таблицы маршрутизации стандартом не предусмотрена.

Основное отличие сетей FC-AE-ASM-RT от сетей AFDX в части контроля трафика заключается в том, что контроль осуществляется на уровне сообщений, а не отдельных кадров. Также не используется разреженная передача кадров каждого виртуального канала с интервалом, определяемым значением BAG_{vl} (введено ниже); кадры сообщения выдаются в физическую линию последовательно, без пауз.

Сети FC-AE-ASM-RT предоставляют ограниченные возможности реконфигурации во время функционирования КБО: в каждый коммутатор может быть заранее загружено несколько таблиц коммутации, переключение между ними поддерживается с кратковременным (несколько десятков миллисекунд) нарушением передачи данных через сеть.

Виртуальный канал vl в сетях AFDX характеризуется следующими атрибутами [2, 6]:

- LM_{vl} – максимальный размер кадра, передаваемого по данному виртуальному каналу;
- BAG_{vl} – минимальный промежуток времени между выдачей последовательных кадров при нулевом джиттере порождения; согласно стандарту, данное значение лежит в промежутке от 1 до 128 мс и является степенью двойки;
- JM_{vl} – максимальный джиттер порождения кадров на узле-отправителе;
- S_{vl} – узел-отправитель кадров виртуального канала;
- $D_{vl} = \{d_{vl}\}$ – множество узлов-получателей кадров виртуального канала ($S_{vl} \notin D_{vl}$);
- $Tree_{vl}$ – маршрут передачи кадров в сети; маршрут представляет собой дерево, корнем которого является S_{vl}, а листьями все элементы множества D_{vl};
- $MSG_{vl} = \{msg\}$ – множество сообщений, передаваемых по виртуальному каналу; сообщения порождаются узлом S_{vl} и принимаются узлами из D_{vl}.

На набор виртуальных каналов в сети AFDX накладываются следующие ограничения:

- Суммарная пропускная способность, зарезервированная для всех виртуальных каналов, проходящих через физическую линию e, не превышает пропускной способности этой линии:
$$\sum_{vl \in e} \frac{LM_{vl}}{BAG_{vl}} \leqslant R_e, \tag{1}$$
где R_e – пропускная способность физической линии e.
- Требуемая для передачи сообщений через виртуальный канал частота выдачи кадров не превосходит максимальной частоты выдачи кадров в этот канал:
$$\forall vl \in VL : \sum_{msg \in MSG_{vl}} (\lceil \frac{size_{msg}}{LM_{vl} - c} \rceil \ / \ T_{msg}) \leqslant \frac{1}{BAG_{vl}} \ . \tag{2}$$
Данное ограничение вытекает из того, что все кадры одного экземпляра сообщения msg должны быть выданы в физическую линию до начала выдачи следующего экземпляра сообщения msg, то есть за время, не превышающее период T_{msg}. Учитывая, что сообщение msg делится на количество кадров $\lceil size_{msg} \ / \ (LM_{vl} - c) \rceil$, где c — размер служебных заголовков кадра, получаем ограничение (2).
- Максимальный джиттер выдачи кадра на оконечной системе-отправителе не превосходит заданного ограничения (0.5 мс для стандарта AFDX):
$$\forall vl \in VL : JM_{vl} \leqslant 0.5. \tag{3}$$

- Максимальная длительность передачи каждого сообщения и максимальный джиттер не превосходят заданных ограничений (определяемых логикой функционирования КБО, а не стандартом сети):

$$\forall msg \in MSG : Dur(msg) \leqslant \tau_{msg}; Jit(msg) \leqslant J^*_{msg}, \tag{4}$$

где $Dur(msg)$ и $Jit(msg)$ — длительность передачи и максимальный джиттер сообщения соответственно.

К сетям FC-AE-ASM-RT применимы схожие ограничения с тем уточнением, что в этих сетях они применимы на уровне сообщений, а не отдельных кадров.

Существует несколько подходов к проектированию виртуальных каналов для сетей AFDX [6, 7]. Эти подходы принимают в качестве входных данных множество сообщений и строят набор виртуальных каналов, удовлетворяющий ограничениям (1) — (4) и гарантирующий передачу сообщений с учетом ограничений реального времени 1 — 3 из раздела 1. При этом для оценки длительностей и джиттеров передачи сообщений через сеть используются такие математические аппараты, как Network Calculus [8] и Trajectory Approach [9]. Эти подходы и аппараты допускают адаптацию и для сетей FC-AE-ASM-RT.

3. Ограничения на формирование и смену режимов КБО, связанные с использованием сетей AFDX и FC-AE-ASM-RT

Примером смены режима функционирования КБО может служить переключение КБО летательного аппарата из режима предполетной подготовки в режим взлета. Штатная реакция КБО на одиночный отказ компонента оборудования (вычислительного модуля, коммутатора, линии связи) также представляет собой переход в один из заранее определенных отказных режимов, в котором не задействуется отказавший компонент, а нагрузка с него перераспределена на оставшиеся в строю компоненты.

При определенных обстоятельствах может потребоваться динамическое (в ходе работы КБО) формирование режима, например:

- при появлении нового задания для управляемого аппарата, в связи с чем в КБО загружаются новые функциональные задачи, при условии что прерывание функционирования КБО для перенастройки недопустимо;
- при множественных отказах оборудования КБО.

Оба случая особенно актуальны для автономных аппаратов с большой длительностью непрерывного функционирования, таких как космические аппараты и беспилотные летательные аппараты.

Использование сетей AFDX и FC-AE-ASM-RT накладывает следующие ограничения на формирование и изменение режимов функционирования КБО:

1. Невозможно динамическое формирование режимов функционирования КБО. Доступны только статические режимы, в которых набор прикладных задач и сообщений (включая распределение задач по модулям) определен на стадии проектирования или модернизации.

2. Невозможна гибкая реконфигурация КБО в случае множественных отказов оборудования.

3. Бесшовное переключение между статическими режимами КБО возможно только в случае одновременного (в одной таблице виртуальных каналов на каждом коммутаторе) резервирования пропускной способности под виртуальные каналы всех режимов. Бесшовным будем называть такое переключение режимов, при котором не прерывается передача сообщений, которые должны передаваться как в прежнем, так и в новом режиме.

Поясним, в связи с чем возникают перечисленные ограничения.

Смена режима функционирования КБО приводит к смене набора исполняемых прикладных задач и передаваемых сообщений; изменяется также набор виртуальных каналов, используемых для передачи этих сообщений. Для сетей AFDX, которые не поддерживают изменение таблиц маршрутизации коммутаторов в ходе работы системы, необходимо зарезервировать пропускную способность для всех режимов работы так, как если бы эти режимы функционировали одновременно. Это сильно ограничивает суммарную пропускную способность виртуальных каналов для всех поддерживаемых режимов и приводит к неэффективному использованию пропускной способности физической сети. Коммутаторы сетей FC-AE-ASM-RT поддерживают переключение в ходе работы системы между несколькими заранее заданными таблицами маршрутизации, которые могут соответствовать нескольким различным режимам функционирования. При таком переключении, выполняемом в процессе смены режима функционирования КБО, передача сообщений между задачами, относящимися как к новому, так и к старому режиму, временно прерывается.

Для поддержки динамического формирования режимов функционирования КБО, основанного на сетях AFDX или FC-AE-ASM-RT, необходимо заранее заложить в таблицы маршрутизации всю информацию по виртуальным каналам, используемым во всех потенциально доступных режимах функционирования.

Если режим КБО динамически формируется в связи с появлением новых прикладных задач и потоков данных, невозможно заранее заложить информацию о виртуальных каналах для этих потоков данных в таблицы коммутаторов, в связи с тем, что эта информация недоступна на момент начала функционирования КБО. Следовательно, в таком случае невозможно динамическое формирование режимов функционирования КБО на основе сетей AFDX или FC-AE-ASM-RT.

При отказе вычислительного модуля интегрированного модульного КБО, существует возможность перераспределить его задачи между оставшимися модулями (миграция задач). В этом случае изменяются маршруты виртуальных каналов, соединяющих данные задачи. В сетях AFDX поддержка миграции задач приводит к исключительно неэффективному использованию пропускной способности, поскольку необходимая для взаимодействия двух задач пропускная способность должна

быть зарезервирована многократно — по одному разу на каждое возможное распределение задач между модулями. Для сетей FC-AE-ASM-RT существует возможность рассчитать и заложить в коммутаторы таблицы маршрутизации для всех одиночных отказов; в то же время попытка учесть варианты множественных отказов повлечет за собой резкий рост числа вариантов миграции задач, что приводит к невозможности заложить все соответствующие наборы виртуальных каналов в таблицы маршрутизации коммутаторов. Необходимость парирования отказов сетевых устройств (коммутаторов, линий связи) только усугубляет описанную ситуацию. Тем самым, практически (при числе компонентов КБО порядка 10 и более) невозможно динамическое формирование режимов для парирования множественных отказов в КБО на основе сетей AFDX или FC-AE-ASM-RT.

Даже для поддержки только статических режимов функционирования, бесшовная смена режимов КБО требует одновременного резервирования пропускной способности для виртуальных каналов всех режимов, требующих бесшовного переключения. Для КБО на основе сетей AFDX это единственный способ поддержки переключения режимов; сети FC-AE-ASM-RT поддерживают также переключение между заранее заданными таблицами маршрутизации, что приводит к временному прерыванию обменов между модулями КБО и не может считаться бесшовной сменой режима функционирования.

4. Передача данных в реальном времени по виртуальным каналам в сети ПКС

Для устранения описанных в разделе 3 ограничений, возникающих при использовании в КБО сетей AFDX и FC-AE-ASM-RT, целесообразно использовать ПКС вместо этих сетей. Данная целесообразность обусловлена тем, что описанные ограничения вызваны ограниченными (в случае FC-AE-ASM-RT) или полностью отсутствующими (в случае AFDX) возможностями изменения конфигурации сети, а именно набора виртуальных каналов и их параметров, включая маршруты, в ходе функционирования системы. С точки зрения реконфигурации сети ключевой особенностью ПКС является поддержка динамической, осуществляемой в ходе функционирования, перенастройки коммутаторов и наличие контроллера сети, приложения которого могут отвечать за эту перенастройку.

Авторами настоящей работы предлагается подход к использованию ПКС в качестве бортовых сетей, основанный на реализации в ПКС схемы передачи данных в реальном времени, обладающей следующими основными чертами:

- для контроля трафика и разделения пропускной способности между потоками данных используется механизм виртуальных каналов;
- контроль трафика виртуальных каналов организован по аналогии с сетями AFDX и FC-AE-ASM-RT;
- сохраняется применимость существующих подходов к оценке задержек и джиттеров передачи данных, разработанных для сетей AFDX;
- контроллер сети функционирует в проактивном (пассивном) режиме;

- поддерживается динамическое изменение набора виртуальных каналов (включая изменение их параметров и маршрутов) без прекращения передачи данных по не изменяемым виртуальным каналам.

Использование известной и практически отработанной в существующих бортовых сетях AFDX и FC-AE-ASM-RT схемы управления трафиком, с одной стороны, ориентировано на упрощение возможной модернизации существующих КБО и создания новых КБО на основе существующих, а с другой стороны, позволяет использовать известные подходы к обоснованию соблюдения требований реального времени к передаче данных через сеть.

Проактивный режим функционирования контроллера гарантирует отсутствие избыточного трафика между коммутаторами и контроллером, за исключением трафика, необходимого для мониторинга и перенастройки сети. Это существенно как с точки зрения минимизации нагрузки на сеть, так и для обеспечения прогнозируемости времен передачи данных через сеть, поскольку не требуется модифицировать существующие методы оценки задержек и джиттера для учета воздействия служебного трафика. Выбор между активным и проактивным режимами работы контроллера подробнее рассмотрен в [10].

Предложенная схема использует стек протоколов TCP/IP за исключением маршрутизации на основе IP-адресов. Передаваемое сообщение разделяется на множество UDP-пакетов, каждый из которых соответствует одному кадру канального уровня (т.е. Ethernet). Эти пакеты посылаются узлом-отправителем в соответствии со схемой, используемой в сетях AFDX, с интервалами, равными параметру виртуального канала BAG_{vl}. Номер виртуального канала указывается в поле vlan заголовка пакета. Маршрутизация пакета коммутаторами производится на основе номера виртуального канала, как и в сетях AFDX и FC-AE-ASM-RT. Для осуществления маршрутизации в таблицы коммутаторов заносятся специальные правила, которые в зависимости от значения поля с номером виртуального канала отправляют пакет на соответствующий выходной порт коммутатора. Пакеты, размер которых превышает заданный лимит для виртуального канала, так же как и пакеты, пришедшие на порт, не соответствующий указанному в их заголовках виртуальному каналу, сбрасываются коммутатором.

Контроль трафика производится при помощи измерителей (meter) — технологии, введенной в протоколе OpenFlow 1.3 [11]. Измеритель ассоциируется с потоком; в рассматриваемом случае поток определяется как последовательность принятых пакетов с определенным номером виртуального канала, т.е. пакетов заданного виртуального канала. Для каждого измерителя на коммутаторе определен свой набор интервалов скоростей потока. В зависимости от принадлежности текущей измеренной скорости потока к одному из интервалов, к полученному пакету применяется заданное для этого интервала действие, например, дальнейшая передача или сброс.

Способ измерения скорости потока не определяется протоколом OpenFlow 1.3 и зависит от реализации механизма измерителей на конкретном коммутаторе. В рамках предлагаемой в данной работе схемы передачи данных, от коммутатора требуется поддержка измерения скорости потока на основе алгоритма текущего ведра, аналогичного используемому в AFDX. Это позволяет контролировать не только непревышение скоростью потока пропускной способности виртуального канала, но и то, что джиттер передачи сообщений не превышает максимально допустимого.

Примером коммутатора, поддерживающего требуемую схему расчета скорости потока, является программный коммутатор *Ofsoftswitch13*.

Измерители настраиваются на коммутаторах при помощи протокола OpenFlow следующим образом. Для каждого коммутатора, для каждого виртуального канала, проходящего через этот коммутатор:

- задается измеритель виртуального канала при помощи сообщения *FlowMod*;
- измерителю при помощи сообщения *MeterMod* задается максимальное значение кредита и величина LM_{vl} / BAG_{vl} (в качестве полей $burst\_size$ и $rate$, соответственно).

В соответствии с предлагаемым подходом, виртуальный канал в ПКС имеет тот же набор параметров, что и виртуальный канал сети AFDX (см. раздел 2). Поскольку схема выдачи данных абонентом в физическую линию и схема контроля трафика на коммутаторе заимствованы из AFDX, для предлагаемого подхода применимы разработанные для сетей AFDX схемы оценки задержек и джиттера передачи данных через сеть [8, 9], а также методы построения систем виртуальных каналов [6].

5. Апробация предлагаемого подхода

Авторами было проведено экспериментальное исследование предлагаемой схемы передачи данных в реальном времени в ПКС. Эксперименты были проведены в виртуальной среде, построенной на основе эмулятора сети mininet, ПКС-контроллера RUNOS и программного коммутатора Ofsoftswitch13.

В среде контроллера RUNOS было реализовано служебное приложение, поддерживающее настройку коммутаторов на заданный набор виртуальных каналов, а также снятие и добавление отдельных виртуальных каналов. Собственно сценарии экспериментов реализованы в виде приложений контроллера, обращающихся к служебному приложению для построения и модификации конкретных систем виртуальных каналов.

Экспериментальное исследование преследовало следующие цели:

- оценить задержки и джиттер передачи сообщений через сеть, подтвердить их соответствие расчетам по методикам, принятым для сетей AFDX;
- подтвердить, что предложенная схема позволяет осуществлять передачу различных потоков данных с гарантированной пропускной способностью для каждого из потоков;
- подтвердить, что попытки узла передавать данные по виртуальному каналу со скоростью, превышающей максимально допустимую пропускную способность, блокируются коммутатором;
- провести динамическую модификацию набора виртуальных каналов в соответствии с различными сценариями, типичными для КБО, а также подтвердить, что передача данных по виртуальным каналам, не затронутым модификацией, не прерывалась.

Количество виртуальных каналов и их параметры были взяты из работы по проектированию сетей AFDX [6]. Количество виртуальных каналов, в зависимости от эксперимента, достигало нескольких десятков.

В качестве топологий сети были взяты обе описанные в [6], а также топологии, типичные для интегрированных модульных КБО, например, топология «множественная звезда» (Рис. 3) и топология с подключением каждого абонента к двум коммутаторам (Рис. 4).

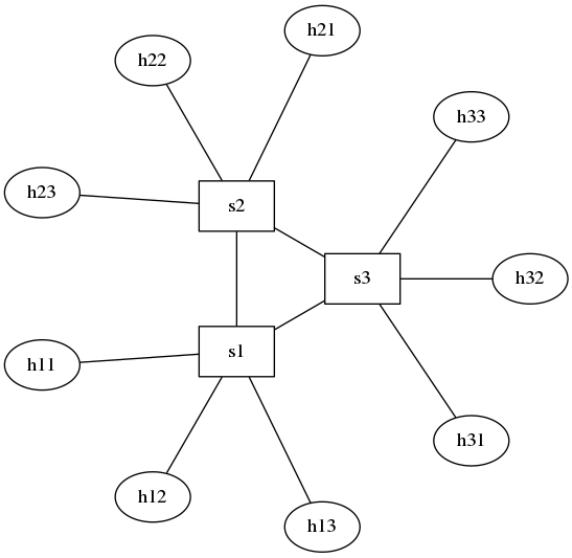


Рис. 3: Топология «множественная звезда»
Figure 3: Multiple star network topology

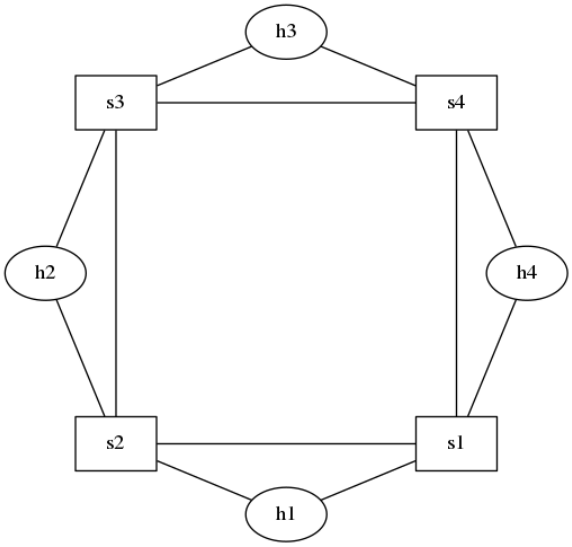


Рис. 4: Топология с подключением каждого абонента к двум коммутаторам
Figure 4: Network topology with every node connected to two switches

Проведенные эксперименты показали, что для рассмотренных потоков данных и топологий сети предложенная схема передачи данных в ПКС гарантирует обеспечение требуемой пропускной способности для каждого потока данных с задержкой и джиттером, удовлетворяющими заданным верхним ограничениям. В экспериментах, где некоторые абоненты формировали трафик, выходящий за ограничения для заданного виртуального канала (слишком большие UDP-пакеты и/или слишком малые интервалы между пакетами), коммутатор осуществлял сброс пакетов этого виртуального канала, тем самым предотвращая превышение ограничений на пропускную способность физической линии.

В экспериментах использовались следующие сценарии динамической модификации систем виртуальных каналов:

1. "обычная" смена режима КБО: часть виртуальных каналов сохраняет функционирование, остальные каналы удаляются, а вместо них добавляются новые каналы;

2. миграция задач в случае выхода из строя вычислительного модуля: виртуальные каналы, входящие или исходящие для вышедшего из строя узла, перекладываются в соответствии со своими новыми маршрутами;

3. реконфигурация сети в случае выхода из строя физической линии передачи данных или коммутатора: виртуальные каналы, проходившие через сбойный элемент сети, меняют свои маршруты на обходные.

Во всех перечисленных сценариях новые маршруты виртуальных каналов были заданы в составе исходных данных. Поддержка построения новых маршрутов виртуальных каналов в ходе функционирования КБО является предметом дальнейших исследований.

Эксперименты со сценарием 1 показали, что при удалении/добавлении виртуальных каналов передача данных через не изменяющиеся виртуальные каналы не нарушается. Также не нарушалась передача данных через не изменяющиеся виртуальные каналы в экспериментах со сценариями 2 и 3.

Основное отличие сценариев 2 и 3 от сценария 1 в том, что выдача данных в изменяемые виртуальные каналы не прекращалась. Настройка коммутаторов на новый маршрут виртуального канала начиналась с первого из коммутаторов (считая по направлению передачи данных), после которого канал идет по новому маршруту. Эксперименты показали, что при настройке коммутаторов на новый маршрут виртуального канала терялись несколько пакетов; конкретное число зависит от частоты выдачи пакетов, т.е. от значения параметра BAG_{vl}. Нарушения передачи данных через другие виртуальные каналы при этом не происходило.

Проведенные эксперименты показали, что предлагаемый подход к построению бортовых сетей передачи данных в реальном времени позволяет сохранить предсказуемость временных характеристик, характерную для сетей AFDX и FC-AE-ASM-RT, при этом позволяет устранить накладываемые этими сетями ограничения на формирование и смену режимов КБО.

6. Необходимый объем адаптации ПКС-технологий для применения в КБО

КБО является критически важным элементом управляемых систем, таких как летательные или космические аппараты. К аппаратно-программным средствам в составе КБО, в т.ч. средствам передачи данных, предъявляются жесткие требования по надежности функционирования в сложных условиях эксплуатации (повышенная температура, вибрация, воздействие радиации). В связи с этим при реализации контроллеров и коммутаторов ПКС для КБО необходимо использовать технологии, не уступающие по уровню устойчивости к жестким условиям эксплуатации остальным компонентам КБО, в том числе вычислительным модулям. Следует отметить, что данное условие может быть реализовано автоматически за счет использования штатных вычислительных мощностей (вычислительных модулей) в составе интегрированных модульных КБО для реализации программных коммутаторов и контроллеров ПКС. Однако при переходе к использованию аппаратных коммутаторов проблема «бортового» исполнения оборудования встанет вновь.

Для использования ПКС-технологий в КБО таких аппаратов, как автомобили и пассажирские самолеты, аппаратный и программный компоненты ПКС, а также процессы их разработки, должны быть сертифицированы в соответствии с рядом промышленных стандартов, таких как DO-178C [11] для авиационных комплексов.

Заключение

В существующих интегрированных модульных комплексах бортового оборудования (КБО) используются коммутируемые сети AFDX и FC-AE-ASM-RT, реализующие основанный на виртуальных каналах подход к передаче данных в реальном времени. Основным недостатком этих сетей являются ограниченные или отсутствующие возможности динамической реконфигурации виртуальных каналов, приводящие к невозможности динамического формирования режимов функционирования КБО, в частности при множественных отказах оборудования.

Для снятия выявленных ограничений авторами предложен подход к использованию ПКС для построения бортовых сетей реального времени. Предложенный подход основан на реализации в сети ПКС, поддерживающей протокол OpenFlow1.3, механизма виртуальных каналов, аналогичного используемому в сетях AFDX и FC-AE-ASM-RT. Подход реализован в виде функционального прототипа и экспериментально апробирован в виртуальной сетевой среде, основанной на программных ПКС-коммутаторах Ofsoftswitch13 и сетевом контроллере RUNOS. Эксперименты показали, что предложенная схема передачи данных позволяет передавать сообщения с соблюдением заданных ограничений на задержку и джиттер, а также не допускает превышения ограничения на пропускную способность виртуального канала. Эксперименты также подтвердили, что динамическая реконфигурация виртуальных каналов в ПКС не нарушает передачу данных по не изменяемым виртуальным каналам.

К направлениям дальнейших работ можно отнести:

- разработку алгоритмов для динамического формирования новых маршрутов

виртуальных каналов в процессе реконфигурации КБО, в т.ч. при парировании отказов вычислительных модулей и сетевых устройств;

- реализацию созданных алгоритмов в рамках приложения для ПКС-контроллера;
- интеграцию созданного приложения со средствами мониторинга состояния КБО, в т.ч. мониторинга элементов сети.

Объединяющей целью перечисленных работ является создание на основе ПКС сетевой технологии, обеспечивающей как передачу данных в реальном времени, так и автоматическое переконфигурирование сети при смене режимов функционирования КБО, в том числе при парировании множественных отказов.

Список литературы / References

[1] Gaska T., Watkins C., and Chen Y., "Integrated Modular Avionics — Past, present, and future", *IEEE Aerospace and Electronic Systems Magazine*, **30**:9 (2015), 12–23.

[2] *Aircraft Data Network. Part 7. Avionics Full Duplex Switched Ethernet (AFDX) Network*, Aeronautical Radio, 2012.

[3] *INCITS 373. Information Technology — Fibre Channel Framing and Signaling Interface (FC-FS)*, International Committee for Information Technology Standards, 2003.

[4] Осипов Ю.С., Першин А.С., Пустовой Ю.В., *Способ передачи информации в реальном времени с использованием локальных сетей ограниченного размера на базе модификации протокола FC-AE-ASM*, патент RU2536659, ОАО "НТЦ ГРЭК", 2013; [Osipov Y., Pershin A., Pustovoy Y., *Method for Real-Time Information Transmission Using Small-Scale Local Area Networks Based on FC-AE-ASM Protocol Modification*, Patent RU2536659C1, NTC GREK, 2013, (in Russian).]

[5] Shalimov A., Nizovtsev S., Morkovnik D., Smeliansky R., "The Runos OpenFlow Controller", *2015 Fourth European Workshop on Software Defined Networks*, IEEE Computer Society, 2015, 103–104, https://ieeexplore.ieee.org/document/7313624.

[6] Вдовин П.М., Костенко В. А., "Организация передачи сообщений в сетях AFDX", *Программирование*, **43**:1 (2017), 5–20; in English: Vdovin P.M., Kostenko V.A. , "Organizing Message Transmission in AFDX Networks", *Programming and Computer Software*, **43**:1 (2017), 1–12.

[7] Al Sheikh A., Brun O., and Hladik P.-E., "Optimal Design of Virtual Links in AFDX Networks", *Real-Time Systems*, **49**:2 (2013), 308–336.

[8] Boyer M., Fraboul C., "Tightening End to End Delay Upper Bound for AFDX Network Calculus with Rate Latency FIFO Servers Using Network Calculus", *IEEE International Workshop on Factory Communication Systems*, IEEE, 2008, 11–20.

[9] Bauer H., Scharbarg J. L., and Fraboul C., "Applying and Optimizing Trajectory Approach for Performance Evaluation of AFDX Avionics Network", *IEEE Conference on Emerging Technologies and Factory Automation*, IEEE, 2009, 1–8.

[10] Balashov V., Kostenko V., Vdovin P., Smeliansky R., Shalimov A., "An Analysis of Approaches to Onboard Networks Design", *2014 International Science and Technology Conference on Modern Networking Technologies*, IEEE, 2014, 20–14.

[11] *DO-178C Software Considerations in Airborne Systems and Equipment Certification*, Committee SC-205, RTCA, 2011.

Balashov V. V., Kostenko V. A., Ermakova T. I., "Design of Onboard Real-Time Networks Based on SDN Technology", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 23–38.

DOI: 10.18255/1818-1015-2019-1-23-38

Abstract. Modern onboard equipment complexes (OEC) utilize AFDX and FC-AE-ASM-RT switched networks implementing a virtual link-based approach to real-time data transfer. The main drawback of these networks is their limited or absent support for dynamic reconfiguration of virtual links, which makes impossible the dynamical recomposition of OEC operation modes, particularly in case of multiple equipment failures. To remove these drawbacks, in this paper an approach is proposed to use software-defined networks (SDN) as onboard real-time networks. The proposed approach is based on implementation of a virtual link-based technology (similar to those used in AFDX and FC-AE-ASM-RT) in an SDN supporting OpenFlow 1.3 protocol. The approach was implemented as a functional prototype and experimentally evaluated in a virtual network environment based on Ofsoftswitch13 software SDN switches and RUNOS controller. The experiments indicated that the proposed data exchange scheme allows the transfer of messages within the given limits on delay and jitter, and does not allow violation of constraints on a virtual link bandwidth. The experiments also confirmed that dynamic reconfiguration of virtual links in SDN does not interrupt the data transfer through unchanged virtual links. An important direction for future work is development of algorithms for dynamic creation of virtual link routes in course of OEC reconfiguration. The final goal of the work is to create an SDN-based network technology supporting both real-time data transfer and automatic network reconfiguration in case of OEC mode change, including parrying multiple failures.

Keywords: software defined networks, real-time, onboard computer systems

On the authors:
Vasily V. Balashov, Ph.D. in Mathematics, senior research fellow, orcid.org/0000-0001-5211-805X,
Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics,
1-52, Leninskiye Gory, Moscow, GSP-1, 119991 Russia, e-mail: hbd@cs.msu.su

Valery A. Kostenko, Ph.D. in Technology, associate professor, orcid.org/0000-0002-7895-2322,
Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics,
1-52, Leninskiye Gory, Moscow, GSP-1, 119991 Russia, e-mail: kostmsu@gmail.com

Tatiana I. Ermakova, master student, orcid.org/0000-0003-0224-9136,
Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics,
1-52, Leninskiye Gory, Moscow, GSP-1, 119991 Russia, e-mail: tanyaerm@mail.ru

Acknowledgments:
This work was partially supported by the Russian Foundation for Basic Research, Grant № 17-07-01566.

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 39–62
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 39–62

DOI: 10.18255/1818-1015-2019-1-39-62

УДК 517.9

Об одном подходе к построению сетевого процессорного устройства

Беззубцев С. О., Васин В. В., Волканов Д. Ю., Жайлауова Ш. Р., Мирошник В. А., Скобцова Ю. А., Смелянский Р. Л.

Поступила в редакцию 10 января 2019
После доработки 12 февраля 2019
Принята к публикации 15 февраля 2019

Аннотация. В работе предложена архитектура и основные требования к сетевому процессору для OpenFlow коммутаторов программно-конфигурируемых сетей (ПКС). Представлен анализ архитектур известных сетевых процессоров – NP-5 компании EZchip (в настоящее время Mellanox) и Tofino компании Barefoot Networks. Рассмотрены достоинства и недостатки двух разных вариантов архитектур сетевого процессора: на основе конвейеров, ячейки которых представлены набором процессорных ядер общего назначения, и на основе конвейеров, ячейкам которых соответствуют ядра, специализированные под конкретные операции обработки пакета. На основе выделенного набора наиболее общих сценариев обработки пакетов предложена новая архитектура сетевого процессорного устройства (СПУ) с функционально специализированными ячейками (стадиями) конвейера. В статье представлено описание имитационной модели СПУ предложенной архитектуры. Имитационная модель построена на языке C++ с использованием открытой библиотеки SystemC. Для проведения функционального тестирования полученной модели СПУ были реализованы описанные сценарии обработки пакетов на языке C. Для оценки производительности предложенной архитектуры СПУ в ходе исследования были использованы программные средства компании KM211, а также семейство микроконтроллеров KMX32. Оценка производительности СПУ проводилась на основе имитационной модели. Получены оценки времени обработки одного пакета и средняя пропускная способность модели СПУ для каждого сценария. Эти оценки показали, что полученная скорость СПУ позволяет их использование в коммутаторах уровня распределения (агрегации).

Ключевые слова: сетевое процессорное устройство, коммутатор, компьютерные сети, программно-конфигурируемые сети, архитектура компьютера, имитационное моделирование, протокол OpenFlow

Для цитирования: Беззубцев С.О., Васин В.В., Волканов Д.Ю., Жайлауова Ш.Р., Мирошник В.А., Скобцова Ю.А., Смелянский Р.Л., "Об одном подходе к построению сетевого процессорного устройства", *Моделирование и анализ информационных систем*, **26**:1 (2019), 39–62.

Об авторах:
Беззубцев Станислав Олегович, технический специалист, orcid.org/0000-0002-2419-7632
Центр прикладных исследований компьютерных сетей,
Ленинские горы, 1, стр. 77, г. Москва, 119992 Россия, e-mail: stas.bezzubtsev@gmail.com

Васин Вячеслав Викторович, ведущий программист-разработчик, orcid.org/0000-0002-5759-3105
Центр прикладных исследований компьютерных сетей,
Ленинские горы, 1, стр. 77, г. Москва, 119992 Россия, e-mail: vvasin@arccn.ru

Волканов Дмитрий Юрьевич, канд. физ.-мат. наук, доцент, orcid.org/0000-0001-9940-5822
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: volkanov@lvk.cs.msu.su

Жайлауова Шынар Рустембековна, аспирант, orcid.org/0000-0001-5725-7040
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, Россия, e-mail: szhaylau@cs.msu.ru

Мирошник Владислав Александрович, программист-разработчик, orcid.org/0000-0003-0883-0116
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: miroshnikv@lvk.cs.msu.su

Скобцова Юлия Александровна, студент-магистр, orcid.org/0000-0001-8351-3191
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: xenerizes@lvk.cs.msu.su

Смелянский Руслан Леонидович, чл.-кор. РАН, д-р физ.-мат. наук, проф., orcid.org/0000-0003-2311-4513
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: smel@cs.msu.su

Благодарности:
Работа выполнена при финансовой поддержке гранта РФФИ (№ 19-07-01076).

Введение

Целью данной работы являлась разработка архитектуры сетевого процессорного устройства (СПУ). Под СПУ понимаем программируемое устройство, настраиваемое под стек протоколов. Разрабатываемая архитектура должна учитывать требования к коммутаторам как для традиционных TCP/IP сетей (IP/MPLS маршрутизаторов), так и для программно-конфигурируемых сетей (ПКС) с коммутацией пакетов и позволять настройку устройства на работу в одной из вышеупомянутых видов сетей программными методами. Рассматривается проблема построения коммутаторов для ПКС сетей [1]. В контексте данной статьи наиболее существенно то, что коммутаторы в ПКС программно управляемы. А именно, рассматриваются ПКС сети с протоколом OpenFlow [2]. Такие ПКС сети предполагают, что центр управления сетью – ПКС контроллер со своими приложениями – формирует программы обработки пакетов данных для каждого коммутатора в контуре передачи данных.

В качестве устройства обработки и передачи трафика в таких коммутаторах может использоваться:

- универсальный микропроцессор;
- сетевой процессор;
- специализированная интегральная схема.

СПУ занимает промежуточное место между универсальными микропроцессорами (также называемыми центральными процессорными устройствами ЦПУ), выполняющими практически любые задачи на низкой скорости, и специализированными интегральными схемами (ASIC), разработанными для эффективного решения конкретного набора задач, объединяя в себе гибкость первых и при правильном проектировании мощность вторых.

Ниже представлена последовательность задач обработки пакета (см. Рисунок 1), выполняемых СПУ: выделение заголовка пакета, классификация и модификация. Одним из способов ускорения обработки данных является конвейеризация выполняемых над ними действий.

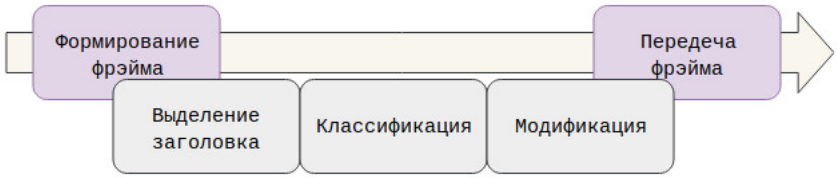


Рис. 1. Классификация задач СПУ. Белым цветом выделены задачи СПУ

Fig. 1. Taxonomy of network processing functions.
The network processor tasks are highlighted in white

К примеру, в случае классического коммутатора может быть достаточно выполнения одной цепочки действий:

- выделение заголовка;
- классификация;
- модификация.

Как будет видно из раздела 3., число цепочек обработки может возрасти.

В коммутаторах, используемых в традиционных компьютерных сетях, архитектура процессора обычно оптимизирована для решения определённого круга задач и работы по определённым протоколам. В случае программируемого СПУ в составе коммутатора, возможно "переиспользование" ячеек конвейера для разных сценариев. Это означает, что действия, выполняемые в каждой из ячеек, некоторым образом параметризуют, и в дальнейшем используют для программирования поведения ячейки. Программист может перепрограммировать работу ячейки с помощью библиотеки функций на языке общего назначения, например на языке C для сетевого процессора EZChip NP5 [5], или с помощью особого предметно-ориентированного языка описания поведения сетевого устройства, например языка P4 [6] для сетевого процессора Barefoot Tofino [7].

Современные крупные сети очень сложны, поскольку определяются множеством протоколов, конфигурациями и технологиями. Используя иерархическую модель, можно легко упорядочить все компоненты сети согласно характеристикам каждого иерархического уровня. Примером такой модели может служить трехуровневая иерархическая модель сети [4], определяющая подход к проектированию сетей и включающая в себя три логических уровня:

- уровень ядра ($>$ 10 Гбит/с);
- уровень распределения/агрегации (1 – 10 Гбит/с);
- уровень доступа ($<$ 1 Гбит/с).

Данная работа организована следующим образом. В разделе 1. описаны две характерные архитектуры СПУ. Сценарии обработки пакетов представлены в разделе 2. Предлагаемая архитектура СПУ описана в разделе 3. Реализованная имитационная модель СПУ рассматривается в разделе 4. Раздел 5. содержит результаты экспериментов, проведенных с имитационной моделью. В заключение проделанной работы в шестом разделе приводится краткое описание направлений предстоящих исследований.

1. Обзор существующих архитектур СПУ

Можно выделить два основных варианта конвейерных архитектур СПУ: использование внутри ячеек конвейера большого числа простых процессорных ядер, которые в явном виде не специализированы под конкретные задачи, и, наоборот, использование ядер, специализированных под конкретные операции обработки пакета. Каждый из этих подходов имеет свои достоинства и недостатки [3].

В настоящем разделе будут рассмотрены обе архитектуры. P4-программируемая абстрактная модель коммутатора (которая может быть рассмотрена как очень общая модель конвейера СПУ) со своей реализацией в процессоре Tofino [7] (компании Barefoot Networks) является примером первого подхода с одинаковыми стадиями конвейера. В качестве примера второго подхода рассмотрим процессор NP-5 [5] (компании EZchip) с функционально специализированными стадиями. Также будут описаны особенности структуры пути обработки данных и некоторые особенности физического устройства.

В основе архитектуры сетевых процессоров EZchip NP-5 лежит сеть специализированных ядер (см. Рисунок 2).

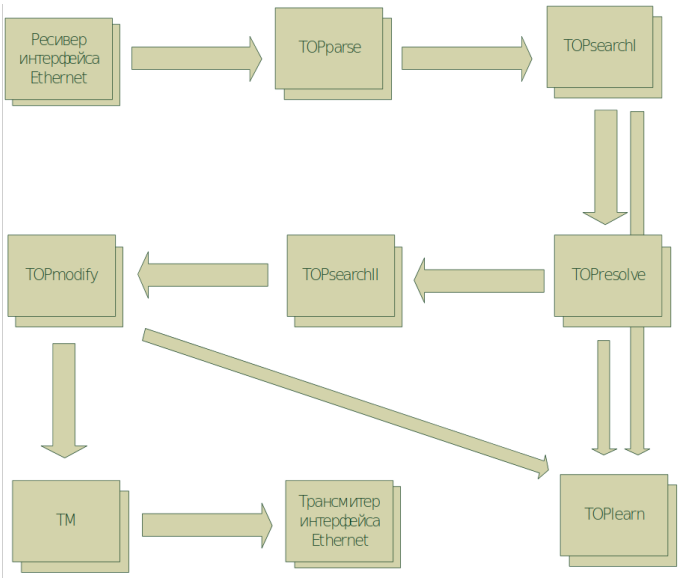


Рис. 2. Логические связи между блоками коммутатора, построенного с использованием сетевого процессора NP-5

Fig. 2. Diagram of links between the objects of the switch based on NP-5

Ядра относятся к разным классам, представители каждого из классов ориентированы на выполнение определённого набора действий: по разбору заголовков сетевых пакетов, классификации сетевых пакетов, преобразованию, управлению потоками. Каждый экземпляр микропроцессора обладает набором ячеек памяти и подключен к ограниченному набору шин данных, необходимых для выполнения определенной задачи.

По тракту обработки данных сетевого процессора NP-5 видно, что он включает в себя одну цепочку действий вида выделение заголовка – поиск – модификация. В случае, если для сценария обработки пакетов этого недостаточно, происходит «заворот» на первую ячейку конвейера, и ячейки используются повторно для реализации последующей обработки пакета. Подобный подход, несмотря на значительную гибкость, понижает скорость работы конвейера. Если для трафика, проходящего по конвейеру сетевого устройства, характерно прохождение двух таких цепочек действий, т. е. всегда используется хотя бы один заворот, скорость обработки пакетов понижается в два раза. Это происходит за счет того, что новые пакеты вынуждены ожидать своей очереди, пока ранее полученные пройдут два цикла обработки.

Модель абстрактного СПУ предложена консорциумом языка программирования сетевых устройств P4 [6]. Стоит отметить, что с точки зрения традиционных сетей подобное устройство может представлять собой и маршрутизатор в зависимости от сценария использования, для работы с которым оно запрограммировано. Предлагаемая модель связана с архитектурой PISA (Protocol Independent Switch Architecture) [6] и опирается на предположение об универсальности ячеек конвейера СПУ и возможность описания их функциональности средствами языка P4. Реализацией данной модели является сетевой процессор Tofino, созданный компанией Barefoot Networks [7].

Тракт обработки данных (см. Рисунок 3) модели абстрактного СПУ подразумевает наличие двух этапов конвейера: входной и выходной обработки, при этом эти этапы, в свою очередь, состоят из трёх программируемых ячеек: ячейка парсера, ячейка конвейера, ячейка депарсера.

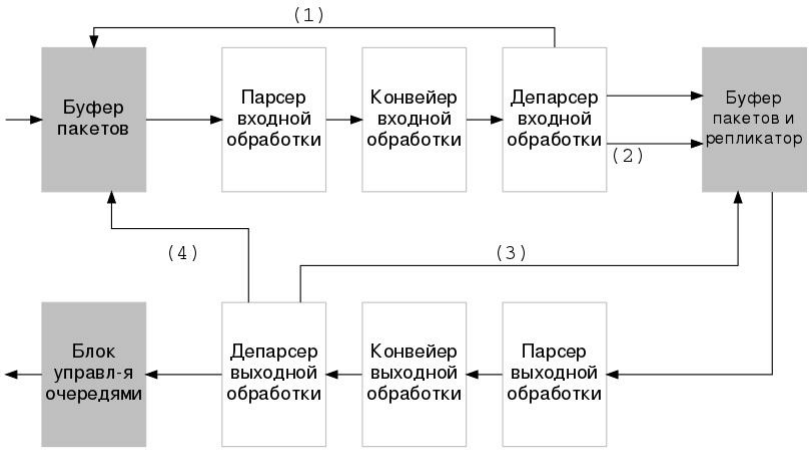


Рис. 3. Схема тракта обработки данных абстрактного СПУ P4

Fig. 3.The data path of the abstract switch model

Устройство ячеек, выполняющих буферизацию, репликацию пакетов и управление очередями, в описании архитектуры не специфицируется. Стоит отметить, что рассматриваемые ячейки и этапы являются абстракциями для разработчика программного обеспечения СПУ и могут значительно отличаться от аппаратной реализации.

Каждая из рассмотренных архитектур имеет достоинства и недостатки. Сетевой процессор NP-5 оптимизирован для использования в коммутаторе, имеющем одну цепочку вида выделение данных — классификация – модификация. В случае, если для сценария использования этого недостаточно, необходимо «завернуть» пакет на вход первой ячейки конвейера, что влечет дополнительные накладные расходы на передачу контекста пакета назад и уменьшение эффективной пропускной способности из-за увеличения времени занятости конвейера. В случае использования MPLS меток возможно большое число «заворотов» внутри конвейера, что приводит к деградации производительности. Сетевой процессор Tofino, претендующий на решение широкого круга задач, допускает большее число таких цепочек, не требуя при этом «заворота» к первой ячейке. В подходе P4 можно производить разбор любого заголовка, разработчик определяет свои структуры для заголовка, что позволяет более гибко работать при использовании MPLS. Тем не менее, здесь тоже может возникнуть проблема физического ограничения на число цепочек для сложных сценариев. Однако здесь не будет возможности сделать «заворот» – будет получена ошибка компиляции программы на P4. Также отсутствуют возможности для оптимизации для какого-либо сценария из-за универсальности ячеек и ограниченности средств языка P4.

Исследования, представленные в данной статье, заключены в попытке скомбинировать оба вышеописанных подхода, когда ячейки конвейера специализированы частично для оптимизации архитектуры с точки зрения сценариев обработки пакетов, описанных в следующем разделе.

2. Описание сценариев обработки пакетов

В ходе работы был выделен набор наиболее общих сценариев обработки пакетов, на которые должно быть рассчитано проектируемое СПУ. В данном разделе приведено описание сценариев обработки пакетов, необходимых для выполнения функционального тестирования конвейера обработки данных и его модели. Такой конвейер может быть использован для разработки семейства ПКС коммутаторов и IP/MPLS маршрутизаторов.

Сценарии обработки пакетов можно разделить на четыре группы (в скобках представлены их ссылочные названия):

1. Сценарии работы классического L2-коммутатора с обучением:
   - простейший Ethernet-коммутатор для локальной сети (L2 switch);
   - Ethernet-коммутатор для виртуальных локальных сетей, протокол 802.1Q (L2 switch + VLAN).
2. Сценарии работы L2/L3 коммутатора:

- L2-коммутатор с обучением для виртуальных локальных сетей и маршрутизацией L3-unicast трафика (L2 switch + VLAN + L3-unicast);
- L2-коммутатор с обучением для виртуальных локальных сетей и маршрутизацией L3-multicast трафика (L2 switch + VLAN + L3-multicast).

3. Сценарии работы сетевых приложений на ПКС-коммутаторе:
   - сценарий обработки L3-multicast трафика (Multicast);
   - виртуальные домены B2C (для коммутаторов с ролями AR, DR), P2P, Multipoint.
4. Сценарии агрегирования, очередизации и перенаправления трафика на ПКС-коммутаторе:
   - сценарий агрегирования каналов (LAG);
   - сценарий разбора поля Ethertype для LACP/LLDP/QoS;
   - сценарий зеркалирования трафика (Mirror);
   - передача трафика уровня управления по каналам уровня передачи данных (Inband).

Далее опишем по одному сценарию из каждой группы.

2.1. Сценарий L2 switch

Данный сценарий соответствует работе простейшего L2-коммутатора для локальной сети. Сценарий предполагает наличие таблицы, которая содержит записи вида:

$$< \text{MAC-адрес} >, < \text{номер порта} > .$$

Каждая из таких записей означает, что за данным портом находится устройство с данным MAC-адресом.

В данном сценарии необходимо выполнить два действия (см. Рисунок 4). Во-первых, определить порт, в который пакет должен быть отправлен. Во-вторых, запомнить пару (MAC-адрес отправителя, номер порта), чтобы в дальнейшем эффективнее коммутировать пакеты.

Для выполнения первой задачи проводится поиск по таблице. Ключом поиска является MAC-адрес получателя. В случае успешного завершения поиска по таблице получаем искомый номер порта, в который будет отправлен пакет. Иначе данный пакет будет отправлен во все порты, кроме порта, через который этот пакет был получен.

Для выполнения второй задачи в той же таблице проводится поиск записи:

$$< \text{MAC-адрес отправителя} >, < \text{номер порта} > .$$

Если поиск завершился успешно, то у найденной записи обнуляется параметр age_time. Иначе такая запись добавляется в таблицу, и обнуляется соответствующий ей параметр age_time.

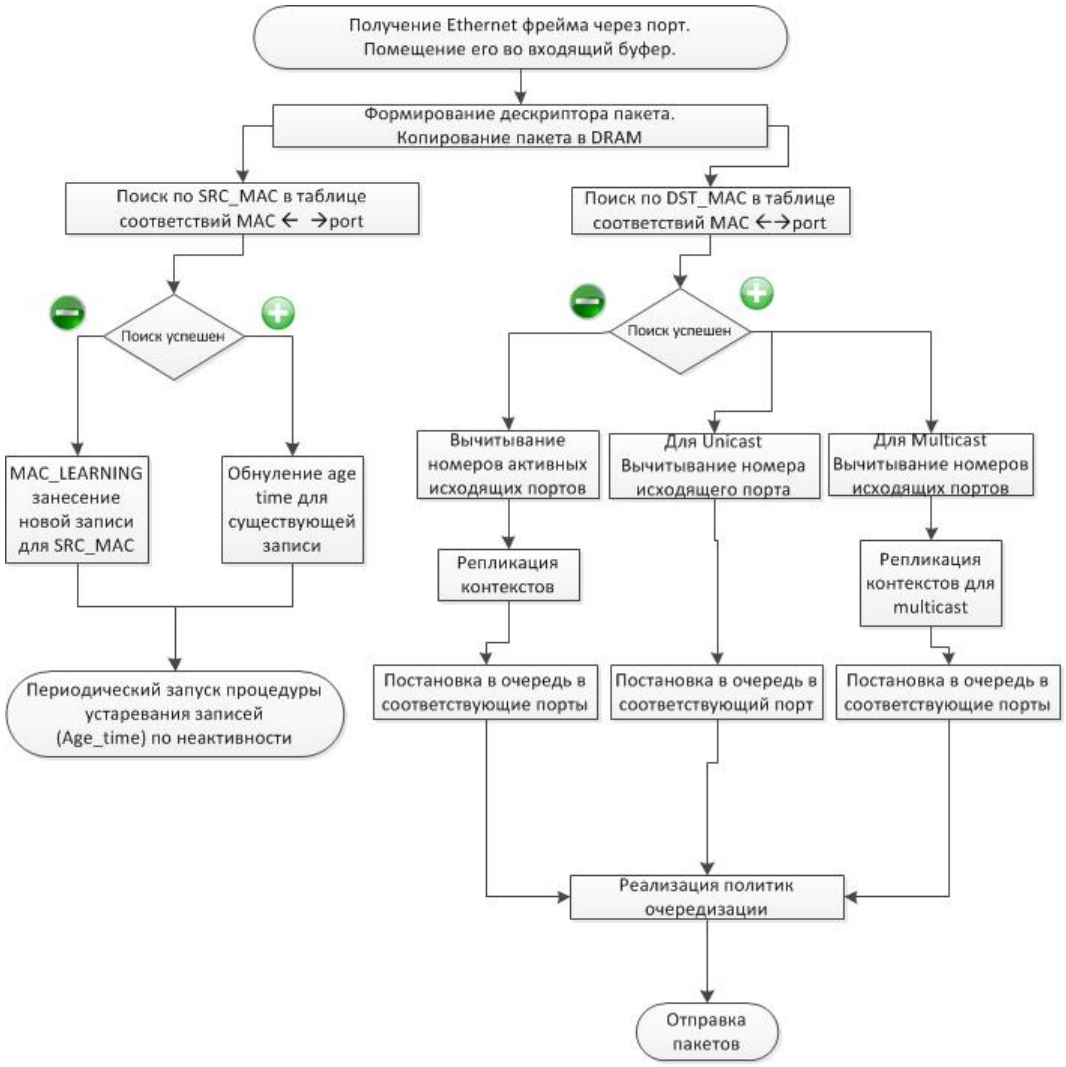


Рис. 4. Схема работы процессора для сценария L2 switch

Fig. 4. The use case scenario for L2-switch

Как правило, к сети постоянно подключаются новые устройства и отключаются старые. Поэтому на коммутаторе работает механизм поддержки актуальности записей в таблице. Для этого периодически запускается процедура устаревания записей по неактивности. Она просматривает все существующие записи, и если параметр age_time превышает заданный порог, то запись из таблицы удаляется.

2.2. Сценарий L2 switch + VLAN + L3-unicast

Данный сценарий является расширением сценария L2 switch за счет добавления дополнительной обработки VLAN-тегов, а также ветки маршрутизации пакета на уровне L3 [11], если полученный пакет необходимо маршрутизировать, а не коммутировать (см. Рисунок 5). Решение об этом принимается после присвоения пакету соответствующего номера vlan и снятия тега 802.1Q, если он присутствует в пакете.

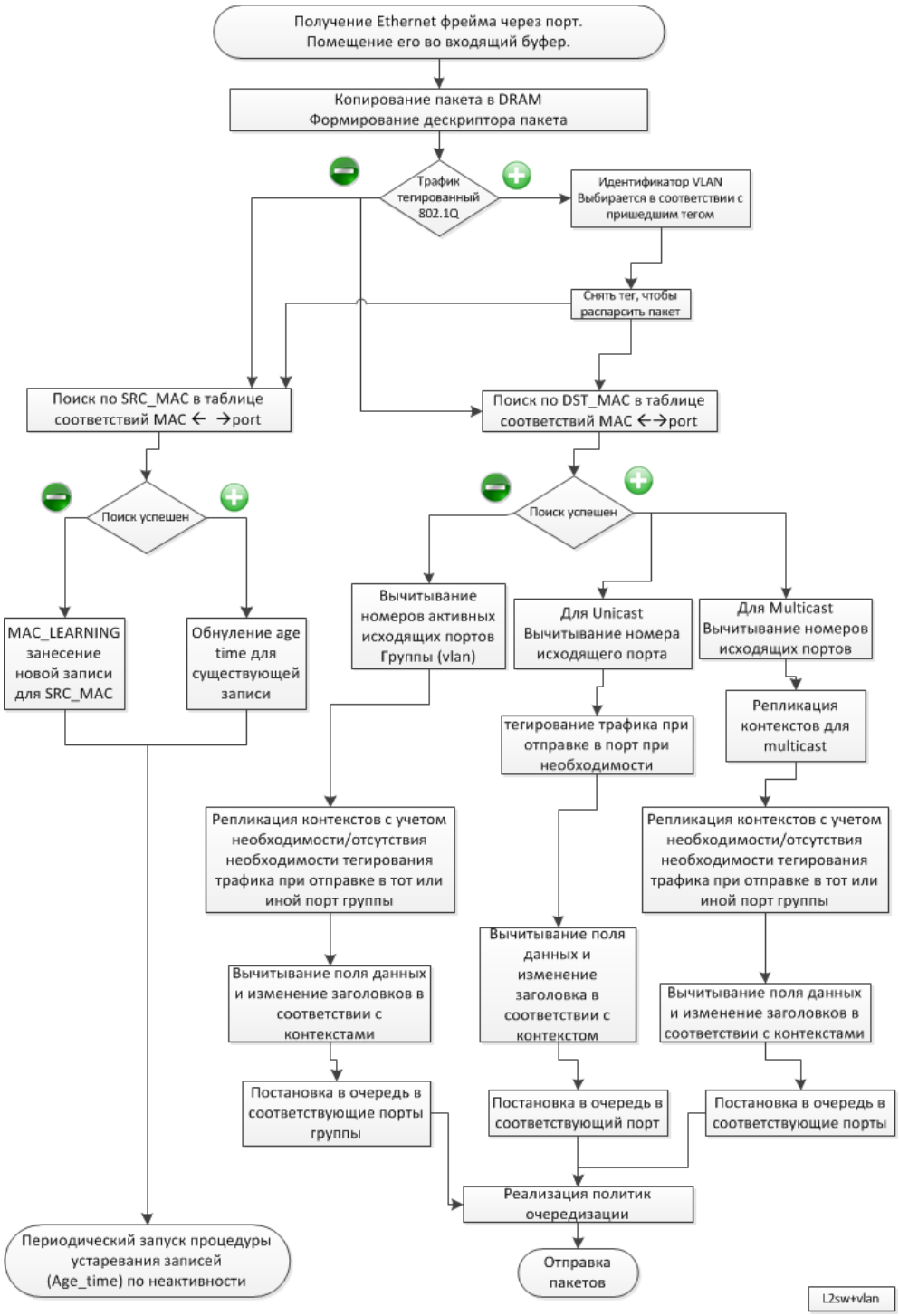


Рис. 5. Схема работы процессора для сценария L2 switch + VLAN + L3-unicast

Fig. 5. The use case for L2 switch + VLAN + L3-unicast

Для этого производится поиск по таблице, содержащей MAC-адреса получателей широковещательного трафика, которая заполняется при получении коммутатором IGMP запросов определенного типа. Если MAC-адрес получателя из принятого пакета был найден в таблице, то принимается решение о маршрутизации пакета. Ина-

че выполнение сценария идет по ветке коммутации, где все действия совпадают с описанным выше сценарием.

Далее рассчитывается контрольная сумма CRC заголовка L3. Если полученное значение не совпадает со значением поля контрольной суммы пакета, то он сбрасывается. Затем поле TTL уменьшается на единицу, и если оно стало равно нулю, то пакет сбрасывается. Далее необходимо определить, в какие порты отправить данный пакет. Для этого есть специальная таблица Multicast маршрутизации, которая содержит записи вида:

$$< \text{IP-адрес получателя} >, < \text{IP-адрес отправителя} >,$$

$$< \text{номер входящего интерфейса} >, < \{\text{список исходящих интерфейсов}\} > .$$

Ключом поиска является IP-адрес получателя, содержащийся в заголовке пакета. Если поиск был неуспешен, пакет сбрасывается. Иначе рассчитывается новая контрольная сумма L3, так как ранее поле TTL было уменьшено, и в L2 заголовке данного пакета MAC-адрес отправителя заменяется на MAC-адрес исходящего L3 интерфейса коммутатора. Затем происходит репликация контекстов для отправки в каждый порт с учетом необходимости или отсутствия необходимости тегирования заголовка пакета.

2.3. Сценарий B2C-AR

Сценарии виртуального домена B2C отвечают за предоставление отказоустойчивого сервиса клиенту. Клиент идентифицируется по MAC-адресу и VLAN-тегу. Для клиента, подключенного к коммутатору, реализующему сценарий B2C-DR, предоставляется сервис (например, доступ в Интернет) через коммутатор, реализующий сценарий B2C-AR. Коммутатор AR выбирает работающий порт по VLAN трафика клиента, и передает трафик на него. Этот сценарий требует одной таблицы, которая содержит поля:

$$< \text{номер порта} >, < \text{номер vlan} > .$$

Поиск по таблице ведется по принципу полного совпадения полей. В случае неудачи пакет передается на обработку по другой схеме (см. Рисунок 6).

Если поиск завершается успешно, то далее проводится поиск по групповой таблице. Группа в этом сценарии — множество портов, из которого по специальному алгоритму выбирается один порт, куда будет отправлен пакет. Алгоритм состоит в том, что сначала проверяется порт, указанный первым в списке соответствующей группы. Если он находится в рабочем состоянии, пакет отправляется на этот порт. Если он в неработоспособном состоянии, то проверяется следующий по списку порт. Эти действия продолжаются до тех пор, пока не будет найден работающий порт. Если все порты оказываются в нерабочем состоянии, то пакет сбрасывается.

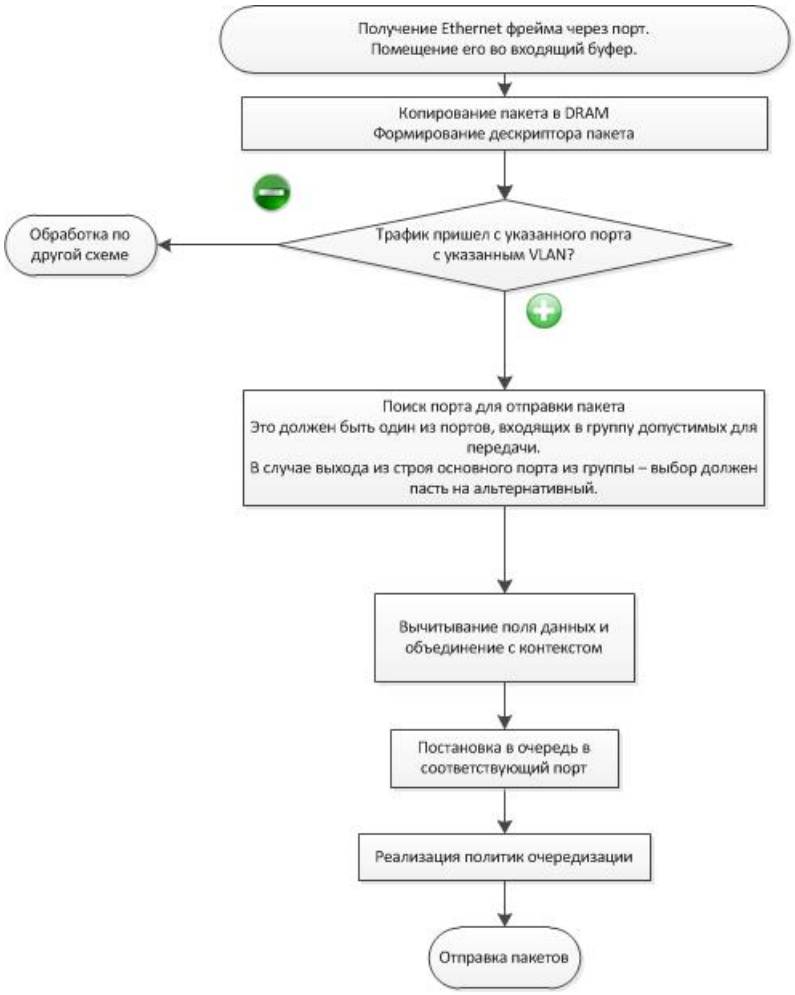


Рис. 6. Схема работы процессора для сценария B2C-AR

Fig. 6. The use case scenario for B2C-AR

2.4. Сценарий Mirror

Сценарий Mirror реализует зеркалирование трафика на коммутаторе программно-конфигурируемой сети. Для данного сценария достаточно одной таблицы, которая содержит поля:

$$< \text{номер порта} >, < \text{s\_vlan} >, < \text{c\_vlan} >, < \text{набор действий} > .$$

Обработка пакета происходит следующим образом (см. Рисунок 7). Сначала происходит проверка наличия тега 802.1Q в заголовке пакета. Если он есть, то значение vlan запоминается в контексте как s_vlan, а тег снимается. Далее проверяется наличие второго тега и происходят аналогичные действия, а значение vlan запоминается как c_vlan. В результате заголовок пакета не содержит тегов, и все нужные значения записаны в контекст. На следующем шаге просматривается описанная выше таблица, в которой ищутся полные совпадения по полям. В случае успеха производится копирование пакета, и к данной копии добавляются теги при их наличии в

изначальном пакете, а затем она отправляется на нужный порт. Тело оригинального пакета передается на обработку по другой схеме.

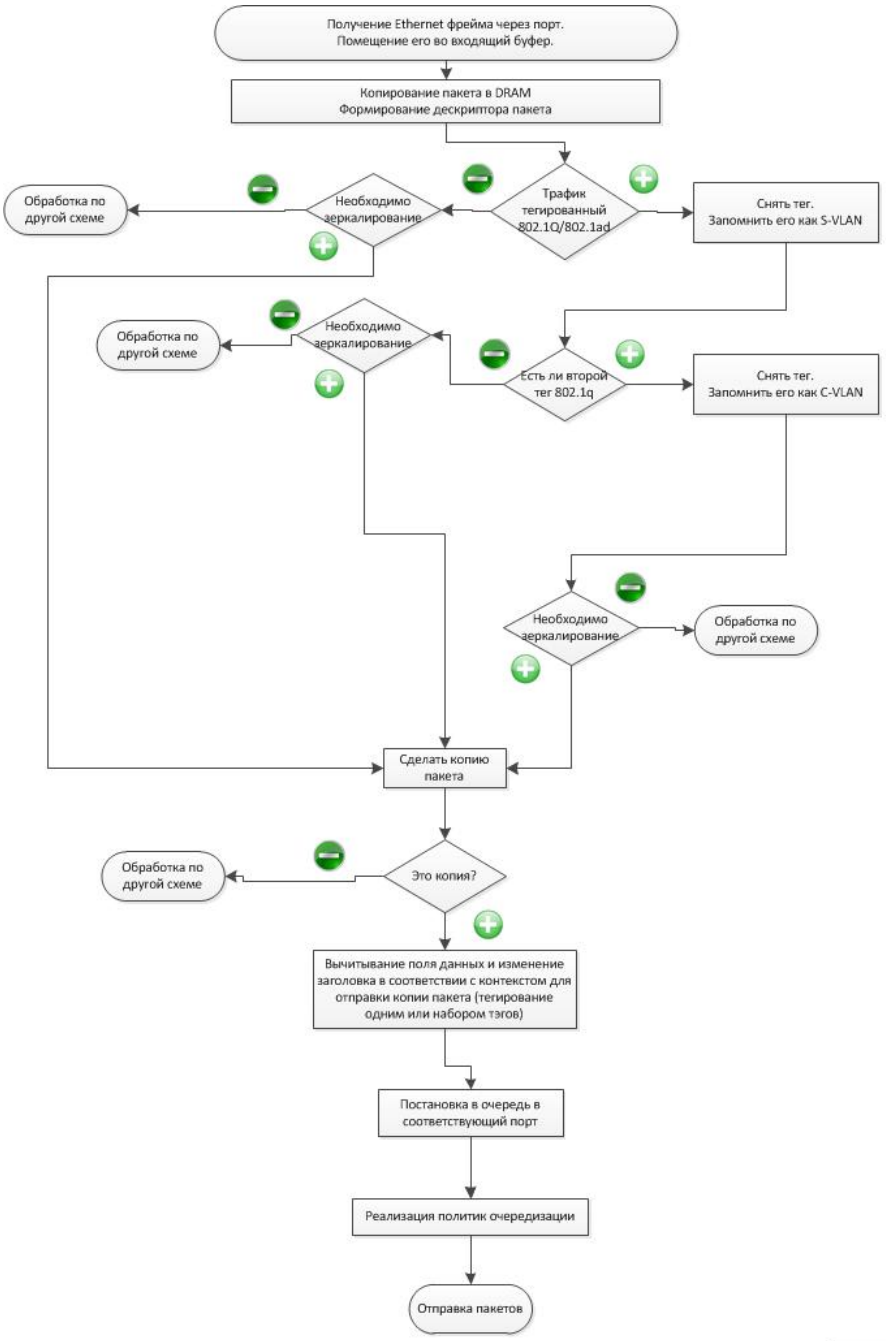


Рис. 7. Схема работы процессора для сценария Mirror

Fig. 7. The use case scenario for Mirror

3. Описание предлагаемой архитектуры

В данном разделе представлено описание архитектуры СПУ, основанного на конвейере обработки данных с функционально специализированными ячейками (см. Рисунок 8).

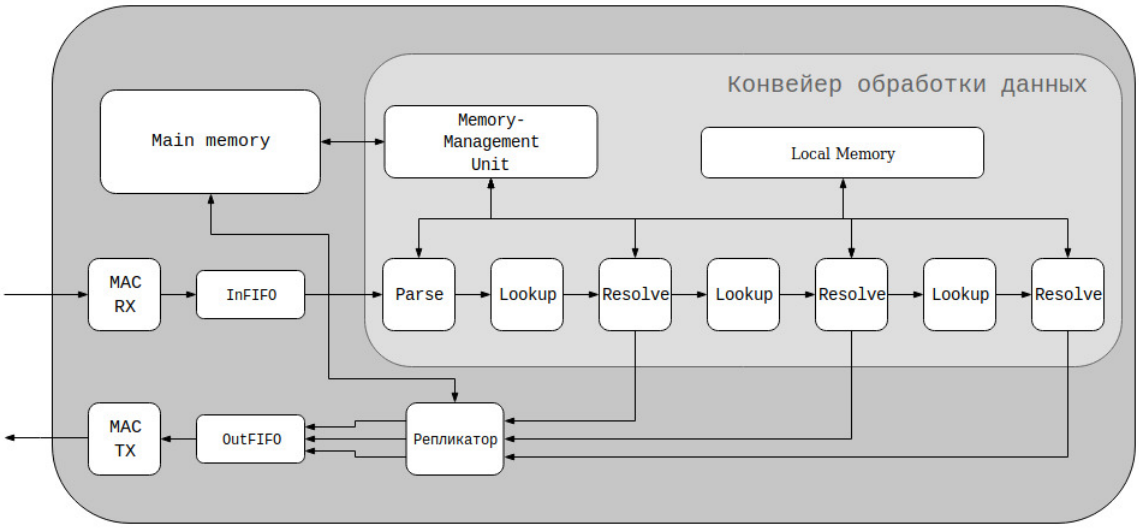


Рис. 8. Описание предлагаемой архитектуры

Fig. 8. Proposed NPU architecture

Основными блоками предлагаемого СПУ являются следующие:

- MAC_RX – ресивер интерфейса Ethernet.
- InFifo/OutFifo – буферизирующие устройства входных/выходных пакетов.
- Конвейер обработки данных - конвейер с семью ячейками, состоящий из ячеек следующих типов:
  - Parse – считывание данных заголовка пакета, необходимых для классификации пакета.
  - Lookup – поиск подходящей записи в одной из таблиц потоков (классификация).
  - Resolve – обработка результата поиска: выполнение модификаций заголовка и принятие решения о завершении обработки пакета либо о её продолжении.
- Resolve – обработка результата поиска: выполнение модификаций заголовка и принятие решения о завершении обработки пакета либо о её продолжении.
- Memory-Management unit – блок управления доступом к основной памяти.
- Main Memory – основная память, предназначенная для хранения тел пакетов.

- Local Memory - внутренняя память, предназначенная для хранения контекстов, таблиц маршрутизации.
- Репликатор – блок, выполняющий репликацию пакетов.
- MAC_TX – трансмиттер интерфейса Ethernet.

При прохождении конвейера обработки данных с пакетом ассоциируется контекст, который далее передается блоками друг другу и модифицируется ими. Таким образом, фактически по тракту перемещается только контекст пакета. Контекст состоит из заголовка пакета и/или сопутствующих метаданных (как стандартных специфицированных, так и заданных разработчиком). Состав контекста, т.е. наборы полей входных и выходных метаданных, для различных блоков отличается. Любая копия пакета обрабатывается как отдельный пакет со своим собственным контекстом.

Блок MAC_RX отвечает за формирование фреймов. Преобразование фрейма включает в себя конвертацию битов в данные пакета посредством группировки битов в логические единицы (фреймы, ячейки и пакеты). В зависимости от используемого протокола логические единицы определяются как фреймы, ячейки и пакеты. Блок MAC_TX является зеркальным отображением блока MAC_RX, то есть выполняет все функции блока MAC_RX с точностью до наоборот. Он отвечает за «расщепление» фрейма на биты и дальнейшую передачу последовательностей битов на физический интерфейс. Блоки InFIFO/OutFIFO представляют собой логические устройства, осуществляющие буферизацию пакетов, помещая их во входную и выходную очереди соответственно.

Конвейер обработки данных состоит из семи функционально специализированных ячеек. Все ячейки можно разбить на три типа: Parse, Lookup и Resolve. Каждый из этих типов ячеек предназначен для определенных действий:

1. Тип ячейки Parse – анализ заголовка пакета, необходимого для дальнейшей классификации пакета, запись данных в контекст, передача контекста в следующую ячейку.
2. Тип ячейки Lookup – классификация пакета по данным из контекста (поиск в одной из таблиц), запись результата поиска в контекст пакета и передача в следующую ячейку.
3. Тип ячейки Resolve – в случае успешного поиска выполнение преобразования контекста в соответствии с записью таблицы и передача контекста в следующую ячейку, если обработка еще не завершена, либо обработчику очередей, если обработка завершена и указан(ы) порт(ы) для отправки. В случае неудачного поиска выполняется действие по умолчанию (например, отправка на порт контроллера).

Блок Memory-Management Unit (MMU) отвечает за размещение тела пакета в основную память. Являясь посредником между конвейером и основной памятью, блок MMU соединен с некоторыми ячейками конвейера, однако для каждой такой ячейки доступен не весь интерфейс блока MMU. Этот интерфейс состоит из следующих действий: выделение буфера для тела пакета, получение указателя на этот

буфер с целью записи или чтения, размещение тела пакета в буфере, освобождение буфера. Ниже приведены действия, выполнимые MMU для каждой ячейки:

- Типу ячейки Parse доступны операции:
  - выделение буфера для тела пакета;
  - получение указателя на буфер для записи;
  - запись – размещение тела пакета в буфере;
  - освобождение буфера;
- Типы ячейки Lookup не имеют доступа к основной памяти.
- Типы ячейки Resolve не имеют доступа к основной памяти.
- Репликатору доступны операции:
  - получения указателя на буфер для чтения;
  - считывание пакета из буфера;
  - освобождение буфера.

Получив контекст пакета, предназначенный для отправки на порт, репликатор анализирует его. Репликатор считывает весь пакет из основной памяти и освобождает память из-под считанного пакета. Если необходимо копирование пакета, например, для широковещательной или мультивещательной рассылки, создается указанное количество копий с соответствующими изменениями заголовка. Вся необходимая репликатору информация содержится в контексте пакета. Результирующие пакеты передаются выходному буферизирующему устройству.

4. Описание модели

4.1. Описание имитационной модели

Предлагаемая имитационная модель работы СПУ выполняет следующие действия:

- передача поступившего пакета сетевому процессору;
- запись поступившего пакета в локальную память;
- обработка заголовка пакета;
- в зависимости от результата обработки принятие решения о дальнейшей отправке пакета на соответствующие порты коммутатора;
- проверка работы сетевого процессора.

Рассматриваемая модель состоит из трех компонентов:

1. поставщика (provider);

2. проверяемого устройства (DUT, Device Under Test) – СПУ;

3. потребителя (consumer).

Все три компонента модели связаны между собой каналами, организованными по принципу FIFO (см. Рисунок 9).

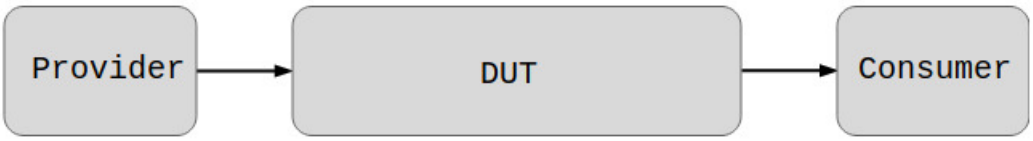


Рис. 9. Основные компоненты модели сетевого процессора

Fig. 9. Main components of the network processor model

Поставщик передает пакеты на обработку сетевому процессору, имитируя поступление пакетов на порты коммутатора. Поставщик каждый такт с начала симуляции поставляет пакет и номер порта коммутатора, на который поступил пакет, и отправляет его СПУ. СПУ, получив пакет и номер порта, начинает его обработку. Потребитель, в свою очередь, ответственен за проверку результатов работы СПУ. Потребитель связан каналами с каждым из этапов обработки пакета, по которым передается информация о том, на каком этапе завершилась обработка пакета.

4.2. Реализация модели

Имитационная модель сетевого процессора реализована на языках C/C++ с использованием открытой C++ библиотеки SystemC [9]. Базовым элементом в SystemC является модуль, включающий в себя процессы и другие модули. Модель в SystemC состоит из нескольких модулей, которые общаются друг с другом через порты. В нашем случае модель состоит из трех основных модулей (блоков): Provider, DUT и Consumer. Каждый блок оснащен портами, с помощью которых происходит взаимодействие между блоками. В построенной модели порты каждого блока делятся на два типа: порты, по которым блок получает данные, и порты, через которые блок отсылает данные. Коммуникация между блоками осуществляется с помощью библиотеки SystemC TLM. Блоки соединены друг с другом однонаправленными каналами из библиотеки TLM.

Блок Provider посылает блоку DUT каждый такт сообщение, содержащее пакет и номер порта, на который якобы пришел пакет. Блок DUT начинает свою работу, как только сообщение поступило на его входящий порт. Блок Consumer аналогично блоку DUT начинает работу, как только получает пакет на входящие порты. Во время моделирования процессы, соответствующие блокам, выполняются асинхронно по отношению друг к другу. В представленной модели процессы прерываются только в двух случаях: задержки, моделирующей время работы отдельного блока, и задержки, связанной с ожиданием поступления сообщения на входные порты блока.

Типы ячеек конвейера и остальные объекты представлены в виде C++ классов, экземпляры которых, в свою очередь, являются полями класса DUT.

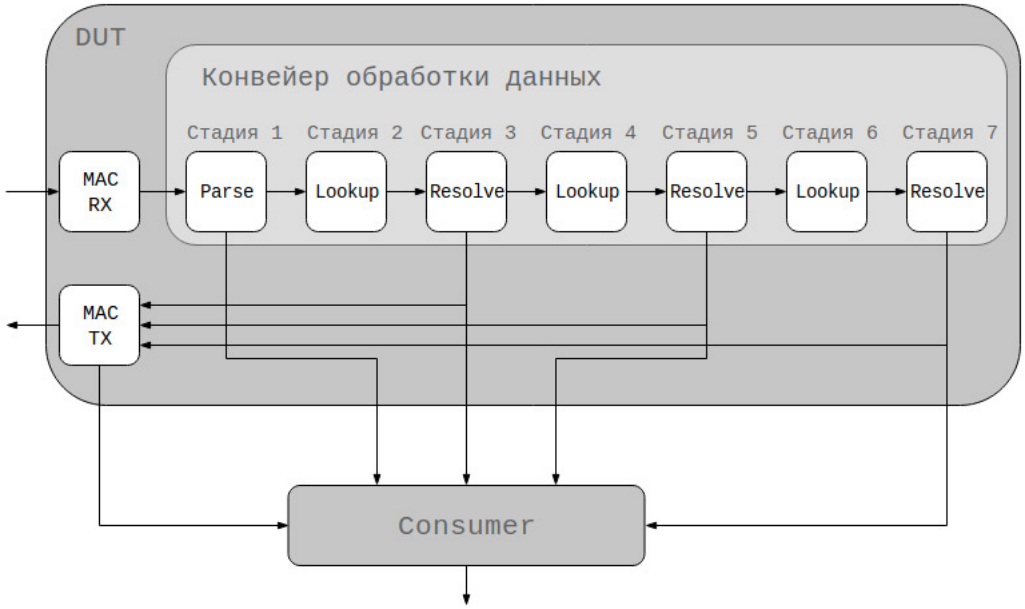


Рис. 10. Схема каналов, по которым передаются сообщения о сбросе пакета

Fig. 10. Notification channels between consumer and NPU units in implemented simulation model

На Рисунке 10 представлены внутренние связи блока DUT, а также связи между блокоми DUT и Consumer. Связи между блокоми MMU и DUT опущены.

В модели используются динамические библиотеки, содержащие реализацию всех ячеек типов Parse, Lookup и Resolve для каждого сценария. При запуске модели указывается, какой сценарий будет промоделирован. Выбор сценария осуществляется выбором реализаций ячеек типов Parse, Lookup и Resolve из динамической библиотеки. Также во время запуска в качестве опций задаются следующие параметры: период тактового сигнала, количество тактов, в течение которого происходит моделирование, набор задержек для каждой ячейки. Если данные параметры при запуске не указываются, то при моделировании используются значения этих параметров по умолчанию.

4.3. Реализация сценариев обработки пакетов

Сценарии обработки пакетов, представленные в разделе 2., реализованы на языке C. Код для каждого сценария состоит из функций обработки пакета, соответствующих ячейкам конвейера и реализованных с использованием следующего интерфейса (API) модели СПУ:

- библиотека функций и структур для работы с памятью (такие, как memcpy или memcmp);
- библиотека функций, имитирующих получение и передачу пакетов с физического интерфейса на коммутатор;

- библиотека функций, вычисляющих контрольную сумму L3-заголовка и хэш-функций для адресов;
- библиотека контекстов;
- функции, обрабатывающие результаты поиска и принимающие решение о дальнейших действиях, таких, как
  - удаление, добавление или замена тега;
  - модификация полей пакета;
  - копирование пакета;
  - размещение пакета в буфер коммутатора;
  - отправка контекста на следующую ячейку конвейера;
  - сброс пакета.

5. Оценка производительности

Данный раздел содержит результаты оценки производительности рассматриваемой архитектуры СПУ.

5.1. Экспериментальный трафик

При проведении тестовых прогонов учитывался опыт нагрузочного и функционального тестирования, проводимого в ПАО Ростелеком.

Распределение трафика при тестировании:

1. Трафик групповой адресации (сценарий Multicast): 400 потоков (с разной пропускной способностью, то есть с разным качеством) давали 1.5 Гбит/с.
2. B2C трафик (сценарий B2C-AR B2C-DR): 100 потоков при идентификации их по внешнему VLAN давали 3.5 Гбит/с (реально порядка 10000 потоков).
3. B2B трафик (сценарий P2P): 50 потоков при идентификации их по внешнему VLAN потоков давали 4 Гбит/с (реально порядка 2500 потоков).
4. B2B трафик (сценарий MP): 1000 потоков с разными MAC давали 1 Гбит/с.
5. Inband трафик (сценарий INBAND): до 1000 пакетов в секунду на сети из 5 коммутаторов (до 200 пакетов/с – обмен между контроллером и CPU коммутатора).
6. LLDP трафик (сценарий LLDP): на каждый интерфейс каждого коммутатора по 1 пакету в секунду (тоже своего рода обмен между контроллером и коммутаторами).

Кроме того, в ходе данного исследования был адаптирован и использован набор программных средств, разработанных компанией KM211. Также рассматривалась возможность использования семейства микроконтроллеров KMX32 (спроектированных компанией KM211).

5.2. Методика получения оценок

Для измерения времени работы процессора с применением процессоров семейства KMX32 применялась следующая методика:

1. Измерение времени работы функций, вызываемых на каждой из ячеек, для каждого теста сценария на эмуляторе процессора KMX32.

2. Запуск имитационной модели сетевого процессора с каждым из тестов. В качестве исполняемых ячейками функций на вход модели подаются адреса из динамической библиотеки функций сценария. В качестве задержек работы каждой из ячеек указываются значения, полученные на шаге 1.

3. Измерение итогового времени работы на каждой из ячеек с учётом моделируемых задержек.

Измерение пессимистичной оценки пропускной способности конвейера представленной архитектуры производилось для пакетов размером 80 байт (в худшем случае). В каждом сценарии выделялось узкое место, т.е. ячейка конвейера, требующая наибольшего времени для обработки контекста. Далее подсчитывалось число пакетов, обработанных за одну секунду, и умножалось на размер пакета (в худшем случае). Полученное значение пропускной способности для каждого сценария является искомой средней пропускной способностью.

5.3. Результаты оценок

В данном разделе представлены результаты оценки производительности, проведенной по вышеописанной методике, для каждого сценария (см. Таблица 1 – Таблица 8), описанного в разделе 2.

Таблица 1. Время обработки пакета для сценария: a) B2C-DR, b) Mirror

Table 1. Packet processing time for a) B2C-DR and b) Mirror scenarios

a)

Test	**RX**	**P**	**L**	**R**	**TX**
1	6495	311	175	94	
2	6549	313	696	1388	2540
3	6551	309	1932	3922	7255
4	6547	309	1445	1388	2539
5	6547	309	1172	1388	2539
6	5945	278	1335	94	

b)

Test	**RX**	**P**	**L**	**R**	**TX**
1	6146	195	353	3043	1794
2	6488	1583	298	4453	1874
3	6858	3195	243	5891	1954
4	6148	192	253	51	

Для сценариев, представленных в Таблицах 3 – 8, в некоторых тестах происходит отправка нескольких пакетов.

В Таблицах 1 – 2 столбцы соответствуют ячейкам конвейера обработки данных, ресиверу (блок MAC_RX) и трансмиттеру (MAC_TX), а строки – тестам, запущенным на предложенной модели конвейера.

Таблица 2. Время обработки пакета для сценария Multicast

Table 2. Packet processing time for Multicast scenario

Test	**RX**	**P**	**L1**	**R1**	**L2**	**R2**	**TX**
1	6165	261	523	110	208	5895	3530
2	6219	231	259	1340			1778
3	6251	263	336	55			

Таблица 3. Время обработки пакета для сценария Inband

Table 3. Packet processing time for Inband scenario

Test	**RX**	**P**	**L**	**R**	**TX**
1	3687.2	480.8	1328	846	1194
2	6281.8	474	1292	1310	1834
3	6275	474	1160	48	
4					
packet 1	3739.2	3.8	1658	846	1194
packet 2	3694	448	1997	847	1193

Таблица 4. Время обработки пакета для сценария B2C-AR

Table 4. Packet processing time for B2C-AR scenario

Test	**RX**	**P**	**L**	**R**	**TX**
Failover					
packet 1	6495	224	358	1369	2546
packet 2	6549	196	358	1368	2540
packet 3	6500	200	451	1368	2540
packet 4	6496	200	544	1368	2540
No match					
packet 1	6504	226	146	48	
packet 2	6551	226	146	48	

В полях таблиц представлено время, затраченное блоком, соответствующим названию столбца, на обработку одного пакета при выполнении теста, указанного в строке. Результаты представлены в наносекундах. Пустые поля в таблице свидетельствуют о том, что соответствующий блок не был задействован при обработке пакета.

На основе результатов экспериментов, представленных в Таблицах 1 – 8, для каждого сценария был произведен расчёт пессимистичной оценки пропускной способности СПУ. Таблица 9 содержит результаты для различных сценариев и вариантов построения СПУ.

Таблица 5. Время обработки пакета для сценария P2P

Table 5. Packet processing time for P2P scenario

Test	**RX**	**P**	**L**	**R**	**TX**
Failover					
packet 1	6495	261	722	1466	2521
packet 2	6549	232	723	1465	2515
packet 3	6500	236	1156	1466	2514
packet 4	6496	237	1589	1466	2514
No match					
packet 1	6504	263	1679	48	
packet 2	6551	262	165	48	
packet 3	6551	262	165	48	

Таблица 6. Время обработки пакета для сценария LACP/LLDP/QoS

Table 6. Packet processing time for LACP/LLDP/QoS scenario

Test	**RX**	**P**	**L**	**R**	**TX**
Tag					
packet 1	6795.2	3221.8	1	1153	
packet 2	6508.8	1644	1	1146	
Ethernet	6221	223	1	1146	
LACP/LLDP					
packet 1	6302	181	1	1289	2515
packet 2	6302	161	1	1289	2514

Таблица 7. Время обработки пакета для сценария LAG

Table 7. Packet processing time for LAG scenario

Test	**RX**	**P**	**L**	**R**	**TX**
1	6197	62	167	48	182
2					
packet 1	6190	68	994	1293	1785
packet 2	6175	62	1036	1294	1784
packet 3	6173	62	1107	1294	1784
packet 4	6170	62	1330	1319	1785

Данные результаты показывают, что в пессимистичном варианте при трафике, состоящем преимущественно из пакетов небольшого размера, такой вариант архитектуры не достигает требуемых показателей. Таким образом, для достижения необходимых показателей нужна доработка имеющихся процессорных ядер.

Таблица 8. Время обработки пакета для сценария L2 switch + VLAN

Table 8. Packet processing time for L2 switch + VLAN scenario

Test	**RX**	**P**	**L1**	**R1**	**L2**	**R2**	**L3**	**R3**	**TX**
1									
packet 1	6209	328	205	188	477	817	936	11926	12830
packet 2	6474	441	193	186	477	843	735	4793	4714
2	6529	437	193	187	476	843	873	6457	8771
3	6235	326	212	188	476	843	880	3293	8570
4	6295	354	205	188	477	843	894	3293	8570
5									
packet 1	6525	439	193	70					
packet 2	6528	441	193	70					
6	6240	358	205	188	476	843	839	1709	4713

Таблица 9. Оценка пропускной способности СПУ для рассмотренных сценариев

Table 9. NPU pipeline throughput for considered scenarios

Сценарии	**1 конвейер × 1 порт, Мбит/с**	**2 конвейера × 1 порт, Мбит/с**	**1 конвейер × 24 порта, Гбит/с**	**2 конвейера × 24 порта, Гбит/с**
B2C-AR	97.7	195.4	2.3	4.7
B2C-DR	97.7	195.4	2.3	4.7
Inband	101.9	203.8	2.4	4.5
LACP/ LLDP/ QoS	94.2	188.4	2.2	4.5
LAG	103.3	206.6	2.5	5
Mirror	93.3	186.6	2.2	4.5
Multicast	102.3	204.6	2.5	4.9
P2P	97.7	195.4	2.3	4.7
L2 swtich VLAN	98	196	2.3	4.7

Заключение

Целью данной работы было изучение различных подходов к разработке архитектуры сетевого процессора и предложение подхода к созданию СПУ. Были изучены архитектуры конвейеров СПУ EZChip NP-5 и Barefoot Tofino. Учитывая их достоинства и недостатки, была предложена архитектура конвейера СПУ, ориентированная на работу со сценариями, описанными в разделе 2. В работе предложена архитектура сетевого процессорного устройства на основе разработанного конвейера. Для тестирования сетевого процессора на наборе сценариев обработки пакета была разработана его имитационная модель на языке C ++. В качестве ядер процессора

в модели использовался процессор KMX32, разработанный компанией KM211. Затем было проведено экспериментальное исследование, результаты которого были использованы для оценки времени выполнения каждой ячейки конвейера.

Результаты исследований показали, что скорость СПУ позволяет применить их в коммутаторах уровня распределения в трехуровневой иерархической сети. Для использования СПУ в коммутаторах ядра сети требуется дальнейшее совершенствование предлагаемой архитектуры. Это возможно достичь двумя способами. Первый из них заключается в усовершенствовании существующих процессорных ядер с точки зрения добавления специализированных программируемых устройств для обработки сетевого трафика. Другой способ – создать такое специализированное программируемое устройство с нуля.

Список литературы / References

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

[2] Смелянский Р. Л., “Технологии реализации программно конфигурируемых сетей: Overlay vs OpenFlow”, *Журнал сетевых решений LAN*, 2014, №4, 53–55; [Smeliansky R.L., “Tekhnologii realizacii programmno konfiguriruemyh setej: Overlay vs OpenFlow”, *Zhurnal setevyh reshenij LAN*, 2014, №4, 53–55, (in Russian).]

[3] Kornaros G., *Multi-core embedded systems*, Boca Raton, FL: CRC Press, 2010.

[4] Cisco Networking Academy, *Connecting Networks Companion Guide*, Cisco Press, 2014.

[5] *EZchip NP-5 Product Brief*, http://www.ezchip.com.

[6] Bosshart P., et al., “P4: Programming protocol-independent packet processors”, *ACM SIGCOMM Computer Communication Review*, **44**:3 (2014), 87–95.

[7] “Tofino: World’s fastest P4-programmable Ethernet switch ASICs”, *Barefoot*, https://barefootnetworks.com/products/brief-tofino/.

[8] Kaushalram A., Budiu M., Kim C., *Data-plane stateful processing units in packet processing pipelines*, US Patent App 14864088, 2017, http://www.freepatentsonline.com/y2017/0093987.html.

[9] Accellera Standarts: SystemC, http://www.accellera.org/downloads/standards/systemc.

[10] “Семейство микроконтроллеров KMX32”, http://km211.ru; [“KMX32 family microcontrollers”, http://km211.com, (in Russian).]

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

Bezzubtsev S. O., Vasin V. V., Volkanov D. Yu., Zhailauova Sh. R., Miroshnik V. A., Skobtsova Yu .A., Smeliansky R. L., "An Approach to the Construction of a Network Processing Unit", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 39–62.

DOI: 10.18255/1818-1015-2019-1-39-62

Abstract. The paper proposes the architecture and basic requirements for a network processor for OpenFlow switches of software-defined networks. An analysis of the architectures of well-known network processors is presented – NP-5 from EZchip (now Mellanox) and Tofino from Barefoot Networks. The advantages and disadvantages of two different versions of network processor architectures are considered: pipeline-based architecture, the stages of which are represented by a set of general-purpose processor cores, and pipeline-based architecture whose stages correspond to cores specialized for specific packet processing operations. Based on a dedicated set of the most common use case scenarios, a new architecture of the network processor unit (NPU) with functionally specialized pipeline stages was proposed. The article presents a description of the simulation model of the NPU of the proposed architecture. The simulation model of the network processor is implemented in C ++ languages using SystemC, the open-source C++ library. For the functional testing of the obtained NPU model, the described use case scenarios were implemented in C. In order to evaluate the performance of the proposed NPU architecture a set of software products developed by KM211 company and the KMX32 family of microcontrollers were used. Evaluation of NPU performance was made on the basis of a simulation model. Estimates of the processing time of one packet and the average throughput of the NPU model for each scenario are obtained.

Keywords: network processor, network processing unit, switch, computer networks, SDN, computer architecture, simulation modeling, Open Flow protocol

On the authors:
Stanislav O. Bezzubtsev, technical expert, orcid.org/0000-0002-2419-7632
Applied Research Center for Computer Networks,
1, bd. 77 Leninskie Gory, Moscow, 119992 Russia, e-mail: stas.bezzubtsev@gmail.com

Vyacheslav V. Vasin, senior software developer, orcid.org/0000-0002-5759-3105
Applied Research Center for Computer Networks,
1, bd. 77 Leninskie Gory, Moscow, 119992 Russia, e-mail: vvasin@arccn.ru

Dmitry Yu. Volkanov, PhD, assistant professor, orcid.org/0000-0001-9940-5822
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: volkanov@lvk.cs.msu.su

Shynar R. Zhailauova, PhD student, orcid.org/0000-0001-5725-7040
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: szhaylau@cs.msu.ru

Vladislav A. Miroshnik, software developer, orcid.org/0000-0003-0883-0116
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: miroshnikv@lvk.cs.msu.su

Yuliya A. Skobtsova, M.A. student, orcid.org/0000-0001-8351-3191
Lomonosov Moscow State University,
1 Leninskie Gory str., Moscow, Russia, e-mail: xenerizes@lvk.cs.msu.su

Ruslan L. Smeliansky, Corresponding Member of Russian Academy of Sciences, professor, doctor of sciences,
orcid.org/0000-0003-2311-4513
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: smel@cs.msu.su

Acknowledgments:
This work was supported by the Russian Foundation for Basic Research, Grant No 19-07-01076.

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 63–74
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 63–74

DOI: 10.18255/1818-1015-2019-1-63-74

УДК 004.72

Визуальная веб-ориентированная среда динамического управления потоками данных в кампусных программно-конфигурируемых сетях

Корячко В. П., Перепелкин Д. А., Иванчикова М. А., Бышов В. С.

Поступила в редакцию 10 января 2019
После доработки 14 февраля 2019
Принята к публикации 18 февраля 2019

Аннотация. В настоящее время в области компьютерных сетей (КС) широкую популярность получают инновационные подходы, основанные на технологии программно-конфигурируемых сетей (ПКС). ПКС позволяют обеспечить гибкий подход в обработке и управлении потоков данных в КС за счет разделения плоскости управления и передачи данных, а также централизации представления всей сети. В данной работе предложен прототип программной инфраструктуры и визуальной веб-ориентированной среды (ПИВС) динамического управления потоками данных в ПКС на основе протокола OpenFlow. Предложено использовать ПИВС в качестве интегрированного сегмента кампусной сети Рязанского государственного радиотехнического университета. Целью работы является разработка архитектуры ПИВС в виде описания UML диаграмм классов, а также создание программных методов для организации эффективного сетевого взаимодействия различных программных систем в ПКС на основе протокола OpenFlow. Для подтверждения эффективности и надежности предложенной ПИВС разработан программно-аппаратный стенд на базе оборудования HP Aruba 2920-24G. Предлагаемая в работе ПИВС является основой для разработки большого класса программных систем и компонентов ПКС на основе протокола OpenFlow.

Ключевые слова: программно-конфигурируемые сети, программная инфраструктура, визуальная веб-ориентировнная среда, OpenDayLight, ПИВС, SDN Topology, сетевое взаимодействие, программный компонент

Для цитирования: Корячко В. П., Перепелкин Д. А., Иванчикова М. А., Бышов В. С., "Визуальная веб-ориентированная среда динамического управления потоками данных в кампусных программно-конфигурируемых сетях", *Моделирование и анализ информационных систем*, **26**:1 (2019), 63–74.

Об авторах:
Корячко Вячеслав Петрович, orcid.org/0000-0003-0272-673X, д-р техн. наук, профессор,
Рязанский государственный радиотехнический университет
ул. Гагарина, 59/1, г. Рязань, 390005 Россия, e-mail: koryachko.v.p@rsreu.ru

Перепелкин Дмитрий Александрович, orcid.org/0000-0003-4775-5745, д-р техн. наук, профессор,
Рязанский государственный радиотехнический университет, e-mail: dmitryperepelkin@mail.ru

Иванчикова Мария Александровна, orcid.org/0000-0002-9615-2898, аспирант,
Рязанский государственный радиотехнический университет, e-mail: ivanchikova.masha@yandex.ru

Бышов Владимир Сергеевич, orcid.org/0000-0001-7673-9803, аспирант,
Рязанский государственный радиотехнический университет, e-mail: b.v.s.12@yandex.ru

Благодарности:
Работа выполнена при финансовой поддержке гранта Президента РФ МД-1826.2019.9.

Введение

В настоящее время идеи и подходы технологии программно-конфигурируемых сетей (ПКС) возродили интерес научного сообщества к программируемым сетям, предлагая сделать процессы проектирования и контроля компьютерными сетями (КС) более инновационными и упрощенными по сравнению с современным общепризнанным подходом. Высокий уровень сложности проектирования и управления современными КС делает данную задачу труднорешаемой. Тесные связи между управляющим уровнем сети (где принимаются решения об обработке трафика) и передающим уровнем сети (где происходит фактическая пересылка трафика) серьезно усложняют процесс управления и совершенствования сети. Введение в сеть новых функциональных возможностей, например инструментов для балансировки нагрузки, требует вмешательства в инфраструктуру сети, что напрямую влияет на ее логику и требует длительной отладки и стандартизации.

Технология ПКС представляет новую парадигму КС, которая позволяет проектировать и управлять сетями с помощью введения абстракций и разделения уровней управления и передачи данных [1] – [4]. Технология ПКС также обеспечивает гибкость в управлении трафиком без использования отдельного специализированного оборудования. В ПКС вся интеллектуальная часть реализуется на центральном сервере или контроллере, что упрощает выполнение сложных операций. Идея заключается в использовании для управления такими сетями стандартных серверов или контроллеров, работающих отдельно от сетевых устройств, благодаря чему сетевые администраторы могут получать более детализированный контроль над трафиком. В настоящее время известно большое количество работ по созданию программной инфраструктуры распределенной обработки данных в ПКС на основе протокола OpenFlow [5] – [7]. Практической реализации концепции ПКС также уделяется большое внимание [8] – [12].

Целью данной работы является разработка и создание новой платформы для распределенной обработки и динамического управления потоками данных в ПКС. В составе ПИВС (программная инфраструктура и визуальная веб-ориентированная среда) предложено использовать ПКС на базе оборудования HP Aruba 2920-24G. Для централизованного управления состоянием всей сети предложено использовать OpenDayLight контроллер [13], установленный на специализированном сетевом вычислительном комплексе. Для тестирования и настройки сетевой конфигурации в OpenDayLight контроллере предлагается использовать встроенный модуль виртуальной среды MiniNet [14]. В основу разработанной ПИВС положены идеи, предложенные в работах [15, 16]. Для решения задачи динамической обработки и управления потоками данных предложено использовать различные программные компоненты адаптивной маршрутизации [17] – [20], многопутевой маршрутизации [21] – [23], балансировки потоков данных [24] – [28] и сегментации структур ПКС [29] – [34]. Архитектура ПИВС представлена в виде описания UML диаграмм классов и соответствующих программных методов.

Предложенная в работе ПИВС является основой для разработки большого класса программных систем и компонентов для ПКС.

Разработка программной инфраструктуры и визуальной веб-ориентированной среды

Программная инфраструктура и визуальная веб-ориентированная среда динамического управления потоками данных представлена на рис. 1. ПКС построена на базе оборудования HP Aruba 2920-24G с поддержкой протокола OpenFlow версии 1.3 и ниже. HP Aruba 2920-24G – управляемые гибридные коммутаторы 3-го уровня (Layer 3), предназначенные для развертывания на периферийных участках корпоративных сетей. Поддерживают модульное стекирование, интерфейс 10GbE, маршрутизацию по протоколу RIP, PoE+, списки контроля доступа (ACL), sFlow и IPv6. Коммутаторы обеспечивают стабильную работу проводных и беспроводных сетей с помощью различных унифицированных средств управления. Коммутаторы построены на базе новейшей архитектуры ProVision ASIC, которая гарантирует малое время задержки, повышенную буферизацию пакетов и адаптивное энергопотребление.

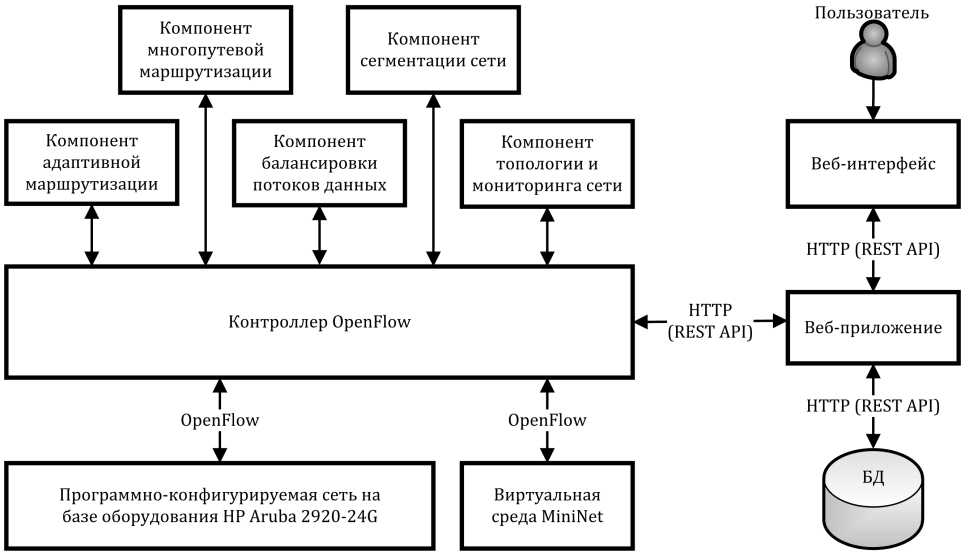


Рис. 1. Программная инфраструктура и визуальная среда SDN Topology
Fig. 1. Software infrastructure and visual web-oriented environment SDN Topology

Данная архитектура позволяет поддерживать важные для бизнеса мультимедийные приложения с высокими требованиями к пропускной способности. Возможность выбора различных конфигураций и уровня буферизации памяти, соответствующего требованиям к сетевым приложениям, повышает производительность всей сети. Управление коммутатором HP Aruba 2920-24G осуществляется с помощью веб-интерфейса и/или консольной строки через последовательный порт коммутатора. В качестве контроллера OpenFlow используется OpenDayLight контроллер, который установлен на сервере. Сервер представляет собой специализированный сетевой вычислительный комплекс на базе процессора Intel Xeon E5-2630 v4 LGA 2.2 ГГц, объем оперативной памяти 16 Гбайт DDR4 2133 МГц, объем жесткого диска 2 Тбайта. На материнской плате сервера имеются дополнительные слоты расширения объемов оперативной памяти и жесткого диска. Взаимодействие между серве-

ром и коммутаторами HP Aruba 2920-24G осуществляется по протоколу OpenFlow. Для корректной работы протокола OpenFlow сетевому администратору необходимо обеспечить разделение плоскости управления от плоскости передачи данных с целью предотвращения сбоев в работе ПКС. Для этого созданы два VLANa: VLAN 1 и VLAN 2. VLAN 1 – это сеть или плоскость передачи данных, к которой применяются правила OpenFlow и происходит передача пакетов по этим правилам. С помощью VLAN 2 (плоскость управления) происходит обмен управляющими элементами между контроллером OpenFlow и коммутаторами OpenFlow. Контроллер сообщает конфигурационные данные и правила обработки потоков данных коммутаторам OpenFlow. Структурная схема ПКС приведена на рис. 2.

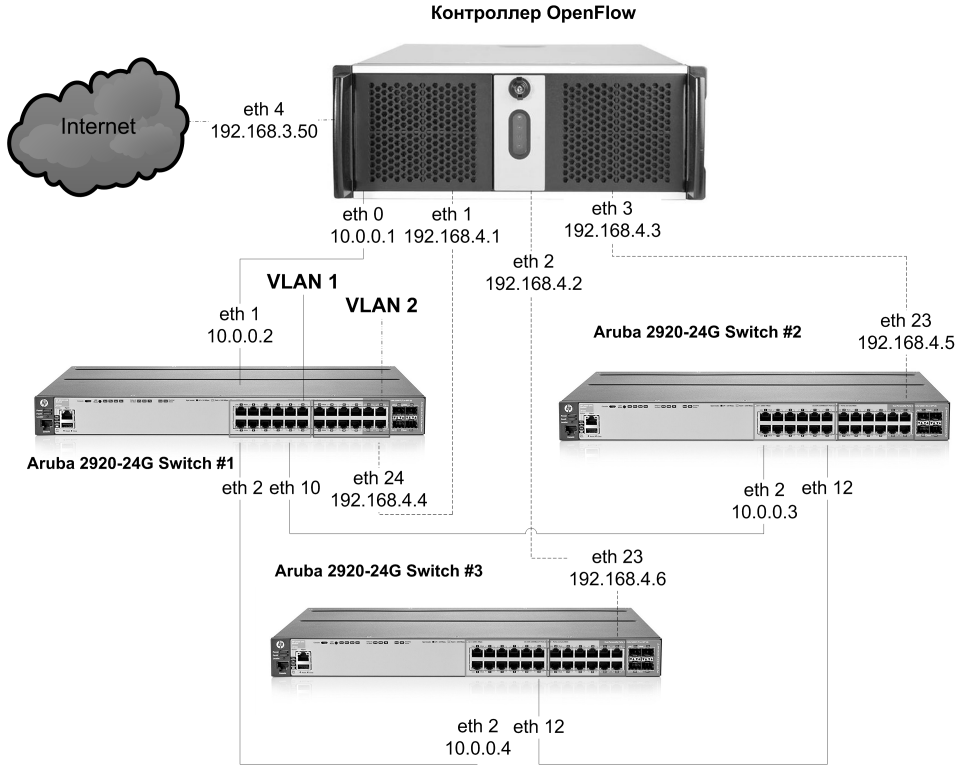


Рис. 2. Структурная схема кампусной ПКС
Fig. 2. Structure scheme of campus SDN

ПИВС состоит из двух основных частей.

1. Первая часть – это OpenDaylight OpenFlow контроллер. OpenDaylight (ODL) – это модульная платформа Open SDN для сетей любого размера и масштаба. ODL предоставляет сетевые услуги для всего спектра оборудования в мультивендорной среде. Микросервисная архитектура позволяет пользователям контролировать приложения, протоколы и плагины, обеспечивая взаимодействие между внешними потребителями и поставщиками. ODL – это общая платформа, которую можно настроить любым способом для решения проблем сети. ODL сочетает в себе открытый исходный код, открытые стандарты и открытый API.

   Преимуществами этой платформы являются:

- Микросервисная архитектура. Модель следующего поколения (YANG) используется для описания состояния сети и функций, которые будут выполняться в ней. ODL позволяет объединять отдельные сервисы для решения более сложных задач, используя единую структуру данных модели YANG в инфраструктуре общего хранилища и обмена сообщениями.
- Поддержка нескольких протоколов сетевого взаимодействия. Платформа похожа на традиционные протоколы, включая NETCONF, BGP / PCEP и CAPWAP, а также протокол OpenFlow и другие расширения.

Для ODL контроллера были разработаны следующие программные компоненты:

a. Компонент адаптивной маршрутизации отвечает за разработку оптимальных схем маршрутов в ПКС.

b. Компонент многопутевой маршрутизации отвечает за многопутевую маршрутизацию в ПКС.

c. Компонент балансировки нагрузки отвечает за балансировку потоков данных между конечными узлами.

d. Компонент сегментации сети отвечает за построение сегментов сети. Формирование сегментов позволяет снизить вычислительную сложность расчета маршрутов и распараллелить этот процесс.

e. Компонент топологии и мониторинга сети. Этот модуль отвечает за обновление информации о структуре сети и параметрах линий связи.

Вышеперечисленные компоненты в своей работе используют ODL контроллер. Благодаря возможностям ODL контроллера, программные компоненты взаимодействуют в сети как с реальным оборудованием HP Aruba 2920-24G, так и с виртуальной средой MiniNet.

2. Вторая часть системы представляет собой компонент управления в форме веб-приложения с графическим интерфейсом, через который контролируются другие компоненты системы. Вы можете настроить компоненты, используя RESTful API. Графический интерфейс системы доступен в браузере пользователя. Основными задачами компонента управления являются: отображение топологии сети и статистических данных, а также настройка компонентов системы, реализованных в виде модуля для ODL контроллера. Статистика записывается в базу данных, из которой статистическая информация считывается управляющим компонентом и отображается в графическом интерфейсе. Системные требования для работы компонента управления:

   - Протокол связи: HTTP (REST API);
   - Требуемая пропускная способность сервера для веб-приложения: 15 Мбит/с;
   - Поддерживаемые операционные системы: Ubuntu версия 14.04 или 16.04;
   - Оперативная память: не менее 2 Гбайт;

- Жесткий диск: не менее 300 Мбайт;
- Процессор: не менее 2 ядер и не менее 2 ГГц.

Основные сущности ПИВС предоставляются следующими классами: Topology, Switch, Link and Metrics (рис. 3).

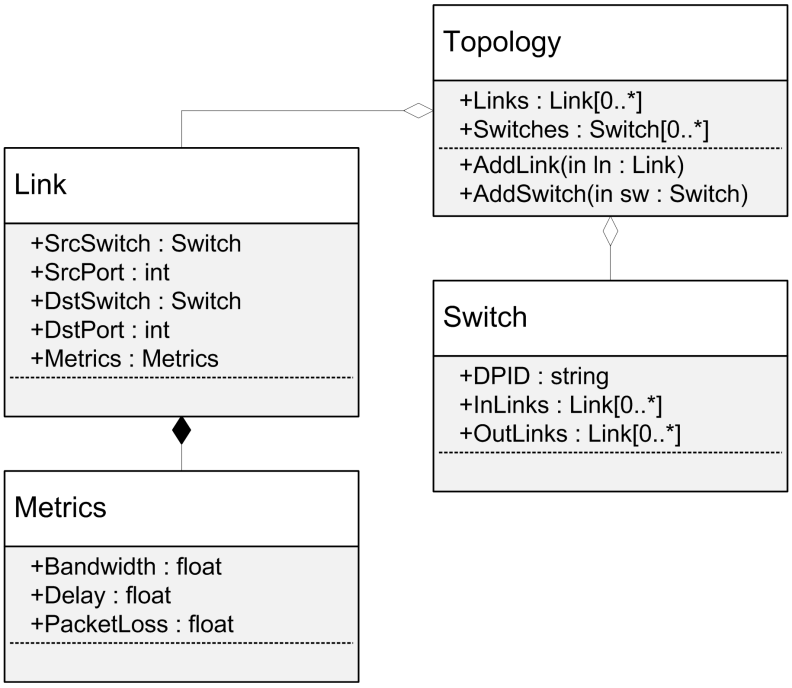


Рис. 3. Диаграмма основных классов ядра ПИВС SDN Topology
Fig. 3. Diagram of the SIVE SDN Topology main classes

Структура Topology представляет топологию сети: множество каналов связи Links – срез структур типа Link, и множество коммутаторов Switches – срез структур типа Switch. Структура содержит два метода. Метод AddLink(ln Link) добавляет в топологию канал связи. Метод AddSwitch(sw Switch) добавляет коммутатор к топологии. Топология формируется на основе данных, полученных контроллером ПКС.

Структура Switch представляет коммутатор OpenFlow, который имеет уникальный идентификационный номер – DPID, заданный строковым типом. Каждый коммутатор содержит два множества каналов связи. InLinks – срез структур типа Link. Данные каналы связи представляют ребра, направленные в данный коммутатор. OutLinks – срез структур типа Link. Данные каналы связи представляют ребра, направленные из данного коммутатора.

Структура Link представляет канал связи. Структура ссылается на коммутатор SrcSwitch – структура типа Switch, из которого направлено ребро, представляемое данным каналом связи. Коммутатор DstSwitch – структура типа Switch, в него направлено ребро, представляемое данным каналом связи. Для канала связи заданы номера портов, к которым присоединен данный канал: SrcPort – номер порта SrcSwitch, к которому подключен данный канал связи, DstPort – номер порта DstSwitch, к которому подключен данный канал связи. Метрики канала связи представлены структурой типа Metrics.

Структура Metrics содержит информацию о метриках канала связи: пропускная способность, задержка, процент потерь пакетов. Структура состоит из следующих компонентов: Bandwidth – текущая пропускная способность канала связи, Delay – текущая задержка канала связи. PacketLoss – текущий процент потерь пакетов канала связи.

Основные классы для организации сетевого взаимодействия ПИВС и контроллера OpenDayLight приведены на рис. 4.

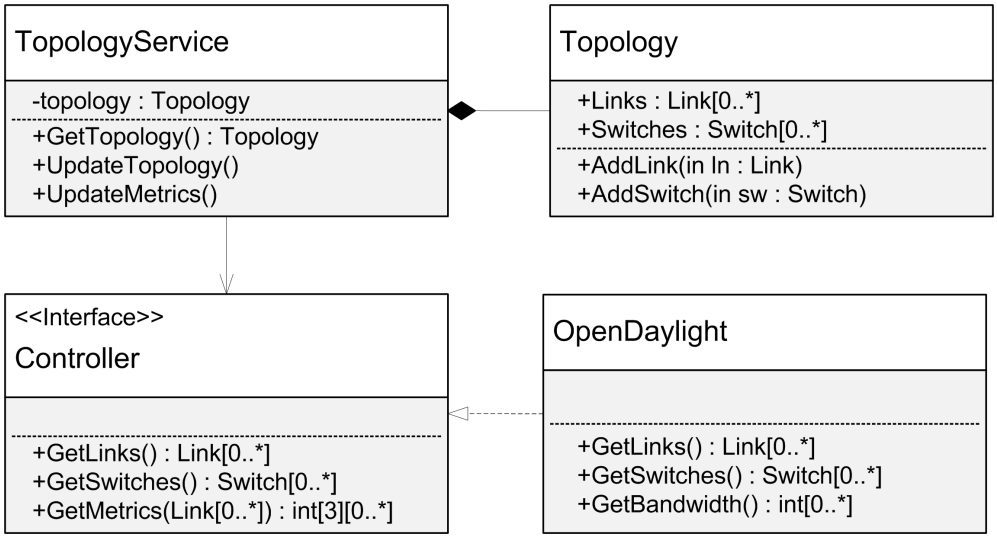


Рис. 4. Диаграмма основных классов для организации сетевого взаимодействия ПИВС и контроллера OpenDayLight

Fig. 4. The main classes for organization of network interaction of SDN Topology and ODL controller

Структура TopologyService обновляет сетевую топологию, хранящуюся в topology – структуре типа Topology. Метод GetTopology() Topology – возвращает сетевую топологию. Метод UpdateTopology() – обновляет структуру сетевой топологии (каналы связи и коммутаторы). Метод UpdateMetrics() – обновляет метрики каналов связи (пропускную способность, процент потерь пакетов, задержку передачи). Интерфейс Controller задает поведение абстрактного контроллера ПКС. Конкретная реализация интерфейса должна реализовать следующие методы: GetLinks() Link[0..*] – возвращает срез каналов связи сети, GetSwitches() Switch[0..*] – возвращает срез коммутаторов сети и GetMetrics(Link[0..*]) int[3][0..*] – возвращает метрики каналов связи. Структура OpenDaylight реализует интерфейс Controller с помощью REST API контроллера OpenDaylight.

Графический интерфейс ПИВС (рис. 5) условно можно разделить на следующие части:

- Программное меню (Menu);
- Инструменты управления (Instruments);
- Графический интерфейс (Graphic Interface).

Графический редактор отображает элементы проектируемой топологии. Компьютерная модель ПКС, используемая в SDN Topology, состоит из следующих элементов:

- коммутаторы OpenFlow;
- хосты;
- интерфейсы между хостами и коммутаторами.

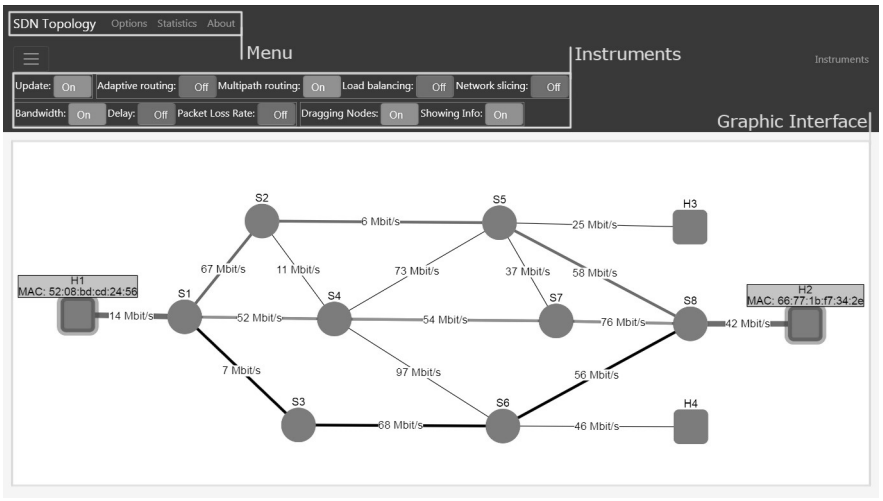


Рис. 5. Интерфейс ПИВС SDN Topology
Fig. 5. SIVE SDN Topology interface

ПИВС SDN Topology обеспечивает динамическое управление потоками данных в ПКС. Выполнение компонента адаптивной маршрутизации показано на рис. 6.

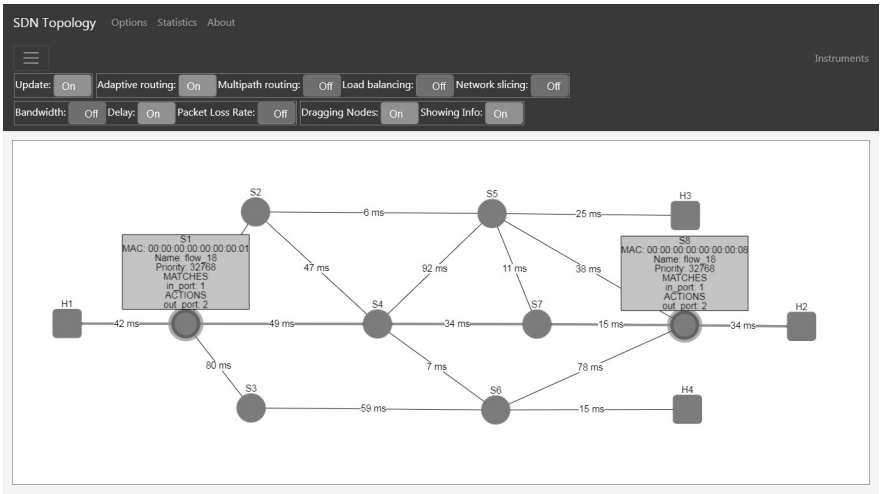


Рис. 6. Выполнение компонента адаптивной маршрутизации
Fig. 6. The execution of adaptive routing component

Заключение

В работе представлена программная инфраструктура и визуальная среда распределенной обработки и динамического управления потоками данных в программно-конфигурируемых сетях на основе протокола OpenFlow. Разработанную ПИВС предлагается использовать в качестве интегрированного сегмента кампусной сети Рязанского государственного радиотехнического университета.

Программно-конфигурируемая сетевая инфраструктура построена на базе оборудования HP Aruba 2920-24G. Архитектура разработанной ПИВС представлена в виде описания UML диаграмм классов и соответствующих программных фрагментов их реализации.

Разработанная ПИВС является основой для реализации большого класса программных систем и компонентов обработки данных в ПКС. Результаты экспериментального исследования различных программных компонентов динамической обработки потоков данных на основе разработанной ПИВС будут детально рассмотрены в последующих работах.

Список литературы / References

[1] McKeown N. et al., “Openflow: enabling innovation in campus networks”, *ACM SIGCOMM Computer Communication Review*, **38**:2 (2008), 69–74.

[2] Kobayashi M. et al., “Maturing of OpenFlow and Software-Defined Networking Through Deployments”, *Computer Networks*, **61** (2014), 151–175.

[3] Egilmez H.E., *Adaptive Video Streaming over OpenFlow Networks with Quality of Service*, Thesis for Degree of Master Science in Electrical and Electronics Engineering, Koc University, 2012.

[4] Ongaro F., *Enhancing quality of service in software-defined networks*, Thesis for Degree of Master Science in Computer Engineering, University of Bologna, 2014.

[5] *GENI: Exploring Network of the Future*, http://www.geni.net.

[6] Choumas K. et al., “Testbed Innovations for Experimenting with Wired and Wireless Software Defined Networks”, *IEEE 35th International Conference*, 2015, 87–94.

[7] Antonenko V. et al., “Towards SDI-bases Infrastructure for supporting science in Russia”, Proceedings SDN and NFV: Next Generation of Computational Infrastructure – 2014 (International Science and Technology Conference MoNeTec 2014), 2014, 1–7.

[8] Егоров В.Б., “Некоторые вопросы практической реализации концепции SDN”, *Системы и средства информатики*, **26**:1 (2016), 109–120; [Egorov V.B., “Some issues of the SDN concept practical implementation”, *Systems and Means of Informatics*, **26**:1 (2016), 109–120, (in Russian).]

[9] Гузев О.Ю., Чижов И.В., “Балансировка нагрузки в защищенных сетях с использованием технологии SDN”, *Системы и средства информатики*, **28**:1 (2018), 123–138; [Guzev O.Yu., Chizhov I.V., “SDN load balancing for secure networks”, *Systems and Means of Informatics*, **28**:1 (2018), 123–138, (in Russian).]

[10] Гузев О.Ю., Чижов И.В., “SDN-балансировка на криптографические маршрутизаторы при объединении центров обработки данных”, *Системы и средства информатики*, **28**:1 (2018), 139–155; [Guzev O.Yu., Chizhov I.V., “SDN load balancing on L3-VPN gateways in data centers interconnection”, *Systems and Means of Informatics*, **28**:1 (2018), 139–155, (in Russian).]

[11] Малахов С.В., Тарасов В.Н., Карташевский И.В., “Теоретическое и экспериментальное исследование задержки в программно-конфигурируемых сетях”, *Инфокоммуникационные технологии*, **13**:4 (2015), 409–413; [Malakhov S.V., Tarasov V.N.,

Kartashevsky I.V., "Theoretical and experimental research of packet delays in software defined networks", *Infokommunikacionnye tehnologii*, **13**:4 (2015), 409–413, (in Russian).]

[12] Бахарева Н.Ф. и др., "Управление корпоративными программно-конфигурируемыми сетями", *Вестник Оренбургского государственного университета*, 2015, № 13(188), 108–113; [Bakhareva N.F, et al., "Management of enterprise software defined networks", *Vestnik Orenburgskogo gosudarstvennogo universiteta*, 2015, № 13(188), 108–113, (in Russian).]

[13] *OpenDayLight Controller*, https://www.opendaylight.org/.

[14] *Virtual Network MiniNet*, http://mininet.org.

[15] Perepelkin D., Byshov V., "Visual design environment of dynamic load balancing in software defined networks", *2017 27th International Conference Radioelektronika*, IEEE, 2017, 183–186.

[16] Perepelkin D., et al., "Development of architecture of visual program system for distributed data processing in software defined networks", *2018 28th International Conference Radioelektronika*, IEEE, 2018, 281–284.

[17] Корячко В.П., Перепелкин Д.А., *Анализ и проектирование маршрутов передачи данных в корпоративных сетях*, Горячая линия – Телеком, Москва, 2012; [Koryachko V.P., Perepelkin D.A., *Analiz i proektirovanie marshrutov peredachi dannyh v korporativnyh setyah*, Gorjachaja linija – Telekom, Moscow, 2012, (in Russian).]

[18] Перепелкин Д.А., Перепелкин А.И., "Алгоритм адаптивной ускоренной маршрутизации в условиях динамически изменяющихся нагрузок на линиях связи в корпоративной сети", *Информационные технологии*, 2011, № 3, 2–7; [Perepelkin D.A., Perepelkin A.I., "The Accelerated Algorithm Adaptive Routing in Dynamically Changing Loads on the Lines of Communication in Corporate Network", *Information Technologies*, 2011, № 3, 2–7, (in Russian).]

[19] Перепелкин Д.А., "Динамическое формирование структуры и параметров линий связи корпоративной сети на основе данных о парных перестановках маршрутов", *Информационные технологии*, 2014, № 4, 52–60; [Perepelkin D.A., "Dynamic Corporate Network Structure and Communication Links Loading Formation Based on Routes Pairs Permutations Data", *Information Technologies*, 2014, № 4, 52–60, (in Russian).]

[20] Перепелкин Д.А., "Концептуальный подход динамического формирования трафика программно-конфигурируемых телекоммуникационных сетей с балансировкой нагрузки", *Информационные технологии*, **21**:8 (2015), 602–610; [Perepelkin D.A., "Conceptual Approach of Dynamic Traffic Formation of Software-Defined Telecommunication Networks with Load Balancing", *Information Technologies*, **21**:8 (2015), 602–610, (in Russian).]

[21] Корячко В. П., Перепелкин Д. А., "Разработка и исследование математической модели многопутевой адаптивной маршрутизации в сетях связи с балансировкой нагрузки", *Электросвязь*, 2014, № 12, 27–31; [Koryachko V.P., Perepelkin D.A., "Development and research of the mathematical model multipath adaptive routing in telecommunication networks with load balancing", *Electrosvyaz*, 2014, № 12, 27–31, (in Russian).]

[22] Перепелкин Д.А., "Математическая модель многопутевой адаптивной маршрутизации с балансировкой неоднородной нагрузки в условиях динамических подключений узлов и линий связи в телекоммуникационных сетях", *Радиотехника*, 2015, № 5, 46–54; [Perepelkin D.A., "Mathematical model of multipath adaptive routing with heterogeneous load balancing in the course of nodes and communication links dynamic connections in telecommunication networks", *Radioengineering*, 2015, № 5, 46–54, (in Russian).]

[23] Перепелкин Д.А., "Модель отказоустойчивой многопутевой адаптивной маршрутизации с балансировкой неоднородной нагрузки в сетях связи", *Радиотехника*, 2015, № 11, 40–47; [Perepelkin D.A., "Model of fault-tolerant multipath adaptive routing with load balancing of heterogeneous traffic in communication networks", *Radioengineering*, 2015, № 11, 40–47, (in Russian).]

[24] Koryachko V.P., Perepelkin D.A., Byshov V.S., "Improved multipath adaptive routing model in computer networks with load balancing", *Proceedings IEEE 2016 International Siberian Conference on Control and Communications (SIBCON)*, 2016, 1–4.

[25] Koryachko V.P., Perepelkin D.A., Byshov V.S., "Multipath adaptive routing in computer networks with load balancing", *2016 Mediterranean Conference on Embedded Computing*, IEEE, 2016, 281–285.

[26] Koryachko V.P., Perepelkin D.A., Byshov V.S., "Development and research of improved model of multipath adaptive routing in computer networks with load balancing", *Automatic Control and Computer Sciences*, **51**:1 (2017), 63–73.

[27] Перепелкин Д.А., Бышов В.С., "Балансировка потоков данных в программно-конфигурируемых сетях с обеспечением качества обслуживания сетевых сервисов", *Радиотехника*, 2016, №11, 111–119; [Perepelkin D.A., Byshov V.S., "Load balancing in software defined networks with quality of services", *Radioengineering*, 2016, №11, 111–119, (in Russian).]

[28] Koryachko V. P., Perepelkin D. A., Byshov V. S., "Enhanced dynamic load balancing algorithm in computer networks with quality of services", *Automatic Control and Computer Sciences*, **52**:4 (2018), 268–282.

[29] Перепелкин Д.А., "Динамическое формирование трафика корпоративных сетей на основе метода маршрутизации по подсетям", *Вестник Рязанского государственного радиотехнического университета*, 2015, №51, 35–41; [Perepelkin D.A., "Dynamic traffic formation of corporate networks based on routing method of subnets", *Vestnik of Ryazan State Radio Engineering University*, 2015, №51, 35–41, (in Russian).]

[30] Перепелкин Д.А., Цыганов И.Ю., "Усовершенствованный алгоритм сегментации структур корпоративных сетей по критерию минимальной стоимости", *Вестник Рязанского государственного радиотехнического университета*, 2015, №53, 48–57; [Perepelkin D.A., Tsyganov I.Yu., "Improved algorithm of stuctures segmentation with minimum cost criterion in corporate networks", *Vestnik of Ryazan State Radio Engineering University*, 2015, №53, 48–57, (in Russian).]

[31] Perepelkin D.A., Tsyganov I.Yu., "Proactive backup scheme of routes in distributed computer networks", *Proceedings IEEE 2016 International Siberian Conference on Control and Communications (SIBCON)*, 2016, 1–4.

[32] Perepelkin D.A., Tsyganov I.Yu., "Paired transitions algorithm of communication links in computer networks based on subnet routing method", *5th Mediterranean Conference on Embedded Computing*, IEEE, 2016, 260–263.

[33] Perepelkin D., Ivanchikova M., Ivutin A., "Fast rerouting algorithm in distributed computer networks based in subnet routing method", *2017 27th International Conference Radioelektronika*, IEEE, 2017, 187–190.

[34] Perepelkin D. et al., "Algorithm and software of virtual slices formation in software defined networks", *2018 28th International Conference Radioelektronika*, IEEE, 2018, 265–268.

Koryachko V. P., Perepelkin D. A., Ivanchikova M. A., Byshov V. S., "Visual Web-Oriented Environment of Dynamic Control of Data Flow in Campus of Software Defined Networks", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 63–74.

DOI: 10.18255/1818-1015-2019-1-63-74

Abstract. Nowadays new innovative approaches based on the technology of software defined networks (SDN) are gaining popularity in the field of computer networks (CN). SDN provide a flexible approach to the processing and control of data flows in CN by separating the control plane and data plane, as well as centralizing the representation of the entire network. In this paper, we propose a software infrastructure and a visual web-oriented environment (SIVE) for dynamic control of data flows in campus SDN based on OpenFlow protocol. It was proposed to use the SIVE as an integrated segment of the campus network of Ryazan State Radio Engineering University. The aim of the work is the development of the SIVE architecture in the form of UML class diagram description, as well as the creation of software methods for organizing effective network interaction of various software systems in SDN based on OpenFlow protocol. A hardware-software test bench based on HP Aruba 2920-24G equipment was developed to confirm the efficiency and reliability of the proposed SIVE. The offered

SIVE is the basis for the development of a large class of software systems and SDN components based on OpenFlow protocol.

Keywords: software defined networks, software infrastructure, visual web-oriented environment, OpenDayLight, SIVE, SDN Topology, network interaction, program component

On the authors:
Vyacheslav P. Koryachko, orcid.org/0000-0003-0272-673X, PhD,
Ryazan State Radioengineering University,
59/1 Gagarina str., Ryazan 390005, Russia, e-mail: koryachko.v.p@rsreu.ru

Dmitry A. Perepelkin, orcid.org/0000-0003-4775-5745, PhD,
Ryazan State Radioengineering University,
59/1 Gagarina str., Ryazan 390005, Russia, e-mail: dmitryperepelkin@mail.ru

Maria A. Ivanchikova, orcid.org/0000-0002-9615-2898, postgraduate student,
Ryazan State Radioengineering University,
59/1 Gagarina str., Ryazan 390005, Russia, e-mail: ivanchikova.masha@yandex.ru

Vladimir S. Byshov, orcid.org/0000-0001-7673-9803, postgraduate student,
Ryazan State Radioengineering University,
59/1 Gagarina str., Ryazan 390005, Russia, e-mail: b.v.s.12@yandex.ru

Acknowledgments:
The work was supported by the Grant of the President of the Russian Federation (МД-1826.2019.9).

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 75–89
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 75–89

DOI: 10.18255/1818-1015-2019-1-75-89

УДК 004.7

Эффективный алгоритм разрешения коллизий в правилах политики безопасности

Моржов С. В., Соколов В. А.

Поступила в редакцию 15 декабря 2018
После доработки 14 января 2019
Принята к публикации 18 февраля 2019

Аннотация. Межсетевой экран является основным классическим инструментом для контроля и управления сетевым трафиком в локальной сети. Его задача — сравнивать проходящий через него сетевой трафик с установленными правилами безопасности. Эти правила, которые часто также называют политикой безопасности, могут быть определены как до, так и во время работы межсетевого экрана. Управление политикой безопасности крупных корпоративных сетей является сложной задачей. Для того чтобы правильно ее реализовать, правила фильтрации межсетевого экрана должны быть написаны и организованы аккуратно и без ошибок. Кроме того, процесс изменения или вставки новых правил должен выполняться только после тщательного анализа отношений между изменяемыми или вставляемыми правилами, а также правилами, которые уже существуют в политике безопасности. В данной статье авторы рассматривают классификацию отношений, в которых могут находиться правила политики безопасности между собой, и дают определение возможных коллизий между ними. Авторы представляют также новый эффективный алгоритм обнаружения и устранения коллизий в правилах межсетевого экрана на примере контроллера ПКС Floodlight.

Ключевые слова: список контроля доступа, межсетевой экран, программно-конфигурируемая сеть, ПКС, дерево политики безопасности

Для цитирования: Моржов С. В., Соколов В. А., "Эффективный алгоритм разрешения коллизий в правилах политики безопасности", *Моделирование и анализ информационных систем*, **26**:1 (2019), 75–89.

Об авторах:
Моржов Сергей Владимирович, аспирант, orcid.org/0000-0001-6652-3574
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, Ярославль, 150003, Россия, e-mail: smorzhov@gmail.com

Соколов Валерий Анатольевич, д-р физ.-мат. наук, профессор, orcid.org/0000-0003-1427-4937
НОМЦ Центр интегрируемых систем, Ярославский государственный университет им. П.Г. Демидова
ул. Советская, 14, Ярославль, 150003, Россия, e-mail: valery-sokolov@yandex.ru

Благодарности:
Работа выполнена при финансовой поддержке РФФИ, научный проект № 17-07-00823 А, и в рамках государственного задания Министерства образования и науки РФ, проект № 1.10160.2017/5.1

Введение

Межсетевой экран является основным классическим инструментом для контроля и управления сетевым трафиком в локальной сети. Его задача — сравнивать прохо-

дящий через него сетевой трафик с установленными правилами безопасности. Эти правила, которые часто также называют политикой безопасности, могут быть определены как до, так и во время работы межсетевого экрана. Управление политикой безопасности крупных корпоративных сетей является сложной задачей. Для того чтобы правильно ее реализовать, правила фильтрации межсетевого экрана должны быть написаны и организованы аккуратно и без ошибок. Кроме того, процесс изменения или вставки новых правил должен выполняться только после тщательного анализа отношений между изменяемыми или вставляемыми правилами, а также правилами, которые уже существуют в политике безопасности.

В данной статье авторы рассматривают классификацию отношений, в которых могут находиться правила политики безопасности между собой, и дают определение возможных коллизий между ними. Авторы представляют также новый эффективный алгоритм обнаружения и устранения коллизий в правилах межсетевого экрана на примере контроллера ПКС Floodlight.

1. Формализация возможных коллизий в правилах политики безопасности

Основными источниками установки правил на сетевое оборудование в традиционной компьютерной сети являются межсетевой экран, список контроля доступа, система предотвращения вторжений, сервер IP-телефонии, системный администратор и т.п. Добавление нового правила без координации с другими источниками может привести к коллизиям с существующими правилами. Коллизия в правилах политики безопасности — это ситуация, когда в политике безопасности существуют два или более правила, значения фильтрующих атрибутов которых пересекаются. Наличие коллизий в правилах политики безопасности серьезно затрудняет анализ установленных правил политики безопасности, тем самым осложняя работу сетевого администратора. Кроме того, коллизии потенциально могут приводить к появлению брешей в политике безопасности, а следовательно, напрямую влияют на функционирование сети в целом.

Таким образом, при создании правил коммутации пакетов в традиционной сети могут возникать коллизии, которые необходимо разрешать.

1.1. Определение отношений между двумя правилами безопасности контроллера ПКС Floodlight

Правило политики безопасности[1] представляет собой кортеж, элементами которого являются пары атрибут/значение. Так как значением в общем случае является множество, правило может быть представлено, как совокупность или набор этих множеств. Таким образом, для сравнения правил между собой можно использовать отношения над множествами, представленные на рисунке 1.

Возможные отношения между двумя правилами могут быть определены следующим образом [1]:

[1] Здесь и далее, если не будет указано другого, под правилом политики безопасности будет пониматься правило политики безопасности контроллера ПКС Floodlight.

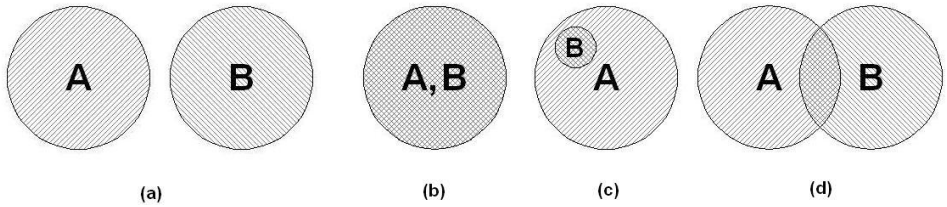


Рис. 1: (a) $A \cap B = \emptyset$; (b) $A = B$; (c) $B \subset A$; (d) $A \cap B \neq \emptyset$, $A \not\subset B \wedge B \not\subset A$

Fig. 1: (a) $A \cap B = \emptyset$; (b) $A = B$; (c) $B \subset A$; (d) $A \cap B \neq \emptyset$, $A \not\subset B \wedge B \not\subset A$

Определение 1. *Два правила r и s не пересекаются ($\mathfrak{R}_D$), если они имеют хотя бы один атрибут, для которого заданы непересекающиеся значения. Формально это можно записать так:*

$$r\mathfrak{R}_D s, \exists a \in attr, a_r \cap a_s = \emptyset \ . \tag{1}$$

Например, правила 1 и 2 ниже не пересекаются, так как имеют разные значения атрибута *«src-port»* (21 и 9050):

1. tcp, 193.168.*, 192.168.0.1, 21, allow
2. tcp, 193.168.*, 192.168.0.1, 9050, deny

Определение 2. *Два правила r и s полностью совпадают ($\mathfrak{R}_{EM}$), если значения всех их атрибутов равны. Формально это можно записать так:*

$$r\mathfrak{R}_{EM} s, \forall a \in attr, a_r = a_s \ . \tag{2}$$

Например, правила 1 и 2 полностью совпадают, так как значения всех атрибутов равны:

1. tcp, 193.168.*, 192.168.0.1, 21, allow
2. tcp, 193.168.*, 192.168.0.1, 21, allow

Определение 3. *Правило r является подмножеством правила s ($\mathfrak{R}_{IM}$), если существует хотя бы один атрибут правила r, значение которого является подмножеством к соответствующему атрибуту правила s, а остальные атрибуты правил равны. Формально это можно записать так:*

$$r\mathfrak{R}_{IM} s, \exists a \in attr, a_r \subset a_s \wedge \forall b \subset attr \setminus \{a\}, b_r = b_s \vee b_r \subset b_s \ . \tag{3}$$

Например, правило 1 является подмножеством правила 2, так как все атрибуты правила 1 равны соответствующим им атрибутам правила 2 за исключением атрибута «src-ip». У правила 1 значение атрибута *«src-ip»* является подмножеством значения атрибута *«src-ip»* правила 2:

1. tcp, 193.168.0.1, 192.168.0.1, 21, deny
2. tcp, 193.168.*, 192.168.0.1, 21, deny

Определение 4. *Два правила r и s коррелируют ($\mathfrak{R}_C$), если не выполнено условие определения 1, а также правила не являются подмножеством или надмножеством друг друга. Формально, это можно записать так:*

$$r\mathfrak{R}_C s, (r \neg \mathfrak{R}_D s) \wedge (r \neg \mathfrak{R}_{EM} s) \wedge (r \neg \mathfrak{R}_{IM} s) \ . \tag{4}$$

Например, правила 1 и 2 коррелируют, так как значения атрибутов *«nw-proto»* и *«src-port»* равны, у правила 1 значение атрибута *«src-ip»* является подмножеством значения соответствующего атрибута правила 2, а значение атрибута *«dst-ip»* правила 1 является надмножеством значения соответствующего атрибута правила 2:

1. tcp, 193.168.*.*, 21, deny
2. tcp, *, 192.168.0.1, 21, deny

Лемма 1. *Любые два правила, имеющие два атрибута, могут находиться в одном из четырех отношений:* $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}$ *или* $\mathfrak{R}_C$.

Доказательство. Рассмотрим правила $R_x = \langle x_1, x_2\rangle$ и $R_y = \langle y_1, y_2\rangle$. Отношение между ними определяется отношением между соответствующими значениями их атрибутов, то есть $x_i \mathfrak{R} y_i$, где $\mathfrak{R} \in \{=, \subset, \supset, \bowtie\}, i = 1, 2$. Оператор $\bowtie$ определим следующим образом: $x \bowtie y \iff x \neq y \wedge x \not\subset y \wedge x \not\supset y \wedge x \cap y \neq \emptyset$. Рассмотрим все возможные отношения между атрибутами R_x и R_y:

Если $x_1 = y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_{EM} R_y$
Если $x_1 = y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 = y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 = y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 = y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \subset y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \subset y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \subset y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \subset y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \subset y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \supset y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \supset y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \supset y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_{IM} R_y$
Если $x_1 \supset y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \supset y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \bowtie y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_C R_y$
Если $x_1 \bowtie y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 = y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \subset y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \supset y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \bowtie y_2$, то $R_x \mathfrak{R}_D R_y$
Если $x_1 \neg\mathfrak{R} y_1$ и $x_2 \neg\mathfrak{R} y_2$, то $R_x \mathfrak{R}_D R_y$

Таким образом, было показано, что R_x и R_y всегда находятся в одном из четырех отношений: $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}$ или $\mathfrak{R}_C$. □

Лемма 2. *Добавление одного атрибута к любым двум правилам* R_x *и* R_y*, находящимся в отношении* $\mathfrak{R} \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$*, оставляет правила* R_x *и* R_y *в прежнем отношении* $\mathfrak{R}$ *или переводит их в новое отношение*

$$\mathfrak{R}' \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}.$$

Доказательство. Рассмотрим правила $R_x = \langle x_1, ..., x_k \rangle$ и $R_y = \langle y_1, ..., y_k \rangle$. Добавим к R_x атрибут x_{k+1}, а к R_y атрибут y_{k+1}. Обозначим новые правила как $\mathfrak{R}'_x$ и R'_y соответственно. Если R_x и R_y были в отношении $\mathfrak{R} \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$, то $\mathfrak{R}'_x$ и R'_y могут быть в одном из следующих отношений:

Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_D R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_{EM} R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_{EM} R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_{IM} R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_{IM} R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} = y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \subset y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \supset y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \bowtie y_{k+1}$, то $R'_x \mathfrak{R}_C R'_y$
Если $R_x \mathfrak{R}_C R_y$ и $x_{k+1} \neg\mathfrak{R} y_{k+1}$, то $R'_x \mathfrak{R}_D R'_y$

Таким образом, было показано, что R'_x и R'_y всегда находятся в одном из четырех отношений: $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}$ или $\mathfrak{R}_C$. □

Теорема 1. *Отношения $\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C$ образуют универсальное множество отношений между двумя правилами R_x и R_y.*

Доказательство. Докажем теорему методом математической индукции.

Для $k = 2, R_x = \langle x_1, x_2 \rangle, R_y = \langle y_1, y_2 \rangle$ — верно (см. Лемму 1).

Пусть утверждение теоремы верно для $R_x = \langle x_1, ..., x_k \rangle, R_y = \langle y_1, ..., y_k \rangle$ и $R_x \mathfrak{R} R_y$, где $\mathfrak{R} \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$. Добавим к R_x атрибут x_{k+1}, а к R_y атрибут y_{k+1} и обозначим новые правила как R'_x и R'_y соответственно. Так как правила R'_x и R'_y были получены путем добавления одного атрибута к правилам R_x и R_y таким, что $R_x \mathfrak{R} R_y$, то по Лемме 2 $R'_x \mathfrak{R}' R'_y$, где $\mathfrak{R}' \in \{\mathfrak{R}_D, \mathfrak{R}_{EM}, \mathfrak{R}_{IM}, \mathfrak{R}_C\}$. □

1.2. Возможные коллизии между двумя правилами

Исходя из отношений между правилами, обозначенных в разделе 1.1., возможные коллизии можно определить следующим образом [2]:

Определение 5. *Перекрытие. Правило r перекрывается правилом s, если s имеет более высокий приоритет и фильтрует все пакеты, которые может отфильтро-*

вать r. Как следствие, правило r никогда не будет использовано. Формально правило r перекрывается правилом s, если s имеет более высокий приоритет, $r\mathfrak{R}_{EM}s$ и $action_r \neq action_s$, или s имеет более высокий приоритет, $r\mathfrak{R}_{IM}s$ и $action_r \neq action_s$.

Определение 6. *Корреляция. Два правила r и s коррелируют, если они фильтруют пересекающиеся множества пакетов, то есть найдутся такие пакеты, которые могут быть отфильтрованы правилом r и s одновременно. Формально правила r и s коррелируют, если $r\mathfrak{R}_C s$ и $action_r \neq action_s$.*

Определение 7. *Избыточность. Избыточное правило r фильтрует те же пакеты, что и правило s, и политика безопасности не пострадает при удалении правила r. Формально правило r избыточно по отношению к правилу s, если s имеет более высокий приоритет, $r\mathfrak{R}_{EM}s$ и $action_r = action_s$, или s имеет более высокий приоритет, $r\mathfrak{R}_{IM}s$ и $action_r = action_s$; в то же время, правило s избыточно к правилу r, если s имеет более высокий приоритет, $s\mathfrak{R}_{IM}r$ и $action_r = action_s$ и не существует t, такого что s имеет более высокий приоритет по отношению к t, а t имеет более высокий приоритет по отношению к r, $s\{\mathfrak{R}_{IM}, \mathfrak{R}_C\}t, action_r \neq action_t$.*

2. Разрешение коллизий

При разработке алгоритмов разрешения и недопущения возникновения коллизий в правилах межсетевого экрана или списке контроля доступа (Access Control List — ACL) разумно потребовать соблюдение следующих условий:

1. Алгоритмы должны корректно разрешать все возможные коллизии, определенные в разделе 1.2. как в уже существующей политике безопасности, так и в режиме реального времени. Последнее необходимо для реализации алгоритмов, например, в виде сетевого приложения, способного осуществлять постоянный мониторинг устанавливаемых правил, не допуская возникновения всех возможных коллизий.

2. Полученная в результате работы алгоритмов новая политика безопасности, свободная от всех возможных коллизий, должна фильтровать те же пакеты, что и исходная политика безопасности. Таким образом, новая политика безопасности не должна содержать уязвимостей, которых была лишена исходная политика безопасности.

3. Алгоритмы должны обеспечивать по возможности максимальную быстроту следующих операций над правилами политики безопасности: операции добавления, удаления, итерирования правил.

Принимая во внимание вышеизложенные условия, разработку алгоритма следует начать с выбора оптимальной структуры данных для хранения и представления правил политики безопасности.

2.1. Дерево правил

Будем называть правило «хорошим», если при добавлении его в политику безопасности она остается свободной от всех возможных коллизий. Если же последнее условие не выполняется, то правило будем называть «плохим». Разумно допустить, что «хорошие» правила добавляются в политику безопасности чаще, чем «плохие». Кроме этого, стоит отметить, что понять, «хорошее» перед нами правило или «плохое», нельзя, не сравнив его с правилами, ранее добавленными в политику безопасности. Очевиден факт: понять, что перед нами «хорошее» правило вычислительно не проще, а скорее всего сложнее, чем понять, что перед нами «плохое» правило, так как в общем случае вывод, что правило «хорошее», может быть сделан только после того, как мы убедимся, что новое правило не создает коллизий с каждым правилом, добавленным в политику безопасности ранее. Отсюда следует, что операция добавления «хорошего» правила должна быть как можно более вычислительно простой.

Рассмотрим базовые структуры данных, пригодных для хранения правил политики безопасности. Первым и наиболее очевидным выбором может быть хранение правил в виде массива. Данный подход обеспечит быстроту и удобство итерирования правил. Так как операция добавления наиболее частая, то трудоемкость алгоритма в случае хранения правил в массиве не может быть лучше $O(n)$ [2], потому что «хорошие» правила будут сравниваться со всеми, ранее добавленными. Кроме того, операции добавления и удаления правил в этом случае — вычислительно сложные, так как сопряжены с изменением размера массива. Отсюда общая трудоемкость будет хуже, чем $O(n)$. Использование словарей, хеш-таблиц или даже баз данных кардинально не улучшит оценочную трудоемкость, так как все эти структуры данных принципиально не отличаются от массива и не решают главной проблемы — каждое новое правило нельзя отнести к «хорошим» или «плохим», не сравнив его в общем случае со всеми ранее добавленными в политику безопасности правилами.

Рассмотрим древовидную структуру данных и покажем, что она способна удовлетворить всем поставленным условиям и что она является оптимальной для поставленной задачи. Так как алгоритм разрабатывается для контроллера ПКС Floodlight, при описании структуры данных будем иметь в виду правило модуля межсетевого экрана контроллера ПКС Floodlight. Оно состоит из 12 атрибутов (см. таблицу 1).

Каждый атрибут может принимать одно значение из определенного множества. Например, атрибут «dl-type» может принимать только значения из множества $\{ipv4, arp, *^3\}$, мощность которого — 3, а атрибут *«src-ip»* может быть равен любому из $(2^8 + 1)^4$ IP-адресов (учитываются возможные символы-джокеры). Упорядочим атрибуты (все, кроме *«priority»* и *«action»*) в порядке возрастания мощностей множеств их возможных значений. Атрибуты *«priority»* и *«action»* поместим в конец полученной последовательности, так как при определении возможной коллизии они играют второстепенную роль, и их следует учитывать в последнюю очередь. Более подробно об этом будет сказано дальше при описании алгоритмов разрешения коллизий.

[2] Здесь и далее для упрощения выкладок будет приводиться трудоемкость операции добавления одного правила в политику безопасности фиксированного размера n.

[3] Символ-джокер (wildcard) для замены любой строки символов

Таблица 1: Атрибуты правила межсетевого экрана контроллера ПКС Floodlight [5]

Table 1: Rule fields of Floodlight SDN controller

Атрибут	**Значение**	**Описание**
dl-type	ARP или IPv4	Протокол канального уровня
nw-proto	TCP или UDP или ICMP	Протокол сетевого уровня
switchid	xx:xx:xx:xx:xx:xx:xx:xx	Идентификационный номер коммутатора
src-inport	short	Номер входящего порта коммутатора
tp-src	short	Номер порта источника
tp-dst	short	Номер порта получателя
src-ip	A.B.C.D/M	IP-адрес источника
dst-ip	A.B.C.D/M	IP-адрес получателя
src-mac	xx:xx:xx:xx:xx:xx	MAC-адрес источника
dst-mac	xx:xx:xx:xx:xx:xx	MAC-адрес получателя
priority	integer	Приоритет правила
action	allow or deny	Разрешить или запретить пакеты, удовлетворяющие правилу

Итак, в результате упорядочивания получен 12-мерный кортеж. Так как правила будут храниться в дереве, то 12-мерному кортежу должен соответствовать один путь от вершины до листа в этом дереве, а каждому пути из вершины до листа в дереве должен соответствовать один 12-мерный кортеж, представляющий правило политики безопасности. Установим взаимно однозначное соответствие между значениями 12-мерного кортежа и уровнями дерева: первое значение кортежа будет соответствовать вершине дерева, второе — вершинам на первом уровне, третье — вершинам на втором уровне и т.д., двенадцатый элемент кортежа будет соответствовать листьям на одиннадцатом уровне дерева (см. рисунок 2). Так как атрибут *«dl-type»* может иметь всего три значения, то он будет соответствовать корню дерева. Атрибут *«nw-proto»* имеет четыре различных значения. Ему будут соответствовать вершины первого уровня дерева правил. Листья на одиннадцатом уровне дерева будут содержать одно из двух возможных значений: *allow* – для разрешающего правила и *deny* – для запрещающего. Нумерация правил на рисунке 2 вставлена исключительно для удобства восприятия.

Пусть каждая вершина дерева правил [3] [4] содержит хеш-таблицу (пару ключ-значение). В качестве ключа в хеш-таблице будут храниться значения соответствующих уровню атрибутов правил политики безопасности, а в качестве значений для каждого из ключей в таблице будут храниться адреса памяти смежной вершины на уровень ниже. Таким образом в хеш-таблице корня дерева, представленного на рисунке 2, будут содержаться два ключа: *«ipv4»* и *«arp»*. Значения этих ключей — адреса на левое и правое поддеревья соответственно.

Отметим, что порядок элементов в кортеже, представляющем собой правило политики безопасности, выбран не случайно. Все атрибуты, за исключением *«priority»* и *«action»*, были упорядочены по возрастанию мощностей множеств их возможных значений с той целью, чтобы минимизировать длины нисходящих путей, возникающих при добавлении правил в дерево. Покажем, что такой подход обеспечивает наиболее эффективное расходование памяти, необходимой для хранения дерева правил.

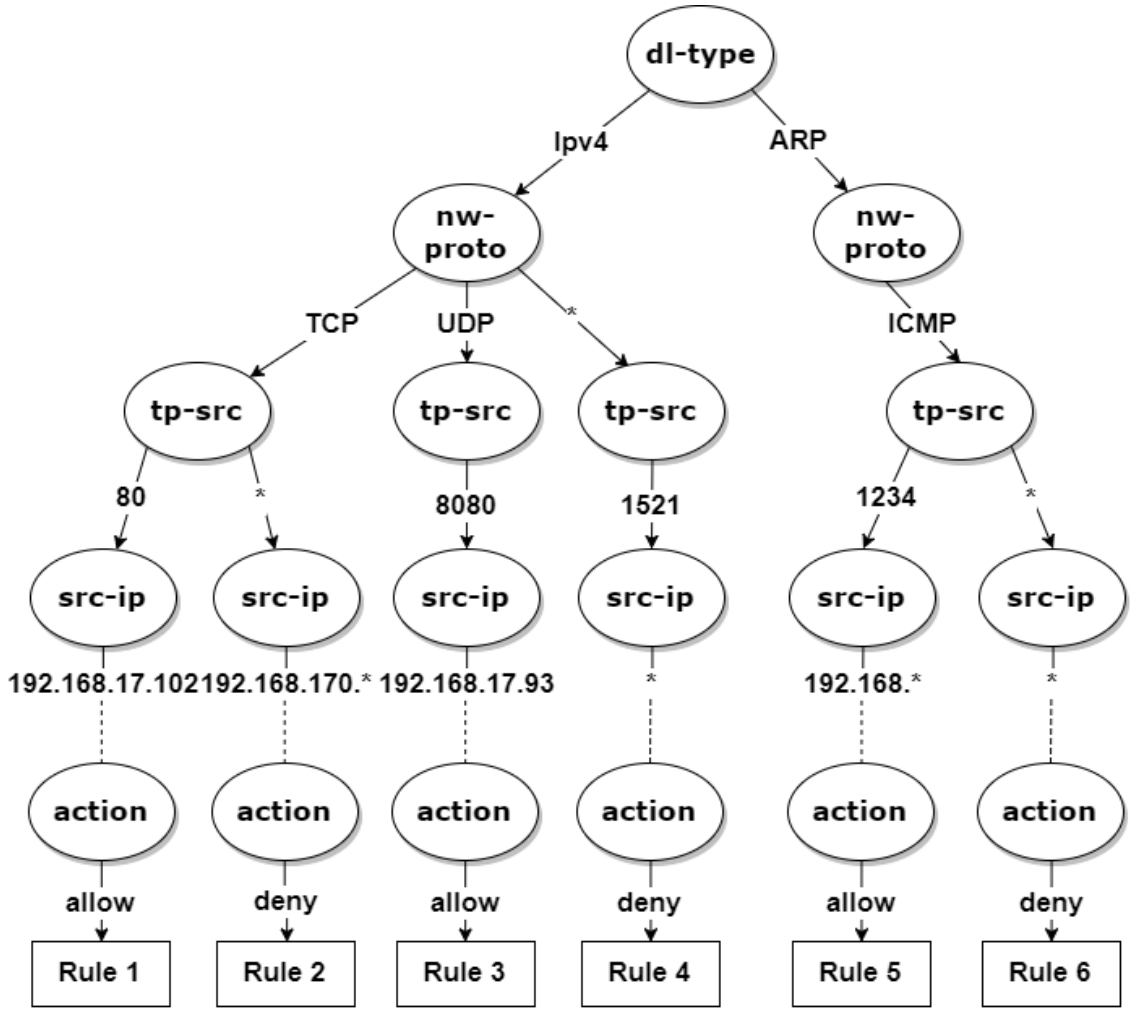


Рис. 2: Пример дерева правил

Fig. 2: Policy tree example

Пусть имеется два дерева правил. Пусть первое построено так, как было описано выше, а второе — так же, но с той лишь разницей, что атрибуты в кортеже упорядочены в порядке убывания мощностей множеств их возможных значений. Таким образом, первое дерево расширяется от корня к листьям, а второе дерево в общем случае становится широким сразу. Оценим максимально возможное количество хеш-таблиц для первого и второго дерева. При оценке, для простоты выкладок, без ограничения общности, предположим, что оба дерева хранят правила с пятью атрибутами — *«dl-type»*, *«nw-proto»*, *«tp-src»*, *«src-ip»* и *«action»*.

Рассмотрим первое дерево. В хеш-таблице первого уровня максимально может быть три записи — *«ipv4»*, *«arp»* и *«*»*. Отсюда следует, что на втором уровне будет содержаться 3 хеш-таблицы, каждая из которых может содержать по четыре записи — *«tcp»*, *«udp»*, *«icmp»* и *«*»*. Следовательно, на третьем уровне максимально может быть $3 * 4 = 12$ хеш-таблиц, содержащих 2^{16} записей, так как для хранения номера порта (третий уровень дерева соответствует атрибуту *«tp-src»* —

номер порта источника) выделяется 16 бит памяти. На четвертом уровне хранятся IP-адреса, а значит, хеш-таблица максимально может содержать $(2^8 + 1)^4$ записей, учитывая возможные символы-джокеры. Таким образом, первое дерево будет содержать $1 + 3 + 3 * 4 + 3 * 4 * 2^{16} = A$ хеш-таблиц.

Аналогично подсчитав количество хеш-таблиц для второго дерева, получим $1 + (2^8 + 1)^4 + (2^8 + 1)^4 * 2^{16} + (2^8 + 1)^4 * 2^{16} * 4 = B$, $B \ggg A$.

Очевидно, что, как бы ни упорядочивались атрибуты, количество хеш-таблиц A в первом дереве правил будет наименьшим.

Для построения дерева правил может быть использован следующий алгоритм [4].

Algorithm 2.1: Алгоритм BuildPolicyTree

```
1   input: rule, field, node
2   begin
3       value_found ← FALSE
4       if field ≠ ACTION then
5       foreach branch in node->branch_list do
6           if branch->value = rule->field->value then
7               value_found ← TRUE
8               BuildPolicyTree(rule, field->next, branch->node)
9           elif branch->value ⊂ rule->field->value
10              or branch->value ⊃ rule->field->value then
11              BuildPolicyTree(rule, field->next, branch->node)
12      if value_found = FALSE then
13          new_branch ← new TreeBranch(rule, rule->field, rule->filed->value)
14          node->branch_list->add(new_branch)
15          if field ≠ ACTION then
16              BuildPolicyTree(rule, field->next, new_branch->node)
17          end if
18      end if
19  end
```

2.2. Алгоритм разрешения коллизий

На основе определенных в разделе 1.1. отношений между правилами, а также возможных коллизий, определенных в разделе 1.2. был разработан алгоритм борьбы с коллизиями. Основная идея алгоритм состоит в том, что новое правило нужно добавить в дерево правил, свободных от всяческих коллизий. Если путь в дереве, соответствующий новому правилу, совпадает с каким-нибудь путем в дереве, представляющим собой ранее добавленное правило, то была найдена коллизия, которую необходимо разрешить. В противном случае добавление нового правила оставит политику безопасности свободной от всяческих коллизий. Данная логика может быть имплементирована в алгоритм построения дерева правил. Ниже приведен алгоритм *DiscoverAnomaly*, представляющий собой усовершенствованный алгоритм *BuildPolicyTree* [4].

Algorithm 2.2: Алгоритм DiscoverAnomaly

```
1  input: rule, field, node, anomaly_state
2  begin
3      if field ≠ ACTION then
4          value_found ← FALSE
5          foreach branch in node->branch_list do
6              if branch->value = rule->field->value then
7                  value_found = TRUE
8                  if anomaly_state = NOANOMALY then
9                      anomaly_state ← REDUNDANT
10                     DiscoverAnomaly(rule, field->next, branch->node, anomaly_state)
11                 end if
12             else
13                 if rule->field->value ⊂ branch->value then
14                     if anomaly_state = GENERALIZATION then
15                         DiscoverAnomaly(rule, field->next, branch->node, CORRELATION)
16                     else
17                         DiscoverAnomaly(rule, field->next, branch->node, SHADOWING)
18                     end if
19                 elif rule->field->value ⊃ branch->value
20                     if anomaly_state = SHADOWING then
21                         DiscoverAnomaly(rule, field->next, branch->node, CORRELATION)
22                     else
23                         DiscoverAnomaly(rule ,field->next ,branch->node ,GENERALIZATION)
24                     end if
25                 end if
26             end if
27         if value_found = False then
28             new_branch ← new TreeBranch(rule, rule->field, rule->filed->value)
29             node->branch_list->add(new_branch)
30             DiscoverAnomaly(rule, field->next, new_branch->node, NOANOMALY)
31         end if
32     else
33         DecideAnomaly(rule, field, node, anomaly_state)
34     end if
35 end
```

Алгоритм *DecideAnomaly* [4], который выносит конечный вердикт относительно наличия или отсутствия коллизий, вызывается, когда путь добавляемого правила в дереве правил был полностью определен и достигнут атрибут *«action»*.

Algorithm 2.3: Алгоритм DecideAnomaly

```
1  input: rule, field, node, anomaly
2  begin
3      if node in branch_list then
4          branch ← node->branch_list->first()
5          if anomaly = CORRELATION then
6              if rule->action ≠ branch->value then
7                  report rule rule->id is in correlation with rule branch->rule->id
8              end if
9          elif anomaly = GENERALIZATION and rule->action ≠ branch->value then
10             report rule rule->id is a generalization of rule branch->rule->id
11         elif anomaly = GENERALIZATION and rule->action = branch->value then
12             branch->rule->setAnomaly(REDUNDANCY)
13             report rule branch->rule->id is redundant to rule rule->id
14         elif rule->action = branch->value then
15             anomaly ← REDUNDANCY
16             report rule rule->id is redundant to rule branch->rule->id
17         elif rule->action ≠ branch->value then
18             anomaly ← SHADOWING
19             report rule rule->id is shadowed by rule branch->rule->id
20         end if
21     end if
22     rule->setAnomaly(anomaly)
23 end
```

После того как тип коллизии был определен, новое правило может быть добавлено в политику безопасности (коллизии отсутствуют), не добавлено (перекрытие или избыточность) или добавлено частично (корреляция). Последний случай является наиболее интересным и сложным, поэтому его стоит рассмотреть отдельно.

Если было определено, что два правила коррелируют, то их можно «разбить» на непересекающиеся части и добавить полученные новые правила в политику безопасности. С этой целью сначала находится множество атрибутов, значения которых у данных правил различны. После этого для каждого из найденных атрибутов вызывается алгоритм *Split*, приведенный ниже, который изменяет правила r и s таким образом, чтобы они не пересекались. Алгоритм *Split* получает на вход два пересекающихся правила r и s и атрибут a, значение которого у данных правил не равно.

Как видно из рисунка 3, общая часть значения атрибута всегда начинается с $max(r.a.start, s.a.start)$ и заканчивается $min(r.a.end, s.a.end)$. Непересекающаяся часть до общей части всегда начинается с $min(r.a.start, s.a.start)$ и заканчивается $max(r.a.start, s.a.start) - 1$. Непересекающаяся часть после общей части всегда начинается с $min(r.a.end, s.a.end) + 1$ и заканчивается $max(r.a.end, s.a.end)$. Непересекающиеся части правил r и s могут быть добавлены в дерево правил посредством алгоритма *DiscoverAnomaly*, так как не гарантировано, что они не конфликтуют с уже существующими правилами.

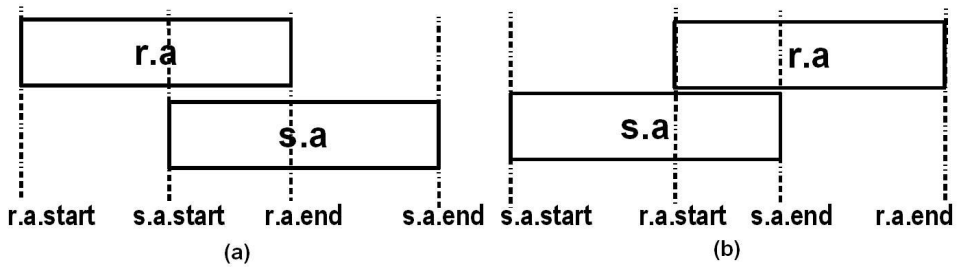


Рис. 3: (a) $r.a.start < s.a.start \wedge r.a.end < s.a.end$, таким образом, интервал может быть разбит на $[r.a.start, s.a.start - 1]$, $[s.a.start, r.a.end]$ и $[r.a.end + 1, s.a.end]$; (b) $r.a.start > s.a.start \wedge r.a.end > s.a.end$, таким образом, интервал может быть разбит на $[s.a.startr.a.start - 1]$, $[r.a.start, s.a.end]$ и $[s.a.end + 1, r.a.end]$

Fig. 3: (a) $r.a.start < s.a.start \wedge r.a.end < s.a.end$, thus, this interval can be split into $[r.a.start, s.a.start - 1]$, $[s.a.start, r.a.end]$, and $[r.a.end + 1, s.a.end]$; (b) $r.a.start > s.a.start \wedge r.a.end > s.a.end$, thus, this interval can be split into $[s.a.startr.a.start - 1]$, $[r.a.start, s.a.end]$, and $[s.a.end + 1, r.a.end]$

В алгоритме *Split* [4] сначала вычисляется общая часть двух пересекающихся правил, затем вычисляется непересекающаяся часть до общей части и добавляется в дерево правил как новое правило. После этого вычисляется непересекающаяся часть после общей части и также добавляется в дерево правил.

Algorithm 2.4: Алгоритм Split

```
1   input: r, s, a
2   begin
3       left ← min(r->a->start, s->a->start)
4       right ← max(r->a->end, s->a->end)
5       common_start ← max(r->a->start, s->a->start)
6       common_end ← min(r->a->end, s->a->end)
7       if r->a->start > s->a->start then
8           DiscoverAnomaly(⟨(left, common_start-1), s' rest attributes ⟩,
9               first_filed, first_node, NOANOMALY)
10      elif r->a->start < s->a->start then
11          DiscoverAnomaly(⟨(left, common_start-1), r's rest attributes ⟩,
12              first_filed, first_node, NOANOMALY))
13      elif r->a->end > s->a->end then
14          DiscoverAnomaly(⟨(common_end+1, right), r's rest attributes ⟩,
15              first_filed, first_node, NOANOMALY))
16      elif r->a->end < s->a->end then
17          DiscoverAnomaly(⟨(common_end+1, right), s' rest attributes ⟩,
18              first_filed, first_node, NOANOMALY))
19      r ← ⟨(common_start, common_end), r's rest attributes ⟩
20      s ← ⟨(common_start, common_end), s' rest attributes ⟩
21  end
```

После завершения работы алгоритмов дерево правил будет свободно от всех коллизий, обозначенных в разделе 1.2.

2.3. Анализ сложности алгоритмов разрешения коллизий

Сложность алгоритма определения и разрешения коллизий *DiscoverAnomaly* во многом зависит от правил, которые подаются на вход. Если алгоритм получает на вход правило, которое не пересекается или является надмножеством для всех правил в политике безопасности, то данное правило добавляется в политику безопасности сразу. Сложность данной операции в среднем — $O(1)$ (в худшем $O(n)$), так как в общем случае понадобится на всех 12 уровнях дерева правил по одному разу добавить в хеш-таблицу новое значение либо удостовериться, что значение уже присутствует и спуститься на уровень ниже в дереве (приведенный анализ трудоемкости не учитывает возможное перехеширование). Если добавляемое правило является подмножеством какого-либо ранее добавленного правила, такое правило будет добавлено в политику безопасности. Исходя из вышесказанного, сложность данной операции в среднем $O(1)$ (в худшем $O(n)$).

Поступающее правило будет отброшено, если оно равно какому-либо ранее установленному правилу. Чтобы это понять, в худшем случае придется пройти по дереву от корня и до листа, что, учитывая фиксированную высоту дерева, тоже имеет трудоемкость в среднем $O(1)$ (в худшем $O(n)$).

Если поступающее правило коррелирует с каким-либо ранее установленным, то данные правила будут разбиты на непересекающиеся части. В худшем случае число сгенерированных таким образом правил будет в два раза превышать число атрибутов правил, а установлены данные правила будут с помощью вызовов метода DiscoverAnomaly. Так как количество атрибутов у правил фиксировано, трудоемкость данной операции в среднем — $O(1)$ (в худшем $O(n)$).

Таким образом, мы получили, что сложность предложенного алгоритма в среднем — $O(1)$ (в худшем $O(n)$).

Заключение

Основным преимуществом алгоритма разрешения коллизий, представленного в разделе 2.2., является, в общем случае, его константная трудоемкость. Однако данный алгоритм имеет один существенный недостаток. Без кардинальных изменений его трудно адаптировать для работы с правилами, содержащими символы-джокеры. Слово «трудно» здесь следует понимать в том смысле, что задача адаптации представленного алгоритма таким образом, чтобы он корректно обрабатывал символы-джокеры, сохраняя при этом константную трудоемкость, является нетривиальной. В будущих исследованиях планируется решить данную проблему.

Помимо всего прочего, вопрос о необходимости устранения всех типов коллизий остается открытым. Часто системные администраторы намеренно включают в политику безопасности правила, которые коррелируют с другими. Разрешение таких коллизий является ошибкой, а следовательно, в дальнейших работах по усовершенствованию представленного алгоритма следует это учитывать.

Список литературы / References

[1] Al-Shaer E., et al., “Automated Firewall Analytics. Design, Configuration and Optimization”, Proceedings of 2016 IEEE Trustcom/BigDataSE/ISPA, 2014, 15–18.

[2] Abedin M., Nessa S., Khan L., Thuraisingham B., “Detection and Resolution of Anomalies in Firewall Policy Rules”, *Data and Applications Security XX*, Springer, 2006, 15–29.

[3] Morzhov S., Nikitinskiy M., “Development and research of the PreFirewall network application for Floodlight SDN controller”, 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT) (Moscow, March 14–16), 2018, 1–4.

[4] Morzhov S., Sokolov V., Nikitinskiy M., Chaly D., “Building a Security Policy Tree for SDN Controllers”, 2018 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTec) (Moscow, October 25), 2018, 1–6.

[5] Morzhov S., Alekseev I., Nikitinskiy M., “Firewall application for Floodlight SDN controller”, XII International Siberian Conference on Control and Communications (SIBCON-2016) (Moscow, May 12–14), 2016, 1–5.

Morzhov S. V., Sokolov V. A., "An Effective Algorithm for Collision Resolution in Security Policy Rules", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 75–89.

DOI: 10.18255/1818-1015-2019-1-75-89

Abstract. A firewall is the main classic tool for monitoring and managing the network traffic on a local network. Its task is to compare the network traffic passing through it with the established security rules. These rules, which are often also called security policy, can be defined both before and during the operation of the firewall. Managing the security policy of large corporate networks is a complex task. In order to properly implement it, firewall filtering rules must be written and organized neatly and without errors. In addition, the process of changing or inserting new rules should be performed only after a careful analysis of the relationship between the rules being modified or inserted, as well as the rules that already exist in the security policy. In this article, the authors consider the classification of relations between security policy rules and also give the definition of all sorts of conflicts between them. In addition, the authors present a new efficient algorithm for detecting and resolving collisions in firewall rules by the example of the Floodlight SDN controller.

Keywords: access control list, firewall, software defined network, ACL, SDN, security policy tree

On the authors:
Sergey V. Morzhov, graduate student, orcid.org/0000-0001-6652-3574
P.G. Demidov Yaroslavl State University,
14 Sovetskay st., Yaroslavl, 150003, Russia, e-mail: smorzhov@gmail.com

Valeriy A. Sokolov, Doctor, Professor, orcid.org/0000-0003-1427-4937
Centre of Integrable Systems, P.G. Demidov Yaroslavl State University,
14 Sovetskay st., Yaroslavl, 150003, Russia, e-mail: valery-sokolov@yandex.ru

Acknowledgments:
The work was funded by Russian Foundation for Basic Research, according to the research projects No. 17-07-00823 A, and was carried out within the framework of the state program of the Ministry of Education and Science of the Russian Federation, project № 1.10160.2017/5.1

Моделирование и анализ информационных систем. Т. 26, №1 (2019), с. 90–100
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 90–100

DOI: 10.18255/1818-1015-2019-1-90-100

УДК 004.7

Разработка и исследование алгоритмов формирования правил для узлов сетевой безопасности в мультиоблачной платформе

Парфёнов Д. И., Болодурина И. П., Торчин В. А.

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

Аннотация. В рамках исследования рассмотрены существующие решения, направленные на обеспечение безопасности сетевого периметра мультиоблачной платформы. Установлено, что наиболее острой является проблема эффективного формирования правил на межсетевых экранах. Существующие подходы не позволяют оптимизировать список правил на узлах, контролирующих доступ к сети. Целью исследования является повышение эффективности средств межсетевого экрана путем бесконфликтной оптимизации правил безопасности и применения нейросетевого подхода в программно-определяемых сетях. Предлагаемое решение основано на совместном использовании интеллектуальных математических подходов и современных технологий виртуализации сетевых функций. В ходе экспериментальных исследований проведен сравнительный анализ традиционных средств формирования правил, нейросетевого подхода, а также генетического алгоритма. Для автоматического построения правил сетевой безопасности рекомендуется применять нейросетевой классификатор архитектуры «многослойный персептрон», поскольку он даёт лучшие результаты с точки зрения производительности, и уменьшать размерность списка правил безопасности межсетевого экрана при помощи сети Кохонена, поскольку это средство показывает лучшую производительность. В спроектированную архитектуру был внедрен алгоритм бесконфликтной оптимизации, который производит конечную оптимизацию путем ранжирования и выведения наиболее часто встречаемых исключений из больших запретительных правил, что позволяет увеличить защиту от атак, которые направлены на выявление правил безопасности, находящихся внизу списка межсетевого экрана. На базе предложенного решения в рамках исследования реализован модуль адаптивного межсетевого экрана.

Ключевые слова: адаптивный межсетевой экран, программно-конфигурируемая сеть, мультиоблачные платформы, нейронная сеть, виртуализация сетевых функций, кибербезопасность

Для цитирования: Парфёнов Д. И., Болодурина И. П., Торчин В. А., "Разработка и исследование алгоритмов формирования правил для узлов сетевой безопасности в мультиоблачной платформе", *Моделирование и анализ информационных систем*, **26**:1 (2019), 90–100.

Об авторах: Парфёнов Денис Игоревич, канд. техн. наук, orcid.org/0000-0002-1146-1270
Оренбургский государственный университет,
пр. Победы, 13, г. Оренбург, 460018 Россия, e-mail: parfenovdi@mail.ru

Болодурина Ирина Павловна, д-р техн. наук, профессор, orcid.org/0000-0003-0096-2587
Оренбургский государственный университет, e-mail: prmat@mail.osu.ru

Торчин Вадим Александрович, студент, orcid.org/0000-0002-5315-6047
Оренбургский государственный университет, e-mail: vadim.torchin@gmail.com

Благодарности:
Работа выполнена при финансовой поддержке РФФИ, научные проекты № 18-07-01446 и № 16-29-09639.

Введение

В настоящее время активно развивается рынок телекоммуникационных услуг. Крупные организации для более эффективного ведения бизнеса арендуют виртуальные центры обработки данных (ЦОД) для размещения собственной ИТ-инфраструктуры. Поэтому наиболее востребованным сегментом таких услуг является предоставление пользователям сетевых сервисов на базе мультиоблачных платформ. Популярность использования таких технических решений привела к тому, что пользователи и поставщики телекоммуникационных услуг ежедневно сталкиваются с проблемами, связанными с угрозами в сфере кибербезопасности.

Согласно аналитическим данным ведущих поставщиков сетевого оборудования, таких как Cisco и Huawei, количество активных угроз кибербезопасности ежегодно увеличивается на 15–25%. Оценивая вектор атак на ИТ-инфраструктуру, которая поддерживает работу информационных систем в крупных компаниях, можно составить следующий рейтинг угроз: ограничение доступа легитимных пользователей к ключевым ресурсам компании (25%); нарушение работы технологического оборудования (35%); получение несанкционированного доступа к служебной или конфиденциальной информации, а также ее намеренное или случайное раскрытие, искажение либо уничтожение вследствие нарушения политики безопасности предприятия (45%). На практике список кибератак, направленных на корпоративную сеть, можно разделить на четыре основные группы. Он включает вектор атаки типа отказ в обслуживании (DDoS), вектор атаки от удаленных подключений на локальные (пользовательские) узлы (R2L), вектор атаки от локального пользователя к корневому узлу сети (U2R), а также вектор атаки типа грубого перебора узлов (Probing) [9].

Для предотвращения активных угроз провайдерам необходимы эффективные средства, которые позволяют осуществлять контроль процессов, протекающих в сети, мониторинг сервисов, размещенных в ней, а также проактивное управление элементами безопасности. На сегодняшний день наиболее востребованным и эффективным подходом к организации сети для предоставления услуг на базе виртуальных центров обработки данных является использование технологии программно-конфигурируемых сетей (Software-Defined Network, SDN). Применение данной технологии обусловлено рядом преимуществ. В первую очередь SDN значительно упрощает проектирование и эксплуатацию сети, поскольку она позволяет осуществлять централизованное интеллектуальное управление на уровне контроллера. Во-вторых, SDN позволяет сетевым администраторам быстро конфигурировать и оптимизировать сетевые ресурсы на основе агрегированного набора данных, собираемых в едином центре. В-третьих, использование SDN позволяет обеспечивать защиту с помощью динамического анализа потоков данных, циркулирующих в виртуальном центре обработки данных [10].

Еще одной технологией, применяемой для организации сети на базе виртуальных центров обработки данных, является технология виртуализация сетевых функций (Network Function Virtualization, NFV). Технология NFV предлагает новый способ проектирования, развертывания сетевых сервисов на базе мультиоблачной платформы. Виртуализация сетевых функций позволяет не только отделить сетевые функции, такие как NAT, firewall, IDS, IPS, DNS, от аппаратного уровня, но и объ-

единить все сетевые компоненты, необходимые для поддержки виртуализованной инфраструктуры на программном уровне [11]. Как и SDN, технология NFV также дает преимущества при проектировании защищенной сетевой среды в виртуальном центре обработки данных. Одним из основных преимуществ технологии NFV является масштабируемость. Для быстрого удовлетворения динамически меняющихся потребностей пользователей и предоставления новых услуг, провайдеры телекоммуникационных услуг должны иметь возможность адаптировать свою сетевую архитектуру, не меняя при этом состав аппаратного обеспечения.

Технологии SDN и NFV относятся к технологиям компьютерных сетей нового поколения и могут сосуществовать в одной сетевой среде, используя при этом общую аппаратную платформу на физическом уровне. Поэтому в рамках настоящего исследования нами предложено решение, основанное на гибридном использовании SDN и NFV для организации сетевой безопасности для мультиоблачной платформы, развернутой на базе виртуального центра обработки данных. Целью исследования является повышение эффективности средств межсетевого экрана путем бесконфликтной оптимизации правил безопасности и применения подхода нейронной сети в программно-определяемых сетях.

1. Обзор исследований

На сегодняшний день разработано достаточно много решений, позволяющих осуществлять защиту облачных платформ от кибератак. В рамках настоящего исследования нами проведен обзор таких решений. Большинство из них основано на использовании механизмов защиты на базе межсетевых экранов. Это обусловлено тем, что большинство атак поступают в сеть извне.

Механизм защиты на базе межсетевых экранов основан на использовании листов правил, разрешающих или запрещающих доступ к определенным ресурсам. Основной проблемой безопасности традиционной архитектуры межсетевых экранов является конфликт правил, возникающий в результате неправильной конфигурации. Кроме того, в больших сетях существует проблема длинных листов правил. Это особенно актуально для сетей, обеспечивающих работу платформы облачных вычислений. В таких сетях, как правило, присутствует достаточно большое множество наложенных и пересекающихся потоков, и разграничение правил в таких сетях является не тривиальной задачей. В рамках своего исследования Thawatchai Chomsiri предложил механизм Tree-Rule, который не сталкивается с такими конфликтами правил в пределах своего набора правил и работает быстрее, чем традиционные брандмауэры [1]. Тем не менее, авторы исследования отмечают, что проблемы правильности конфигурирования остаются не решенными в данной работе.

Еще одной не менее важной проблемой обеспечения безопасности на основе Firewall является единая точка отказа. Firewall, как правило, располагается на границе сети и осуществляет фильтрацию входящего трафика. Однако при высокой интенсивности потока трафика, например при DOS или DDOS атаке, Firewall может не справиться с нагрузкой. Авторами исследования [2] предложено решение на базе кластерного подхода, позволяющее осуществлять балансировку нагрузки между узлами при возрастании нагрузки на средства обеспечения безопасности. Однако

авторы не затрагивают в своем исследовании вопросов репликации правил между узлами кластера. Кроме того, данный подход не решает проблему длинных ACL правил.

Еще одним узким местом работы механизмов защиты на базе межсетевых экранов является процесс фильтрации пакетов. При высокой интенсивности запросов очень важным параметром является время принятия решения. В работе [3] предложен подход, позволяющий ускорить процедуру анализа пакетов. В основе решения применен гибридный подход, основанный на совместном использовании двух алгоритмов эффективного геометрического сопоставления и оптимизации смены состояний межсетевого экрана (GEM-iptables & nftables algorithm). Использование GEM-iptables & nftables позволяет ускорить фильтрацию пакетов, при этом не изменяя существующую схему работы Firewall. Однако такой подход эффективен только для традиционных сетей.

На сегодняшний день большинство провайдеров облачных услуг используют для обеспечения работы сети решения, основанные на программно-конфигурируемых сетях. Использование SDN позволяет решить сразу несколько проблем безопасности облачной платформы. Например, группой исследователей под руководством S. Kaur предложен распределенный межсетевой экран (Distributed Firewall), использующий в своей основе технологию SDN. В предложенном решении каждый коммутатор OpenFlow в сети может выступать в качестве брандмауэра [4]. В другом исследовании авторы используют функциональные возможности протокола OpenFlow для записи информации о текущих потоках в сетях для автоматизированного формирования правил на элементах безопасности [5]. Несмотря на все преимущества данной системы, она не в состоянии проводить оптимизацию списка правил, что сказывается на ее работе в больших сетях.

При большом количестве правил на границе сети межсетевой экран вынужден последовательно проверять весь список. Это вносит существенную задержку в работу легитимных пользователей. Для ускорения данного процесса исследователями в работе [6] предложено решение, использующее в качестве анализируемого параметра MAC-адреса. Данный метод более эффективный, чем фильтрация по IP-адресам, но он подходит только для внутренней сети и совершенно не применим для границы сети.

Другое решение в данном направлении предложено в работе [7]. Исследователи предлагают модель объединения правил фильтрации, используя алгоритм, основанный на анализе потоков трафика, классифицируя их по сервисам доступа. Однако такой подход не решает вопросов конфликтов правил для разных групп пользователей.

При возникновении инцидентов безопасности очень важно исследовать журналы доступа и применяемые списки правил межсетевого экранирования. Исследователи F. Ertam и M. Kaya предложили ряд решений для поиска закономерностей в лог-файлах с использованием классификатора векторных машин с поддержкой мультиклассов (SVM) [8]. Данное решение позволяет предотвращать новые неизвестные типы атак.

Обзор исследований показал, что для эффективного построения списка правил для межсетевого экрана необходимо решить две задачи:

– автоматизировать построение правил для межсетевого экрана на основе данных о трафике, циркулирующем в сети;

– осуществить оптимизацию полученного списка правил для межсетевого экрана.

Для решения первой задачи необходимо выполнить классификации сетевого трафика, проходящего через корпоративную сеть. Для решения второй задачи применим итерационный метод, в рамках которого выделим два основных этапа. На первом этапе выполним кластеризацию правил, полученных в результате решения первой задачи, и следующее за ней выведение правил из кластеров. На втором этапе для решения второй задачи применим алгоритмом бесконфликтной оптимизации списка правил межсетевого экранирования.

2. Модель создания адаптивных правил для межсетевого экрана

Для реализации представленного плана необходимо в первую очередь определить модель правил для межсетевого экрана, используемых для обеспечения безопасности корпоративной мультиоблачной платформы.

Правило межсетевого экрана обычно представляет собой строку, состоящую из определенных характеристик сетевого соединения и решения о допустимости такого соединения. В рамках исследования из множества характеристик, описывающих сетевые соединения, были выбраны следующие: IP-адрес отправителя и получателя, соответствующие им порты и протокол, по которому осуществляется передача данных. При выборе этих характеристик правило межсетевого экрана в общем виде будет иметь вид

$$< rn, id\_src\_ip, id\_dst\_ip, src\_p, dst\_p, p\_id, tg, p\_cnt >, \tag{1}$$

где rn – номер правила в списке; id_src_ip – идентификатор IP-адреса отправителя пакета; id_dst_ip – идентификатор IP-адреса получателя пакета; sr_p – порт отправителя пакета; dst_p – порт получателя пакета; p_id – сетевой протокол, через который осуществляется соединение; tg – решение о допустимости или недопустимости соединения; p_cnt – количество пакетов трафика.

Список записей о пакетах представим в виде множества следующего вида:

$$X = \{x_k\}, k = \overline{1, n}, \tag{2}$$

где n – длина списка правил межсетевого экранирования, применяемого для обеспечения безопасности мультиоблачной платформы.

Определим области допустимых значений для элементов данного вектора, а именно характеристики трафика, по которым будет осуществляться фильтрация передаваемых данных в сети мультиоблачной платформы. Они представлены в виде списка строк вида (1). Каждую такую строку представим в виде вектора вида

$$x_k = \{x_{k1}, x_{k2}, x_{k3}, x_{k4}, x_{k5}\}, \tag{3}$$

где k – номер записи в списке; $x_{k1} \in [0; 2^{32} - 1]$ – IP-адрес отправителя пакета; $x_{k2} \in [0; 2^{32} - 1]$ – IP-адрес получателя пакета; $x_{k3} \in [0; 65535]$ – порт отправителя пакета; $x_{k4} \in [0; 65535]$ – порт получателя пакета; $x_{k5} \in \{0, 1, 2\}$ – сетевой протокол, по которому осуществляется соединение.

Далее представим список правил межсетевого экрана, где правила соответствуют модели (2) в виде множества

$$R = \{r_i : r_i = \{r_{i,1}, r_{i,3}, r_{i,4}, r_{i,5}, r_{i,6}, r_{i,7}, r_{i,8}\}\} \tag{4}$$

где $r_{i,1} \in [0; m]$ – номер правила в списке; $r_{i,2} \in [0; 2^{32} - 1]$ – IP-адрес отправителя пакета; $r_{i,3} \in [0; 2^{32} - 1]$ – IP-адрес получателя пакета; $r_{i,4} \in [0; 65535]$ – номер порта отправителя пакета; $r_{i,5} \in [0; 65535]$ – номер порта получателя пакета; $r_{i,6} \in N$ – номер условного обозначения протокола, хранящегося в БД контроллера сети; $r_{i,7} \in \{0; 1\}$ – решение о допустимости или недопустимости соединения, где 0 – соединение запрещено, 1 – соединение разрешено; $r_{i,8} \in N$ – количество пакетов трафика.

После приведения входных данных к необходимому виду, сформулируем постановку задачи. Пусть дано множество записей о трафике, проходящем через сеть мультиоблачной платформы, в виде множества вида (3). Требуется построить множество неконфликтных правил $R = r_i, i = \overline{1, m}, m \to min$ вида (4).

Данная работа предполагает решение задачи итерационным методом в два этапа. На первом этапе осуществляется построение первоначального списка правил R_1 путем классификации множества X на два класса. На втором этапе осуществляется построение множества R_{opt} путем кластеризации множества R_1 с последующим выведением правил из кластеров, то есть построением множества R_2 с последующей оптимизацией этого множества алгоритмом бесконфликтной оптимизации, то есть построением множества R_{opt}.

Пусть дано множество X вида (4), требуется построить список правил, формирующих множество R_1. Это означает, что нужно построить классифицирующую функцию вида $f(x) : X \to R$.

Существует множество различных методов для решения данной задачи. В рамках настоящего исследования будем использовать классификацию на основе нейронной сети. Это обусловлено тем, что классификация является классической задачей для нейросетевых методов. Оптимальной для решения подобной задачи является архитектура нейронной сети типа многослойный перцептрон.

Количество нейронов на входном слое вычисляется в зависимости от входных данных. В данном случае размер входного слоя нейронной сети составит 99 нейронов. На запись IP-адреса приходится по 32 бита, а на запись портов по 16 бит, число рассматриваемых протоколов = 7, следовательно, для их записи необходимо 3 бита. Для обучения выбранной модели нейронной сети будем использовать алгоритм обратного распространения ошибки.

2.1. Кластеризация правил межсетевого экранирования

Одним из нередко встречаемых типов атак на компьютерные сети является постоянно повторяющаяся попытка получить доступ к определенному ресурсу с расчетом, что типовой пакет трафика данной атаки на сеть будет удовлетворять определенному правилу межсетевого экрана. Цель атаки – найти правило, находящееся

внизу списка. В результате такой атаки происходит стремительный рост нагрузки на процессор и увеличение объема потребляемой оперативной памяти на межсетевом экране. Это приводит к падению производительности системы безопасности в целом. В целях снижения затрат памяти и времени обхода списка правил межсетевым экраном, необходимо решить задачу сокращения списка правил, не потеряв при этом характеристики, отвечающие за защищенность мультиоблачной платформы. Для этих целей имеет смысл разбить правила на кластеры с целью выведения новых, обобщенных правил из них.

Сформулируем математическую постановку задачи кластеризации правил межсетевого экранирования. Пусть дано множество правил $R = \{r_i\}, i = \overline{1, m}$, требуется построить разбиение выборки на непересекающиеся подмножества, называемые кластерами, таким образом, чтобы каждое подмножество состояло из близких по некоторой метрике объектов, другими словами, построить кластеризующую функцию $f(r) : R \to Y$, которая ставит каждому элементу множества R в соответствие элемент множества $Y = \{y_1, y_2, \ldots\}$ – множества номеров кластеров.

Важным аспектом при решении задачи кластеризации является выбор функции расстояния, или метрики. Метрика является мерой близости, которой пользуются алгоритмы. В рамках исследования в качестве метрики было взято евклидово расстояние по следующим признакам: IP-адреса отправителя и получателя, соответствующие им порты, протокол, по которому осуществляется соединение и решение о допустимости соединения. Таким образом, формула функции расстояния имеет следующий вид:

$$D(r_1, r_2) = \sqrt{a(r_{1,2} - r_{2,2})^2 + b(r_{1,3} - r_{2,3})^2 + c(r_{1,5} - r_{2,5})^2 + d(r_{1,7} - r_{2,7})^2}. \quad (5)$$

Эта функция использовалась с эмпирически подобранными параметрами $a = 0.55$; $b = 0.55$, $c = 38745.6$; $d = 2^4 - 1$.

2.2. Алгоритм выведения правил межсетевого экрана из кластеров

Следующим этапом является выведение правил из кластеризованного списка правил. Входными данными будет являться список правил, разбитый на кластеры $R_{k,l}$. На выходе алгоритма ожидается список обобщенных правил R_{opt}. В рамках данного исследования был разработан алгоритм, приведенный в виде псевдокода ниже. Список кластеров обозначим u. Тогда для всех кластеров из u справедливо:

Вход: Кластеризованный список правил R_{kl}, список кластеров u

Выход: Общий список правил R_{opt}

$r_{i,1} = min(r_{k,1})/mask = 32 - log_2(max(r_{k,1}) - min(r_{k,1}))$; $r_{i,2} = min(r_{k,2})/mask = 32 - log_2(max(r_{k,2}) - min(r_{k,2}))$; $r_{i,3} = \{r_{1,3}, \ldots, r_{k,3}\}$; $r_{i,4} = \{r_{1,4}, \ldots, r_{k,4}\}$; $r_{i,5} = \{r_{1,5}, \ldots, r_{k,5}\}$; $r_{i,6} = r_{i,6}$.

2.3. Алгоритм оптимизации списка правил ранжированной сортировкой

Важным параметром для списка правил, помимо широты охвата защищаемых ресурсов и величины списка, является также расстановка правил. Защитить сеть мультиоблачной платформы от атак, направленных на выполнение правила, находящегося в конце списка, возможно при помощи сортировки списка правил. Цель сортировки – поместить наиболее часто используемые правила вверх списка, чтобы исключить расходование ресурсов системы на последовательную проверку большого количества правил. За счет оптимизации данного процесса возможна существенная экономия и машинного времени, и снижение нагрузки на устройстве, осуществляющем функции межсетевого экрана.

Для решения этой задачи был разработан следующий алгоритм, представленный ниже

Дано:

– Множество заголовков трафика, проходящего по правилу «Запретить всё» – X;

– Неконфликтный список правил $R = \{R_i\}, i = 1, n$, где Rn – «Запретить всё».

Шаг 1: Задать каждому правилу и каждому элементу множества X вес по формуле $w_i = \frac{k_i}{k}$, где k_i – количество трафика, проходящего по правилу i, k – общее количество пакетов, проходящих через сеть.

Шаг 2: IF $w_n > w^*$ THEN перейти к **шагу 3**, ELSE перейти к **шагу 5**.

Шаг 3: Создать запрещающее правило R_{n-1}, по данным элемента X.

Шаг 4: Отсортировать множество R по величине, поставить правило «Запретить всё» на последнее место. Перейти к **шагу 2**.

Шаг 5: Конец алгоритма.

Данный алгоритм не уменьшает размер списка, но позволяет составить оптимальный список правил. При этом обеспечивается защищенность против атак, направленных на возникновение сбоев в работе средств межсетевого экранирования.

3. Экспериментальные исследования

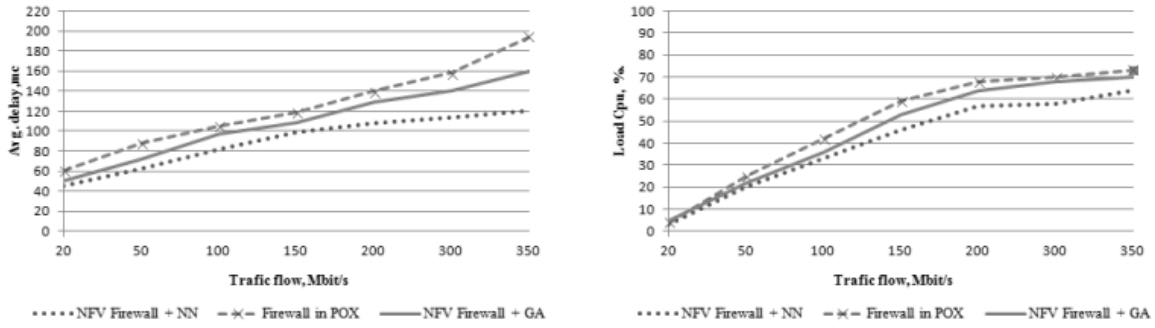
На базе предложенного решения в рамках исследования реализован модуль адаптивного межсетевого экрана. Программное обеспечение реализовано в виде виртуальной сетевой функции на базе Open Platform for NFV (OPNFV), собранной в виде контейнера Docker. Для сравнения был выбран традиционный межсетевой экран, являющийся пакетным фильтром, реализованный внутри платформы POX.

Сравнительный анализ проводился путем сопоставления результатов работы традиционного межсетевого экрана и разработанного программного модуля, а также с использованием оптимизации списка правил сетевой безопасности на основе генетического алгоритма. Перед проведением эксперимента разработанный программный модуль был обучен, а также был разработан сопоставимый набор правил для традиционного межсетевого экрана, что позволяет проводить корректное сравнение сопоставляемых средств. В рамках исследования оценивалась работа при различной нагрузке по двум ключевым показателям: время отклика в сети; нагрузка центрального процесса на межсетевом экране.

Для проведения нагрузочного тестирования сценариев экспериментальных атак была создана виртуальная сеть в облачной системе OpenStack. Она включает 4 коммутатора OpenFlow (2 HP 3500yl, 2 Netgear GSM7200), 8 вычислительных узлов (32 ГБ ОЗУ, 4 ядра), 1 сервер (32 ГБ ОЗУ, 8 ядер) с контроллером OpenFlow и 1 сервер (32 ГБ ОЗУ, 4 ядра) для мониторинга работы виртуальных сетевых функций. В качестве топологии выбрана fat tree с тремя уровнями. Маршрутизаторы имеют скорость соединения 1000 Мбит/с. Вычислительные узлы подключаются к маршрутизатору третьего уровня через сетевые соединения второго уровня со скоростью 1000 Мбит/с. В развернутой инфраструктуре было подготовлено 100 виртуальных машин (атакующих узлов). Среди них случайным образом выбирался один узел, который контролировал атаку. Так же случайным образом выбирались пять атакованных виртуальных машин (сервисный хост).

На выбранные узлы при помощи специально предназначенного для этого средства – hping3 создавалась нагрузка, характерная для сетевых атак типа DDoS. В рамках исследования генерировались пакеты различной длины с предварительным выбором узла назначения. Это позволило максимально приблизить структуру экспериментального трафика к реальным данным. Сам эксперимент состоял в замере показателей производительности сети и оборудования при последовательном увеличении интенсивности потоков трафика в рамках разовой нагрузки в диапазоне от 20 до 350 Мбит/с.

В рамках эксперимента проведена оценка времени отклика – одного из ключевых параметров, применяемых при анализе производительности сети и сетевых устройств. Она показывает, сколько времени проходит с момента отправки запроса пользователем до момента ответа на запрос сервисом. В соответствии с правилами эксперимента были сняты данные о времени отклика с сети.



a) b)

Рис. 1. Результаты экспериментального исследования характеристик сети и параметров работы модулей узлов формирования правил для межсетевого экрана

Fig 1. The results of an experimental study of the characteristics of the network and the parameters of the modules of the nodes forming the rules for the firewall

Список правил безопасности, созданный построенным модулем формирования правил для межсетевого экрана при нагрузке, в том числе лавинообразной, дает сокращение во времени отклика до 20%, что говорит об эффективности разра-

ботанного в исследовании подхода (Рис. 1, a). Нагрузка на центральный процессор межсетевого экрана является не менее важным параметром при рассмотрении средств безопасности. Это показатель того, является ли устройство загруженным или неэффективно используемым. Исходя из полученных результатов имеем, что оптимизированный набор правил, построенный с использованием рассмотренного в исследовании подхода, существенно снижает нагрузку на процессор межсетевого экрана (Рис. 1, b).

Заключение

В ходе исследования был проведен обзор существующих решений в области межсетевого экранирования, был осуществлен выбор архитектуры нейронной сети для достижения поставленной цели, а именно гибридной искусственной нейронной сети, состоящей внутри из двух сетей – классификатора на основе многослойного персептрона и кластеризатора на основе нейронной сети Кохонена, исследованы существующие подходы и алгоритмы для решения задачи обеспечения безопасности информационных ресурсов сети мультиоблачной платформы. В рамках исследования была разработана модель описания правил безопасности межсетевого экрана и на базе нее разработаны алгоритмы автоматического построения и оптимизации списка безопасности межсетевого экрана.

В ходе испытаний было выявлено, что предлагаемый данным исследованием подход дает прирост производительности по двум ключевым параметрам: по времени отклика – в среднем на 20% с ростом нагрузки и по нагрузке центрального процесса экранирующего устройства – в среднем на 4,5% с ростом нагрузки. Таким образом, разработанный подход является эффективным для решения практических задач.

Список литературы / References

[1] Chomsiri T., et al., “An Improvement of Tree-Rule Firewall for a Large Network: Supporting Large Rule Size and Low Delay”, *Proceedings of 2016 IEEE Trustcom/BigDataSE/ISPA* (Tianjin, Aug 23–26), IEEE, 2016, 178–184.

[2] Zhichao P., et al., “A Load-Balancing and State-Sharing Algorithm for Fault-Tolerant Firewall Cluster”, *Proceedings of 2017 4th International Conference on Information Science and Control Engineering (ICISCE)* (Changsha, July 21–23), IEEE, 2017, 34–37.

[3] Nivedita, Kumar R., “An improved Linux firewall using a hybrid frame of netfilter”, *2017 International Conference on Trends in Electronics and Informatics (ICEI)* (Tirunelveli, May 11–12), IEEE, 2017, 657–662.

[4] Kaur S., et al., “Implementing openflow based distributed firewall”, *Proceedings of 2016 International Conference on Information Technology (InCITe) – The Next Generation IT Summit on the Theme – Internet of Things: Connect your Worlds* (Noida, Oct 6–7), IEEE, 2016, 172–175.

[5] Papagrigoriou A., et al., “A firewall module resolving rules consistency”, *Proceedings of 2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES)* (Hamburg, June 12–13), IEEE, 2017, 47–50.

[6] Rengaraju P., et al., “Investigation of security and QoS on SDN firewall using MAC filtering”, *Proceedings of 2017 International Conference on Computer Communication and Informatics (ICCCI)* (Coimbatore, Jan 5–7), IEEE, 2017, 1–5.

[7] Zhang L., Huang M., "A Firewall Rules Optimized Model Based on Service-Grouping", *Proceedings of 2015 12th Web Information System and Application Conference (WISA)* (Jinan, Sept 11–13), IEEE, 2015, 142–146.

[8] Ertam F., Kaya M., "Classification of firewall log files with multiclass support vector machine", *Proceedings of 2018 6th International Symposium on Digital Forensic and Security (ISDFS)* (Antalya, March 22–25), IEEE, 2018, 1–4.

[9] Atighetchi M., Adler A., "A Framework for Resilient Remote Monitoring", *Proceedings of 2014 7th International Symposium on Resilient Control Systems (ISRCS)* (Denver, Aug 19–21), IEEE, 2014, 1–8.

[10] Parfenov D., Bolodurina I., "Methods and algorithms optimization of adaptive traffic control in the virtual data center", *Proceedings of 2017 International Siberian Conference on Control and Communications (SIBCON 2017)* (Astana, June 29–30), IEEE, 2017, 1–6.

[11] Bolodurina I., Parfenov D., "Development and research of models of organization distributed cloud computing based on the software-defined infrastructure", *Procedia Computer Science*, **103** (2017), 569–576.

Parfenov D. I., Bolodurina I. P., Torchin V. A., "Development and Study of Algorithms for the Formation of Rules for Network Security Nodes in the Multi-Cloud Platform", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 90–100.

DOI: 10.18255/1818-1015-2019-1-90-100

Abstract. As part of the study, existing solutions aimed at ensuring the security of the network perimeter of the multi-cloud platform were considered. It is established that the most acute problem is the effective formation of rules on firewalls. Existing approaches do not allow optimizing the list of rules on nodes that control access to the network. The aim of the study is to increase the effectiveness of firewall tools by conflict-free optimization of security rules and the use of a neural network approach in software-defined networks. The proposed solution is based on the sharing of intelligent mathematical approaches and modern technologies of virtualization of network functions. In the course of experimental studies, a comparative analysis of the traditional means of rule formation, the neural network approach, and the genetic algorithm was carried out. It is recommended to use the multilayer perceptron neural network classifier for automatic construction of network security rules since it gives the best results in terms of performance. It is also recommended to reduce the size of the firewall security rule list using the Kohonen network, as this tool shows the best performance. A conflict-free optimization algorithm was introduced into the designed architecture, which produces finite optimization by ranking and deriving the most common exceptions from large restrictive rules, which allows increasing protection against attacks that are aimed at identifying security rules at the bottom of the firewall list. On the basis of the proposed solution, the adaptive firewall module was implemented as part of the research.

Keywords: adaptive firewall, software-defined network, multi-cloud platforms, neural network, network function virtualization, cyber security

On the authors:
Denis I. Parfenov, PhD, orcid.org/0000-0002-1146-1270
Orenburg State University,
13 Pobedy pr., Orenburg 460018, Russia, e-mail: parfenovdi@mail.ru

Irina P. Bolodurina, PhD, orcid.org/0000-0003-0096-2587,
Orenburg State University,
13 Pobedy pr., Orenburg 460018, Russia, e-mail: prmat@mail.osu.ru

Vadim A. Torchin, graduate student, orcid.org/0000-0002-5315-6047
Orenburg State University,
13 Pobedy pr., Orenburg 460018, Russia, e-mail: vadim.torchin@gmail.com

Acknowledgments:
The work was supported by Russian Foundation for Basic Research, Projects No. 18-07-01446, No. 16-29-09639.

Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 101–121
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 101–121

DOI: 10.18255/1818-1015-2019-1-101-121

УДК 517.9

Распределенная отказоустойчивая платформа управления для программно-конфигурируемых сетей

Пашков В. Н.

Поступила в редакцию 10 января 2019
После доработки 15 февраля 2019
Принята к публикации 18 февраля 2019

Аннотация. В рамках исследования рассматривается проблема обеспечения отказоустойчивости распределенной платформы управления для программно-конфигурируемых сетей. Целью исследования является разработка архитектуры и принципов организации отказоустойчивой распределенной платформы управления для ПКС. Отказоустойчивость распределенной платформы управления ПКС достигается за счет резервирования контроллеров, резервирования активных соединений между коммутатором и несколькими контроллерами, резервирования вычислительных ресурсов и использования дополнительных программных инструментов для обнаружения отказов, предотвращения перегрузок и восстановления управления сетью. В работе приводится алгоритм распределения управления коммутаторами между контроллерами платформы управления, выбора резервных контроллеров для каждого коммутатора, что позволяет минимизировать время восстановления в случае одиночных отказов контроллеров. Алгоритм балансировки нагрузки между контроллерами позволяет динамически переконфигурировать платформу управления с минимальным количеством операций передачи управления коммутаторами, чтобы предотвратить перегрузку контроллера. Представлены результаты экспериментального исследования предложенных алгоритмов.

Ключевые слова: программно-конфигурируемые сети, ПКС, распределенная платформа управления, РПУ, отказоустойчивость, балансировка нагрузки, OpenFlow, сетевая архитектура

Для цитирования: Пашков В. Н., "Распределенная отказоустойчивая платформа управления для программно-конфигурируемых сетей", *Моделирование и анализ информационных систем*, **26**:1 (2019), 101–121.

Об авторах:
Пашков Василий Николаевич, программист, orcid.org/0000-0001-5783-4557
Московский государственный университет им. М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: pashkov@lvk.cs.msu.su

Введение

Концепция программно-конфигурируемых сетей (ПКС, SDN) [1–3] предполагает логически централизованное управление сетью, при котором набор функций управления сосредоточен в сетевой операционной системе (или контроллере) и сетевых

приложениях, работающих на выделенном сервере. Контроллер поддерживает и контролирует глобальное состояние сети. Основываясь на глобальном состоянии сети, сетевые приложения контроллера осуществляют логически централизованное управление сетевыми устройствами, политиками и потоками данных в сети.

В статье рассматриваются программно-конфигурируемые сети, в которых взаимодействие между контроллером и сетевыми устройствами осуществляется по протоколу OpenFlow [4]. Архитектура и реализация контроллера не определены спецификацией OpenFlow, поэтому существует множество различных реализаций контроллеров с открытым исходным кодом и проприетарных сетевых операционных систем (NOX [5, 6], Beacon [7, 8], Floodlight [9], OpenMUL [10], RUNOS [11] и другие). Характеристики производительности, масштабируемости и надежности всей ПКС/OpenFlow сети и качество сетевых сервисов для конечных пользователей определяются функциональностью, стабильностью, производительностью контроллера, а также поведением контроллера в критических ситуациях, характеристиками масштабируемости, надежности и безопасности контура управления ПКС.

В то же время контроллер является единственной точкой отказа в ПКС сетях. Сбои сервера и его программные ошибки, перебои с питанием, ошибки контроллера и его сетевых приложений, уязвимости в системе безопасности или несанкционированный доступ к контроллеру могут привести к отказу ПКС контроллера и недоступности сетевых сервисов контроллера для коммутаторов сети. Следовательно, это может привести к частичной или полной потере управления над сетью, отказу сети и недоступности пользовательских сетевых сервисов.

Другим источником проблем для контура управления ПКС является перегрузка контроллера. Это может быть вызвано сбоями сервера и ограничениями его физической производительности, увеличением количества коммутаторов в сети, значительным увеличением количества новых потоков в сети или DDoS-атакой на контроллер. Перегрузка контроллера трафиком управления может привести к увеличению времени отклика контроллера на запросы от коммутаторов и, следовательно, к увеличению времени установления новых потоков для пользовательских сетевых сервисов.

Для внедрения подхода ПКС/OpenFlow в реальных сетях необходимо устранить эти недостатки и обеспечить отказоустойчивость контура управления для коммутаторов в случае отказа контроллера или его перегрузки.

Основным способом достижения отказоустойчивости является резервирование ресурсов в контуре управления. Это может быть реализовано путем распределения управления коммутаторами между несколькими ПКС контроллерами в контуре управления.

Для реализации этой идеи необходимо решить следующие задачи:

1. Как распределить управление коммутаторами между несколькими ПКС контроллерами?

2. Как перераспределить управление в случае отказа одного контроллера в распределенной платформе управления?

3. Как предотвратить перегрузку контроллера в распределенной платформе управления?

В разделе 1 статьи проводится краткий обзор существующих реализаций распределенных контроллеров для ПКС сетей. В разделе 2 предлагается архитектура отказоустойчивой распределенной платформы управления. В разделах 3 и 4 предлагается проактивный алгоритм распределения управления коммутаторами между контроллерами и алгоритм балансировки нагрузки между ними. В разделе 5 приводятся результаты экспериментальных исследований.

1. Обзор распределенных платформ

Приложение HyperFlow (C++, Python, OpenFlow 1.0) [13] для контроллера NOX – это первая попытка реализации распределенного контура управления ПКС с использованием распределенной файловой системы при сохранении логически централизованного управления и поддержки согласованного глобального состояния сети. Каждый коммутатор подключен к контроллеру, до которого задержка является наименьшей, или к контроллеру, указанному сетевым администратором (чтобы минимизировать затраты).

Onix (C++, Python, Java, OpenFlow 1.0) [14] – первая распределенная платформа управления ПКС, изначально предназначенная для виртуализации сетей в центрах обработки данных. Приложения Onix поддерживают согласованное глобальное состояние сети и используют базовые примитивы управления состоянием, предоставляемые платформой. Onix предоставляет общий API для сетевых приложений, позволяя им самостоятельно находить компромиссы между согласованностью, надежностью и масштабируемостью. Для достижения отказоустойчивости платформа Onix устойчива к четырем типам отказов: отказ коммутатора, отказ канала связи между коммутаторами, отказ контролера Onix и отказ канала связи между коммутатором и контроллером Onix (и между контроллерами Onix). Тем не менее, Onix остается закрытой разработкой, и дальнейшее развитие не описано в публикациях.

Контроллер Kandoo (C, C++, Python, OpenFlow 1.0) [15] является иерархически распределенной платформой управления ПКС. Kandoo имеет два уровня контроллеров (и сетевых приложений контроллеров соответственно): корневой (root) контроллер и набор локальных контроллеров. Ключевая особенность контроллера Kandoo – это снижение нагрузки на root-контроллер за счет обработки значительного количества запросов на установление новых потоков от коммутаторов на локальных контроллерах. В свою очередь, root-контроллер представляет собой распределенную платформу управления, такую как Onix. В случае отказа локального контроллера сегмент сети, оставшийся без контроллера, переключается на корневой контроллер.

Реализация распределенных контроллеров ElastiCon, HAC и ONOS использует возможность поддержки активных соединений коммутатора одновременно с несколькими контроллерами, которая была добавлена в протокол OpenFlow, начиная с версии 1.2.

Протокол OpenFlow [12] позволяет коммутатору поддерживать несколько активных защищенных каналов управления с несколькими контроллерами одновременно. Каждый контроллер по отношению к коммутатору может выступить в одной из трех ролей: MASTER, SLAVE или EQUAL. В ролях MASTER и EQUAL контроллер име-

ет полный доступ к коммутатору и может принимать все асинхронные сообщения (например, packet-in сообщения) от коммутатора. В роли SLAVE контроллер имеет доступ только для чтения состояния коммутатора и не может получать от него асинхронные сообщения. Контроллер может изменить свою собственную роль, используя особый тип OpenFlow сообщений для запроса роли (role-request). Эта новая возможность протокола OpenFlow позволяет коммутатору не устанавливать новый защищенный канал в случае сбоя контроллера. Протокол полностью возлагает ответственность за изменение роли на контроллер. Однако для стабильной работы сети для каждого коммутатора должен быть определен свой MASTER контроллер.

ElastiCon (Java, OpenFlow 1.3) [17] – экспериментальная реализация распределенного контроллера ПКС на основе контроллера с открытым исходным кодом Floodlight, который обеспечивает поддержку согласованного глобального состояния сети посредством использования распределенного in-memory хранилища Hazelcast. Ключевой особенностью ElastiCon является динамическая балансировка нагрузки в платформе управления: количество экземпляров контроллера динамически увеличивается или уменьшается в зависимости от условий и количества управляющего трафика. Нагрузка на платформу управления динамически перераспределяется между экземплярами контроллера на основе операции миграции коммутатора.

Контроллер высокой готовности (HAC) (C++, OpenFlow 1.3) [18] является экспериментальной реализацией распределенной платформы управления высокой готовности, основанной на контроллере nox13oflib для корпоративных сетей ПКС/OpenFlow. Для достижения высокой готовности платформа управления на основе контроллера HAC использует стратегию "активный/резервный" для распределения состояния основного контроллера и автоматического переключения на резервный контроллер в случае сбоя основного контроллера. В [18] представлена архитектура платформы управления HAC и реализация прототипа контроллера HAC. В случае отказа основного контроллера резервный контроллер автоматически берет на себя управление сетевой инфраструктурой и управление потоками данных. Эта процедура обеспечивает отказоустойчивость контроллера. Для обнаружения отказа HAC контроллер использует алгоритм Heartbeat с периодической отправкой сообщений основному контроллеру для проверки его работоспособности. Процедура восстановления после отказа основного контроллера может быть реализована двумя способами: либо путем настройки того же сетевого интерфейса, что и на основном контроллере, либо путем изменения роли резервного контроллера на MASTER на каждом сетевом коммутаторе.

Контроллер ONOS (Java, OpenFlow 1.0 и 1.3) [19, 20] – это распределенная сетевая операционная система с открытым исходным кодом для ПКС операторов связи. Высокая производительность платформы управления ONOS достигается за счет разделения сети на сегменты, их распределения между экземплярами контроллера и использования многопоточной обработки сообщений. Для достижения высокой готовности платформа управления ONOS использует избыточные защищенные каналы связи между коммутаторами и контроллерами, глобальное распределение состояния сети между контроллерами платформы, распределение ролей контроллера для каждого коммутатора и процедуру изменения роли контроллера в случае отказа основного контроллера. Однако ONOS не поддерживает процедуру балансировки нагрузки между экземплярами контроллера.

2. Предлагаемая архитектура отказоустойчивой распределенной платформы управления

Программно-конфигурируемая сеть с отказоустойчивым распределенным контуром управления должна включать в себя четыре основных уровня (см. Рисунок 1):

1. Контур данных.
2. Инфраструктура подключения OpenFlow.
3. Распределенная платформа управления.
4. Межконтроллерная коммуникационная инфраструктура.

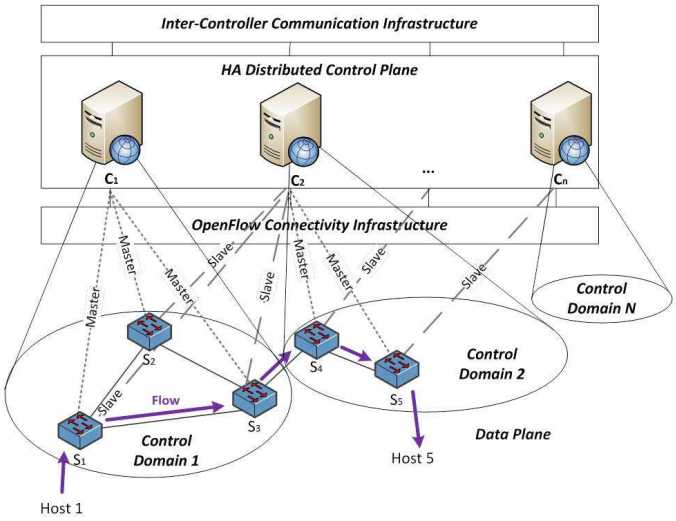


Рис. 1. Модель отказоустойчивой распределенной платформы управления ПКС

Fig. 1. The model of high-availability distributed control plane for software-defined networks

Контур данных (Data Plane) состоит из объектов управления: сетевых устройств (коммутаторов) и потоков данных. Контур данных предназначен для предоставления сетевых сервисов конечным пользователям. Мы предполагаем, что коммутаторы работают в соответствии со спецификацией коммутатора OpenFlow и поддерживают протокол OpenFlow (не ниже версии 1.2).

Инфраструктура подключения OpenFlow предназначена для обеспечения взаимодействия между контуром данных и контуром управления на основе протокола OpenFlow. Инфраструктура связи обеспечивает управление сетевыми устройствами и передачей трафика; управляющий трафик физически отделен от трафика данных. Также этот уровень включает в себя совокупность логических защищенных каналов связи "контроллер-коммутатор" [12].

Распределенная платформа управления (РПУ) включает физически распределенную и логически централизованную сетевую операционную систему с набором сетевых приложений для управления коммутаторами и потоками данных.

Распределенная сетевая операционная система (РСОС) состоит из набора контроллеров. РСОС поддерживает актуальное глобальное состояние сети, которое содержит состояние коммутаторов, каналов, портов, таблиц потоков и другие данные.

Инфраструктура связи между контроллерами включает в себя выделенную сеть для обеспечения связи между контроллерами распределенной платформы управления и протокол взаимодействия контроллеров. Он предназначен для мониторинга состояния контроллеров РПУ, координации контроллеров РПУ, обмена сообщениями между контроллерами и распределения глобального состояния сети между контроллерами РПУ.

На основе этой архитектуры РПУ можно выделить домены и домены управления для каждого контроллера. Домен включает в себя один экземпляр контроллера и набор коммутаторов, с которыми экземпляр контроллера поддерживает активное соединение. Домен управления включает в себя набор коммутаторов и один экземпляр контроллера, который является мастером для этих коммутаторов. Каждый коммутатор должен быть включен в домен управления, но только в один. Таким образом, можно определить Control Domain View (CDV) как набор всех доменов управления в РПУ. И для каждого коммутатора может быть определен групповой контроллер как набор экземпляров контроллера РПУ, которые поддерживают активное соединение (защищенный канал) с коммутатором и имеют согласованную информацию о текущем состоянии коммутатора. Таким образом, любой из экземпляров группового контроллера может быть определен как потенциально основной контроллер для этого коммутатора.

Потоки данных в сети можно классифицировать следующим образом: внутридоменные, транзитные, входящие и исходящие потоки данных. Внутридоменный поток данных не выходит за пределы сетевого сегмента и не требует участия других экземпляров контроллера для маршрутизации и передачи данных. Транзитные, входящие и исходящие потоки требуют координации между контроллерами и взаимодействия между доменами. Таким образом, загрузка РПУ и отдельных экземпляров контроллера зависит от распределения коммутаторов между доменами управления.

Для поддержки отказоустойчивости каждый экземпляр контроллера РПУ должен включать следующие дополнительные программные компоненты:

1. сервис обнаружения отказов контроллера,
2. сервис обнаружения перегрузки контроллера,
3. сервис восстановления управления,
4. сервис координации контроллеров,
5. сервис синхронизации контроллеров.

Сервис обнаружения отказов контроллера используется для мониторинга работоспособности контроллеров РПУ на основе метода Heartbeat.

Сервис обнаружения перегрузки контроллера включает в себя возможность отслеживать загрузку экземпляров контроллера (количество коммутаторов и потоков в каждом домене управления, частоту пакетов и задержку установления

новых потоков для каждого экземпляра контроллера, использование центрального процессора и памяти каждого сервера). Каждый индикатор загрузки экземпляра контроллера имеет фиксированное или плавающее предельное значение. Когда предел превышен, экземпляр контроллера регистрирует факт потенциальной перегрузки контроллера.

Сервис восстановления управления включает в себя набор алгоритмов и сценариев поведения экземпляра контроллера в случае реализации угроз (отказ одного экземпляра контроллера РПУ, перегрузка контроллера, потеря соединения между коммутатором и его основным контроллером).

Сервис координации контроллеров использует распределенный алгоритм консенсуса на основе Paxos [23] для координации событий смены ролей между контроллерами, событий обнаружения и восстановления и других событий в распределенной платформе управления.

Сервис синхронизации контроллеров использует распределенный алгоритм для синхронизации хранилищ данных с информацией о глобальном состоянии сети между контроллерами РПУ.

3. Распределение управления коммутаторами между контроллерами с учетом резервирования

В данной работе рассматривается ПКС/OpenFlow сеть с распределенной платформой управления, как показано на Рис. 1. $G = (S, E)$ – это граф сети. Контур данных включает в себя множество коммутаторов $S = \{s_1, s_2, ..., s_m\}$ из m коммутаторов. Распределенная платформа управления включает в себя множество $C = \{c_1, c_2, ..., c_n\}$ из n ПКС/OpenFlow контроллеров, запущенных на одинаковых серверах.

$E = \{e_{ij} = (s_i, s_j) | s_i, s_j \in S\}$ – множество каналов связи между коммутаторами.

$E_{sc} = \{e_{ij} = (s_i, c_j) | s_i \in S, c_j \in C\}$ – множество всех возможных логических защищенных каналов связи между коммутаторами и контроллерами. Для каждого защищенного канала связи может быть определена одна из четырех ролей $R = \{Master, Equal, Slave, None\}$ и отношение $role(c_i, s_j) \in R$. $None$ означает, что соединение между контроллером и коммутатором отсутствует.

Для обеспечения стабильного функционирования сети необходимо, чтобы для каждого коммутатора был определен единственный контроллер в роли $Master$. Для каждого коммутатора s_i существует, и только один, контроллер $c_j : role(c_j, s_i) = Master$.

Следовательно, для каждого контроллера может быть определен сегмент контура данных, который включает в себя множество коммутаторов, для которых он является $Master$ контроллером. Пусть $group(c_i) = \{s_j \in S \mid role(c_i, s_j) = Master\}$ – множество коммутаторов, для которых c_i – это $Master$ контроллер. И для каждого коммутатора s_i можно определить групповой контроллер $GC(s_i) = \{c_j \in C \mid role(c_j, s_i) \neq None\}$.

Для обеспечения отказоустойчивости для каждого коммутатора s_i должно быть $|GC(s_i)| \geq 2$.

$S' = \{S'_1 \dots S'_{|C|}\}$ — множество групп коммутаторов $S'_j \subseteq S$, где контроллер

$c_j \in C$ является $Master$ контроллером для каждого коммутатора $s_i \in S'_j$. Таким образом, $S'_j = \{s_i \in S \mid role(c_j, s_i) = Master\}$.

Свойства корректности организации управления в ПКС с распределенным контроллером можно сформулировать следующим образом:

Свойство безопасности: В любой момент времени для каждого коммутатора $s_i \in S$ существует не более одного $Master$ контроллера c_j: $role(c_j, s_i) = Master$.

Свойство живучести: В конечном итоге для каждого коммутатора $s_i \in S$ некоторый активный контроллер $c_j : role(c_j, s_i) \neq None$ должен стать $Master$ контроллером для него.

Пусть физическое размещение контроллеров РПУ ПКС статически определено в сети. Коммутаторы могут быть непосредственно соединены с контроллерами с использованием out-of-band каналов управления, остальные коммутаторы – посредством in-band каналов управления. Каждый контроллер может быть в состоянии активный или неактивный. Пусть каждый коммутатор поддерживает логическое соединение (защищенный канал) с каждым активным контроллером РПУ. В соответствии со спецификацией OpenFlow 1.3 для каждого коммутатора должен быть определен по крайней мере один $Master$ контроллер. Предполагается, что другие контроллеры для этого коммутатора находятся в роли $Slave$. И для каждого коммутатора в сети необходимо определить резервный контроллер – один из контроллеров в роли $Slave$, который перехватит управление сегментом сети в случае отказа $Master$ контроллера.

Таким образом, можно сформулировать задачу распределения управления коммутаторами между контроллерами для РПУ ПКС: для данной конфигурации активных контроллеров необходимо определить основной ($Master$) и резервный контроллер для каждого коммутатора.

3.1. Алгоритм выбора основного контроллера

Введем следующие обозначения:

- $d^0_{i,j}$ — задержка канала связи e_{ij} между коммутаторами s_i и s_j,
- d^j_i — задержка кратчайшего пути между коммутаторами s_i и s_j как сумма задержек каналов связи, входящих в кратчайший путь,
- N_{max} — максимальное количество коммутаторов, которыми может управлять один контроллер. Этот параметр одинаков для всех контроллеров,
- $N(c_j) = |S'_j|$ - количество коммутаторов, для которых контроллер c_j является $Master$ контроллером, для каждого контроллера c_j: $N(c_j) \leq N_{max}$.

Метрики задержки для основного контроллера могут быть определены следующим образом:

1. Средняя задержка между контроллерами и коммутаторами в сети:

$$L_{avg}(S') = \frac{1}{|S|} \sum_{s_i \in S} d^j_i, \quad c_j \in C, \quad s_i \in S'_j. \tag{1}$$

2. Задержка в худшем случае в сети между контроллерами и коммутаторами:

$$L_{wc}(S') = \max_{s_i \in S} d_i^j, \quad c_j \in C, \quad s_i \in S'_j. \tag{2}$$

Пусть S' – текущее размещение контроллеров в сети.

Можно сформулировать задачу определения основных контроллеров РПУ для коммутаторов следующим образом:

$$\begin{cases} \min L_{avg}(S') \quad or \quad \min L_{wc}(S') \\ N(c_j) \leq N_{max} \quad \forall j = \overline{1, |C|} \end{cases} \tag{3}$$

Решением этой задачи является распределение управления коммутаторами между контроллерами РПУ ПКС: $S' = \{S'_1 \dots S'_{|C|}\}$ — множество групп коммутаторов $S'_j \subseteq S$, где контроллер $c_j \in C$ является $Master$ контроллером для каждого коммутатора $s_i \in S'_j$. То есть $S'_j = \{s_i \in S \mid role(c_j, s_i) = Master\}$.

Для решения задачи выбираются потенциальные узлы сети, в которых могут быть размещены активные контроллеры РПУ ПКС, на основе использования алгоритма поиска двусвязных компонентов в графе. Для решения проблемы определения основных контроллеров для коммутаторов используются методы k-средних или k-medians в зависимости от выбранной метрики минимизации задержек.

3.2. Алгоритм выбора резервного контроллера

Для обеспечения отказоустойчивости управления в случае отказа контроллера или отказа соединения контроллер–коммутатор (защищенного канала связи) необходимо определить контроллер РПУ ПКС, который будет резервным контроллером для данного коммутатора.

Требования к резервным контроллерам в сети:

- Для каждого коммутатора s_i расположение резервного контроллера c_j отличается от основного контроллера c_k, то есть $j \neq k$.
- Для каждого контроллера c_j количество управляемых коммутаторов в сети не более N_{max}: $N(c_j) \leq N_{max}$.
- Платформа управления поддерживает механизм распределения глобального состояния сети между контроллерами РПУ.

Используя следующие обозначения, можно сформулировать задачу выбора резервного контроллера:

- $x_i^j \in \{0, 1\}$ — индикатор, $x_i^j = 1$, если контроллер $c_j \in C$ назначен резервным контроллером для коммутатора $s_i \in S$, и $x_i^j = 0$ — в противном случае;
- d_i^j — задержка кратчайшего пути между контроллером c_i и коммутатором s_j (из D^N – матрица кратчайших путей);
- $k_i \in C$ — основной контроллер для коммутатора s_i (k_i в $Master$ роли для коммутатора);

- S_i — набор коммутаторов, для которых контроллер c_i является основным контроллером;
- $M_i = |S_i| - N_{max}$ — максимальное количество коммутаторов, для которых контроллер c_i может быть назначен как резервный контроллер.

Выбор целевой функции зависит от выбранного критерия оптимизации:

1. Минимизация средней задержки между контроллером и коммутатором (L_{avg}):
$$F = \sum_{i=\overline{1,|S|}} \sum_{j=\overline{1,|C|}} d_i^j x_i^j \ . \tag{4}$$
2. Минимизация задержки в худшем случае между контроллером и коммутатором (L_{wc}):
$$F = \max_{i=\overline{1,|S|}} \sum_{j=\overline{1,|C|}} d_i^j x_i^j \ . \tag{5}$$

Формулировка проблемы выбора резервных контроллеров

Можно сформулировать следующую **логическую задачу линейного программирования**, чтобы определить резервные контроллеры для каждого коммутатора в сети:
$$\begin{cases} \min F & \\ x_i^j = 0 & \forall i,j:\ k_i = c_j \\ \sum\limits_{j=\overline{1,|C|}} x_i^j = 1 & \forall i = \overline{1,|S|} \\ \sum\limits_{s_i \in S_p} x_i^j \le M_j & \forall j, p = \overline{1,|C|} \end{cases} \tag{6}$$

Можно оценить минимальное количество контроллеров для поддержки резервных контроллеров для каждого коммутатора в сети. Для сети из $|S|$ коммутаторов необходимо следующее количество контроллеров, чтобы обеспечить отказоустойчивость контура управления к одиночному отказу контроллера:
$$|C| \ge \left\lceil \frac{|S|}{N_{max}} \right\rceil + 1 \ . \tag{7}$$

В этом случае для любых начальных распределений коммутаторов между экземплярами контроллера (таких, что в любой группе контроллеров не более N_{max} коммутаторов), можно назначить резервный контроллер для каждого коммутатора. А в случае сбоя любого экземпляра контроллера в сети количество коммутаторов, управляемых одним контроллером, не превышает N_{max}.

В общем случае для устойчивости к сбоям N контроллеров в сети требуется следующее количество контроллеров:
$$|C| \ge \left\lceil \frac{|S|}{N_{max}} \right\rceil + N \ . \tag{8}$$

Поскольку проблема выбора резервного контроллера была сведена к булевой задаче линейного программирования, был реализован алгоритм, основанный на методе Балаша [22]. Алгоритм позволяет найти точное решение со значительным сокращением пространства решения. Алгоритм обеспечивает сходимость с использованием конечного числа итераций.

4. Балансировка нагрузки между контроллерами

Загрузка экземпляра контроллера создается путем обработки входящих сообщений. Для установления нового потока данных на контроллере генерируется набор правил для коммутаторов. Таким образом, загрузка контроллера – это количество входящих сообщений Packet-In в секунду, а производительность или пропускная способность контроллера – это максимальное количество Packet-In сообщений, которые он может обработать в секунду.

Во время работы сети характер, динамика, объемы трафика и количество потоков данных динамически изменяются, и поэтому также изменяется нагрузка на контур управления, которая генерируется коммутаторами. Для достижения отказоустойчивости контура управления необходимы специальные механизмы для предотвращения перегрузки любого контроллера РПУ. А в случае низкой загрузки контура управления количество активных контроллеров должно быть минимальным.

Эти проблемы в контуре управления могут быть решены путем динамического перераспределения управления коммутаторами между контроллерами. Для реализации перераспределения управления коммутатором между контроллерами может использоваться алгоритм передачи управления коммутатором из [17].

Чтобы решить проблему распределения нагрузки между экземплярами контроллера РПУ, введем следующие допущения и ограничения:

1. Каждый коммутатор подключен к каждому экземпляру контроллера.
2. Все экземпляры контроллера имеют одинаковую максимальную производительность.
3. Для каждого коммутатора в каждый момент времени должен быть определен только один мастер.
4. Экземпляры контроллера используют проактивный режим установки правил, то есть контроллер устанавливает правила в коммутаторы, для которых он определен как мастер.

Таким образом, можно декомпозировать задачу на следующие подзадачи:

1. Подзадача группировки коммутаторов: необходимо разделить все коммутаторы на группы в соответствии с их нагрузкой, чтобы любая групповая нагрузка не превышала производительность контроллера.
2. Подзадача распределения групп между экземплярами контроллеров: необходимо распределить группы коммутаторов между контроллерами с минимальным количеством передач управления коммутаторами.

4.1. Алгоритм группировки коммутаторов

Пусть в контуре управления p_i является максимально допустимой пропускной способностью/производительностью контроллера c_i (количество входящих сообщений в секунду), а $load(c_i)$ – средняя текущая нагрузка операций установки потоков для контроллера $c_i \in C$.

Поскольку используется набор идентичных экземпляров контроллера, то $p_i = p$ для всех $c_i \in C$. Значение p задано.

Пусть задана некоторая статистическая картина потоков в контуре данных: $inf(s_j)$ – среднее число новых потоков в секунду в коммутатор s_j и $f_{ij} = f(s_i, s_j)$ – это среднее число потоков данных в секунду от коммутатора s_i к коммутатору s_j.

Проблема группировки коммутаторов: распределить коммутаторы по группам так, чтобы для каждого контроллера c_i: $load(c_i) \leq p$. И требуется достичь минимального количества активных контроллеров в контуре управления. Математическая формулировка **подзадачи группировки коммутаторов** следующая:

$$\min \sum_{i=1}^{n} y_i$$

$$\text{при условии} \quad load(c_i) \leq p,\ c_i \in C,$$

$$\sum_{i=1}^{n} x_{ij} = 1, j \in \{1, \ldots, n\}$$

где

$$y_i = \begin{cases} 1, \text{если контроллер } c_i \text{ активен,} \\ 0, \text{в противном случае.} \end{cases}$$

$$x_{ij} = \begin{cases} 1, \text{если } c_i - Master \text{ для коммутатора } s_j, \\ 0, \text{в противном случае.} \end{cases}$$

Эта задача является NP-трудной. Для решения этой задачи предлагается жадный алгоритм для распределения коммутаторов по минимальному количеству групп и соответственно минимальному числу контроллеров для них.

Алгоритм группировки коммутаторов включает в себя следующие шаги:

1. Вычислить общее количество потоков данных, проходящих через каждый канал связи $e_{ij} \in E$ в контуре данных:

$$l_{ij} = l(s_i, s_j) = f_{ij} + f_{ji}.$$

2. Вычислить общее количество входящих потоков данных для каждого коммутатора $s_i \in S$:

$$f(s_i) = \sum_{s_k \in S} f(s_k, s_i) + inf(s_i).$$

3. Отсортировать каналы связи e_{ij} из E по убыванию общего количества l_{ij} потоков данных.

4. Проинициализировать новый граф $G' = (S, E')$, где $E' = \emptyset$. G' является несвязным графом с $|S|$ компонентами связности.

5. Для группы коммутаторов, пока набор каналов связи E не пустой, необходимо выполнять следующие действия:

   (a) Выбрать линк $e_{ij} \in E$ с максимальным l_{ij},

(b) Удалить линк e_{ij} из E: $E = E \setminus \{e_{ij}\}$,

(c) Добавить линк e_{ij} в E: $E' = E' \cup \{e_{ij}\}$,

(d) Рассчитать нагрузку для каждой группы коммутаторов, которая соответствует компоненту связности графа G', используя следующую формулу:

$$p_j = f_j + \sum_{k=1}^{n} c_{kj}(1 - t_{jk}),$$

$$\text{где } t_{jk} = \begin{cases} 1, \text{если } s_j \text{ и } s_k \text{ в одной группе,} \\ 0, \text{в противном случае.} \end{cases}$$

(e) Добавить коммутатор в группу.

Таким образом, получаем граф G' с разбиением сети на сегменты, распределение коммутаторов в группы, для которых требуется минимальное количество контроллеров РПУ ПКС.

4.2. Алгоритм распределения групп коммутаторов между контроллерами

Чтобы сократить время балансировки нагрузки в контуре управления, необходимо минимизировать количество операций переключения управления коммутаторами между текущим распределением Мастер-контроллеров и требуемым распределением Мастер-контроллеров после перебалансировки. Предполагается, что количество активных контроллеров может быть изменено после процедуры балансировки нагрузки.

Пусть контур управления включает в себя m активных контроллеров, которые управляют контуром данных из n коммутаторов. Пусть матрица $\tilde{X} = [\tilde{x}_{ij}]_{m \times n}$ – это текущее представление распределения Мастер-контроллеров в сети, где $\tilde{x}_{ij} = 1$, если контроллер c_i управляет коммутатором s_j, $\tilde{x}_{ij} = 0$ – в противном случае.

Пусть X – матрица групп коммутаторов, полученная из алгоритма группировки коммутаторов. $x = \{x_1, \ldots, x_n\}$ – группа коммутаторов, где $x_j = 1$, если коммутатор s_j включен в группу и $x_j = 0$ в противном случае. И необходимо предложить набор операций передачи управления коммутаторами для преобразования матрицы $\tilde{X}$ в матрицу X.

Если $|X| \neq m$, то

- Если $|X| < m$, то добавляется $(m - |X|)$ нулевых векторов(строк) в X,
- Если $|X| > m$, то добавляется $(|X| - m)$ нулевых векторов(строк) в $\tilde{X}$.

Чтобы решить эту задачу, необходимо пронумеровать векторы X, чтобы минимизировать следующую функцию:

$$f = \sum_{i=1}^{n} d(x_i, \tilde{x}_i),$$

где $d(x_i, \tilde{x}_i)$ – расстояние Хэмминга между x_i и $\tilde{x}_i$.

Для решения этой задачи предлагается **алгоритм распределения групп между контроллерами** в контуре управления. Он включает в себя следующие шаги:

1. Для каждого коммутатора s_i просмотрим все позиции от 1 до n, чтобы найти в матрице $\tilde{X}$ и в матрице X позицию i, в которой стоит единица. Вычислим расстояние Хэмминга между этими строками.

2. Если добавляются дополнительные строки в $\tilde{X}$, вычислить расстояние Хэмминга между каждым вектором X и нулевым вектором.

3. Сортировать эти пары в порядке возрастания расстояния Хэмминга.

После выполнения этих шагов для каждого x из X найдем $\tilde{x}_i$ из $\tilde{X}$. Это будет означать, что данному элементу x следует присвоить номер i. И определяются контроллеры для новых групп коммутаторов после перебалансировки.

5. Экспериментальные исследования

5.1. Оценка алгоритма распределения коммутаторов между основными и резервными контроллерами

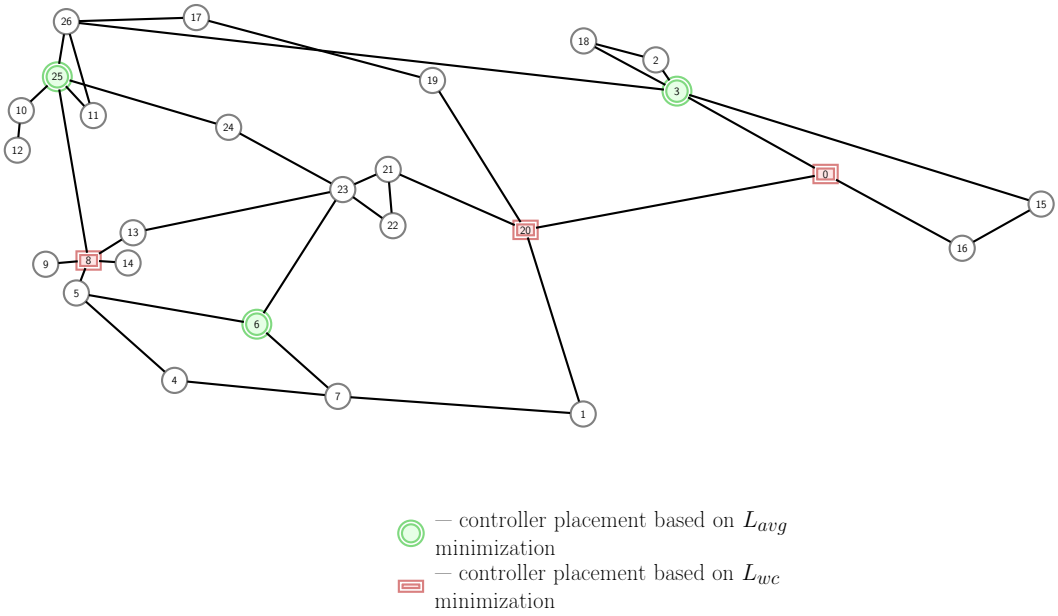


Рис. 2. Размещение контроллеров отказоустойчивой РПУ ПКС в узлах сети Integra Telecom по критериям средней задержки L_{avg} и наихудшей задержки L_{wc}

Fig. 2. Controllers' placement of high-availability distributed control platform for Integra Telecom network by criteria L_{avg} and L_{wc}

Алгоритмы выбора основного и резервного контроллеров реализованы на C ++ с использованием QT5. Для экспериментов используются топологии сети из библиотеки Интернет-топологий TopologyZoo [24] реальных сетей провайдеров.

На рисунке 2 можно увидеть топологию реальной сети Integra Telecom (август 2010 г.), которая состоит из 27 узлов (коммутаторов). Все узлы пронумерованы от 0 до 26. Предполагаем, что контроллеры РПУ ПКС могут быть расположены в любом узле сети. Предполагается, что сеть Integra использует SDN/OpenFlow для управления, а размещение основных контроллеров определяется для случаев минимизации метрик средней задержки между коммутатором и контроллером и задержки в худшем случае.

Результаты алгоритма выбора резервных контроллеров для сети Integra SDN показаны в Таблице 1 и Таблице 2 для средней задержки (L_{avg}) и наихудшей задержки (L_{wc}). В первом столбце можно увидеть пару контроллеров (Master, Slave), а во втором столбце – набор коммутаторов под управлением этих контроллеров.

Таблица 1. Распределение ролей контроллеров отказоустойчивой РПУ ПКС для управления коммутаторами в сети Integra Telecom для случая L_{avg}

Table 1. Assignment of controllers' roles (Master/Slave) for switches in Integra Telecom network in case of L_{avg}

Контроллеры (Master, Slave)	**Коммутаторы**
(3, 6)	0, 15, 16
(3, 25)	2, 3, 18
(6, 3)	1
(6, 25)	4, 6, 7, 20 – 23
(25, 3)	17
(25, 6)	5, 8 – 14, 19, 24, 25, 26

Таблица 2. Распределение ролей контроллеров отказоустойчивой РПУ ПКС для управления коммутаторами в сети Integra Telecom для случая L_{wc}

Table 2. Assignment of controllers' roles (Master/Slave) for switches in Integra Telecom network in case of L_{wc}

Контроллеры (Master, Slave)	**Коммутаторы**
(0, 8)	
(0, 20)	0, 2, 3, 15, 16, 18
(8, 0)	6
(8, 20)	4, 5, 8 – 14, 17, 25, 26
(20, 0)	1, 19 – 24
(20, 8)	

Увеличение задержки в среднем (L_{avg}) и задержки в худшем случае (L_{wc}) при отказе контроллера приведено для различных сетей в Таблице 3 и Таблице 4 (после

распределения происходит переключение между контроллерами с использованием алгоритма выбора резервных контроллеров).

Таблица 3. Увеличение максимальной задержки L_{avg}
в случае отказа контроллера для различных реальных топологий сетей

Table 3. Increasing the average latency L_{avg}
in case of controller failure for different real world network topologies

Название сети	**|S|**	**|C|**	**min, %**	**max, %**	**mid, %**
HiberniaCanada	10	2	60,08	61,82	60,95
Abilene	11	2	117,02	218,67	167,85
Compuserve	11	2	43,97	188,98	116,48
Navigata	13	2	103,28	193,78	148,53
Nsfnet	13	2	65,53	189,65	127,59
Claranet	15	2	77,48	186,82	132,15
Garr199901	16	2	88,72	107,90	98,31
Peer1	16	2	45,95	120,17	83,06
Ernet	16	2	764,00	1427,25	1095,63
Goodnet	17	2	102,56	212,38	157,47
Arpanet19719	18	2	4,48	40,82	22,65
Ibm	18	2	19,38	20,14	19,76
Internetmci	19	2	31,52	51,45	41,49
GtsRomania	19	2	69,68	577,23	323,45
Quest	20	2	147,99	526,47	337,23
BtEurope	22	3	11,50	34,46	20,54
York	23	3	45,11	98,10	78,85
Funet	24	3	147,71	233,42	187,62
Psinet	24	3	1,89	21,11	11,45
Agis	25	3	12,79	31,26	24,58
Integra	27	3	45,17	91,87	69,95
Biznet	28	3	102,10	745,80	519,16
Darkstrand	28	3	22,05	39,01	28,96
Digex	31	3	27,97	84,29	59,05
Bics	33	3	22,06	73,46	39,63
BtNorthAmerica	33	3	16,38	57,86	36,16
Grnet	34	3	0,18	14,59	5,68
NetworkUsa	35	3	6,65	24,51	13,67
Geant2012	37	3	3,73	7,77	6,13
Renater2010	37	3	25,95	191,51	127,11
Cesnet200706	38	3	36,10	126,26	66,29
Chinanet	38	3	16,67	21,41	19,01
Garr200912	42	4	4,73	14,76	10,53
Garr201101	44	4	8,00	41,67	20,16

Таблица 4. Увеличение максимальной задержки L_{wc}
в случае отказа контроллера для различных реальных топологий сетей

Table 4. Increasing the average latency L_{wc}
in case of controller failure for different real world network topologies

Название сети	**|S|**	**|C|**	**min, %**	**max, %**	**mid, %**
HiberniaCanada	10	2	0	30,78	15,39
Abilene	11	2	156,70	204,02	180,36
Compuserve	11	2	86,38	105,66	96,02
Navigata	13	2	132,4	153,8	143,1
Nsfnet	13	2	70,9	105,76	88,33
Claranet	15	2	83,49	113,98	98,74
Garr199901	16	2	71,99	87,33	79,66
Peer1	16	2	10,72	31,37	21,04
Ernet	16	2	335,68	435,68	385,68
Goodnet	17	2	116,16	160,11	138,13
Arpanet19719	18	2	0	0,04	0,02
Ibm	18	2	0	33,474	16,74
Internetmci	19	2	0	33,58	16,792
GtsRomania	19	2	93,57	97,796	95,68
Quest	20	2	304,38	318,8	311,59
BtEurope	22	3	0	15,22	5,08
York	23	3	97,83	151,43	115,7
Funet	24	3	364,52	393,05	376,67
Psinet	24	3	0	10,42	4,67
Agis	25	3	0	9,72	3,24
Integra	27	3	109,99	121,23	114,77
Biznet	28	3	537,52	565,87	550,04
Darkstrand	28	3	0	21,55	11,96
Digex	31	3	15,38	115,38	76,07
BtNorthAmerica	33	3	51,37	151,37	85,31
Bics	33	3	0	57,21	20,97
Grnet	34	3	0	5,75	1,92
NetworkUsa	35	3	0	0,24	0,08
Geant2012	37	3	0	2,28	0,76
Cesnet200706	38	3	4,93	99,03	67,33
Uunet	42	4	0	19,46	8
Garr201101	44	4	0	6,1	1,53
Surfnet	50	4	0	55,9	27,73

В таблицах приведен набор различных сетей, количество коммутаторов и контроллеров в каждой сети, а также минимальное, среднее и максимальное время восстановления в процентах после отказа одного контроллера в сети.

Как видно из этих таблиц, для некоторых сетей средняя задержка между коммутатором и его основным контроллером после сбоя одного контроллера и перераспределения управления в сети увеличивается незначительно. Но для некоторых сетей средняя задержка увеличивается во много раз. Таким образом, для таких сетей общая производительность контура управления может быть достигнута с учетом резерва в случае отказа одного контроллера, но для сохранения приемлемого отклика (времени ответа контроллера) необходимо расширить контур управления, добавив дополнительные контроллеры.

Предложенный алгоритм выбора резервных контроллеров может использоваться не только в режиме реального времени для контура управления для генерации сценариев восстановления в случае отказа одного контроллера, но также и на этапе проектирования контура распределенного управления для конкретной ПКС/OpenFlow сети для выбора разумного количества контроллеров.

5.2. Оценка алгоритма балансировки нагрузки

Предварительные результаты алгоритма группировки коммутаторов показаны на рисунке 3.

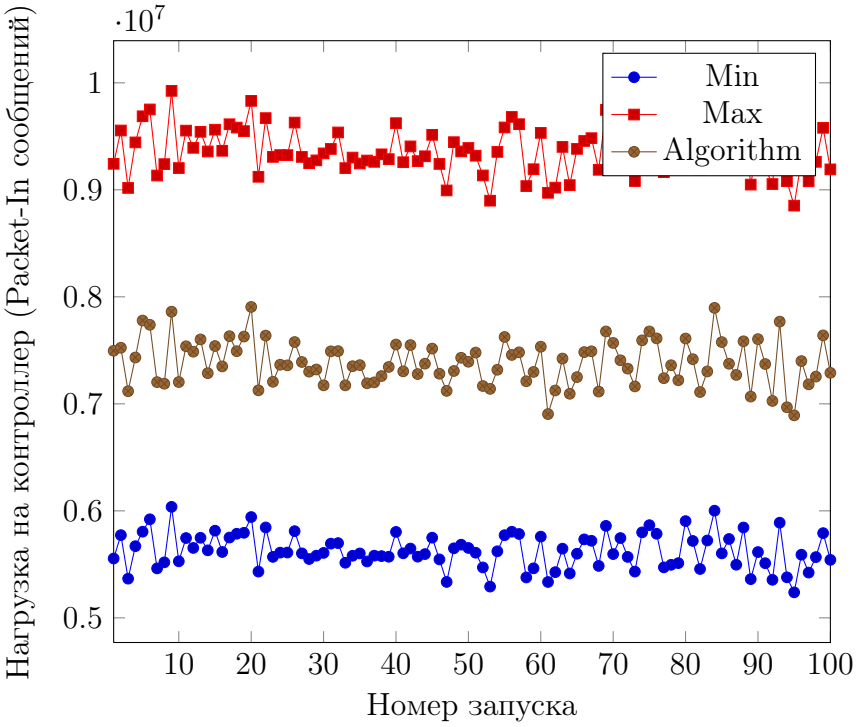


Рис. 3. Общая нагрузка на платформу управления в зависимости от номера запуска

Fig. 3. Total control plane loading depending on the launch number

Исходными данными для алгоритма является топология сети [25], которая включает 141 коммутатор и 748 каналов связи между ними. Для генерации нагрузки для каждого коммутатора в сети использовали равномерное распределение. Алгоритм

был выполнен 100 раз. Для каждого запуска рассчитывается минимальная нагрузка в сети в случае одного контроллера в контуре управления, а максимальная нагрузка в случае одного контроллера в каждом узле сети. И можно наблюдать результат управления загрузкой трафика с помощью алгоритма группировки коммутаторов для балансировки нагрузки между минимальными и максимальными значениями. На рисунке 4 приведено количество сегментов в сети для этого случая.

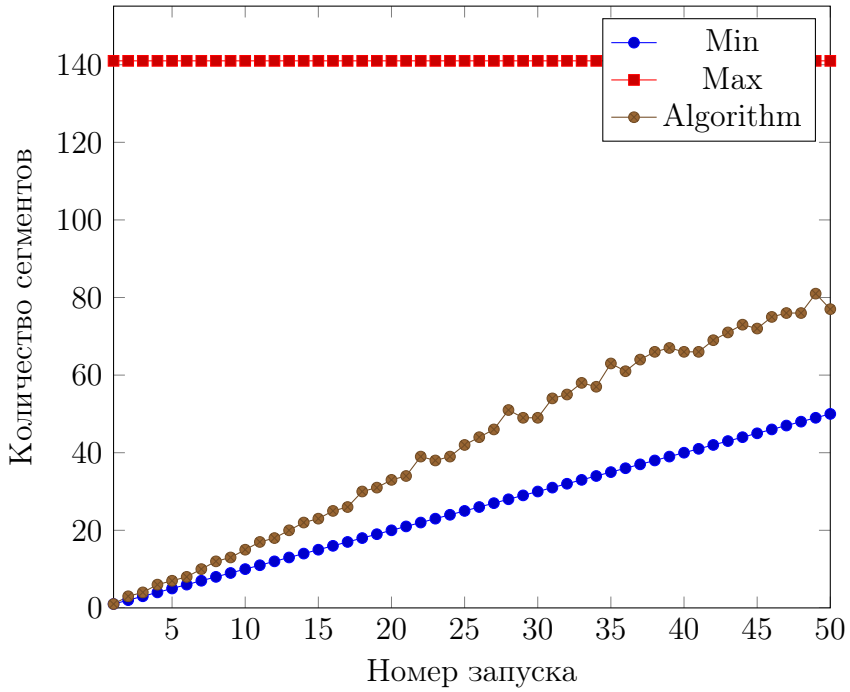


Рис. 4. Количество сегментов в сети в зависимости от номера запуска

Fig. 4. Quantity of segments in the network depending on the launch number

6. Заключение

В этой статье была предложена архитектура отказоустойчивой распределенной платформы управления для глобальных ПКС/OpenFlow сетей. Для достижения отказоустойчивости используется резервирование контроллеров платформы управления, резервирование активных соединений коммутаторов с контроллерами. В работе предложены механизмы для обнаружения и восстановления управления РПУ ПКС. В случае одиночного отказа контроллера РПУ или перегрузки контроллера управление сетью восстанавливается за счет перераспределения управления коммутаторами между оставшимися функционирующими контроллерами РПУ. Предложен проактивный алгоритм выбора резервных контроллеров для каждого коммутатора, который позволяет минимизировать время восстановления в случае сбоя одного контроллера. Предложен алгоритм балансировки нагрузки контроллеров РПУ, чтобы предотвратить перегрузки контроллеров платформы. Эти алгоритмы являются частью сервиса восстановления для каждого РПУ ПКС.

Список литературы / References

[1] McKeown N., et al., "Openflow: Enabling innovation in campus networks", *ACM Computer Communication Review*, **38**:2, (2008), 69–74.

[2] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks", *ONF White Paper*, 2012.

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

[4] Open Networking Foundation, "OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01)", *ONF*, 2009.

[5] Gude N., et al., "NOX: towards an operating system for networks", *SIGCOMM Computer Communication Review*, **38**:3 (2008), 105–110.

[6] *NOX OpenFlow Controller*, http://http://www.noxrepo.org/.

[7] Erickson D., "The Beacon OpenFlow controller", *Proceedings HotSDN*, August, 2013.

[8] "Beacon OpenFlow Controller", https://openflow.stanford.edu/display/Beacon.

[9] "Floodlight OpenFlow Controller", http://floodlight.openflowhub.org.

[10] "OpenMul OpenFlow/SDN Controller", http://www.openmul.org/.

[11] "RUNOS OpenFlow Controller", https://github.com/ARCCN/runos.

[12] Open Networking Foundation, "OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04)", *ONF*, 2012.

[13] Tootoocian A., Ganjali Y., "HyperFlow: A distribute control plane for OpenFlow", *Proceedings of the 2010 INM conference/WREN workshop*, 2010, 3.

[14] Koponen T., et al., "Onix: A distributed control platform for large-scale production networks", *OSDI'10*, USENIX, 2010.

[15] Yeganeh S.H., Kandoo Y.G., "A Framework for Efficient and Scalable Offloading of Control Applications", *Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12*, ACM, New York, NY, USA, 2012, 19–24.

[16] Phemius K., Bouet M., Leguay J., "Disco: Distributed multi-domain sdn controllers", *Network Operations and Management Symposium (NOMS)*, IEEE, 2014, 1–4.

[17] Dixit A., et al., "Towards an Elastic Distributed SDN Controller", *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13*, ACM, New York, NY, USA, 2013, 7–12.

[18] Pashkov V., Shalimov A., Smeliansky R., "Controller Failover for Enterprise SDN", *Proceedings of the Modern Networking Technologies (MoNeTec'2014)*, IEEE, 2014, 27–29.

[19] Lantz B., et al., "ONOS: Towards an Open, Distributed SDN OS", *ACM SIGCOMM HotSDN Workshop*, August, 2014.

[20] "ONOS: Open Network Operating System", https://github.com/opennetworkinglab/onos.

[21] Heller B., Sherwood R., McKeown N., "The Controller Placement Problem", *Proceedings of the first workshop on Hot topics in software-defined networks*, ACM, 2012.

[22] Chinneck J.W., "Practical optimization: a gentle introduction", 2012, https://sce.carleton.ca/faculty/chinneck/po.html.

[23] Bolosky W., et al., "Paxos Replicated State Machines as the Basis of a High-Performance Data Store", *Proceedings of the NSDI*, 2011.

[24] Knight S., et al., "The internet topology zoo", http://www.topology-zoo.org.

[25] "Rocketfuel: An ISP Topology Mapping Engine", https://research.cs.washington.edu/networking/rocketfuel.

Pashkov V. N., "Fault-Tolerance Distributed Control Plane for Software-Defined Networks", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 101–121.

DOI: 10.18255/1818-1015-2019-1-101-121

Abstract. The architecture of the high availability distributed control plane for SDN/OpenFlow networks are considered. High availability is achieved by redundancy of controller instances, active switch-controller communications, computing resources and tools for a controller instance failure and overloading detection and recovery. The proactive backup controller allocation algorithm which allows to minimize the time to repair in the case of a single controller instance failure is discussed. The algorithm for controller load-balancing allows dynamically reconfigure the control plane with a minimum number of switch control transfer operations to avoid controller instance overloading. The initial experimental results of the proposed algorithms for the HA distributed SDN control plane are described.

Keywords: software-defined networking, SDN, distributed control plane, DCP, fault tolerance, load-balancing, OpenFlow, data plane, network architecture

On the authors:
Vasily N. Pashkov, Programmer, orcid.org/0000-0001-5783-4557
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: pashkov@lvk.cs.msu.su

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

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

УДК 004.7

Алгоритм минимизации количества правил маршрутизации в ПКС

Петров И. С.

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

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

Ключевые слова: ПКС, сетевая статистика, счетчики правил маршрутизации

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

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

Введение

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

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

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

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

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

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

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

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

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

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

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

1. Анализ сетевой статистики

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

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

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

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

2. Представление заголовков пакетов

2.1. Header Space

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

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

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

2.2. Представление заголовков

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

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

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

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

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

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

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

Это представление соответствует набору заголовков:

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

Также оно может быть представлено как:

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

2.3. Операции с заголовками

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

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

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

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

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

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

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

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

2.4. Минимизация

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

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

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

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

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

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

3. Алгоритм минимизации

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

3.1. Метод Блейка

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

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

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

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

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

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

для всех $i \in [1, k]$.

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

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

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

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

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

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

3.2. Минимизация масок

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

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

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

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

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

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

4. Реализация

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

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

Algorithm 1 Минимизация *wildcard*-выражений

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

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

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

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

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

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

5. Экспериментальное исследование

5.1. Эксперименты

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

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

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

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

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

5.2. Результаты

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

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

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

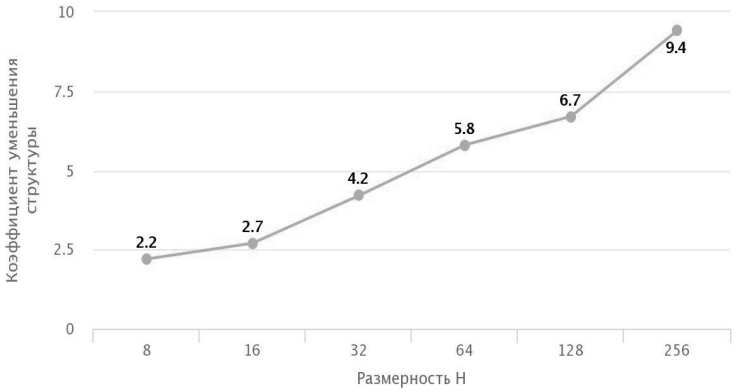


Рис. 1. Зависимость коэффициента уменьшения структуры от размера заголовка

Fig. 1. Reduction rate depending on the header size

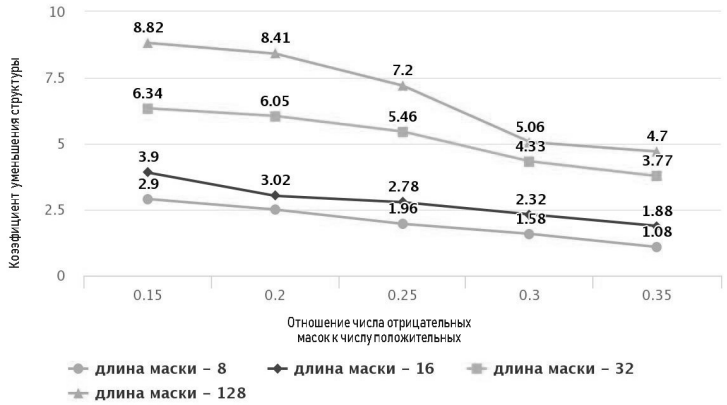


Рис. 2. Зависимость коэффициента уменьшения структуры от относительного числа отрицательных элементов

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

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

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

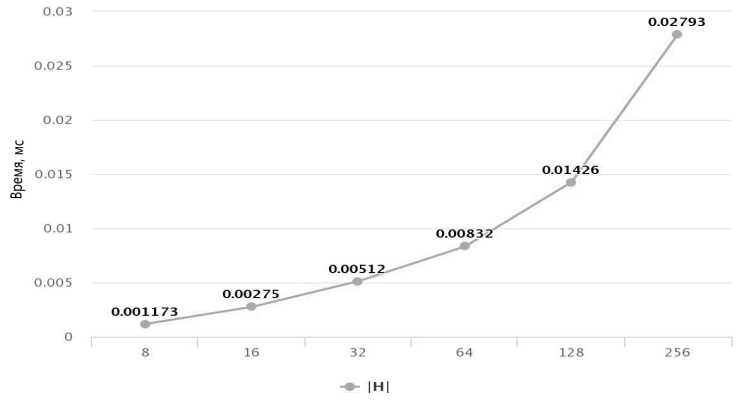


Рис. 3. Зависимость времени (в мс) от размера заголовка

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

Заключение

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

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

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

Список литературы / References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Keywords: SDN; network statistics; forwarding rule counters

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

Моделирование и анализ информационных систем. Т. 26, №1 (2019), с. 134–145
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 134–145

DOI: 10.18255/1818-1015-2019-1-134-145

УДК 004.056.5(076)

«Общие критерии» и безопасность программно-конфигурируемых сетей

Петухов А. Н., Пилюгин П. Л.

Поступила в редакцию 10 января 2019
После доработки 17 февраля 2019
Принята к публикации 18 февраля 2019

Аннотация. «Общие критерии» (ISO 15408) – общепризнанный и широко применимый подход к управлению и оценке решений в области информационной безопасности. «Общие критерии» опираются на разработку общей концептуальной основы для ключевых решений безопасности, включая профили защиты и целевые объекты безопасности. Концептуальная основа разработки подразумевает определение следующих элементов: цели и предположения безопасности (для среды и объекта), угрозы и политики безопасности, а также функциональные требования и требования к обеспечению безопасности. Специфика решений по обеспечению безопасности SDN во многом обусловлена фундаментальными архитектурными принципами самой технологии SDN – в первую очередь разделением потоков управления и данных, а также условиями применения протокола OpenFlow. Тем не менее, проактивные (угрозы и политики), пассивные (цели и предположения) и реактивные (требования) аспекты управления безопасностью остаются весьма актуальными для такого типа решений безопасности. В статье рассматриваются особенности применения единых критериев оценки безопасности SDN и практического опыта Московского технического университета связи и информатики при разработке профиля защиты. Новый класс сетевых атак на коммутаторы и контроллеры SDN может использовать как данные, так и компоненты управления. В дополнение к традиционным уязвимостям централизация функций управления открывает путь для новых угроз безопасности путем изоляции деятельности контроллера и обмена управляющими сообщениями. Поэтому выявление и анализ угроз, политик и требований, специфичных для безопасности модуля управления SDN, становится новым приоритетом.

Ключевые слова: безопасность программно-конфигурируемых сетей, общие критерии, профиль защиты

Для цитирования: Петухов А. Н., Пилюгин П. Л., "«Общие критерии» и безопасность программно-конфигурируемых сетей", *Моделирование и анализ информационных систем*, **26**:1 (2019), 134–145.

Об авторах:
Петухов Андрей Николаевич, канд. техн. наук, orcid.org/0000-0002-1427-2440
Национальный исследовательский университет «МИЭТ»
пл. Шокина, 1, г. Зеленоград, г. Москва, 124498 Россия, e-mail: anpetukhov@yandex.ru

Пилюгин Павел Львович, канд.техн. наук, orcid.org/0000-0003-0011-7180
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: ppl@mail.ru

Благодарности:
Работа выполнена при поддержке ректората Московского технического университета связи и информатики (МТУСИ): С. Д. Ерохина, Ю. Л. Леохина и А. Ю. Муханова — и при финансировании МТУСИ по направлению «Безопасность критических информационных инфраструктур».

Введение

В статье рассматриваются результаты анализа факторов нарушения информационной безопасности программно-конфигурируемых сетей (ПКС), целей, задач и требований безопасности для среды и условий функционирования ПКС как объекта оценки в рамках концепции «Общих критериев» [1].

1. Профиль защиты

Осмысленное обеспечение любой безопасности предполагает наличие адекватного представления об опасности. В каждом конкретном случае есть специфика структуры и формы такого представления, но общими элементами всегда являются:

– типология проявления опасности, номенклатура идентифицированных и квалифицированных видов такого проявления, внешних (агрессивность среды) и внутренних (несовершенство объекта) событий и ситуаций, являющихся причиной возникновения ущерба (проактивный аспект);

– совокупность и структура активов (объектов и процессов), подвергающихся опасности, речь идет о пространстве информационных ресурсов, их состояний и взаимодействий, а также о характере их подверженности опасности (пассивный аспект);

– элементы и свойства объекта (средства защиты, особенности архитектуры, ограничивающие решения и т.п.), препятствующие реализации опасности (реактивный аспект).

На базе таких сведений строится модель угроз, проводится анализ и управление рисками, исследуются уязвимости, учитываются инциденты, эти данные используются при формировании политик безопасности.

Одним из эффективных форматов использования этих сведений для решения проблем безопасности является профиль защиты. Понятие профиля защиты восходит к концепции управления информационной безопасностью с привлечением моделирования требований и условий (не только для объекта управления, но и для среды его функционирования), а также оценки доверия к средствам реализации этих условий и способам выполнения этих требований («исчисление доверия»). Базовым изложением этой концепции является международный стандарт ISO/IEC 15408-1:2005. (т.н. «Общие критерии») [1].

В профиле защиты стандартом предусмотрено несколько категорий, которые в совокупности исчерпывающе отображают все три аспекта представления об опасности и дают возможность обоснованно выстроить систему защитных мер. В качестве средства отображения проактивного аспекта профилем защиты используется категория *«угрозы»*, которая в стандарте полагается высокоуровневой (т.е. первичной) сущностью, гармонизированной (т.е. взаимодействующей, согласованной, обусловливающей и обусловливаемой) с такими высокоуровневыми сущностями, как «активы», «злоумышленники», «уязвимости» и «риски». Для выражения пассивного

аспекта привлекается производная категория *«предположения»*, с помощью которой ограничивается пространство информационных активов и специфицируется интерфейс с окружающей средой. Кроме того, *«предположения»*, отнесенные к внутренним свойствам и характеристикам, отчасти поддерживают реактивный аспект. Основным же инструментом для представления реактивного аспекта в профиле защиты является категория *«политики»*, которая соответствует возможностям (реализованным и гипотетическим) противодействовать вредоносным факторам агрессивной среды. Для обеспечения взаимной корректности *«угроз», «предположений» и «политик»* в профиле защиты предусматривается специальный механизм, выраженный в виде *«целей безопасности»* для среды и самого объекта.

2. Локализация сервисов безопасности

В сетях традиционной архитектуры основным (но не единственным) применяемым подходом к обеспечению безопасности является вынесение функций безопасности в отдельный компонент-посредник, осуществляющий инспекцию и модификацию трафика, принятие решений, аудит и т.п. (межсетевой экран, IDS/IPS, VPN, VLAN и др.), и вынесенный физически или концептуально в отдельный элемент сети. ПКС не исключает такого подхода, например, в [2] предлагается использовать контроллер для управления вынесенными сервисами безопасности. В этом случае приложения указывают контроллеру, какие функции безопасности должны быть активизированы на том или ином узле-посреднике, расположенном между контроллером и сетью. Таким образом, эти приложения расширяют возможности контроллера путем вынесения функций безопасности за его пределы.

Другой подход [3] к использованию локализованных сервисов безопасности ПКС предполагает взаимодействие устройства-посредника с контроллером посредством специального программного интерфейса. В заголовки пакетов внедряются специальные теги, содержащие информацию о потоке, что позволяет обеспечить контролируемую маршрутизацию этого потока на устройство-посредник и его дальнейшую обработку на нем. Процедура работает на базе предварительно программируемых политик, что делает невозможным динамическую модификацию функций безопасности. Развитием такой схемы [4] является построение механизма трансляции высокоуровневых политик безопасности в правила конфигурирования конечных коммутаторов на узлах-посредниках.

Функции безопасности, реализуемые на узлах-посредниках, предназначены для противодействия угрозам, направленным на линии связи, коммуникационное оборудование и узлы сети:

– Межсетевые экраны для защиты периметра и контроля внутреннего домена.

– Системы обнаружения и предотвращения вторжений, которые контролируют сетевые активности для обнаружения вредоносных действий или нарушений политики безопасности и пытаются предотвратить атаки.

– Виртуальные частные сети (SSL VPN), которые обеспечивают безопасное разделение клиентов и доменов.

– Решения для управления сетью, которые позволяют централизованно управлять многими из функций безопасности через консоль администратора.

– IEEE 802.1X – проверка подлинности и контроль доступа к сети на основе портов (VLAN).

– IPsec для сквозной проверки подлинности и шифрования IP-пакетов в сеансе связи.

– Безопасность транспортного уровня (TLS) для шифрования коммуникаций уровня приложения.

– Безопасность транспортного уровня (TLS) для криптографической защиты коммуникаций между приложениями на транспортном уровне.

– Сервис удаленного доступа пользователя (RADIUS) – сетевой протокол, который обеспечивает централизованное управление аутентификацией, авторизацией и учетом (AAA) для конечных устройств, использующих сетевые услуги.

Неизбежной платой за локализацию функций безопасности, заимствованную у традиционных сетей, и вынесение механизмов реализации этих функций за периметр контроллера являются определенные потери производительности при перенаправлении трафика на дополнительные устройства, что затрудняет применение этого подхода в сетях, чувствительных к задержке.

Для описания безопасности сетевой инфраструктуры и управления сетевыми сущностями на уровне узлов-посредников созданы профили защиты для различных типов межсетевых экранов и систем обнаружения вторжений. Например, в отношении ПКС для контроля потоков плоскости данных практически без доработки могут применяться профили защиты межсетевых экранов уровня логических границ (VLAN) или хостов внутри сети, а также профили защиты систем обнаружения вторжений уровня узла. Эти решения снабжены гармонизированными *«целями безопасности»* для среды и объекта в условиях актуальных *«угроз»* безопасности, для них сформулированы наборы *«предположений»*, позволяющие интегрировать их в состав сетей с различными уровнями и границами доверия. Опытом применения этих профилей защиты практически решены проблемы трансляции *«политик»* безопасности в нотации конфигурационных файлов и других инструментов настройки. Дополнительно отметим, что даже при централизованной инкапсуляции функций безопасности более точно назначению и роли контролера ПКС соответствуют именно системы обнаружения вторжений, которые централизованно собирают и анализируют информацию о состоянии узлов сети (это соответствует системам обнаружения вторжений уровня сети).

Особенности OpenFlow ограничивают возможности применения предопределенных (типовых) профилей защиты для контроля сетевых транзакций плоскости управления. Однако доработка профилей защиты, которая необходима для такого применения, может оказаться незначительной и коснуться лишь той части *«функциональных требований»*, которые непосредственно включают указания на *«атрибуты безопасности»*, т.е. используются для настройки сервисов безопасности на базе узлов-посредников. Это обстоятельство возможно в редком случае совпадения

«угроз», *«целей безопасности среды»* и *«предположений»* для потоков плоскости управления и плоскости данных.

3. Централизованная инкапсуляция сервисов безопасности

Одно из принципиальных архитектурных решений ПКС – это разделение сетевых сервисов для функций управления и функций пересылки пакетов. Согласно концепции ПКС, вся функциональность управления выносится в контроллеры, которые способны отслеживать работу всей сети и управлять сетевыми коммутаторами, выполняя планирование и управление трафиком с использованием программных приложений (что и определило название технологии). Этим обстоятельством можно воспользоваться для организации сервисов безопасности. Однако централизация управления в свою очередь способствует возникновению проблем безопасности, так как, в дополнение к традиционным атакам, обособленность функционирования контроллера и циркуляции сообщений в плоскости управления приводят к возникновению новых угроз безопасности [5], причем новый класс сетевых атак на коммутаторы и контроллеры ПКС возможен как из плоскости данных, так и в плоскости управления.

Рассмотрение безопасности протоколов (включая формирование угроз, политик и требований) целесообразно осуществлять раздельно по направлениям безопасности протокола контроллер – коммутатор (в том числе для аспектов, выходящих за пределы базового уровня, обеспечиваемого локализованными сервисами), безопасности протоколов управляющих приложений и безопасности протоколов контроллер – контроллер.

Анализ структуры устройств и использование модели STRIDE для идентификации угроз позволили сформулировать следующее общее описание угроз для ПКС [6, 7]:

– *Фальсифицированные потоки трафика*: потоки, созданные неисправными или дублирующими устройствами в сети, которые могут использоваться для отказа в ресурсах другим устройствам, либо в плоскости управления, либо в плоскости данных.

– *Эксплуатация уязвимостей коммутаторов*: попытки использования уязвимостей в коммутаторах, чтобы скомпрометировать эти устройства; такие атаки могут привести к другим уязвимостям в сети.

– *Атаки на управляющий канал связи*: любая атака, которая может поставить под угрозу безопасность канала связи между плоскостью данных и плоскостью управления, может нарушить или даже остановить сетевые операции; другие атаки могут просто попытаться перегрузить канал связи, чтобы предотвратить функционирование сети.

– *Использование уязвимостей контроллера*: аналогичны атакам на уязвимости коммутаторов, но гораздо более серьезны; после того как контроллер скомпрометирован, злоумышленник потенциально имеет полный контроль над сетью.

– *Нарушение доверия между контроллерами и приложениями*: большинство контроллеров не устанавливают правила доверия для приложений и не поддерживают механизмы для установления доверия; приложения, запущенные на контроллерах, должны быть доверенными, так как приложение контроллера имеет такое же представление о сети, как и контроллер.

– *Утрата надежности на станциях администрирования*: контроллер должен быть защищен от атак в сети, так же как и средства управления контроллером; если эта система скомпрометирована, злоумышленник может перепрограммировать контроллер, а не пытаться скомпрометировать его.

Эти направления являются основой для формирования проактивного аспекта в профиле защиты для централизованно инкапсулированных сервисов безопасности. Угрозы контроллеру рассматриваются по всем возможным интерфейсам: со стороны приложений, систем управления и устройств сети. Угрозам подвергаются критические информационные ресурсы контроллера, нарушается его работоспособность, на нем осуществляется захват управления. Важно отметить, что другие сетевые устройства и каналы связи в таком профиле защиты не рассматриваются в соответствии с выбранной архитектурой сервисов безопасности. Для них сформулированы соответствующие *«предположения»* защиты, которые ограничивают информационные ресурсы, безопасность которых является прерогативой разрабатываемого профиля защиты (пассивный аспект). Этими *«предположениями»* фактически определяется и поддерживается состав критически важных активов контроллеров ПКС, подлежащих защите [7]:

– Данные учетной записи пользователя и другие учетные данные (например, пароли, сертификаты и т.д.).

– Данные конфигурации и управления (например, ресурсы и политики администрирования; IP-адрес контроллера, порты, версия протокола и т.д.).

– Данные журналов регистрации.

– Операционная система (хостовая ОС).

– Программное обеспечение, включая программное обеспечение контроллера и прикладное программное обеспечение.

– Аппаратное обеспечение (например, плата, блок питания и т.д.), используемое для запуска программного обеспечения контроллера или приложений.

– Ресурсы (например, емкость процессора, возможность обработки памяти и т.д.)

– Интерфейсы контроллеров, включая интерфейс удаленного доступа между контроллером и системой администрирования.

Эти данные являются основой для формирования пассивного аспекта профиля защиты.

Примерами *«целей безопасности»* для профиля защиты ПКС в условиях централизованной инкапсуляции сервисов могут быть поддержание изоляции данных, предотвращение эксплуатации уязвимостей, предупреждение исчерпания ресурсов и др. (перечень не претендует на полноту).

Любая технология коллективного использования системных ресурсов требует изоляции пользовательских данных друг от друга. *«Цель безопасности»* должна учитывать, на каких уровнях модели обработки данных имеет место одновременное участие нескольких пользователей в вычислительном процессе – на уровне инфраструктуры (каналы, коммутаторы), на уровне платформы (контроллеры) или на уровне приложения. Механизмы достижения такой цели – *«политики»* и *«функциональные требования»*, должны выявлять и компенсировать фрагменты технологии, не поддерживающие разделение мандатов и (или) партиционирование, в которых один аппаратный модуль (например, процессор контроллера), фрагмент кода базового программного обеспечения (например, программ формирования таблиц потоков) или экземпляр прикладного приложения (процесс) используется несколькими различными пользователями параллельно.

Передаваемые и хранимые ПКС данные могут быть скомпрометированы или фальсифицированы в обход правил и процессов обеспечения безопасности в результате эксплуатации возможных уязвимостей на различных уровнях. Информация о таких уязвимостях может оказаться общедоступной до того, как проблема будет решена. *«Цель безопасности»*, устраняющая этот риск, может достигаться путем шифрования передаваемых и хранимых данных. Разумеется, соответствующими *«политиками»* и *«функциональными требованиями»* должно быть обеспечено управление ключами и сертификатами, используемыми для шифрования данных.

Поскольку превышение интенсивности запросов к службам над максимально допустимой нагрузкой может привести к недоступности сети для пользователей, соответствующая *«цель безопасности»* может быть выражена в виде гарантий для параметров доступности сети и/или восстановления ее в случае нарушения этих гарантий.

Сложился общий набор принципов, которым должны удовлетворят средства обеспечения безопасности контроллера [7]:

– *Ограниченная доступность контроллера.* Контроллер ПКС – это централизованное решение, атаки на контроллер и приложения очень опасны, поэтому доступ к контроллеру и его приложениям должен строго контролироваться.

– *Создание доверия.* Очень важны защита целостности топологий ПКС, а это означает, что контроллер, приложения и устройства, которыми он управляет, – все должны быть доверенными.

– *Надежная структура управления.* Структура управления должна обеспечивать развертывание ПКС в соответствии с требованиями проекта. Для обеспечения доступности и безопасности необходимо урегулировать возможные конфликты нескольких сетей ПКС под одним и тем же контроллером.

– *Безопасность базовой инфраструктуры.* Программируемые сети могут использовать виртуализации сетевой инфраструктуры для подключения сервиса

на основе использования физических и виртуальных ресурсов. Инфраструктуры ПКС должны быть устойчивыми к DOS-атакам на системы управления, потоки данных пользователей и управления, на приложения.

– *Мониторинг безопасности.* ПКС представляют собой объект, который сложно контролировать. В контексте этой динамически меняющейся сетевой топологии необходимо анализировать информацию счетчиков трафика, статистику в коммутаторах, журналы контроллеров и, возможно, даже трафика в плоскости данных. Также должно контролироваться использование ПКС-API (через ведение журнала обращения приложений). Большой объем данных из различных источников, и динамический характер сети могут потребовать разработки новых методов мониторинга и анализа (например, аналитики больших данных).

– *Защита интерфейса ПКС - API.* Необходимо обеспечить безопасность интерфейса с приложениями, включая проверку подлинности и авторизацию приложений, использующих его, а также предотвращение конфликтов между приложениями. API – это направление атак, которое может позволить приложениям поражать контроллер сети или другие приложения.

Перечисленные направления являются основой для создания *«политик»* (реактивного аспекта) в профиле защиты. Эти *«политики»* в профиле защиты должны гарантировать проектирование безопасности таким образом, чтобы контроллеры ПКС не являлись слабым звеном сети и не допускали возможность компрометации. *«Политики»* включают использование защищенных протоколов, шифрование трафика коммуникаций, реализацию взаимной аутентификации, управление доступом к сервису и др.

4. Функциональные требования безопасности

Различные конфигурации границ и уровней доверия предусматривают рассмотрение сети в разных вариантах [5]:

– в рамках одного пространства доверия – все компоненты принадлежат и управляются одним и тем же субъектом; граница доверия специфицирована и совпадает с периметром сети (режим «А»);

– сеть (возможно распределенная) принадлежит одному субъекту, а приложения ПКС могут принадлежать тому же субъекту, но разрабатываться и публиковаться могут другими субъектами или целой иерархией субъектов-разработчиков приложений; граница доверия специфицирована, но не совпадает с периметром сети, а проходит внутри него (режим «Б»);

– ПКС образует облачную структуру, т. е. допускается, кроме сторонних разработчиков приложений контроллера данной сети, использовать в качестве приложений другие виртуальные ПКС со своими контроллерами, приложениями и системой администрирования, граница доверия не специфицирована (режим «В»).

В настоящей статье в качестве базового рассматривается режим «А» использования сети (один провайдер), когда вся сетевая инфраструктура, средства вычислительной техники, коммуникационное оборудование и каналы связи находятся под единым контролем в доверенной зоне. Кроме того, все приложения также являются доверенными и предоставляются (разрабатываются) тем же провайдером. Такой вариант характерен для развертывания ПКС в едином центре, а более сложные режимы «Б» (наличие вторых и третьих участников разработки приложений, территориальная распределенность системы) и «В» (виртуализация сетевых функций) требуют отдельного рассмотрения.

Исходя из возможных угроз и практик по обеспечению безопасности контроллера, можно сформулировать общие *«функциональные требования»* безопасности, которые могут быть непосредственно не связаны с проектированием и разработкой контроллера, но они важны для его среды, эксплуатации или управления. В формате профиля защиты ГОСТ 15408 это семейства и компоненты требований:

Таблица 1. Общие функциональные требования

Table 1. General functional requirements

Семейство требований **Requirement family**	**Компоненты требований** **Requirement components**
Проверка IP	
Аутентификация пользователя	
Управление учетными записями	
Целостность и согласованность оборудования	
Безопасность гипервизора	
Проверка целостности ПО	
Целостность передаваемых данных	
Функция журнала регистрации	Устойчивость функций Защита доступа к журналам Защита модификации от журнала
Защита конфиденциальности передаваемых данных	
Скрытие отображения пароля и ключей	
Разделение разных типов трафика	
Безопасность виртуальных машин	
Закрытие неиспользуемых служб	
Безопасность физического хоста	
Функция Anti-DoS:	Анти-DoS от исчерпания ресурсов, вычислительной мощности.
Разрешение на использование системных функций	
Разрешение интерфейса для третьих лиц (разработчиков ПО)	
Безопасность хостовой ОС	

Общие *«функциональные требования»* безопасности являются развитием требований к сетевым устройствам и коммуникациям, которые должны использовать-

ся для обеспечения безопасности управляющего контура ПКС. Контроллер ПКС представляет собой сетевую операционную систему, но он оперирует сетевыми сущностями, отличными от объектов операционных систем. По сути он представляет собой реализованную на основе операционной системы АСУ по управлению сетью. И несмотря на то что ряд функций безопасности контроллера ПКС может обеспечиваться функциональностью операционной системы (соответствующего класса защиты), архитектура ПКС выдвигает ряд специфических функциональных требований безопасности:

Таблица 2. Специальные функциональные требования

Table 2. Special functional requirements

Семейство требований **Requirement family**	**Компоненты требований** **Requirement components**
Аутентификация на интерфейсах контроллеров SDN	
Защита данных конфигурации и резервных копий от модификации	
Управление доступом к критической информации	
Скрытие отображения пароля и ключа	
Изоляция приложения	
Функция Anti-DoS	Ограничение пересылки («шторма») IP-пакетов в контроллер от коммутаторов Контроль доступа к таблицам потоков Мониторинг трафика интерфейсов приложений и сетевых устройств и блокировка DoS-атак Ограничение ресурсов для приложений – Анти-DoS от чрезмерного потребления ресурсов
Контроль за привилегиями приложений	
Политика разрешения конфликтов (создания правил потоков, доступа к информации и т.п.) приложений	

Изменяющийся характер конфигураций и связей ПКС требует определения и отражения в виде *«политик»* и *«предположений»* безопасного начального состояния и сохранения заданного уровня безопасности в процессе изменений. Кроме того, такая изменчивость может приводить к конфликту правил, реализуемых различными приложениями. Это вызывает необходимость включения в состав *«функциональных требований»* спецификаций для процедур разрешения таких конфликтов.

Авторы сознательно не рассматривали «требования доверия», поскольку факторы, определяющие эту категорию требований, выходят за рамки инфраструктурной архитектуры и определяются в основном критичностью информационных активов.

Заключение

Таким образом, сформулирован базис для всех аспектов представления объекта и среды и их взаимодействия в контексте проблем безопасности. Это позволило подготовить спецификации конкретных угроз, предположений и политик, объединенных общим целями безопасности, как для объекта (контроллера ПКС), так и для среды его функционирования, и приступить к формированию функциональных требований безопасности. Дифференциация уровней, определяющих состав функциональных требований безопасности, предполагает несколько классов защиты. При разработке профиля защиты контроллера ПКС было принято, что рассматривается класс безопасности с минимальным составом функциональных требований безопасности, а для конкретного использования класс может быть уточнен. В рамках этих условий в МТУСИ был разработан проект профиля защиты контроллера программно-конфигурируемых сетей типа «А» пятого класса защиты (ИТ.КПКС.А5.ПЗ).

Список литературы / References

[1] *ISO/IEC 15408-1:2005 Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model*, https://www.iso.org/standard/40612.html.

[2] Anwer B., et al., “A Slick Control Plane for Network Middleboxes”, *Open Networking Summit*, 2013, http://nextstep-esolutions.com/Clients/ONS2.0/pdf/2013/researchtrack/posterpapers/final/ons2013-final51.pdf.

[3] Fayazbakhsh S., et al., “FlowTags: Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions”, *HotSDN’13*, ACM, 2013, http://www.cs.columbia.edu/~lierranli/coms6998-8SDNFall2013/papers/Flowtags-HotSDN2013.pdf.

[4] Qazi Z.A., et al., “SIMPLE-fying Middlebox Policy Enforcement Using SDN”, *SIGCOMM*, ACM, 2013.

[5] *ONF Threat Analysis for the SDN Architecture. Version 1.0*, TR-530, July 2016, https://www.opennetworking.org/wp-content/Threat_Analysis_for_the_SDN_Architecture.pdf.

[6] Pilyugin P., Smeliansky R., “Modern security issues in SDN”, 2-nd International Conference on Information Technologies, Systems and Networks. ITSN-2017 (Chisinau, Republic of Moldova, 17 – 18 October 2017).

[7] *ONF Security Foundation Requirements for SDN Controllers. Version 1.0*, TR-529, July 2016, https://www.opennetworking.org/wp-content/Security_Foundation_Requirements_for_SDN_Controllers.pdf.

Petukhov A. N., Pilyugin P. L., "«Common Criteria» and Software Defined Network Security", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 134–145.

DOI: 10.18255/1818-1015-2019-1-134-145

Abstract. «Common criteria» (ISO 15408) is a universally recognized and broadly applicable approach to information security solutions management and evaluation. «Common criteria» leans on developing a shared conceptual basis for key security solution modules including protection profiles and security targets. Conceptual basis development implies defining the following elements: security objectives and assumptions (for the environment and the object), threats and security policies, as well as functional and assurance requirements. The specifics of SDN (software defined network) security solutions is largely driven by fundamental architectural principles of SDN technology itself – primarily

by the separation of control and data flows, – and by conditions imposed by Open Flow protocol application. However, proactive (threats and policies), passive (objectives and assumptions) and reactive (requirements) aspects of security management remain highly relevant for this type of security solutions. This paper discusses the Common Criteria application specifics for assessing the SDN security and practical MTUCI (Moscow Technical University of Communications and Informatics) experience in the development of the protection profile. A new class of network attacks on SDN switches and controllers can involve either data or control components. In addition to traditional vulnerabilities, centralization of management functions paves way for new security threats by isolating controller activity and administrative message exchange. Therefore, identifying and analyzing threats, policies and requirements specific to SDN control module security becomes an emerging priority.

Keywords: security of software defined networks, general criteria, security profile

On the authors:
Andrey Petukhov, PhD, orcid.org/0000-0002-1427-2440
National Research University of Electronic Technology – MIET
Bld. 1, Shokin Square, Zelenograd, Moscow, Russia, 124498, e-mail: anpetukhov@yandex.ru

Paul Pilyugin, PhD, orcid.org/0000-0003-0011-7180
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: ppl@mail.ru

Acknowledgments:
The work was supported by MTUCI (Moscow Technical University of Communications and Informatics) rectorate: Erokhin S., Leokhin Yu. and Mukhanov A., and with funding from the MTUCI, in the direction of «Security of critical information infrastructures».

Моделирование и анализ информационных систем. Т. 26, №1 (2019), с. 146–169
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 146–169

DOI: 10.18255/1818-1015-2019-1-146-169

УДК 004.7

Иерархические периферийные вычисления

Смелянский Р. Л.

Поступила в редакцию 10 января 2019
После доработки 12 февраля 2019
Принята к публикации 14 февраля 2019

Аннотация. На смену вычислительной парадигме, основанной на giant-like ЦОДах, идет новая, основанная на сети мелких ЦОДов, образующих инфраструктуру для облачных вычислений. Эта смена объективна. Её актуальность обусловлена требованиями новых приложений, активно использующих видео, интерактивность в реальном времени, новые технологии мобильной связи, которые сегодня невозможно реализовать без облачных вычислений и виртуализации на основе технологий SDN&NFV. В статье рассмотрены требования, предъявляемые этими приложениями, предложена архитектура новой парадигмы, которую мы называем «Иерархическими периферийными вычислениями» (Hierarchical Edge Computing – HEC). Показано, что большинство современных приложений являются распределенными совокупностями сервисов реального времени, которые требуют гарантированного качества обслуживания и возможности динамически быть размещенными при работе на периферии сетей разных операторов. Обсуждаются основные научные проблемы, которые необходимо решить для реализации предлагаемой новой парадигмы.

Ключевые слова: ПКС, программно-конфигурируемые сети, ВСС, виртуализация сетевых сервисов, туманные вычисления, облачные вычисления, ЦОД, центры обработки данных, мобильные периферийные вычисления

Для цитирования: Смелянский Р. Л., "Иерархические периферийные вычисления", *Моделирование и анализ информационных систем*, **26**:1 (2019), 146–169.

Об авторах:
Смелянский Руслан Леонидович, чл.-кор. РАН, д-р физ.-мат. наук, проф., orcid.org/0000-0003-2311-4513
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, стр. 52, г. Москва, 119991, Россия, e-mail: smel@cs.msu.ru

Благодарности:
Работа выполнена при поддержке РФФИ, грант N 18-07-01245.

Введение

Мы быстро движемся из эпохи «создай свое» в эпоху просто «используй услугу». От организации вычислений, когда предприятие вынуждено покупать себе все необходимое сетевое оборудование, подключение и услуги провайдера, нанимать дорогостоящих Cisco Certified Internetworking Experts (CCIE) специалистов, которые должны заставить все это заработать, – к организации вычислений, когда предприятие просто использует «сеть как услугу». В этой новой развивающейся парадигме «сеть

как услуга» можно прокладывать и убирать соединения, разворачивать услуги, тогда и там, где они нужны, и оплачивать только то, что было использовано.

За последние 10 лет облачные вычисления, как вычислительная парадигма, полностью изменили ландшафт информационно-коммуникационных технологий (ИКТ) [1]–[4]. Это значительно способствовало росту как числа центров обработки данных (ЦОД), так и их размера, увеличению пропускной способности магистральных каналов [2], увеличению плотности оборудования: виртуализация ИТ-оборудования в облачных архитектурах позволила втиснуть в одну стойку то, что раньше требовало 10 стоек. Совершенствование и развитие возможностей персональных гаджетов, различных типов датчиков, развитие технологий передачи данных, таких как OTN, сети 5G, сетевая конвергенция, появление технологий программно-конфигурируемых сетей (ПКС)[1] и виртуализации сетевых функций (ВСС)[2] дало толчок развитию большого числа приложений реального времени [5]. Вот лишь некоторые примеры таких приложений: умный город, умный дом, здравоохранение (особенно такие его области, как хирургия, телемедицина, экстренная кардиология), интерактивные игры, обучение, дополненная реальность, сельское хозяйство, инфраструктура для научных многодисциплинарных исследований, социальные коммуникации, системы управления объектами энергетики (умные сети электроснабжения), беспроводные датчики, встроенные в различные роботизированные устройства, мониторинг и управление транспортными системами и объектами, сборочные производственные линии, газопроводы и нефтепроводы. Согласно оценкам IDC, в 2019 году 49% всего трафика наших сетей будет генерироваться такими приложениями.

Все вышеупомянутые приложения чувствительны к задержкам на время реакции, требуют интеграции географически распределенных данных. Эти ограничения могут быть жесткими или нет, в зависимости от приложений. Они проистекают либо из характера самих приложений, либо из требований пользователей, которые готовы ждать ответа лишь ограниченное время. При этом качество представления ответа должно удовлетворять ожиданиям пользователя. Другими словами, все больше и больше наших приложений становятся приложениями реального времени.

Ограничения, которые налагают приложения как жёсткого, так и мягкого реального времени, имеют две основных составляющих: общее время на передачу исходных данных и результатов их обработки, а также собственно время их обработки. Баланс между этими составляющими существенно зависит от архитектуры вычислительной (обрабатывающей) и коммуникационной инфраструктуры (ИКТ-инфраструктуры).

Усиление ограничений на время взаимодействия между приложением и терминальным устройством пользователя привело к противоречию с концепцией вычислений, основанной на giant-like ЦОДах. Дело в том, что в инфраструктуре на основе giant-like ЦОДов наши возможности изменять задержку при передаче данных и их обработке весьма ограничены. В таблице 1 на основе данных из [30] показаны такие характеристики взаимодействия, как задержки, потери пакетов и пропускной способности в зависимости от расстояния между ЦОД и конечным пользователем.

В свое время одним из основных доводов за концепцию giant-like ЦОДов были экономические соображения снижения общей стоимости владения ИКТ-инфра-

[1]Англ. эквивалент Software Defined Network (SDN).
[2]Англ. эквивалент Network Function Virtualization (NFV)

Таблица 1. Характеристики взаимодействия клиента и сервера

Table 1. Interaction characteristics of a client and the server

Расстояние (между клиентом и сервером)	**RTT**	**Потери пакетов**	**Пропускная способность**	**Загрузка 4 GB**
<160 км	1.6 мс	0.6%	44 Mbps	12 мин
<750–1600 км	16 мс	0.7%	4 Mbps	2.2 ч
∼ 4000 км	48 мс	1.0%	1 Mbps	8.2 ч
∼ 9000 км	96 мс	1.4%	0.4 Mbps	20 ч

структурой такого ЦОД. Однако рост числа мобильных устройств (по данным аналитиков, 50 миллиардов штук к 2020 году, т.е. на следующий год) приводит к резкому росту объёма сгенерированных данных, которые необходимо обработать и на которые необходимо должным образом отреагировать [6]. Ярким примером тому является «Интернет вещей» (IoT). Миллиарды IoT-устройств будут наводнять наши сети зеттабайтами данных, которые требуют обработки и ответа в режиме реального времени. Согласно отчету Cisco Systems [7], к 2019 году 500 ZB данных будут «прокачиваться» через наши сети в год, а к 2020 году каждый день – 2,3 ZB. IoT-приложения требуют высокой степени мобильности, строго ограниченной задержки доставки данных и обработки их в режиме реального времени [8]. Для обработки такого потока данных в реальном времени требуются новые решения и новые подходы к организации вычислительной и коммуникационной инфраструктуры.

1. Вычисления на периферии сегодня

На сегодняшний день было предложено несколько подходов к организации ИКТ-инфраструктуры для приложений, для которых решающее значение имеет время отклика: туманные вычисления (Fog computing), облачка – мобильные облачные мини-ЦОДы (Cloudlets), мобильные вычисления на границе (MEC), микро-ЦОДы (MDC). Все они в основном сосредоточены на потребностях IoT и имеют ограниченную интерпретацию концепции периферии (Edge) сети. Например, это может быть зона доступа в сеть, расположенная как можно ближе к пользовательскому устройству или датчику (Customer Premise Equipment – CPE) и оснащенная средствами вычисления, хранения и накопления данных. Ниже кратко рассмотрены перечисленные выше подходы.

Fog computing представляет собой платформу, которая обеспечивает облачные вычисления в непосредственной близости от конечных пользователей. Первоначально термин «туман» был введен Cisco [9]: виртуализированная «туманная» платформа развертывается близко к конечным пользователям – между традиционными giant-like облачными ЦОДами и конечными пользователями. Хотя как облачная, так и «туманная» парадигма поддерживает почти аналогичный набор сервисов (вычисление, хранение, сетевое взаимодействие), между ними существуют разли-

чия. Развертывание «туманных» вычислений предназначено только для определенного географического региона. Кроме того, эта платформа специально создавалась для приложений IoT и для приложений, требующих ответа в реальном времени с минимальной задержкой. С другой стороны, традиционный облачный ЦОД централизован и расположен в основном далеко от пользователя. Ему присущи некоторые ограничения по задержке и времени отклика приложений реального времени. «Туманная» платформа предполагает использование разнообразных устройств, собирающих данные разных типов. Взаимодействие между гетерогенными устройствами – не единственная проблема для «туманных» вычислений. Другими являются оркестрация и управление ресурсами, их балансировка, масштабируемость, безопасность и конфиденциальность. До сих пор нет стандартной модели монетизации для «туманных» вычислений. Это все еще открытая исследовательская проблема.

Cloudlets (облачка) разрабатываются командой Университета Карнеги – Мелона [11], [28]. Они предназначались для приближения облачных сервисов к мобильным пользователям. Внутри «облачка» состоят из набора достаточно мощных ресурсов, таких как многоядерные серверы с высокоскоростным подключением к Интернету и высокоскоростной беспроводной локальной сетью для связи с мобильными устройствами [11]. Мобильное устройство, рассматриваемое как тонкий клиент, может загружать вычислительные задачи через беспроводную сеть в «облачко», находясь при этом от него на расстоянии одного скачка (hop). Наличие облачка вблизи мобильного устройства необходимо для сокращения и предсказуемости времени приема-передачи для исполняемых приложений. Если устройство выходит из зоны действия «облачка», то оно либо должно переключиться на удаленный облачный ЦОД или пользоваться своими собственными ресурсами.

Концепция Micro-DC (μDC; микро-ЦОД) была представлена компанией Microsoft Research [10]. Она рассматривает микро-ЦОД как расширение сегодняшнего большого облачного ЦОД. Подобно «облачкам», микро-ЦОД также разработан для приложений, которые требуют малых задержек на коммуникацию, обработку, и устройств, которые работают в условиях жестких ограничений по энергетике. Микро-ЦОД – это вычислительный комплекс, состоящий из одного или нескольких соединенных между собой стоек, оснащенных всей необходимой инфраструктурой для ИТ-оборудования, собранных и протестированных производителем. Микро-ЦОД представляет собой автономную безопасную вычислительную среду, которая включает в себя все необходимые вычислительные ресурсы, хранилище данных и сетевое оборудование для работы клиентских приложений. Потребляемая мощность микро-ЦОД может составлять от 1 до 100 кВт для удовлетворения требований к масштабируемости и задержкам с учетом рабочей нагрузки, она может меняться, если в будущем потребуется больше энергии.

Следует отметить, что промышленность давно освоила производство таких вычислительных комплексов. Примерами являются UCS от Cisco, V-Blocks компании VCE, Active Systems от Dell, компания Schneider Electric предлагает Smart Bunker и Smart Data Safe. Система VPLEX является продуктом компаний EMC, Microsoft и AVNET [28]. Huawei – еще один производитель, выпускающий микро-ЦОДы [20]. Перечисленные выше микро-ЦОДы в основном предназначены для размещения вычислительных ресурсов и телекоммуникационного оборудования в неподготовленных помещениях (таких как офисы, склады, подсобные помещения или производ-

ственные объекты) и подключения их в корпоративную сеть. По словам поставщиков, время их установки (до начала использования) сократилось до 60–70% по сравнению с классическим решением.

Mobile Edge Computing (MEC; мобильные периферийные вычисления) предназначены для предоставления облачных вычислительных ресурсов и ИТ-услуг на периферии сотовых сетей [14] (см. Рис. 1). MEC обеспечивает малые задержки, близость к оконечному устройству пользователя, знание контекста его работы и местоположения, а также более высокую пропускную способность. Как видно на рисунке 1, серверы MEC развернуты на сотовых базовых станциях, что позволяет гибко и быстро развертывать новые приложения и услуги конечным пользователям. MEC можно рассматривать как облачные серверы, работающие на границе зоны радиодоступа мобильных сетей и реализующие конкретные услуги, которые не могут быть достигнуты с использованием традиционной сетевой инфраструктуры. При использовании MEC весь трафик перенаправляют не на удаленный облачный ЦОД, а на серверы MEC. Таким образом, серверы MEC, работающие с приложениями и выполняющие связанные с ними задачи обработки данных, ближе к сотовым клиентам, уменьшают нагрузку на сети и время отклика приложения. ETSI1 разработал отраслевую спецификацию MEC [21] и опубликовал ее в сентябре 2014 года. Были предложены системная архитектура и стандарты ряда API, необходимые для MEC [21]. Компания Nokia, например, продемонстрировала, что MEC играет ключевую роль в автоматизации вождения автомобилей. В случае подключения автомобиля к традиционному облачному ЦОД задержки будут составлять не менее 100 мс. Базовые станции с распределенными облаками MEC продемонстрировали сквозную задержку в пределах 20 мс.

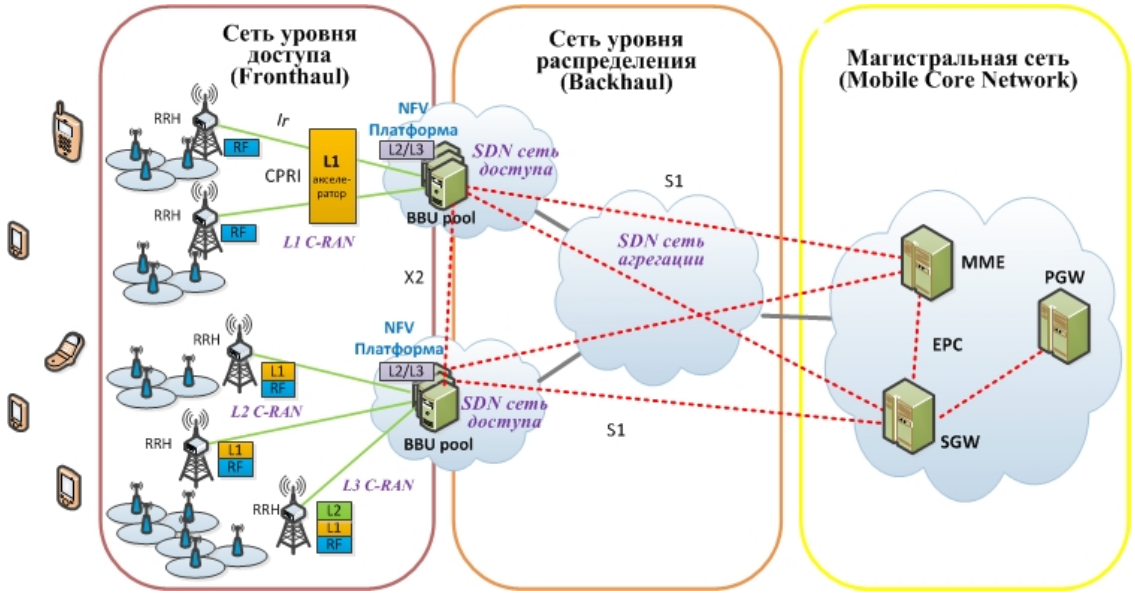


Рис. 1. Mobile Edge Computing для сетей 5G

Fig. 1. Mobile Edge Computing for 5G networks

Анализ приложения CDN. Технологии периферийных вычислений, описанные выше, подходят для тех приложений, для которых можно локализовать необходимые сервисы в непосредственной близости от мобильного устройства, чтобы удовлетворить ограничениям, связанным с задержками или с недостатком вычислительной мощности. Однако есть приложения, для которых ни один из вышеупомянутых вариантов организации ИКТ-инфраструктуры не является достаточным. Например, согласно данным компании Akamai [30], более 650 сетей участвуют в доставке 90% трафика этой компании. Если мы посмотрим на приложение доставки контента как на пример приложения реального времени, то мы увидим, что в доставке этой услуги участвуют несколько разных сетей разных провайдеров.

Рассмотрим в качестве примера организацию трансляции игр на чемпионате мира по футболу 2014 года в Бразилии [23]. Для этого чемпионата была создана специальная инфраструктура по производству видеоконтента, включающая следующие услуги, поддерживаемые компанией EVS (Event Video Service):

- Мультимедийная прямая трансляция матча;
- Выделенный мобильный/мультимедийный канал;
- Дополнительный контент в режиме видео по запросу (Video-on-Demand; VoD);
- Многоракурсный контент;
- Вставка рекламных видеоклипов в контент;
- Мультимедиа и текстовые сообщения (MMS и SMS);
- Интерактивный доступ к данным;
- Визуализация данных.

Техническая инфраструктура EVS объединила множество ведущих технологических решений нескольких компаний, которые работали вместе, предлагая для зрителей лучшие интерактивные и мультимедийные продукты:

- от компании Elemental Cloud для облачной обработки видеопотока в реальном времени;
- от компании Aspera для передачи файлов на высокой скорости через сети от места проведения мероприятия до облачной инфраструктуры;
- от компании Brightcove – облачные сервисы для операций по перекодированию мультимедийного контента;
- от Amazon S3 – хранилище данных;
- от Akamai – сеть доставки контента (CDN);
- от компании NETCO Sports - сервис режима второго экрана («second screen»).

Во время проведения матчей шесть видеопотоков с шести камер (camera angles) записывались на серверах EVS XT3 непосредственно на месте проведения матча, где автоматически обрабатывались средствами EVS C-Cast Agent и отправлялись в виде потоков по 10 Мбит/с каждый компанией IBC (Международная вещательная корпорация) через волоконно-оптическую сеть. В компании IBC они обрабатывались специальным программным обеспечением C-Cast Central, после чего обработанные потоки передавались по волоконно-оптической сети из IBC в хранилище Amazon S3 в Дублине, где было развернуто производство EVS C-Cast. Там каждый входящий поток, при помощи программного обеспечения компании Elemental, был фрагментирован на пакеты, каждый из которых нёс видеофрагмент с фиксированным временем проигрывания. После этого из каждого такого пакета генерировали 10 различных видеопотоков разного качества, которые передавали со скоростью 10 Мбит/с для доставки по CDN сети по конкретному адресу.

В рассматриваемом примере отметим следующее. Сеть доставки контента (CDN) имеет статическую оверлейную структуру, производство 10 видеопотоков различного качества для передачи одного и того же контента было распределено между конкретными устройствами, кэширование производилось только на Edge серверах. Если бы такие сервисы для видеопотоков, как перекодирование, кэширование, трансрейтинг, компрессия, были доступны не только на Original и Edge серверах, но и в сетях, через которые велась трансляция, это значительно снизило бы нагрузку на эти серверы. Также за счет интенсивного использования групповой передачи (multicast) вместо одноадресной (unicast) передачи сократилась бы и нагрузка на сеть. Следует отметить, что эффект от размещения всех перечисленных сервисов в сети значительно возрос бы, если бы размещение определенных видов сервисов в сети можно было менять динамически. Примером такого сервиса может служить кэширование. В зависимости от степени популярности, контент должен быть кэширован как можно ближе к какой-либо локальной группе клиентов, если этот контент представляет интерес в значительной степени для этой локальной группы клиентов. Если же он будет интересен более широкой аудитории, то размещать его надо так, чтобы время доступа и объем передаваемого трафика для клиентов из разных регионов были сбалансированными.

Из рассмотренного примера видно, что между сетями, в которых размещается источник данных и/или инициатор запроса данных, было задействовано несколько сетей с дополнительными услугами по обработке видеопотоков. Возникает вопрос: где та «граница», периферия, о которой мы говорим? Границу какой сети мы имеем в виду, когда говорим о доставке контента?

Другой важный вывод, который можно сделать на основе рассмотренного – приложение более не локализовано в одном ЦОДе. Оно превратилось в систему взаимодействующих сервисов, распределенных, в общем случае, в разных сетях.

2. Концепция иерархических периферийных вычислений

На рисунке 2 показана типичная структура сети доставки контента (CDN), арендованной поставщиками контента (CP) у какого-либо Интернет-провайдера.

Как видно из этого рисунка, существует несколько сетей, подконтрольных различным Интернет-провайдерам, которые предоставляют клиентам доступ к контенту определенного поставщика контента. Естественно, возникает вопрос, о какой периферии какой сети идет речь, какие из этих Интернет-провайдеров и как должны взаимодействовать при доставке контента?

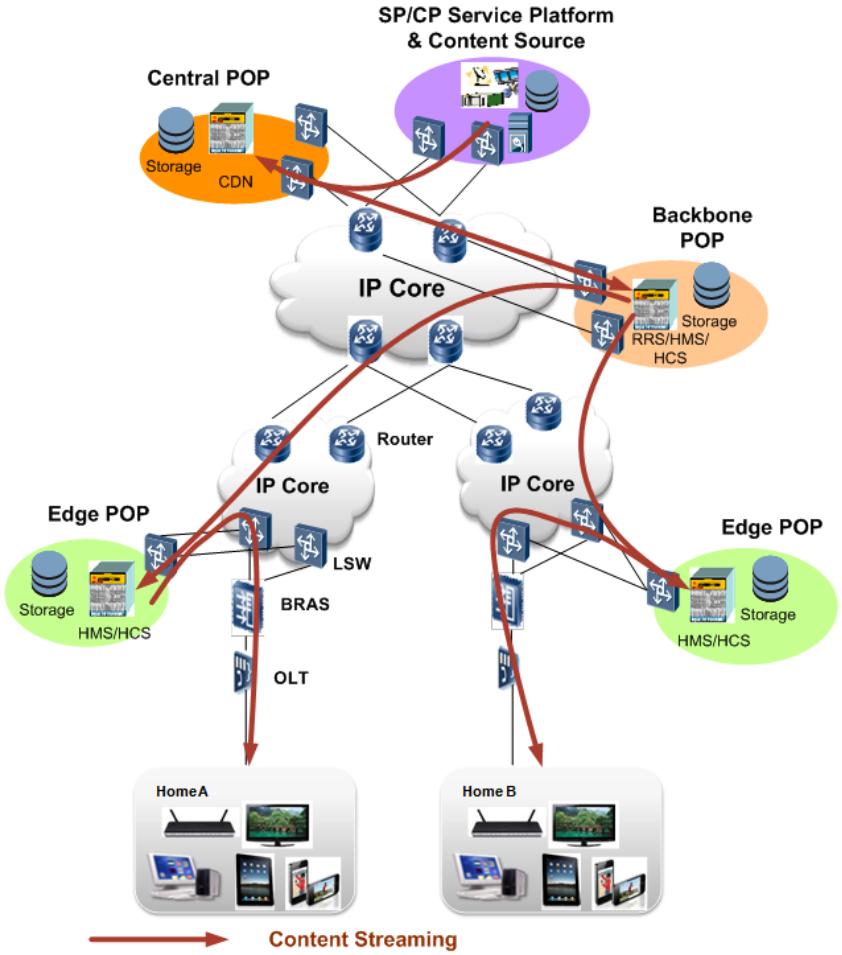


Рис. 2. Типовая структура CDN сети,
арендуемая контент-провайдером (CP) у ISP оператора

Fig. 2. Typical structure of a CDN network
rented by a content provider (CP) from an ISP operator

Из этой иллюстрации и приведенного выше примера можно сделать вывод:

- Во-первых, данные для приложения могут поступать из нескольких источников, расположенных в разных сетях;
- Во-вторых, ограничение на время доставки, обработки полученных данных не всегда может быть соблюдено в случае единого центра обработки данных. В этом случае решением может быть распределение обрабатывающих

мощностей в нескольких периферийных центрах обработки, в сетях разных Интеренет-провайдеров (ISP операторов) с учетом ограничений на время их взаимодействия.

- В-третьих, нужны динамизм и гибкость при выборе топологии виртуальной CDN сети и таких ее параметрах, как пропускная способность, места размещения в ней таких сервисов, как кэширование, сжатие, транскодинг, трансрейтинг и т.д.

Идея иерархических периферийных вычислений основана на размещении микро- или мини-ЦОД (μ/m-DC соответственно), оснащенных сетевыми, вычислительными ресурсами, ресурсами хранения данных, на периферии сетей, через которые проходит поток данных от/до устройств конечного пользователя. Такие сети мы будем называть взаимодействующими сетями. Устройства конечных пользователей обычно представляют собой смартфоны, носимые гаджеты и различные устройства для «интернета вещей», беспроводных сенсорных сетей, требующие реакции в реальном времени. Концепция иерархических периферийных вычислений позволяет динамически выбирать распределение затрат на передачу и обработку данных в зависимости от текущей загрузки ресурсов взаимодействующих сетей, расположенных между источником данных/запроса и местами их обработки, и ограничений, накладываемых приложением.

Развертывание как вычислительных ресурсов, так и ресурсов хранения данных на границах взаимодействующих сетей позволяет реализовать большое количество приложений, требующих реакции в реальном времени. Список таких приложений включает в себя, но не ограничивается перечисленными ниже примерами:

- мониторинг транспортных потоков и навигация, включая передачу информации о трафике и расчет маршрутов для определенного региона непосредственно на периферии сети;

- фильтрация и агрегация данных, при которых выполняется предварительная фильтрация контента и данных на периферии сети, для уменьшения объема данных перед отправкой их в традиционное облако;

- дополненная реальность, интерактивные медиа, распознавание речи, обработка естественного языка [11], [28];

- сеть доставки контента (CDN);

- научные эксперименты.

Важным свойством концепции иерархических периферийных вычислений является возможность динамически определять состав и расположение сервисов в виртуальной ИКТ-инфраструктуре с использованием техники виртуализации сервисов в облаках микро- и мини-ЦОДов на периферии взаимодействующих сетей. Под термином «периферия взаимодействующих сетей» мы будем понимать набор точек входа/выхода в сеть конкретного провайдера, например, точку присутствия провайдера (PoP) в зоне доступа его сети, BGP шлюзы и т.д. Как можно ближе к

этим точкам необходимо разместить микро- и мини-ЦОДы с облачной инфраструктурой, с вычислительными мощностями и хранилищами данных. Термин «ближе» означает контролируемую, небольшую и гарантированную задержку доступа. Здесь мы также будем широко трактовать понятие услуги. В сети все является сервисом: предоставление ресурса – сервис, агрегирование функций обработки – сервис. Такую трактовку сегодня позволяет подход концепций программно-конфигурируемой сети (ПКС) и виртуализации сетевых функций (ВСС).

Для иерархических периферийных вычислений ПКС предлагает уникальную возможность поддерживать глобальное видение сети вплоть до соединений и портов, различных полей заголовков разного уровня. При этом контроллеры ПКС могут запрашивать статистику коммутаторов, обнаруживать потоки с определенными шаблонами заголовков пакетов, динамически реагировать на обнаруживаемые угрозы. Виртуализация сетевых функций предоставляет уникальные возможности динамически размещать виртуализованный сервис в надлежащем периферийном ЦОДе, чтобы поддерживать баланс между задержкой и временем обработки. Обе концепции хорошо дополняют друг друга.

Одной из ключевых проблем такой сетевой организации становится взаимодействие автономных систем (АС) для обмена информацией о том, какие услуги они могут предоставить друг другу, с какими характеристиками, с каким качеством. Традиционно АС обмениваются информацией в точках обмена Интернет-трафиком (IXP), которая в случае использования ПКС называется Software Defined Exchange Point (SDX).

Сегодня основным протоколом, обеспечивающим связь между автономными системами, является BGP. Однако для иерархических периферийных вычислений BGP протокол в традиционной форме не подходит. В своей традиционной форме он предназначен для выбора маршрута для доступа из одной АС к другой АС. Этого для реализации концепции иерархических периферийных вычислений уже недостаточно. Необходимо, чтобы протокол, назовем его «расширенным BGP» (Extended BGP – EBGP), разрешал АС (можно в этом случае использовать термин АС-издатель (AS-publisher)) сообщать другим, внешним АС, какие услуги могут быть им доступны в АС-издателе и в соответствии с какими соглашениями об уровне услуг (SLA). АС-издатель должен предоставлять о каждой опубликованной услуге такую информацию, как: политика выставления счетов, соглашение об уровне услуг, информация о качестве обслуживания (QoS), минимальная ожидаемая производительность (задержка на обработку), через какой шлюз АС-издателя эта услуга доступна. Здесь термин «услуга» можно толковать широко: это предложение некоторых ресурсов, инфраструктуры как услуги (IaaS) и различных виртуализированных сервисов. Ключевым моментом является то, что АС-издатель должен гарантировать условия соглашения об уровне услуг, такие как: доступность услуги, задержка доступа, джиттер, производительность. Одним из возможных способов сделать это является включение в объявление АС некоторой части спецификации TOSCA (Topology and Orchestration Specification for Cloud Applications).

Язык спецификации TOSCA предназначен для предоставления информации о виртуализированной услуге, представляющей собой композицию из виртуализированных функций [33] в виде шаблона услуги. Шаблон описывает различные типы политик услуги, такие как масштабирование, пороговые величины (критическая

загрузка процессора, топология, производительность), некоторые параметры виртуальной машины (число ядер, оперативная память, объем диска, операционная система и т.д.). В описании политики также задается группа узлов сети, для которых будут отслеживаться эти параметры. Это описание также содержит инструкции, что делать, когда критическое значение будет достигнуто хотя бы на одном из узлов из группы. Подробнее см. [29].

Механизм шаблонов TOSCA – это средство описания, спецификации сервиса. Для целей объявления сервиса и связанной с ним политики можно было бы применить средства, созданные в рамках концепции сервис-ориентированной архитектуры [32], такие как язык WSDL (Web Services Description Language), для описания интерфейсов службы, протокол SOAP (Simple Object Access Protocol) для описания формата получаемых и отправляемых сообщений, а также стандарт UDDI (Universal Description, Discovery and Integration) для создания каталогов доступных услуг. Таким образом, концепция иерархических периферийных вычислений позволяет динамически выбирать затраты на передачу и обработку данных в соответствии с текущим уровнем загрузки ресурсов сетей, расположенных между источниками данных и/или запросов данных и местами их обработки, с учетом ограничений, налагаемых приложениями.

Важным моментом любой технологии является ее «уживчивость» с уже существующими. Можно привести много примеров, когда технология с прекрасными характеристиками так и не находила себе практического применения, просто потому, что ставила вопрос: либо я, либо все остальные. Отсутствие стандартов и устоявшейся терминологии приводит к неправильным представлениям о соотношениях между технологиями периферийных вычислений, «интернетом вещей» и облачными вычислениями. Примеры таких заблуждений упоминаются в литературе, где авторы утверждают, что технологии периферийных вычислений будут «теснить» или «заменять» облако туманом или децентрализованной парадигмой облачных вычислений на периферии. Как упоминалось в [28], существует путаница в понимании передовых Edge технологий, например, некоторые авторы рассматривают туманные вычисления как микро-ЦОДы [12], [13], в то время как другие сосредоточены главным образом на идее усиления и оснащения сетевых компонентов дополнительными ресурсами обработки и хранения.

Нужно четко понимать, что технологии периферийных вычислений, включая иерархические периферийные вычисления, не следует рассматривать как замену облачной парадигме. Как показано на рисунке 3, эти технологии дополнят облако и расширят облачные сервисы до самых отдаленных закоулков сетей, так чтобы удовлетворялись потребности приложений по работе в реальном времени. Таким образом, подход иерархических периферийных вычислений не конкурирует ни с «туманными» вычислениями, ни с «облачками», ни с мобильными «граничными» вычислениями, или микро-ЦОДами, ни с традиционным подходом, основанным на giant-like ЦОДах. Напротив, они дополняют друг друга. Это ясно видно на рисунке 3, где представлен список некоторых потенциальных областей применения иерархических периферийных вычислений.

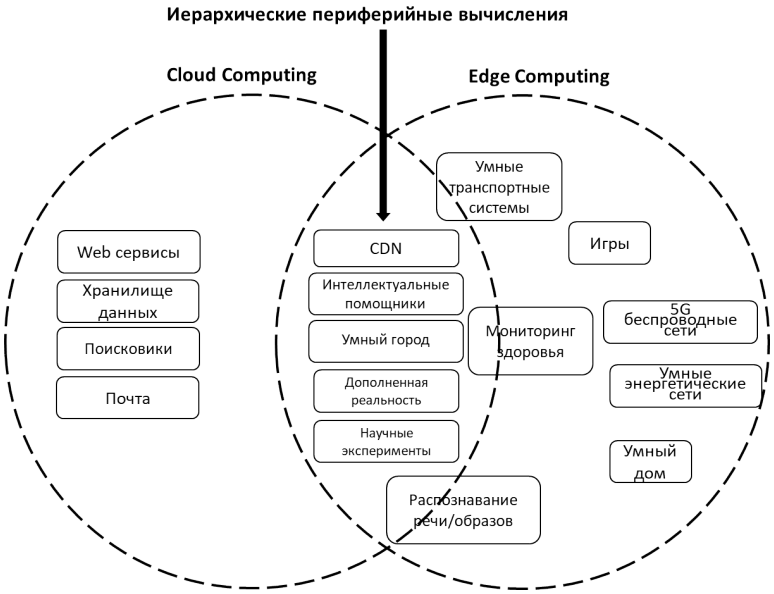


Рис. 3. Соотношение приложений традиционного подхода к облачным вычислениям на основе giant-like ЦОД, технологий Edge Computing и концепции иерархии периферийных вычислений

Fig. 3. Relation among applications of the traditional approach to cloud computing based on giant-like data center, Edge Computing technologies, and the concept of Hierarchical Peripheral Computing

3. Потенциал иерархических периферийных вычислений

Организация ИКТ-инфраструктуры на основе концепции иерархических периферийных вычислений позволяет:

1. Управлять задержкой при взаимодействии пользователя с сетевыми приложениями.

2. Сократить трафик через взаимодействующие сети.

3. Сократить требования к вычислительным возможностям и объему памяти на конечных устройствах (терминалах, мобильных устройствах и датчиках).

4. Минимизировать потребление энергии как на мобильных устройствах, так и на датчиках/сенсорах «интернета вещей».

Например, минимизация трафика в магистральной части одной из промежуточных сетей достигается путем помещения услуг по обработке входящего трафика как можно ближе к точке его входа. Очевидным примером тому является пример сети доставки контента (CDN). В случае «интернета вещей» минимизация трафика в опорной сети достигается за счет предварительной обработки трафика от датчиков на границе сети, т.е. ближе к датчику.

Организация иерархических периферийных вычислений на основе микро- и/или мини-ЦОД позволяет:

1. Свести к минимуму капитальные затраты, используемые площади и энергопотребление такого ЦОД.

2. Значительно сократить время на создание микро-ЦОД и ввод его в эксплуатацию.

3. Уменьшить сроки возврата инвестиций при создании микро-ЦОД.

Сокращение трафика в сети. Традиционная модель взаимодействия пользователя с Интернетом предполагает короткие запросы от пользователей на доступ к Интернет-сервису и получение в ответ иногда больших объемов данных. Например, передача файлов и, в частности, видео по запросу (VoD) или передача потокового видео в реальном времени, состоят из очень маленьких запросов данных от пользователя к поставщику услуг (SP) и большого объема данных, поступающих от поставщика услуг к пользователям. Как мы уже говорили, кэширование видеоконтента на границе разных сетей, образующих инфраструктуру для сети доставки контента (CDN), может значительно сократить объем передаваемых данных и задержки передачи контента от поставщика услуг до пользователя.

Динамическое размещение кэшей в сети доставки контента также может уменьшить эту задержку. Например, наиболее популярный контент сохраняется в кэшах Интернет-провайдеров или в сетях доставки контента (CDN), чтобы свести к минимуму поток данных и задержки во взаимодействующих сетях [24] – [26]. Транскодинг видео на периферии CDN в максимальной близости к конечным пользователям/устройствам позволяет использовать групповую рассылку вместо одноадресной передачи, что значительно сокращает объем трафика во взаимодействующих сетях. Это также помогает минимизировать задержки доступа и максимизировать «качество восприятия» (QoE) зрителей.

Например, при потоковом вещании, в частности потоковом вещании с использованием краудсорсинга, значительные объемы данных передаются от пользователей сервис-провайдеру, а затем распространяются глобально через различных провайдеров, таких как Twitch (on-line игровая система с поддержкой краудсорсинга), YouTube Live [28], Periscope [28] и YouNow [28]. Компания Netflix размещает свою огромную коллекцию развлекательного видеоконтента. Если 10% из 8 миллионов человек в Нью-Йорке захотят одновременно просматривать фильмы от Netflix, для одновременной обработки всех запросов потребуется инфраструктура с пропускной способностью 1,6 Терабит в секунду (Tbps) [28].

Например, рассмотрим, как вариант, финальный матч европейского футбольного турнира, где сеть Akamai обслуживала 3,3 миллиона видеопотоков одновременно, испытывая пиковую нагрузку в 7,3 Тбит/с [22]. В условиях отсутствия групповой рассылки, что является сегодня типичным случаем для CDN сетей из-за проблем с конфигурацией и безопасностью, такой поток данных, проходящий через множество маршрутизаторов между сетью доставки контента и сетью провайдера, приводит к значительным затратам энергии, а также к затратам на поддержание такой сети и управление ею. Кроме того, сеть доставки контента имеет пассивную систему хранения, хранящую большие объемы данных, и очень ограниченные возможности для

обработки данных. Транскодинг видеороликов на лету недоступен в существующих сетях доставки контента. В то же время, кэширование на границе сотовой сети (например, на базовой станции или на eNodeB) может сэкономить значительное количество трафика в Backhaul сегменте сети.

Минимизация задержки. Как отмечено в [28], для сервисов визуальной навигации (visual guiding services) в режиме реального времени предпочтительное время отклика составляет от 25 мс до 50 мс. Задержки в облачных вычислениях в Интернете являются серьезной проблемой для приложений, которым требуется реакция в реальном времени, например, таким как интеллектуальные транспортные системы, игры, приложения, использующие потоковое видео, и другие, важные для безопасности приложения, где такие задержки недопустимы. При использовании традиционного облака задержка на передачу данных от клиента к поставщику облачных услуг может составить от нескольких миллисекунд до секунды (см. Таблицу 1 выше) [28]. Даже небольшая задержка при обработке запроса пользователя может привести к потере абонента и дохода. Например, в [15] отмечено, что замедление обработки запросов всего на 2 секунды привело к сокращению количества запросов на одного пользователя на 1,8% и дохода на пользователя на 4,3%. Опрос, проведенный Forester, показал, что большинство покупателей Интернет-магазинов считают время отклика веб-сайта основным фактором при оценке удовлетворенности [28]. В этом опросе также выяснилось, что более 40% клиентов могут ждать загрузку страницы не более трех секунд, прежде чем они покинут сайт. В обзоре IDC [16] сообщается, что улучшение производительности и надежности услуг предоставляемых компанией Akamai и ориентированных на ускорение корпоративных приложений привело к ежегодному увеличению доходов этой компании с 2 до 3 миллионов долларов США. Поэтому размещение контента у локальных Интернет-провайдеров (граница сети) имеет решающее значение для областей с низкой связностью и высоким временем отклика [30]. В настоящее время многие Интернет-провайдеры создают облачные инфраструктуры на границах своих сетей. В таблице 1 выше показано, что иерархические периферийные вычисления могут снизить задержки, уменьшить трафик в опорной части сети, если виртуализированные сервисы будут расположены на границе сети Интернет-провайдера.

Сокращение потребления энергии. Потребление энергии облачными сервисами обычно зависит от следующих факторов [28]: (а) потребление энергии устройством конечного пользователя, (б) потребление энергии в ЦОДах, используемых для иерархических периферийных вычислений, включая энергию, потребляемую во внутренней сети самого ЦОДа, а также хранилищем данных, (в) объем трафика, передаваемого между пользователем и облаком, использующим иерархические периферийные вычисления, (г) вычислительная сложность выполняемой задачи; (д) количество пользователей, использующих вычислительный ресурс, и (е) потребление энергии транспортной сетью.

В [18] показано, что 14% потребления энергии в Интернете связано с транспортировкой данных. Этот источник также показывает, что интерактивные приложения генерируют значительный объем трафика и потребляют больше энергии из-за накладных расходов, возникающих в результате взаимодействия в реальном времени с традиционным облачным ЦОД. Было показано, что большой дополнительный трафик связан с частой установкой/разрывом TCP сессий и объемом данных, передава-

емых при этом пользователю и от пользователя за сеанс (измеряется от десятков до сотен КБ) [28]. Как мы уже говорили, кэширование на краю сети позволяет значительно сократить задержки доступа и сетевой трафик [28]. Кроме того, технологии периферийных вычислений позволяют размещать специализированные сервисы на границе сети для обеспечения реакции в реальном времени и фильтрации данных. Например, компания Akamai развернула пограничные вычислительные сети для обеспечения распределенного выполнения Java-приложений [30].

Сектор ИКТ является одним из основных потребителей электроэнергии, который, по оценкам, потребил более 271 млрд. КВт/ч энергии в ЦОДах в 2010 году [3]. Сетевая инфраструктура также является одним из основных потребителей электроэнергии. По оценкам [2], в 2010 году она потребила около 15,6 млрд. КВт/ч энергии. Эксперименты с периферийными вычислениями на границах взаимодействующих сетей, через которые проходит трафик мобильных устройств, показали, что потребление энергии может быть уменьшено на 42% [27].

Вычисления, организованные по принципу иерархических периферийных вычислений, могут дополнить традиционные облачные вычисления для определенных приложений, и это может привести к экономии энергии, если приложение или его компоненты могут быть перенесены из традиционного ЦОД в микро-ЦОД в транзитной сети. Кроме того, кэширование данных на граничных устройствах снижает нагрузку на опорную сеть, что позволяет снизить пропускную способность каналов за счет таких «зеленых» технологий, как Adaptive Link Rate (ALR) [4].

Уменьшение нагрузки на традиционные облака и традиционные ЦОДы. Сервисы, с определением своего местоположения (Location aware services), ежедневно генерируют огромное количество данных. Популярны несколько приложений для регистрации и учета спортивной активности, таких как Nike +, Runtastic, Runkeeper и Endomondo [28]. Эти приложения работают на смартфонах и регистрируют ежедневную активность пользователей с помощью различных датчиков, например, акселерометров, GPS, гироскопов и датчиков температуры, обычно устанавливаемых на смартфонах. В основном данные, записанные приложениями, отправляются в облако в виде наборов, где каждый набор содержит несколько записей, таких как идентификатор пользователя, долгота, широта, время, расстояние, скорость, продолжительность, калории, погода и другие параметры. Например, недавнее исследование Endomondo показало, что за одну часовую тренировку в среднем генерируется 170 GPS-записей, а среднее количество записей, генерируемых в месяц, составляет от 2,8 до 6,3 млрд. [17]. С 30 миллионами пользователей число записей, генерируемых в секунду, может достигать 25 000 [17]. Этот огромный объем данных будет закачиваться в облако умным городом. Более того, не все полученные данные полезны. Например, датчики, развернутые в проекте Большого адронного коллайдера (LHC), генерируют около 500 Экзабайт данных в день. Однако 99,999% этих данных отфильтровываются [17]. Поставщики приложений, чтобы отфильтровать локально ненужные данные и обеспечить реакцию в реальном времени для пользователей, находящихся поблизости от точки доступа в сеть, могут использовать периферийные вычисления. Более того, т.к. данные будут отфильтрованы до отправки в облако, то сетевой трафик и нагрузка на обрабатывающие эти данные облачные серверы будет сокращена.

Другим хорошим примером, где технология иерархических периферийных вычислений позволяет успешно справляться с большими объемами трафика и временными ограничениями, являются такие потоковые приложения, как Facebook Live, YouTube Live и Livestream [28], позволяющие пользователям осуществлять прямую трансляцию. Сообщается [6] , [31], что в течение одной минуты пользователи YouTube загружают 72 часа нового видео, пользователи Facebook выкладывают в сеть 2 460 000 фрагментов контента, пользователи WhatsApp выкладывают 347 222 фотографии, пользователи Instagram публикуют 216 000 новых фотографий, а пользователи Vine – 8333 видеороликов. Обычно, когда видео или фото загружается, например, на Facebook или YouTube, для уменьшения размера изображения оно подвергается сжатию с потерями. Загрузка фотографий и видео с высоким разрешением непосредственно с пользовательских устройств в облако потребует значительную долю полосы пропускания канала и может занять много времени в тех областях, где используется Интернет-соединение плохого качества.

Подобные проблемы возникают в приложениях мониторинга состояния здоровья или в приложениях интеллектуального управления городом, где потоки данных с камер наблюдения и других датчиков необходимо загружать в облако. Технология иерархических периферийных вычислений может быть использована для переноса задач, связанных со сжатием данных перед загрузкой в облако, на периферийные мини- или микро-ЦОДы поблизости от конечных пользователей. Более того, в этих ЦОДах также может выполняться шифрование пользовательских данных вместо загрузки необработанных данных в облако, что обеспечивает безопасность и конфиденциальность пользовательских данных при транзитной передаче.

Итак, построение решений для «интернета вещей» в двухуровневой архитектуре с традиционным giant-like облачным ЦОДом на одном конце и устройствами для «интернета вещей» на другом не удовлетворяет требованиям по величине задержек, скорости перемещения терминала (мобильности) и точности определения местоположения терминала [19]. Как отмечалось ранее, удовлетворить перечисленные выше требования позволяет многоуровневая архитектура иерархических периферийных вычислений, показанная на рис. 4.

На первом уровне этой архитектуры находятся сенсоры, датчики сбора данных для приложений «интернета вещей», либо сами приложения, развернутые на соответствующих устройствах, которые являются устройствами конечного пользователя, например транспортным средством.

Вторая часть архитектуры – граничный ТУМАН, связанный с датчиками, конечными пользователями через маршрутизатор, точку доступа, сеть беспроводного доступа или базовую станцию LTE.

Третья часть архитектуры иерархических периферийных вычислений представляет собой сеть неоднородных ЦОДов: облака на периферии взаимодействующих сетей на основе сетей микро-ЦОДов и, наконец, традиционные большие облачные ЦОД.

Организация доступа на границе беспроводных сетей на основе технологий ПКС и ВСС (SDN NFV), равно как и построение опорной (core) части сотовых сетей 5G на основе виртуальных сетевых функций, пока также требует проработки и исследования.

Рис. 4. Архитектура иерархических периферийных вычислений

Fig. 4. Architecture of Hierarchical Peripheral Computing

Заключение

Сегодня технология иерархических периферийных вычислений только зарождается. Нет общепринятой архитектуры, устоявшейся системы понятий, стандартов и протоколов. На рисунке 5 показаны типичные приложения, для которых необходима архитектура иерархических периферийных вычислений. Там же перечислены потенциальные возможности этой архитектуры, которые были обсуждены выше. Благодаря им, эффективность наших сетей может быть значительно повышена. Однако для того, чтобы сделать этот подход осуществимым, необходимо решить много проблем, провести исследования. Некоторые из этих проблем перечислены здесь. Этот список не полон, те, что перечислены, не охватывают их всех.

Безопасность и приватность. Говоря о безопасности иерархических периферийных вычислений, мы должны понимать, что речь идет о безопасности в неоднородной облачной среде, безопасности в контуре управления и в контуре передачи данных ПКС сетей, безопасности, связанной с использованием предложенными выше расширениями протокола BGP – EBGP, уязвимостях, которые могут привнести виртуализованные сервисы от независимых поставщиков [36]. В некоторой степени проблемы безопасности в контуре управления ПКС сравнительно проще из-за централизованного характера и логически единого «органа» управления сетью – контроллера. Важными проблемами являются безопасность как самого ПКС контроллера, так и его приложений, изоляция приложений от взаимного влияния как в контуре управления, так и в контуре передачи данных, выявление скомпрометированных коммутаторов в контуре передачи данных ПКС сетей. Скомпрометированные коммутаторы являются важной угрозой, поскольку они могут быть использованы для разнообразных сетевых атак.

Рис. 5. Приложение, возможности и проблемы иерархических периферийных вычислений

Fig. 5. Application, possibilities and problems of Hierarchical Peripheral Computing

Одним из механизмов, который может быть использован для обнаружения скомпрометированных коммутаторов в ПКС сетях, является прогнозирование значения счетчиков правил коммутации пакетов в коммутаторе. Поскольку контроллер обладает полной информацией об использовании правил коммутации и обработки пакетов в коммутаторе, равно как и о самих правилах, определенных в сети, то можно обнаружить аномальное поведение в сети, предсказав ожидаемые значения счетчиков и сравнив их с реальными значениями счетчиков, полученными контроллером от коммутаторов [34, 35]. На периферии сети миграция сервисов из одной сети в другую, развернутых разными операторами сервисов, может приводить к возникновению уязвимостей, которые пока слабо изучены.

Необходимо также тщательно изучить проблемы обеспечения приватности (privacy), связанные с передачей данных от пользователя в сеть и передачей их из периферии одной сети на периферию другой. Дело в том, что устройства пользователей могут быть недостаточно мощными для надежного шифрования. В таком случае эту задачу могу решить устройства на периферии ближайшей сети. Однако соединение между устройством конечного пользователя и микро-/мини-ЦОДом на периферии ближайшей сети становится удобным местом для организации «атаки чужой в середине». Кроме того, шифрование увеличивает задержку на передачу данных до места их обработки. Однако здесь важнее то, что большинство конечных пользователей обычно не задумываются о приватности и безопасности своих данных. Необходимы автоматические механизмы, обеспечивающие приватность пользовательских данных во время работы. Например, в опросе [6] выяснилось, что 80% из 439 миллионов пользователей Wi-Fi сетями используют в своих беспроводных маршрутизаторах пароли, установленные по умолчанию, а 49% пользовательских

сетей не защищены. Кроме того, 89% общедоступных Wi-Fi точек доступа являются незащищенными. В [9] сообщается, что к 2020 году 10% всех атак будут нацелены на системы «интернета вещей». Также важно сохранять приватность данных пользователя и изолировать их от других данных, собранных сторонними приложениями. Так, данные от приложения, отслеживающего физическую активность пользователя, не должны смешиваться, а само приложение иметь доступ к данным о других видах деятельности пользователя или, например, данным о состоянии электроприборов в его умном доме [7]. Поддержка механизмов контроля доступа к данным разных пользователей на периферии разных сетей – это еще одна проблема обеспечения приватности данных.

Стандартизация архитектуры и протоколов. ПКС – довольно новая концепция сетевой архитектуры. Для ее широкого применения требуется стандартизация. Эти стандарты необходимы для проектирования, интеграции, эксплуатации и технического обслуживания таких сетей. Необходимо разработать политику для взаимодействия доменов ПКС, находящихся под управлением разных контроллеров, друг с другом и с другими, например, унаследованными (legacy) доменами традиционных сетей, а также с виртуализованными сервисами в ПКС сети ЦОДа (там может быть свой контроллер). В контуре управления ПКС сетью используются три интерфейса или API: 1) южный интерфейс, 2) северный интерфейс и 3) интерфейс восток-запад. Интерфейс восток-запад определяет, как контроллеры ПКС доменов взаимодействуют друг с другом для обмена информацией в процессе управления. По сути, мы можем думать об этом интерфейсе как о канале, проходящем через разные ПКС домены, для связи с их контурами управления. Организация интерфейса восток-запад оказывает прямое влияние на отказоустойчивость системы управления ПКС и доступность сервисов, реализуемых приложениями ПКС контроллера. Здесь широкое поле для исследований [36]. В целях масштабируемости, доступности и надежности сервисов в контуре управления сетью необходимо обеспечить консистентность данных в физически распределенном контуре управления, но который является логически централизованным. Это необходимо для того, чтобы, прежде всего, гарантировать, что резервный контроллер сможет корректно принять на себя управление сетью в случае отказа активного контроллера. Кстати, архитектура распределенного контура управления до сих пор остается одной из ключевых проблем в ПКС сетях. Проблемы, возникающие в физически распределенном контуре управления, включают в себя задержки на передачу сообщений в контуре управления, поддельные управляющие сообщения, несогласованные обновления и изменения маршрутизации в сети, в случае, когда пакеты все еще находятся в пути. Например, при создании избыточности для обеспечения отказоустойчивости контура управления сообщения OpenFlow, вызванные задержками в сети, могут привести к неправильным обновлениям информации в контуре передачи данных. Так что здесь есть над чем поработать.

Архитектура SDX и протокол EBGP. Организация точек обмена трафиком между автономными системами Интернета (Internet eXchange Point – IXP) в виде ПКС сети дает много преимуществ по сравнению с традиционными подходами. Прежде всего надо осознавать, что управление такой точкой обмена должно быть частью распределенного контура управления сетью. Такая организация программно-конфигурируемой точки обмена (SDX) позволит автономным системам

(АС) задавать свои политики маршрутизации в виде приложений централизованного ПКС контроллера сети точки обмена (Software Defined eXchange point – SDX), который становится арбитром глобальной политики для нескольких доменов и автономных систем. Обладая таким глобальным видением, контроллер SDX может внедрять методы масштабирования, которые могут позволить объединить большое количество политик разных АС. В результате SDX поможет разрешать проблемы междоменной маршрутизации, с которыми долгое время сталкиваются существующие точки обмена, внедряя новые политики, которые обрабатывают пакеты на более детальном уровне, при сохранении поддержки постоянной рассылки объявлений BGP маршрутов. Преимуществами SDX являются предотвращение нарушений политик обмена между доменами-участниками обмена при возникновении DDoS-атак; оптимизация маршрутов для обеспечения быстрой сходимости в сети, разгрузки каналов передачи данных, анализ трафика на middlebox'ах, инженерия трафика в контуре передачи данных; применение пиринга в зависимости от используемых приложений; дистанционное управление выбором маршрута BGP и балансировкой нагрузки в WAN сетях при передаче разнородного трафика, чувствительного к производительности.

Еще одна проблема, связанная с SDX и мультидоменными ПКС сетями, – это обеспечение соблюдения определенной политики в разных доменах. Поскольку ПКС контроллер в одном домене не может определять и контролировать политики в других доменах, сетевые операторы не могут обеспечить, чтобы их собственные политики применялись в доменах, внешних по отношению к их собственным. Поэтому проблемы проверки политики в SDX и мультидоменных ПКС сетях предоставляют обширную область для исследований, актуальную для операторов связи. Как уже было отмечено, при рассмотрении иерархических периферийных вычислений необходимо, чтобы протокол, который мы назвали Extended BGP (EBGP), разрешал АС объявлять (публиковать) другим АС, какие сервисы доступны в этой АС и в соответствии с каким SLA. АС-заявитель должен предоставлять для каждого анонсируемого сервиса как минимум следующую информацию: политику расчета стоимости оплаты, SLA, QoS, гарантированную задержку доступа, минимальную ожидаемую производительность, через какой шлюз АС-заявителя этот сервис доступен. Здесь термин «сервис» можно толковать широко: это предложение некоторых ресурсов, инфраструктуры как услуги и различных виртуализированных сервисов. Ключевым моментом является то, что АС-заявитель гарантирует условия SLA, такие как: доступность услуги, задержка доступа, джиттер, производительность. Кроме того, для поддержки гибкого сквозного управления и согласования ресурсов и сервисов в средах с несколькими АС необходима общая точка для обмена информацией о доступных ресурсах. Принципы организации такой точки и ее функционирование – самостоятельная проблема в рамках проблематики организации SDX.

Виртуализированные сервисы: интероперабельность и управление. Существующие подходы к организации управления и поддержки (MANO) сервисов, определенные ETSI, не предназначены для использования в мульти-доменных средах. Однако важным требованием гетерогенной среды иерархических периферийных вычислений является обеспечение совместимости между виртуализированными сервисами разных поставщиков при их соединении в цепочки на периферии сетевых доменов разных операторов. Объединение виртуализированных сервисов, раз-

мещенных на периферии разных сетевых доменов и от разных поставщиков, в единый сервис невозможно без четко определенных интерфейсов, в первую очередь потому, что нет общепринятой модели данных для реализации дескрипторов сервисов. Возможными решениями могут быть концепции SOA [32] и TOSCA [29]. Это еще одна проблема, требующая дополнительных исследований. Другой важной проблемой виртуализации сервисов в распределенной мультидоменной среде является поддержка функций конфигурирования, определения работоспособности, производительности, билинга и безопасности. Например, вопросы билинга и управления учетом потребления ресурсов сети и выставления счетов по-прежнему полностью игнорируется почти во всех системах, в то же время средства и методы управления безопасностью и мониторинг производительности до сих пор имеют существенные ограничения [36]. Вообще говоря, для распределенной мультидоменной среды все еще нет системы управления жизненным циклом виртуализированного сервиса.

Оркестровка ресурсов, их инициализация и мониторинг. Поскольку серверы, включая их ограниченные объемы основной памяти, вычислительные ресурсы и ресурсы внешней памяти, могут быть распределены по периферия разных доменов, а пропускная способность междоменной связи также ограничена, то управление этими ресурсами должно отличаться динамизмом, масштабируемостью и быть автоматизированным, чтобы добиться нужного экономического эффекта. Здесь можно выделить три проблемы. Это а) неопределенность задержки от и до «точки присутствия» (PoP) виртуализированного сервиса, б) управление размещением сервисов и в) динамическое управление ресурсами. Централизованный подход к управлению виртуализированными сервисами и их оркестровка, предлагаемые ETSI, налагают ограничения на масштабируемость, что особенно проблематично для услуг в мультидоменных средах из-за накладных расходов на передачу данных и задержек процессов на обработку. В результате возможные направления исследования в этом направлении включают в себя разработку эффективных механизмов мониторинга, которые лучше реагируют на динамику запросов и изменяющиеся требования в обслуживании, лучше учитывают задержки на распространение информации об изменениях в конфигурациях ресурсов, а также предоставляют информацию, необходимую для динамических изменений конфигураций, распределенных объектов. Здесь, по-видимому, нужны будут легкие коммуникационные протоколы для оптимизации использования ресурсов и повышения производительности услуг. SDN и NFV очень хорошо дополняют друг друга, при этом динамизм сетей и сервисов, их изменчивость должны быть хорошо наблюдаемы и хорошо управляемы. Следовательно, кроме традиционного управления виртуализированными вычислительными ресурсами и сервисами, должны быть разработаны дополнительные подходы к управлению и мониторингу в случае совместного использования SDN&NFV. Таким образом, управленческие решения, которые служат для объединения SDN и NFV, являются ключевыми областями исследований.

Управление качеством обслуживания (QoS) и отказоустойчивость. Поддержание необходимых уровней качества обслуживания и отказоустойчивости является важной проблемой. Иерархические периферийные вычисления в первую очередь предназначены для приложений реального времени, поэтому отказоустойчивость должна быть проактивной и должно быть реализовано автоматическое восстановление работы контура управления и контура передачи данных после сбоев.

Устройства на периферии не должны быть перегружены, чтобы поддерживать минимально необходимый уровень качества обслуживания. Следовательно, должен быть реализован надлежащий механизм мониторинга, который контролирует использование периферийных узлов в пиковые часы, тем самым облегчая гибкое распределение и планирование задач. Еще одна проблема в обеспечении качества обслуживания при периферийных вычислениях заключается в том, что в совместной работе участвуют объекты из периферий нескольких, разных сетевых доменов. Например, такой сценарий может возникать в случае сети доставки контента или когда пользователь перемещается из зоны периферии одного домена в периферийную зону другого. В этом случае пользовательские данные должны быть доступны в обеих периферийных зонах. Решением этой проблемы является совместное кэширование пользовательских данных на периферии взаимодействующих сетей. Однако это вызывает рост трафика между взаимодействующими сетями. Следовательно, должны быть разработаны оптимальные стратегии размещения данных и их репликаций, которые уменьшают задержки и собственно трафик до минимально допустимых пределов, определяемых требуемым качеством обслуживания. Сложной проблемой, связанной с качеством обслуживания, является поддержание пропускной способности сети на требуемом уровне [37].

Фильтрация контекста данных. Фильтрация контекста данных – это предварительная обработка данных на периферии сети, предшествующая их дальнейшей передаче. Выше мы приводили несколько примеров, когда устройства для «интернета вещей» или пользовательские устройства в научных экспериментах генерируют огромное количество данных. Прокачка этого объема данных через взаимодействующие сети может привести к перегрузкам в них и перегрузкам ЦОДов на периферии взаимодействующих сетей. Однако фильтрация данных вызывает несколько проблем. Если будет отфильтровано слишком много данных, это может привести к потере некоторой полезной информации, что приведет к снижению точности данных. Если данные будут отфильтрованы слабо, нежелательные данные также могут быть отправлены дальше, вызывая дополнительную нагрузку на ресурсы взаимодействующих сетей.

Список литературы / References

[1] Bilal K., et al., “Trends and challenges in cloud datacenters”, *IEEE Cloud Computing*, **1**:1 (2014), 10–20.

[2] Bilal K., et al., “A taxonomy and survey on Green Data Center Networks”, *Future Generation Computer Systems*, **36** (2014), 189–208.

[3] Bilal K., Khan S.U., Zomaya A.Y., “Green Data Center Networks: Challenges and Opportunities”, *IEEE Conference on Frontiers of Information Technology*, IEEE, 2013, 229–234.

[4] Bilal K., et al., “A survey on green communications using adaptive link rate”, *Cluster Computing*, **16**:3 (2013), 575–589.

[5] Chen Zhuo, et al., “Early implementation experience with wearable cognitive assistance applications”, *Proceedings of the 2015 workshop on Wearable Systems and Applications*, ACM, 2015, 33–38.

[6] Shi Weisong, et al., “Edge computing: Vision and challenges”, *IEEE Internet of Things Journal*, **3**:5 (2016), 637–646.

[7] "Cisco Global Cloud Index: Forecast and Methodology, 2016-2021", https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

[8] "4 Vs of Big Data", http://www.ibmbigdatahub.com/sites/default/files/infographic_file/4-Vs-of-big-data.jpg.

[9] Bonorni F., et al., "Fog computing and its role in the internet of things", Proceedings of the first edition of the MCC workshop T Mobile cloud computing (Helsinki, Finland), 2012, 13–16.

[10] Brown B., "Microsoft researcher: Why Micro Datacenters really matter to mobile's future", http://www.networkworld.com/article/2979570/cloud-computing/microsoft-researcher-why-micro-datacenters-really-matter-to-mobiles-future.html.

[11] Satyanarayanan M., et al., "The Case for VM-Based Cloudlets in Mobile Computing", *IEEE Pervasive Computing*, **8**:4 (2009), 14 – 23.

[12] Aazam M., Huh E., "Dynamic resource provisioning through fog micro datacenter", *The 12th IEEE International Workshop on Managing Ubiquitous Communications and Services*, 2015, 105–110.

[13] Aazam M., Huh E., "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT", *The 29th IEEE International Conference on Advanced Information Networking and Applications (AINA-15)*, IEEE, 2015, 687–694.

[14] Jararweh Y., et al., "The future of mobile cloud computing: Integrating cloudlets and Mobile Edge Computing", *2016 23rd International Conference on Telecommunications (ICT)*, IEEE, 2016.

[15] "Green Clouds", http://www.greenclouds.in/views-and-resources/high-performance-websites/.

[16] Giordano A., Spezzano G., Vinci A., "Smart agents and fog computing for smart city applications", *International Conference on Smart Cities*, Springer, 2016, 137–146.

[17] Cortes R., et al., "Stream processing of healthcare sensor data: studying user traces to identify challenges from a big data perspective", *Procedia Computer Science*, **52** (2015), 1004–1009.

[18] Costenaro D., Duer A., "The megawatts behind your megabytes: going from data-center to desktop", *2012 ACEEE Summer Study on Energy Efficiency in Buildings*, 2012.

[19] "Cisco Data in Motion", https://www.cisco.com/c/m/en_us/solutions/data-center-virtualization/data-motion.html.

[20] "MicroDC Solution", https://actfornet.com/HUAWEI_CLOUD_COMPUTING/Huawei%20MicroDC%20Brochure.pdf.

[21] "Mobile Edge Computing", http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing.

[22] "Akamai", https://blogs.akamai.com/2016/07/portugal-france-sets-live-sports\discretionary{-}{}{}streaming-record-on-akamai.html.

[23] Wang Meisong, et al., "An overview of cloud based content delivery networks: research dimensions and state-of-the-art", *Transactions on Large-Scale Data-and Knowledge-Centered Systems XX*, Springer, 2015, 131–158.

[24] Chu Weibo, et al., "Network delay guarantee for differentiated services in content-centric networking", *Computer Communications*, **76** (2016), 54–66.

[25] Wang Rui, et al., "Mobility-aware caching for content-centric wireless networks: Modeling and methodology", *IEEE Communications Magazine*, **54**:8 (2016), 77–83.

[26] Ahmed Syed Hassan, Bouk Safdar Hussain, Kim Dongkyun, *Content-Centric Networks: An Overview, Applications and Research Challenges*, Springer, 2016, 108 pp.

[27] Gao Ying, et al., *Are cloudlets necessary?*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA, 2015, Tech. Rep. CMU-CS-15-139.

[28] Bilal K., et al., "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers", *Computer Networks*, **130** (2018), 94–120.

[29] "TOSCA Simple Profile in YAML Version 1.2", http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/csprd01/TOSCA-Simple-Profile-YAML-v1.2-csprd01.pdf.

[30] Nygren E., Sitaraman R.K., Sun J., "The akamai network: a platform for high-performance internet applications", *ACM SIGOPS Operating Systems Review*, **44**:3 (2010), 2–19.

[31] "Data Never Sleeps 2.0", https://www.domo.com/learn/data-never-sleeps-2.

[32] Erl T., *Service-oriented architecture: concepts, technology, and design*, 2005.

[33] Antonenko V., et al., "C2: General Purpose Cloud Platform with NFV Life-Cycle Management", *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2017, 353–356.

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

[35] Petrov I., Morgunova O., "Forwarding Rule Minimization for Network Statistics Analysis in SDN", *2018 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTec)*, IEEE, 2018, 1–6.

[36] Cox J.H., et al., "Advancing software-defined networks: A survey", *IEEE Access*, **5** (2017), 25487–25526.

[37] Chemeritskiy E., Stepanov E., Smeliansky R., "Managing network resources with flow (de) multiplexing protocol", *Mathematical and Computational Methods in Electrical Engineering*, Recent Advances in Electrical Engineering Series, **53**, 2015, 35–43.

Smeliansky R. L., "Hierarchical Edge Computing", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 146–169.

DOI: 10.18255/1818-1015-2019-1-146-169

Abstract. The computing paradigm based on the giant-like DC is replaced by a new paradigm. The urgency of this shift is caused by the requirements of new applications that actively use video, real-time interactivity, new mobile communication technologies, which today cannot be implemented without the usage of cloud computing and virtualization based on SDN&NFV technologies. The presentation considers the requirements dictated by these applications, outlines the architecture of this new paradigm which we call Hierarchical Edge Computing (HEC). Attention is focused on the fact that all these applications are distributed, become more and more real-time applications and require guaranteed quality of service in the networking operation. The main scientific problems that need to be solved for implementing this new paradigm are discussed.

Keywords: SDN, NFV, fog computing, cloud computing, data center, mobile edge computing

On the authors:
Ruslan L. Smeliansky, Corresponding Member of Russian Academy of Sciences, professor, doctor of sciences,
orcid.org/0000-0003-2311-4513
Lomonosov Moscow State University,
1-bd. 52 Leninskie gory, Moscow, 119992, Russia, e-mail: smel@cs.msu.ru

Acknowledgments:
This work was supported by the Russian Fund of Basic Research, Grant N 18-07-01245.

Моделирование и анализ информационных систем. Т. 26, №1 (2019), с. 170–190
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 170–190

DOI: 10.18255/1818-1015-2019-1-170-190

УДК 004.7

Анализ эффективности демультиплексирования транспортных потоков

Степанов Е. П.

Поступила в редакцию 10 января 2019
После доработки 12 февраля 2019
Принята к публикации 14 февраля 2019

Аннотация. Известно, что разделение отдельного транспортного потока на несколько независимых транспортных подпотоков может повысить скорость этого потока. Справедливость этого утверждения, верная для одного потока, не очевидна для *массового* случая, когда демультиплексированию (разделению на подпотоки) подвергают все транспортные потоки в сети одного ISP оператора. Возникает вопрос, какое влияние окажет массовое демультиплексирование транспортных потоков на пропускную способность сети ISP оператора. В статье этот вопрос рассмотрен для статического случая, когда каждый поток разделяется статически, т.е. перед его запуском, на одинаковое число подпотоков. Была предложена математическая модель, на основе которой построена имитационная модель с целью получения более точных оценок производительности сети с демультиплексированием потоков и без него. С помощью имитационной модели определены свойства сети, при которых применение демультиплексирования транспортных потоков оправдано. Корректность полученных результатов обосновывается при помощи эмуляции сети и нагрузки в ней на основе виртуализации стека протоколов при тех же входных данных. В статье рассмотрены разные политики маршрутизации, которые могут быть использованы при массовом демультиплексировании. Особое внимание уделяется алгоритмам, позволяющим строить маршруты с минимальными пересечениями, так как использование неоптимальных по длине, но непересекающихся маршрутов может повысить производительность сети. Маршруты, построенные при помощи этих алгоритмов, использовались как для анализа производительности сети на предложенной имитационной модели с демультиплексированными потоками, так и в случае балансировки недемультиплексированных потоков.

Ключевые слова: многопоточная маршрутизация, качество сервиса, MPTCP

Для цитирования: Степанов Е. П., "Анализ эффективности демультиплексирования транспортных потоков", *Моделирование и анализ информационных систем*, **26**:1 (2019), 170–190.

Об авторах:
Степанов Евгений Павлович, orcid.org/0000-0003-3649-0745, аспирант,
Московский государственный университет им. М.В. Ломоносова,
Ленинские горы, 1, стр. 52, г. Москва, 119991 Россия, e-mail: estepanov@lvk.cs.msu.su

Благодарности:
Работа выполнена при частичной поддержке РФФИ, грант № 18-07-01255.

Введение

В настоящее время продолжают активно исследоваться методы ускорения передачи данных на транспортном уровне (транспортных потоков). Добиться ускорения транспортных потоков можно разными способами. Например, при помощи настройки алгоритма управления перегрузкой, который регулирует интенсивность отправки пакетов в сеть. В случае неоднородности среды передачи данных (например, проводной и беспроводной сегменты) можно поделить маршрут потока на фрагменты с однородной средой передачи данных. Тогда на каждом из фрагментов можно использовать индивидуальный алгоритм управления перегрузкой, адаптированный под качество сервиса, достижимого в среде этого фрагмента. Так устроена версия транспортного протокола Split TCP [1]. В этой статье основное внимание уделено методу ускорения транспортных потоков, основанного на их демультиплексировании.

Идея *демультиплексирования потока* заключается в распределении пакетов одного и того же транспортного потока между несколькими транспортными *подпотоками*, у каждого из которых свой маршрут. Два и более подпотоков будем называть *родственными*, если они образованы пакетами одного и того же транспортного потока, который будем называть *родительским*. В настоящей статье техника демультиплексирования рассматривается применительно к сети Интернет-провайдеров (ISP), и исследования проводятся в предположении, что все родственные подпотоки проходят через сеть одного и того же Интернет-провайдера.

В топологии сети ISP всегда присутствует определенная избыточность, которая позволяет соединить любых двух абонентов несколькими маршрутами. Такая избыточность объективно необходима в сети для устойчивости к отказам сетевого оборудования и может быть использована для демультиплексирования пакетов одного и того же транспортного потока между несколькими маршрутами, соединяющими одну и ту же пару (отправитель, получатель).

Демультиплексировать транспортный поток можно несколькими способами. Например, для одной и той же пары (отправитель, получатель) можно заранее проложить несколько маршрутов и использовать балансировщик нагрузки (например, ECMP [2]) для распределения пакетов транспортного потока между построенными маршрутами.

Такой подход, во-первых, может вызвать нарушение порядка поступления пакетов к получателю. Из-за разной задержки пакетов на разных маршрутах ранее посланный пакет может дойти до получателя позже пакета, посланного позднее первого. Нарушение порядка доставки пакетов может быть воспринято алгоритмом управления перегрузкой на отправителе как признак потери пакетов, вследствие чего будет снижена скорость отправки пакетов в сеть. Как следствие, отправитель будет снижать скорость отправки пакетов в сеть и дублировать пакеты.

Во-вторых, в силу одного из основных принципов архитектуры Интернета – принципа “End-To-End” [3] не желательно вмешиваться в работу потока внутри сети, это следует делать на концах соединения. Поэтому балансировку пакетов между родственными подпотоками стоит делать на стороне отправителя. При этом интенсивность передачи пакетов каждого родственного подпотока должен регулировать свой экземпляр алгоритма управления перегрузкой. Благодаря этому автоматиче-

ски будет корректироваться количество пакетов в каждом родственном подпотоке пропорционально текущей пропускной способности каждого из их маршрутов.

В-третьих, балансировщики в своей работе опираются на эвристические предсказания. В случае неправильного предсказания потоки-“слоны” могут разделять один и тот же маршрут, в то время как на других маршрутах будет всего лишь несколько потоков-“мышей” с маленьким объемом передаваемых данных [4]. Как будет показано ниже, есть подходы, основанные на оперативной оценке реальных характеристик маршрутов (назовем такой подход демультиплексорами), что выгодно отличает его от сетевых балансировщиков.

Еще одним существенным недостатком методов демультиплексирования является то, что при условии массового демультиплексирования потоков увеличивается число транспортных потоков (подпотоков) в сети, вследствие чего растет конкуренция за сетевые ресурсы между ними. Здесь важно понимать, что маршруты подпотоков могут пересекаться как у родственных подпотоков, так и у подпотоков разных родительских потоков. Если алгоритм управления перегрузкой какого-то родственного подпотока вызовет временную перегрузку на каком-то участке сети, то она может оказать влияние на большее количество родительских потоков, чем в случае отсутствия демультиплексирования. Поэтому здесь мы исследуем границы применимости массового демультиплексирования потоков, т.е. определение тех условий, при которых массовое демультиплексирование будет давать преимущество.

Основной вклад этой работы в рассматриваемую проблематику состоит в следующем:

- Построена математическая модель, позволяющая оценить эффективность применения массового статического демультиплексирования в конкретной сети.
- С помощью этой модели (см. разделы 1.4. и 2.3.) определены свойства сети, при которых применение демультиплексирования транспортных потоков оправдано.

1. Массовое статическое демультиплексирование в сетях Интернет-провайдеров

1.1. Демультиплексирование потока: современное состояние

Сегодня есть много реализаций демультиплексирования транспортных потоков. Например, широкую популярность получил *Multi-Path TCP (MPTCP)* [5]. Важным достоинством MPTCP явилась его совместимость со стандартными интерфейсами TCP/IP стека как на сетевом, так и на прикладном уровнях [6]. По существу, MPTCP вводит дополнительный демультиплексирующий промежуточный слой, API которого полностью совместим со стандартным интерфейсом TCP сокета и чьи подпотоки существуют в сети как обычные TCP сессии. Этот промежуточный слой послужил основой для тестирования десятка различных идей по демультиплексированию потока [7].

Все подходы к демультиплексированию транспортных потоков можно разделить на статические и динамические. MPTCP является примером статического подхода.

Во всех известных реализациях MPTCP среди всех родственных подпотоков выделяется стартовый, который начинает передачу данных первым. Порождение остальных родственных подпотоков происходит только после того, как получатель заявил о поддержке работы по протоколу MPTCP. В статьях [6,8] было выявлено несколько случаев, когда применение MPTCP оказалось полезным. В этих работах для каждого рассмотренного случая предложена стратегия выделения маршрутов для родственных подпотоков. В работе [6] предложена стратегия *fullmesh*, ориентированная на случай, когда у устройства отправителя есть несколько сетевых интерфейсов, например, если устройство оборудовано как проводным интерфейсом (Ethernet), так и беспроводным (WiFi) или GSM радиоинтерфейсом. Эта стратегия предполагает установление маршрута для каждой пары интерфейсов отправителя и получателя. Другая стратегия допускает существование нескольких маршрутов между одной и той же парой интерфейсов при условии, что у этих маршрутов нет общих "узких" мест. Именно такой случай рассмотрен в работе *Binder* [8]. Примером динамического подхода к демультиплексированию транспортных потоков является протокол *Flow DeMultiplexing Protocol (FDMP)* [9]. Этот протокол выделяет новые маршруты для родственных подпотоков только в случае снижения скорости родительского потока ниже заранее заданного предела. Как только этот предел достигнут, протокол старается закрыть некоторые родственные подпотоки и освободить маршруты. Далее в статье будем рассматривать только MPTCP со статическим количеством используемых подпотоков.

Пакеты родственных подпотоков в MPTCP могут различаться значениями следующих полей заголовка: IP-адреса источника, IP-адреса назначения, порта источника и порта назначения. Так, IP-адреса источника могут различаться, если родственные подпотоки маршрутизируются через разные сетевые интерфейсы, например, WiFi и Ethernet.

Реализация идеи демультиплексирования транспортного потока требует решения ряда сложных проблем, связанных с управлением перегрузкой. Прежде всего это проблема справедливого распределения сетевых ресурсов. Родительский поток благодаря использованию нескольких родственных подпотоков может получить больше сетевых ресурсов по сравнению с недемультиплексированным потоком. Независимое демультиплексирование нескольких разных родительских потоков может привести к тому, что один родительский поток получит сетевых ресурсов больше других родительских потоков. Такой случай распределения ресурса будем считать *несправедливым* [10]. В ряде работ предлагается поддерживать справедливое распределение, установив дополнительные ограничения на работу алгоритмов управления перегрузкой в MPTCP подпотоках. Например, существуют реализации известных алгоритмов управления перегрузкой, которые были адаптированы под MPTCP таким образом, чтобы не нарушать справедливое распределение ресурсов. Такие работы проводились для реализаций алгоритмов New Reno [11,12], Cubic [13], Compound [14], Fast [15] и многих других. Кроме того, ограничения на работу алгоритмов управления перегрузкой можно вводить не для всех родственных подпотоков, а только внутри множеств родственных подпотоков, которые обладают общим «узким» местом. Такие множества можно динамически составлять на основе анализа характеристик родственных подпотоков, как это делает Алгоритм Dynamic Window Coupling [16].

Итак, методы демультиплексирования уже можно активно применять в транспортных сетях. Однако большое количество способов выделения маршрутов говорит о том, что необходима тонкая настройка демультиплексоров в зависимости от параметров сети, где они будут использоваться. Для этого нужно уметь оценивать эффективность методов демультиплексирования в условиях конкретной сети. Об оценке эффективности методов демультиплексирования пойдет речь в следующих подразделах.

1.2. Модель функционирования сети

Для оценки методов демультиплексирования построим математическую модель функционирования сети. Представим сеть в виде неориентированного связного графа $G = (V, E)$, где V – пронумерованный набор вершин графа, а E – множество ребер, представленных парами (v_i, v_j) на V. E опишем в виде матрицы смежности $\Omega \in \mathbb{R}^{n \times n}$ с весами $\omega_{ij} = \omega(v_i, v_j)$. Если канала от v_i к v_j не существует, то $\omega_{ij} \equiv \omega_{ji} \equiv 0$. В опорной сети ISP оператора все каналы связи обычно обладают примерно одинаковой пропускной способностью, поэтому будем предполагать, что вес всех ребер в нашей модели одинаков и равен единице. Так как граф неориентированный, то $\omega_{ij} \equiv \omega_{ji}$ и матрица Ω является симметричной. Поскольку в силу сделанного выше предположения пропускные способности всех каналов считаются одинаковыми, то обозначим её значение как $\psi_{ch} (\psi_{ch} \in \mathbb{R})$.

Среди множества всех вершин V выделим подмножество вершин P, которые будем называть полюсами. На полюсах расположены отправители и получатели транспортных потоков. Для упрощения математической модели введем следующее ограничение для полюсов: каждая вершина, не являющаяся полюсом, может быть соединена не более чем с одним полюсом:

$$\forall v_i \in V \setminus P \colon (\exists! v_j \in P \colon \omega_{ij} > 0) \vee (\forall v_j \in P \colon \omega_{ij} = 0).$$

Таким образом, даже если в сети ISP к узлу агрегации трафика абонентов подключены сотни-тысячи пользователей, то в нашей математической модели они будут представлены одним полюсом, на котором могут терминироваться сотни-тысячи соединений соответственно. Отсюда вытекает ограничение на размер подмножества полюсов: $2 \leq |P| \leq \frac{|V|}{2}$. В математической модели сложно учесть все возможные варианты размещения полюсов. Поэтому оценивать методы демультиплексирования мы будем, варьируя соотношения полюсов к остальным вершинам $\frac{|P|}{|V \setminus P|}$ графа сети.

Пусть $F = f_1, ..., f_m$ – множество всех родительских потоков в сети, где $\forall i \colon f_i = ((v_s^i, v_r^i), \mu_i), v_s^i, v_r^i \in P, \mu_i \in \mathbb{R}, (v_s^i, v_r^i)$ задает пару отправителя и получателя, а μ_i – требование к минимальной пропускной способности, необходимой для потока. Поскольку наибольший интерес представляет случай максимальной нагрузки в сети, поэтому сделаем предположение, что все потоки в сети стартуют и заканчиваются одновременно. Пусть также задан коэффициент демультиплексирования $k \in \mathbb{N}$, который определяет число родственных подпотоков у каждого родительского потока. Отсутствие демультиплексирования соответствует случаю $k = 1$.

Пусть также задана функция выбора маршрута R. Аргументом функции R являются граф сети G, родительский поток f_i и коэффициент демультиплексирования

k. Тогда значением функции R является множество $\{\pi_{ij}\}$ из не более чем k маршрутов от отправителя до получателя для i-го потока. Под маршрутом понимается простоя цепь в графе сети, начинающаяся с полюса-отправителя и заканчивающаяся полюсом-получателем. При этом все маршруты, получаемые функцией R, не должны иметь пересечений (под пересечением маршрутов понимается пересечение по дугам графа). Если всевозможных непересекающихся маршрутов между отправителем и получателем только l, то функция R вернет множество из этих l маршрутов. Если всевозможных непересекающихся маршрутов больше k, то результатом будут случайно выбранные k маршрутов:

$$\pi_{ij} = \{v_1^j, v_2^j, ..., \quad v_{n_j}^j\}, \quad f_i \in F, \quad j = \overline{1,l}, \quad l < k, \quad v_1^j = v_s^i, \quad v_{n_j}^j = v_r^i,$$

$$\forall c \in \overline{1, n_j - 1} : \omega(v_c^j, v_{c+1}^j) = 1;$$

$$\forall i, j \in \overline{1, |F|} : \forall \pi_{iu} \in R(G, f_i, k) \wedge \pi_{jv} \in R(G, f_j, k) : \pi_{iu} \cap \pi_{jv} = \emptyset \vee i \equiv j \wedge v \equiv u.$$

Таким образом, набор $\langle V, \Omega, P, \psi_{ch}, F, R, k\rangle$ определяет набор входных параметров математической модели.

Эффективность применения методов демультиплексирования MPTCP с коэффициентом k (обозначим как $MPTCP(k)$) в математической модели будем оценивать отношением производительности сети с применением демультиплексирования T_D к производительности сети без применения демультиплексирования T_C:

$$MPTCP(k) = \frac{T_D}{T_C}.$$

Для определения понятия производительности сети введем дополнительные обозначения. Степенью η_i вершины $v_i \in V$ назовем количество трансцендентных ей ребер, т.е.

$$\eta_i = \sum_j \omega_{ij}. \tag{1}$$

Тогда максимальная степень вершин: $\eta_{max} = \max_i \eta_i$.

Максимально возможная длина маршрута в графе G в общем случае может быть V (на функцию R мы пока не накладываем каких-либо ограничений на выбор маршрутов, например выбор только оптимальных маршрутов, поэтому мы не можем оценить эту длину через диаметр графа). Количество всех маршрутов от вершины v_i до вершины v_j можем оценить методом индукции. В исходной вершине v_i у нас может быть η_{max} вариантов для второй вершины в маршруте. В m-й вершине каждого варианта маршрута может быть $\eta_{max} - 1 \leq \eta_{max}$ вариантов для выбора $(m+1)$-ой вершины в маршруте. В наихудшем случае надо выбрать $|V| - 2$ вершин из маршрута (так как первая и последняя вершины нам уже известны). Тогда количество всевозможных маршрутов от вершины v_i до вершины v_j (множество таких маршрутов обозначим как $ROUTES_{ALL(i,j)}$) можно оценить как

$$|ROUTES_{ALL(i,j)}| \leq \eta_{max}^{|V|-2}.$$

Количество всевозможных маршрутов в графе между полюсами (множество всевозможных маршрутов между полюсами обозначим как $ROUTES_{ALL}$) можно оценить как

$$|ROUTES_{ALL}| \leq C_{|P|}^2 * max_{v_i, v_j \in P} |ROUTES_{ALL(i,j)}| \leq C_{|P|}^2 * \eta_{max}^{|V|-2}.$$

Обычно под скоростью подпотока понимают минимум из скоростей подпотока на каналах маршрута этого подпотока (полученного при помощи функции R):

$$\lambda_{sf} = min_{\langle v_i, v_j\rangle \in \pi_{sf}} \lambda_{sf;(v_i,v_j)}, \quad \lambda_{sf;(v_i,v_j)} \in \mathbb{R}^+.$$

Однако в силу ранее сделанного предположения о том, что все маршруты, получаемые функцией R, не должны иметь пересечений, получаем

$$min_{\langle v_i, v_j\rangle \in \pi_{sf}} \lambda_{sf;(v_i,v_j)} = \psi_{ch}, \quad \forall i\colon \mu_i > \psi_{ch}.$$

Под скоростью родительского потока понимается сумма скоростей родственных подпотоков

$$\lambda_{f_i} = \lambda_{sf_1^i} + ... + \lambda_{sf_k^i} = k * \psi_{ch}.$$

В случае $k = 1$ подпоток совпадает с родительским потоком. С использованием введенного выше производительность сети для любого k определим как:

$$T_D = T_C = \sum_{f_i \in F} \lambda_{f_i}.$$

В сети в случае пересечения маршрутов пропускная способность общего для этих маршрутов канала обычно распределяется не стихийно, а в соответствии с определенным понятием справедливости. В математической модели будем использовать широко применяемое толкование понятия справедливости – *max-min справедливость*. Напомним, что множество скоростей всех подпотоков в сети $\Lambda = \{\lambda_{sf}\}$ является max-min справедливым для заданной сети, если скорость ни одного из потоков не может быть увеличена без ущемления скорости потоков с меньшей скоростью:

$$\forall \Lambda'\colon \lambda'_{sf} > \lambda_{sf} (\lambda'_{sf} \in \Lambda') \implies \exists sf''\colon \lambda_{sf''} < \lambda_{sf} \wedge \lambda'_{sf''} < \lambda_{sf''}.$$

Так как маршруты функции R не имеют пересечений, то никакие из двух подпотоков не разделяют один и тот же канал, поэтому распределение $\Lambda = \{\lambda_{sf_i} = \psi_{ch}\}$ удовлетворяет понятию max-min справедливости. Производительность сети можно оценить в этом случае как:

$$T_D = \sum_{f_i \in F} \lambda_{f_i} = min(|F| * k, C^2_{|P|} * \eta^{|V|-2}_{max}) * \psi_{ch};$$

$$T_C = \sum_{f_i \in F} \lambda_{f_i} = min(|F|, C^2_{|P|} * \eta^{|V|-2}_{max}) * \psi_{ch}.$$

Минимум необходимо брать, так как максимальное количество маршрутов может оказаться меньше, чем количество родительских потоков или родственных подпотоков.

Таким образом, становится возможным оценить эффективность методов демультиплексирования $MPTCP(k)$ как отношение производительности сети с демультиплексированием к производительности сети без демультиплексирования:

$$MPTCP(k) = \frac{T_D}{T_C} = \frac{min(|F| * k, C^2_{|P|} * \eta^{|V|-2}_{max}) * \psi_{ch}}{min(|F|, C^2_{|P|} * \eta^{|V|-2}_{max}) * \psi_{ch}}.$$

Однако надо понимать, что получившаяся оценка является достаточно грубой. Грубость этой оценки следует из того, что оценка количества всех маршрутов дана для наихудшего случая, а также из-за предположения, что маршруты не пересекаются. В силу общности математической модели она не учитывает особенности каждого конкретного маршрута и топологию графа сети. Поэтому далее на основе математической модели будет предложена имитационная модель, которая будет учитывать то, от чего мы абстрагировались в математической модели, например политику маршрутизации и топологию сети.

Даже для грубой полученной оценки эффективности методов демультиплексирования можно сделать несколько выводов об области применения демультиплексирования. Применение демультиплексирования будет неоправданным, если количество потоков превышает максимальное число маршрутов между полюсами, тем самым задействуя все имеющиеся ресурсы сети, и при этом потоки максимально используют эти ресурсы:

$$(|F| > C^2_{|P|} * \eta^{|V|-2}_{max}) \wedge (\forall f_i \colon \mu_i > \psi_{ch}).$$

При соблюдении описанных в математической модели ограничений и при условии

$$(|F| * k \ll C^2_{|P|} * \eta^{|V|-2}_{max}) \wedge (\forall f_i \colon \mu_i > k * \psi_{ch}),$$

эффективность демультиплексирования будет равняться $\frac{T_D}{T_C} = k$, что является весомым аргументом в пользу демультиплексирования.

1.3. Оценка эффективности массового статического демультиплексирования

Для того чтобы получить более точную оценку эффективности применения метода массового демультиплексирования, построим на основе математической модели имитационную модель. Вначале снимем ограничение на непересекаемость маршрутов, получаемых функцией R. Функция R будет задаваться алгоритмом выбора маршрута, который может основываться на политике маршрутизации в сети (кратчайший маршрут в OSPF), а также на методах балансировки в сети (один из кратчайших маршрутов равной стоимости в ECMP). Так как маршруты теперь могут пересекаться, мы не можем считать, что скорость любого подпотока одинакова и равна ψ_{ch}. Однако на распределение пропускной способности согласно нашей математической модели наложено условие max-min справедливости. Благодаря этому условию мы можем рассчитать распределение пропускной способности согласно алгоритму постепенного заполнения [17]. Таким образом, для расчета производительности сети в имитационной модели необходимо проделать следующие действия:

1. Рассчитать маршруты для каждого потока в сети в случае применения демультиплексирования (обозначим как $ROUTES_D$) и без него (обозначим как $ROUTES_C$):

$$ROUTES_D = \cup_{f_i \in F} R(G, f_i, k);$$

$$ROUTES_C = \cup_{f_i \in F} R(G, f_i, 1).$$

Также обозначим множество всех подпотоков всех потоков как SF_D ($|SF_D| = k * |F|$) и SF_C ($SF_C \equiv F$).

2. При помощи алгоритма постепенного заполнения [17] рассчитать распределение пропускной способности для значений входных параметров $\langle G, \Omega, P, SF_C, ROUTES_C\rangle$ и $\langle G, \Omega, P, SF_D, ROUTES_D\rangle$. Из распределения пропускной способности получим скорость λ_{sf} для каждого подпотока из SF_C и SF_D.

3. Далее согласно введенному ранее определению производительности сети, рассчитать T_C и T_D как сумму соответствующих скоростей подпотоков.

Для оценки эффективности метода статического демультиплексирования имитационная модель исследовалась на ряде входных данных, описанных ниже. Сравнение и анализ полученных результатов приведены в конце этого раздела. В следующем разделе обосновывается корректность полученных результатов при помощи эмуляции сети и нагрузки в ней на основе виртуализации стека протоколов при тех же входных данных.

Вначале опишем набор входных параметров для имитационной модели и множество их значений, для которого мы провели имитационные эксперименты. В качестве первого входного параметра выступает топология сети. По конкретной топологии можно построить множество вершин V и матрицу смежности Ω из математической модели. Пропускная способность каналов была положена равной 1 Гбит/с (для упрощения эмуляции сети). Варианты топологии сети для моделирования брались из библиотеки Topology Zoo [18]. Поскольку эта библиотека не предоставляет информацию о распределении потоков трафика, это распределение было синтезировано следующим образом. Для каждого запуска имитационной модели выбиралось случайное подмножество размера из узлов рассматриваемой топологии сети, к которым подключались дополнительные узлы – *полюса*, имитирующие отправителей и получателей потоков. В отличие от математической модели, полюса не входят в топологию сети, а являются дополнительными вершинами. Поэтому чтобы охватить разные шаблоны нагрузки, варьировалась доля полюсов к узлам топологии сети $\frac{|P|}{|V|}$, а также распределение полюсов по топологии. Еще одним параметром является $|F|$ – количество родительских потоков. Потоки распределялись равномерно по всем полюсам в сети. Требования потоков к пропускной способности возьмем больше, чем $k * 1$ Гбит/с, чтобы максимально задействовать ресурсы сети.

Имитационные эксперименты проводились в предположении, что каждый узел в сети поддерживает метод балансировки ECMP и распределяет входящие потоки между кратчайшими маршрутами до соответствующего получателя. Чтобы сымитировать такую политику маршрутизации, от каждого узла сети были построены кратчайшие маршруты ко всем другим узлам. Тогда для каждого узла можно определить множество альтернативных *следующих скачков* для балансировки, образованное соседями, через которые проходят маршруты до заданного получателя с минимальной стоимостью. В таком случае маршрут каждого потока определяется итерационно, начиная с полюса-отправителя. Для любого узла в маршруте следующий узел выбирается случайным образом из множества альтернативных следующих скачков для соответствующего получателя. Описанный выше метод построения

маршрута назовем *Случайным Следующим Скачком* (CCC) или Random Next Hop (RNH).

Итак, входными параметрами имитационной модели являются:

- Топология (из Topology Zoo) – соответствует V, Ω из математической модели;
- Доля полюсов к узлам топологии (10%, 20%, ..., 100%) – соответствует P из математической модели;
- Пропускная способность каналов (1 Гбит/с) – соответствует ψ_{ch} из математической модели;
- Количество родительских потоков (512, 1024, 2048, 4096) – соответствует $|F|$ из математической модели;
- Размещение отправителя и получателя для каждого потока – определяет F из математической модели;
- Политика маршрутизации (CCC) – соответствует R из математической модели;
- Количество родственных подпотоков у родительского (1, 2, 3) – соответствует k из математической модели.

Для сравнения производительности сети в случае массового статического демультиплексирования потоков с двумя и тремя подпотоками и в случае отсутствия демультиплексирования был проведен анализ скорости потоков для полумиллиона разных значений параметров имитационной модели. Влияние демультиплексирования на производительность сети в случае двух и трех подпотоков отображено на рисунках 1a и 1b соответственно. Каждая метка на графиках представляет подмножество значений входных параметров, для которых совпадает число узлов в топологии и значение утилизации сети. Под утилизацией сети понимается среднее значение утилизации всех каналов. Утилизация канала равна отношению суммы скоростей всех потоков, проходящих через этот канал, к пропускной способности канала и определена для случая недемультиплексированных потоков. Поскольку для разного количества потоков и разных вариантов размещения отправителей и получателей производительность сети может совпадать, то метка на графике может комбинировать результаты для более чем 5 тысяч разных значений входных параметров имитационной модели. Производительность сети для одной метки рассчитывается как среднее значение производительности сети для всех значений входных параметров, соответствующих этой метке. Светлые метки обозначают случаи, где производительность сети с демультиплексированными потоками выше, чем с недемультиплексированными – чем светлее цвет, тем выше преимущество демультиплексирования. Темные метки представляют случаи, где демультиплексирование проигрывает – чем темнее цвет, тем больше падает пропускная способность сети. Серые метки предполагают, что производительность сети для демультиплексированных потоков отличается от производительности сети для недемультиплексированных потоков меньше, чем на один процент.

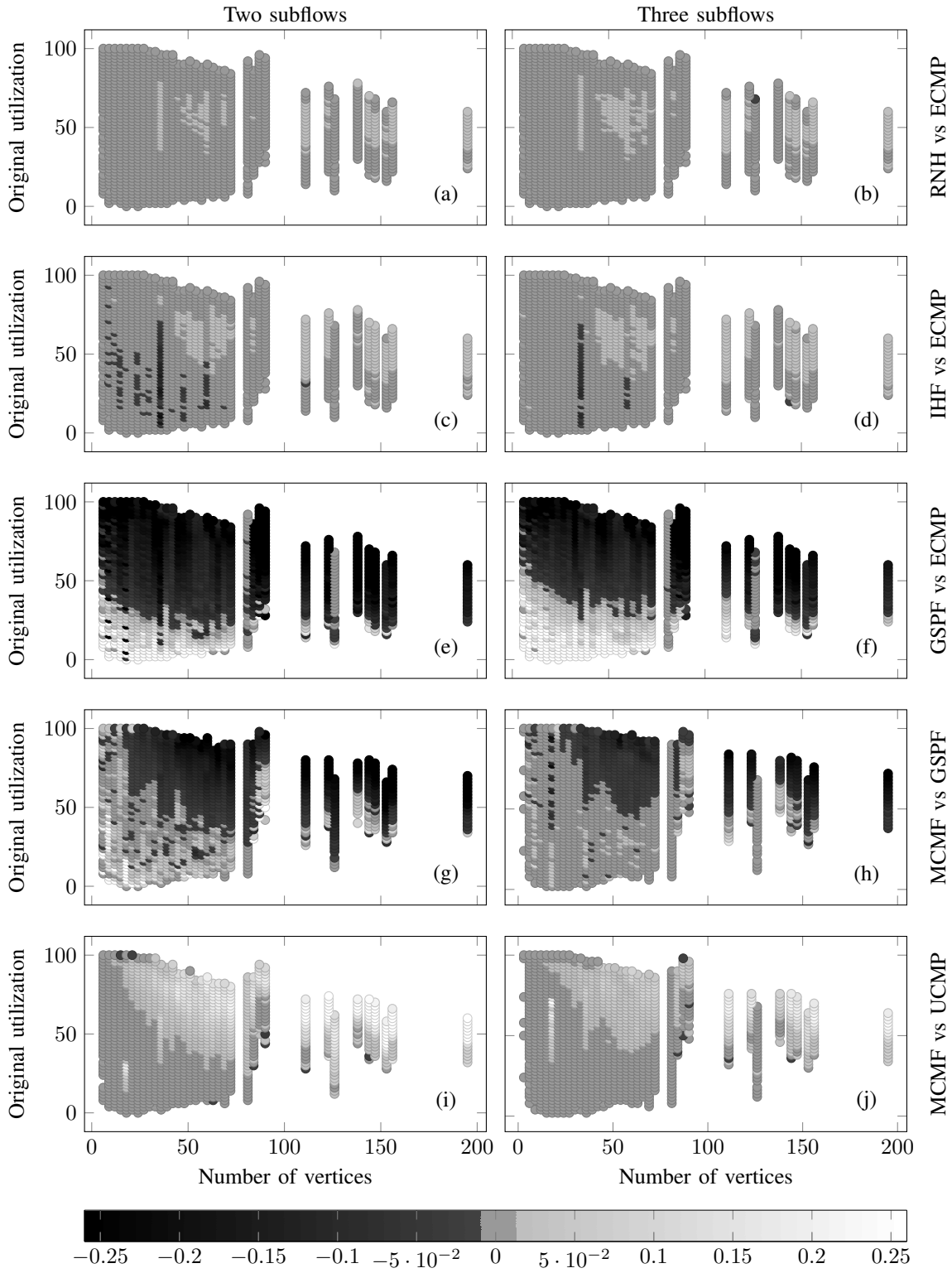


Рис. 1: Сравнение скоростей демультиплексированных и недемультиплексированных потоков с разными политиками маршрутизации

Fig. 1. Comparison of demultiplexed and non-demultiplexed flow rates under different routing policies

Панель на рисунке 1 снизу проводит соответствие между оттенком цвета и величиной преимущества демультиплексирования.

Результаты экспериментального исследования показали, что демультиплексирование обеспечивает увеличение производительности сети больше чем на 1% лишь в случае небольшого числа наборов значений входных параметров. Высокую эффективность метод массового статического демультиплексирования показывает в следующем случае: утилизация сети от 40% до 70%, средняя степень вершин в топологии больше 2.3 и среднее количество кратчайших маршрутов между вершинами больше 1.3. В этом случае производительность сети с применением демультиплексирования будет больше производительности сети без его использования более чем на 1% с вероятностью 78% при демультиплексировании на 2 подпотока и с вероятностью 85% при $k = 3$. Большое количество каналов между узлами сети обеспечивает ECMP балансировщики большим множеством альтернативных следующих скачков по направлению к каждой точке назначения. Например, доминирование демультиплексирования на топологиях с 36 вершинами связано с экспериментами на плотной топологии с названием «Bt North America 2010». Однако лишь небольшое количество рассмотренных топологий обладает достаточным разнообразием маршрутов, которое может эффективно использовать ECMP. Среди 1421 меток на рис. 1, показанных для случая двух подпотоков, демультиплексирование явно выигрывает только 179 раз.

1.4. Обоснование корректности

Для проверки корректности результатов, полученных при помощи имитационной модели, мы провели эмуляцию сети и нагрузки при тех же значениях входных параметров, что и для имитационных экспериментов. Сетевые топологии эмулировались с помощью виртуальных коммутаторов OpenVSwitch и Veth-интерфейсов на экспериментальном стенде, который состоял из одного сервера с 24 ядрами 2 ГГц и оперативной памятью с объемом 64 ГБ. Каждый виртуальный канал между коммутаторами обеспечивал 1 Гбит/с пропускной способности, задержку в 2 мс и 0% потерь. Для задачи генерации трафика (потоков) на сервере размещались 4 гостевые виртуальные машины, выполняющие роль полюсов. Эти виртуальные машины были развернуты с модифицированным ядром, поддерживающим работу по протоколу MPTCP. Каждая гостевая виртуальная машина была подключена к эмулируемой топологии одним сетевым интерфейсом через выделенный под эту задачу коммутатор OpenVSwitch. Этот коммутатор перенаправлял потоки от гостевой виртуальной машины к узлам топологии сети и возвращал потоки обратно к гостевым виртуальным машинам.

Родительские потоки трафика генерировались при помощи клиент-серверной программы iperf3. Одна пара клиент-сервер у iperf3 позволяет открывать до 128 потоков. Чтобы запустить необходимое количество потоков, запускались множественные пары клиент-серверов, которые были равномерно распределены по четырем гостевым виртуальным машинам. Для осуществления демультиплексирования потоков на каждой виртуальной машине выделялось k интерфейсов с IP-адресами из разных подсетей.

Для упрощения настройки экспериментального стенда использовался разработанный нами модуль порождения родственных подпотоков, основанный на страте-

гии full-mesh [6]. Он порождал один подпоток на каждую пару IP адресов внутри одной подсети. Кроме того, порождение дополнительных родственных подпотоков происходило на стороне сервера, а не клиента. Тогда родственные подпотоки имели разные транспортные порты на стороне сервера, но гарантированно имели один и тот же транспортный порт на стороне клиента. Так как программе iperf3 в параметрах можно указать, какие транспортные порты использовать на стороне клиента, то можно узнать принадлежность каждого подпотока к родственным потокам по паре IP-адресов и клиентскому транспортному порту. Порт на стороне сервера для этой задачи не подойдет, так как iperf3 и так запускал до 128 параллельных родительских потоков с одним серверным портом для одной пары клиент-сервера. Используя информацию об IP-адресах и клиентском транспортном порте подпотока, можно реализовать любую политику маршрутизации, вычисляя для каждого подпотока подходящие правила пересылки и загружая их в коммутаторы.

Каждый запуск экспериментального стенда был настроен на создание 1024 одноминутных родительских потоков. Для каждого потока была рассчитана его скорость как среднее значение от ежесекундных замеров скорости потока. При этом статистика за первые 15 секунд каждого запуска была отброшена, чтобы отсечь период стабилизации нагрузки (например, стадия медленного старта у алгоритмов управления перегрузок).

Множество всех значений входных параметров, где алгоритмом маршрутизации был CCC, включало в себя несколько тысяч различных комбинаций сетевых топологий, нагрузки трафика и т.д.

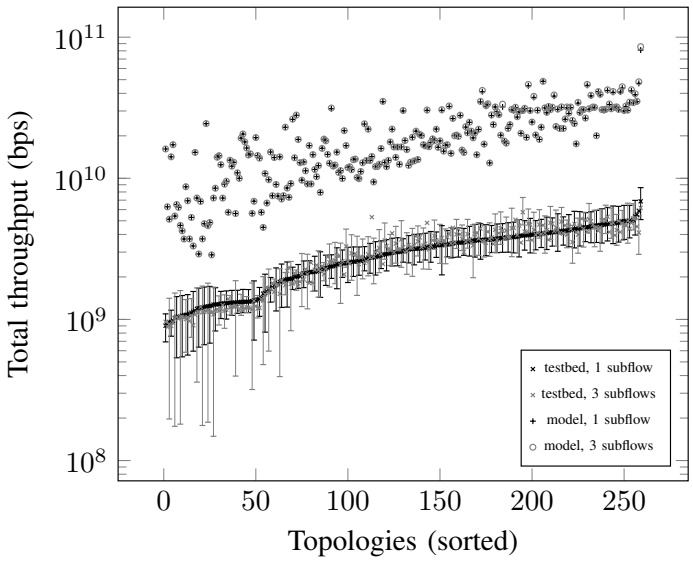


Рис. 2: Производительность сети для CCC, измеренная на тестовом стенде и в имитационной модели

Fig. 2. Total network performance as it is measured by the testbed and estimated by simulation model for RNH

Рисунок 2 показывает среднее значение производительности сети для каждой из тестируемых топологий. Производительность сети в случаях недемультиплексированных потоков и демультиплексирования на три подпотока показана черными и

серыми крестами соответственно. Вертикальный отрезок для каждой отметки ограничивает доверительный интервал для значения производительности сети с уровнем доверия 50 %.

Результаты для демультиплексирования на два подпотока были опущены, потому что они оказались аналогичны случаю с тремя подпотоками. Рисунок 2 также содержит оценку, предоставленную имитационной моделью из раздела 1.3. Оценки в случаях недемультиплексированных потоков и демультиплексирования на три подпотока показаны черными плюсами и серыми окружностями соответственно.

Линейная корреляция между измеренной производительностью сети и оценкой имитационной модели составила 0.79, что является высокой степенью соответствия по шкале Чеддока [19]. Пониженная точность связана с тем, что прогнозы нашей модели чрезмерно оптимистичны – на практике TCP потоки не задействуют все предоставленные ресурсы сети из-за неэффективности работы алгоритмов управления перегрузкой. Тем не менее, модель применима для сравнения производительности сети в случаях демультиплексирования потоков и его отсутствия. И экспериментальный стенд, и имитационная модель не показали существенных улучшений или ухудшений в производительности сети в случае массового демультиплексирования.

2. Иные политики маршрутизации

В предыдущем разделе был сделан вывод, что массовое применение протокола MPTCP дает преимущество лишь для небольшого количества рассмотренных значений входных параметров модели. Причиной этому может быть неэффективность используемой политики маршрутизации. Например, из-за случайности выбора следующего скачка несколько родственных подпотоков могут использовать одно и то же узкое место одного и того же маршрута. В этом разделе приводится исследование альтернативных вариантов политик маршрутизации, которые могут быть использованы при массовом демультиплексировании.

2.1. Обратимые функции хэширования

Чтобы проверить гипотезу о неэффективности алгоритма маршрутизации CCC, была изменена процедура генерации маршрутов. В измененном варианте предотвращается маршрутизация родственных подпотоков через один и тот же следующий скачок, если это возможно. Реализовать предложенный подход в сетях ISP можно при помощи стратегии, предложенной в [20], которую будем называть *Обратимой Функцией Хэширования* (ОФХ) или Invertible Hash Function (IHF). Идея ОФХ заключена в том, чтобы выбор среди множества альтернативных следующих скачков осуществлялся при помощи обратимой хэш-функции от заголовков пакетов. Благодаря обратимости хэш-функции отправитель транспортного потока сможет подобрать значение своего порта таким образом, чтобы на маршрутизаторе хэш-функция от заголовка пакета соответствовала индексу желаемого следующего скачка. В частности, для демультиплексированного потока можно обеспечить распределение i-го подпотока на i-й следующий скачок из множества альтернативных.

Рисунки 1c и 1d сравнивают производительность сети для стратегии ОФХ. Неожиданным результатом оказалось, что ОФХ обеспечило лишь минимальное преимущество для статического демультиплексирования – демультиплексированные потоки выигрывают на 243 метках на рис. 1c из 1421, а проигрывают на 119 из них. На тех значениях входных параметров, на которых демультиплексирование обладало преимуществом в случае ССС, производительность сети в случае ОФХ оказалась намного ниже. Наше исследование показало, что поскольку отправитель не обладает информацией о количестве альтернативных следующих скачков на маршрутизаторе, то он всегда выбирает следующие скачки с наименьшими индексами. Поскольку в этом случае транспортные потоки никогда не используют следующий скачок с высоким индексом, ОФХ не может использовать все доступные альтернативные маршруты, в то время как ССС равномерно распределяет потоки по альтернативным следующим скачкам.

2.2. Маршруты с минимальными пересечениями

До сих пор рассматривались только те политики маршрутизации, которые распределяли подпотоки среди маршрутов с одинаковой наименьшей стоимостью. С использованием ССС и ОФХ могли возникнуть ситуации, когда маршруты родственных подпотоков совпадают или частично накладываются друг на друга, так как нет альтернативных маршрутов той же стоимости. Использование неоптимальных по длине, но непересекающихся маршрутов может повысить производительность сети. В данном разделе проверяется эта гипотеза при помощи нескольких алгоритмов построения маршрутов, которые предпочитают непересекающиеся маршруты кратчайшим. При этом мы предполагаем, что можем контролировать маршрут каждого подпотока (такое возможно, например, в сети SDN), а детали осуществления этого предположения лежат за рамками этой статьи.

Стоит отметить, что построение полностью непересекающихся маршрутов не всегда возможно. Топология сети может быть устроена таким образом, что какой бы маршрут от заданных отправителя и получателя ни был выбран, он обязательно пройдет через некоторый канал в сети. В этом случае предпочтительнее будут маршруты с минимальным пересечением, так как узкое место может находиться в другом месте сети. Под n маршрутами с минимальным пересечением понимается любое множество из n маршрутов, у которых сумма пересечений каждой пары маршрутов этого множества минимальна по сравнению со всеми другими множествами из n маршрутов.

Наше исследование начнем с выбора алгоритмов построения соответствующих маршрутов для родственных подпотоков. Многие статьи [9, 21, 22] предлагают использовать простой, но в то же время эффективный *Greedy Shortest Paths First* (GSPF) алгоритм. На каждой итерации этого алгоритма строится маршрут для одного из родственных подпотоков при помощи одного из алгоритмов построения кратчайшего маршрута (например, [23]). По окончании очередной итерации GSPF вес каждого ребра построенного маршрута увеличивается на определенное штрафное значение. Обычно это значение достаточно велико, чтобы маршруты с повторно используемыми ребрами были менее предпочтительными, чем маршруты без пересечений.

Достоинства GSPF – простота и явный способ вычисления дополнительных маршрутов родственных подпотоков по требованию. Однако алгоритм не даст оптимального решения – количество совпадающих ребер в построенных маршрутах не всегда минимально. В какой-то момент жадный выбор (построение кратчайшего маршрута на каждой итерации) может привести к ситуации, когда не существует еще одного непересекающегося маршрута, хотя можно получить аналогичное количество непересекающихся маршрутов при другом алгоритме построения (подробнее в [24]).

Нами был разработан еще один алгоритм построения маршрутов, который в отличие от GSPF дает оптимальное решение по количеству пересечений маршрутов. Этот алгоритм сводит задачу построения маршрутов к классической задаче нахождения *максимального потока минимальной стоимости* (min-cost max-flow), откуда получил название MCMF алгоритма. MCMF алгоритм начинает свою работу с изменения начального графа сети. Он заменяет каждое ребро в графе на множество k параллельных ребер, где k равно ожидаемому числу родственных подпотоков. Для i-го ребра в этом множестве, $0 \leq i < k$, MCMF задает его стоимость как $1+i|E|$, где $|E|$ обозначает количество всех ребер в исходном графе. Стоимость необходима для того, чтобы в решении было как можно меньше пересечений между маршрутами. Для всех ребер в графе MCMF устанавливает пропускную способность в 1. Наконец, MCMF добавляет специальный узел *сток* и соединяет его с полюсом-получателем ребром с пропускной способностью k.

По измененному графу MCMF алгоритм вычисляет максимальный поток минимальной стоимости между отправителем и стоком. С одной стороны, поток всегда может получить по крайней мере k единиц пропускной способности, так как MCMF заменяет каждое ребро в графе на k ребер с единичной пропускной способностью. С другой стороны, скорость потока никогда не сможет превысить k из-за узкого места, образованного ребром на стоке. Таким образом, максимальная скорость потока всегда эквивалентна k, и она распределяется среди k разных маршрутов через измененный граф.

Результаты, полученные алгоритмом нахождения максимального потока минимальной стоимости для измененного графа, отображаются на соответствующие маршруты в исходном графе. Хотя отображение можно провести неоднозначно, каждый полученный набор маршрутов имеет минимальное пересечение и наименьшую суммарную стоимость среди всех маршрутов с минимальным пересечением. MCMF в отличие от GSPF всегда обеспечивает *оптимальный* результат – пока есть возможность уменьшить число разделяемых каналов, MCMF будет её использовать.

2.3. Оценка производительности сети в случае GSPF и MCMF

Описанные выше стратегии выбора маршрутов GSPF и MCMF были проанализированы на имитационной модели из раздела 1.3. На рисунках 1e и 1f показаны результаты сравнения GSPF с ECMP. Рисунки 1g и 1h демонстрируют улучшение в производительности сети, которое MCMF обеспечивает по сравнению с GSPF. Эксперименты показывают, что GSPF и MCMF обеспечивают существенное увеличение производительности сети по сравнению с ECMP, когда значение утилизации низкое. Если утилизация сети ниже 37%, демультиплексирование на три подпотока увели-

чивает производительность сети с вероятностью большей, чем 50%! Очевидно, что GSPF и MCMF обеспечивают достаточное разнообразие маршрутов для демультиплексированных потоков, чтобы обойти перегруженные каналы, появляющиеся вдоль кратчайших маршрутов. Однако, как только сеть становится сильно загруженной, маршруты от GSPF и MCMF ухудшают производительность сети из-за большей длины их маршрутов. Чем длиннее маршрут, тем больше ресурсов сети необходимо для потока, что труднее удовлетворить при большой утилизации сети. Если сеть использует 80% её полосы пропускания, демультиплексирование на три подпотока ухудшит производительность сети на 18% в среднем.

С целью обоснования корректности расчетов была проведена эмуляция на части значений входных параметров для стратегии маршрутизации GSPF на экспериментальном стенде из раздела 1.4. На рисунке 3 показана производительность сети для демультиплексированных и недемультиплексированных потоков, когда доля полюсов составляет 30%. Эмуляция в целом подтвердила наши предсказания – как математическая модель, так и эмуляция сети показали, что демультиплексирование должно увеличить производительность сети на 25% в среднем. Линейная корреляция между результатами на модели и на стенде составила 0.84, что тоже является высокой степенью соответствия по шкале Чеддока.

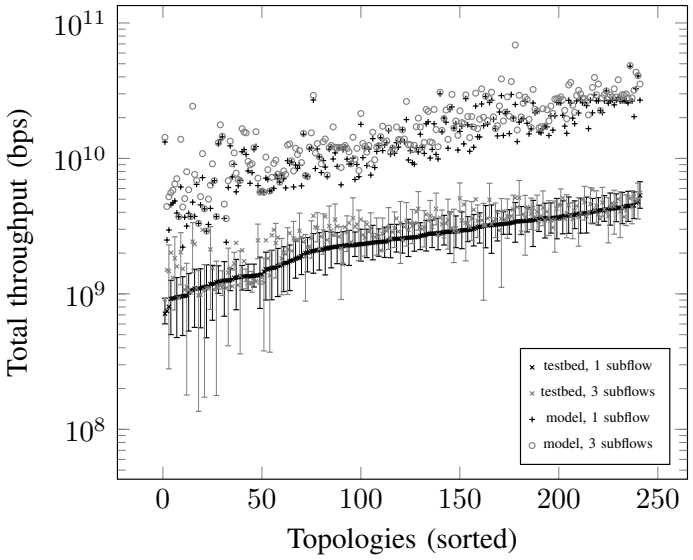


Рис. 3: Производительность сети для GSPF,
измеренная на тестовом стенде и в имитационной модели

Fig. 3. Total network performance as it is measured
by the testbed and estimated by simulation model for GSPF

2.4. Алгоритм выбора маршрутов UCMP

Как было показано выше, демультиплексирование потоков с политикой маршрутизации GSPF или MCMF позволяет увеличить производительность сети в случае её низкой утилизации (до 37%). Однако есть опасение, что этот эффект вызван скорее алгоритмами выбора маршрута, а не техникой демультиплексирования потоков.

Для снятия этого опасения была проведена еще одна серия экспериментов с недемультиплексированными потоками. В этот раз потоки распределялись равномерно по неоптимальным маршрутам, предоставленным GSPF и MCMF, а не только по кратчайшим, как в CCC. Такой алгоритм выбора маршрутов для недемультиплексированных потоков назовем *Unequal-Cost Multi-Path* (UCMP).

Оценка производительности сети с алгоритмом выбора маршрутов UCMP, полученная на имитационной модели из раздела 1.3., была использована как новый эталон для оценки эффективности демультиплексирования. Сравнение производительности сети с алгоритмами выбора маршрутов UCMP для недемультиплексированных потоков и MCMF для демультиплексированных показано на рисунках 1i и 1j. Результаты для GSPF были опущены, так как они схожи с результатами для MCMF. Демультиплексирование улучшает производительность сети, достигнутую балансировкой UCMP, в ряде случаев, однако эти случаи соответствуют сетям с утилизацией выше 60%. Ранее было показано, что маршрутизация по кратчайшему маршруту CCC более эффективна для высоко утилизированных сетей, чем GSPF и MCMF. А при утилизации ниже 60% можно использовать алгоритм выбора маршрутов UCMP для недемультиплексированных потоков, что приводит к тому же результату, как и демультиплексирование с MCMF или GSPF. Поэтому можно сделать вывод, что демультиплексирование не способствует увеличению производительности сети, по крайней мере при неизменяющейся нагрузке.

Интересно, что имитационное моделирование показало уменьшение преимущества демультиплексирования потоков при утилизации сети выше 60%, если количество родственных подпотоков увеличить с 2 до 3. Интуитивно, демультиплексирование на большее количество подпотоков должно увеличивать преимущество в производительности сети над недемультиплексированными потоками. Однако включение третьего маршрута приводит к увеличению утилизации сети и производительности сети для UCMP в среднем на 10%, из-за чего становится намного сложнее увеличить производительность сети в случае демультиплексирования.

Эксперименты показали, что использование маршрутов с минимальным пересечением для родственных подпотоков имеет гораздо более высокий потенциал, чем использование традиционной политики маршрутизации. Однако преимущество демультиплексирования, скорее всего, лежит в более равномерном распределении пропускной способности сети между родительскими потоками, а не в увеличении производительности сети. В зависимости от текущей утилизации каналов вдоль маршрутов скорость транспортных потоков может существенно отличаться. Использование нескольких альтернативных маршрутов родственными подпотоками может смягчить различие в скорости родительских потоков и гарантировать более высокую минимальную скорость для всех потоков в сети. Но исследование этого вопроса находится за рамками данной статьи.

Заключение

В статье проведено исследование, может ли массовое статическое демультиплексирование потоков улучшить производительность сети масштаба ISP. Была предложена математическая модель, на основе которой построена имитационная модель с

целью получения более точных оценок производительности сети с демультиплексированием потоков и без него. Хотя наша имитационная модель и не учитывает всех тонкостей работы алгоритмов управления перегрузкой, её все же можно использовать для прогнозирования того, улучшит демультиплексирование производительность сети или ухудшит. Согласно нашим экспериментам массовое демультиплексирование не обеспечит никакого улучшения для большинства топологий ISP сетей, представленных в библиотеке TopologyZoo.

В то же время было показано, что демультиплексированные соединения имели более высокую эффективность, если маршруты их родственных подпотоков имели минимальное пересечение. Была произведена оценка производительности сети при работе с двумя разными алгоритмами построения маршрутов – GSPF и MCMF. При использовании маршрутизации по каждому из этих алгоритмов демультиплексированные потоки обеспечили ощутимое увеличение производительности сетей ISP с уровнем утилизации ниже 37%. Интересно, что можно добиться аналогичного улучшения, используя MCMF или GSPF при реализации алгоритма выбора маршрутов UCMP для недемультиплексированных потоков. Таким образом, улучшение производительности сети является основным достоинством алгоритмов построения маршрутов, а не демультиплексирования.

Кроме того, в ходе экспериментального исследования стало ясно, что необходимо демультиплексировать разные родительские потоки на разное число родственных подпотоков в зависимости от топологии сети и текущей утилизации каналов. Определить, на какое количество необходимо демультиплексировать поток, можно только в динамике в зависимости от текущего состояния сети. Если скорость родительского потока не увеличивается при создании дополнительного подпотока, то этот подпоток лишний и его можно закрыть, чтобы не возникало лишней конкуренции в сети. Это подтверждает преимущество динамических подходов к выбору маршрутов, таких как FDMP [9].

Дополнительное исследование демультиплексированных потоков может показать преимущества по другим показателям. Например, демультиплексирование может приводить к более равномерному распределению пропускной способности сети по сравнению с UCMP. На практике это означает, что потоки между клиентами одной ISP сети будут обеспечивать более стабильную и прогнозируемую скорость независимо от их относительного местоположения. Кроме того, демультиплексирование также может обеспечить определенную выгоду в сетях с быстро меняющимися матрицами трафика. Быстрое перераспределение данных между подпотоками должно реагировать на перегрузки в сети намного быстрее, чем балансировщики, интегрированные в сетевую инфраструктуру. Однако проверка этого утверждения выходит за рамки данной статьи.

Список литературы / References

[1] Kim Bong Ho, Calin Doru, “On the Split-TCP Performance over Real 4G LTE and 3G Wireless Networks”, *IEEE Communications Magazine*, **55**:4 (2017), 124–131.

[2] Chiesa M., Kindler G., Schapira M., “Traffic Engineering With Equal-Cost-MultiPath: An Algorithmic Perspective”, *IEEE/ACM Transactions on Networking*, **25**:2 (2017), 779–792.

[3] Saltzer J. H., Reed D. P., Clark D. D., "End-to-end Arguments in System Design", *ACM Trans. Comput. Syst.*, **2**:4 (1984), 277–288.

[4] Curtis A. R., Kim W., Yalagandula P., "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection", *2011 Proceedings IEEE INFOCOM*, 2011, 1629–1637.

[5] Ford A., et al., *RFC 6824: TCP Extensions for Multipath Operation with Multiple Addresses*, 2013, 64 pp.

[6] Raiciu C., et al., "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, 399–412.

[7] Habib S., et al., "The Past, Present, and Future of Transport-Layer Multipath", 2016, https://arxiv.org/abs/1601.06043.

[8] Boccassi L., Fayed M.M., Marina M.K., "Binder: A System to Aggregate Multiple Internet Gateways in Community Networks", *Proceedings of the 2013 ACM MobiCom Workshop on Lowest Cost Denominator Networking for Universal Access*, 2013, 3–8.

[9] Chemeritskiy E., Stepanov E., Smeliansky R., "Managing network resources with flow (de) multiplexing protocol", *Mathematical and Computational Methods in Electrical Engineering*, Recent Advances in Electrical Engineering Series, **53**, 2015, 35–43.

[10] Raiciu C., Handley M., Wischik D., "RFC 6356: Coupled Congestion Control for Multipath Transport Protocols", 2011, 12 pp.

[11] Khalili R., et al., "MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution", *IEEE/ACM Transactions on Networking*, **21**:5 (2013), 1651–1665.

[12] Peng Q., et al., "Multipath TCP: Analysis, Design, and Implementation", *IEEE/ACM Transactions on Networking*, **24**:1 (2016), 596–609.

[13] Le T. A., Hong C. S., Lee S., "MPCubic: An extended cubic TCP for multiple paths over high bandwidth-delay networks", *ICTC 2011*, 2011, 34–39.

[14] Ha B. P., et al., "A hybrid multipath congestion control algorithm for high speed and/or long delay networks", *2014 International Conference on Advanced Technologies for Communications (ATC 2014)*, 2014, 452–456.

[15] Bich-Phuong Ha, et al., "A Multipath Cubic TCP Congestion Control with Multipath Fast Recovery over High Bandwidth-Delay Product Networks", *IEICE Transactions on Communications*, **E-95B**:7 (2012), 2232–2244.

[16] Hassayoun S., Iyengar J., Ros D., "Dynamic Window Coupling for multipath congestion control", *2011 19th IEEE International Conference on Network Protocols*, 2011, 341–352.

[17] Jean-Yves Boudec, *Rate adaptation, Congestion Control and Fairness: A Tutorial*, 2018, 54 pp.

[18] Knight S. et al., "The internet topology zoo", *IEEE Journal on Selected Areas in Communications*, **29**:9 (2011), 1765–1775.

[19] Carter H., "Chaddock, Robert, Emmet. Principles and Methods of Statistics. Pp. xvi+471. Houghton Mifflin Company, 1925", *The ANNALS of the American Academy of Political and Social Science*, **123**:1 (1926), 229.

[20] van der Linden S., Detal G., Bonaventure O., "Revisiting Next-hop Selection in Multipath Networks", *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, 420–421.

[21] Navin Kukreja, et al., "SDN based automated testbed for evaluating multipath TCP", *IEEE International Conference on Communication, ICC 2015* (London, United Kingdom, June 8–12, 2015), 2016, 718–723.

[22] Chawanat Nakasan, et al., "A Simple Multipath OpenFlow Controller using topology-based algorithm for Multipath TCP", 2015, https://arxiv.org/abs/1509.08388.

[23] Dijkstra E. W., "A Note on Two Problems in Connexion with Graphs", *Numer. Math.*, **1**:1 (1959), 269–271.

[24] Suurballe J. W., Tarjan R. E., "A quick method for finding shortest pairs of disjoint paths", *Networks*, **14**:2 (1984), 325–336.

Stepanov E. P., "On Analysis of Traffic Flow Demultiplexing Effectiveness", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 170–190.

DOI: 10.18255/1818-1015-2019-1-170-190

Abstract. It is known that the demultiplexing of the individual traffic flow into several independent transport subflows can increase the speed of it. This statement is true for a single flow but its truth for the *massive* case, when demultiplexing technics are applied to all traffic flows in the network of a single Internet Service Provider (ISP), is not obvious. The question arises, what impact the massive demultiplexing of traffic flows will have on the whole ISP network bandwidth. In this paper, this question is considered for the static case, when each flow is demultiplexed statically, i.e. every flow before its launching is demultiplexed into the same number of subflows. We developed a mathematical model that was used to construct a simulation model in order to obtain more accurate estimates of the network performance with and without flow demultiplexing. Using simulation model, network properties are defined, under which the demultiplexing of traffic flows is justified. We proved the correctness of the obtained results by emulating a network load based on a protocol stack virtualization for the same input. We considered various routing policies that can be used for massive demultiplexing. Special attention is paid to algorithms that allow you to build routes with minimal intersections, since using the non-intersecting routes with non-optimal cost can increase the network performance. The routes constructed with these algorithms were used both for the network performance analysis with demultiplexed flows and in the case of balancing non-demultiplexed flows.

Keywords: multipath routing, quality of service, MPTCP

On the authors:
Evgeniy P. Stepanov, orcid.org/0000-0003-3649-0745, graduate student,
Lomonosov Moscow State University,
1, bd. 52 Leninskiye Gory, Moscow 119991, Russia, e-mail: estepanov@lvk.cs.msu.su

Acknowledgments:
The research was supported by the Russian Fund of Basic Research (Project No. 18-07-01255).