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.

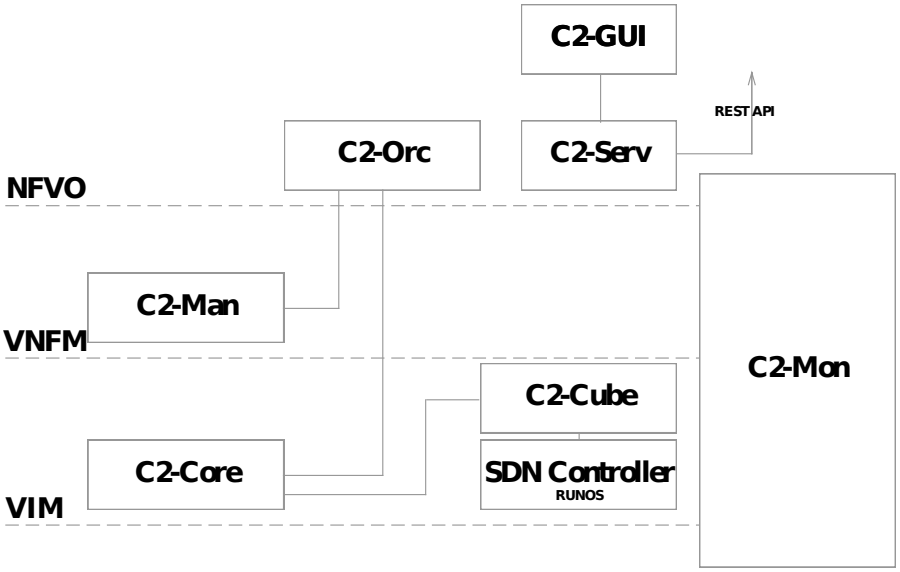

Рис. 1. Архитектура MANO-платформы C2
Fig. 1. Architecture of C2 MANO-platform

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.

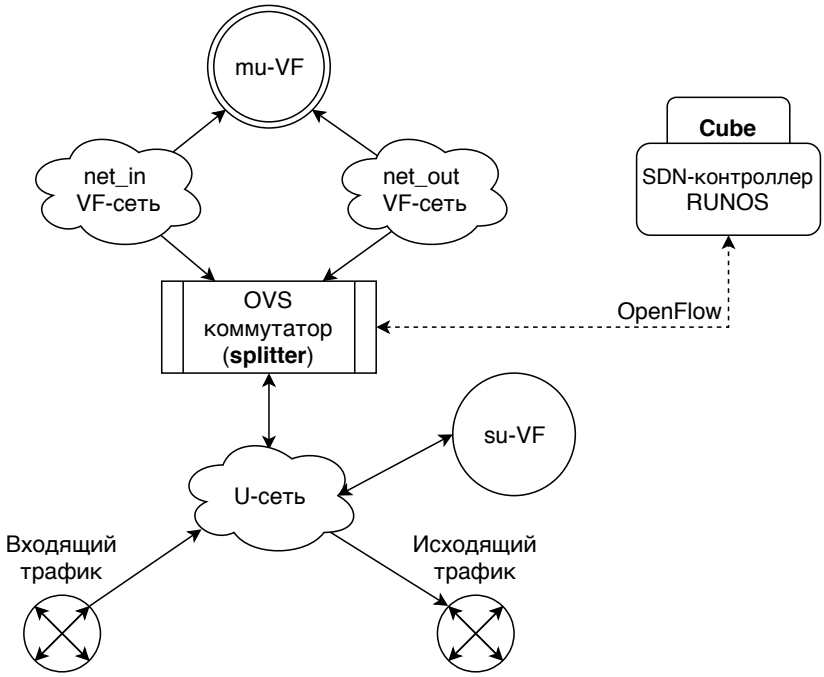

Рис. 2. Схема подсоединения mu-VF
Fig. 2 The mu-VF network connection scheme

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.

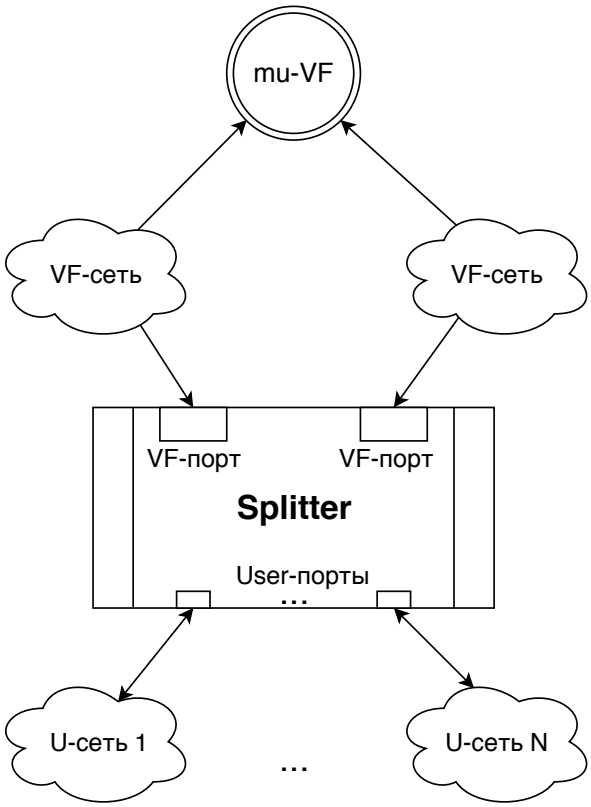

Рис. 3. Схема портов splitter
Fig. 3 Splitter port scheme

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].

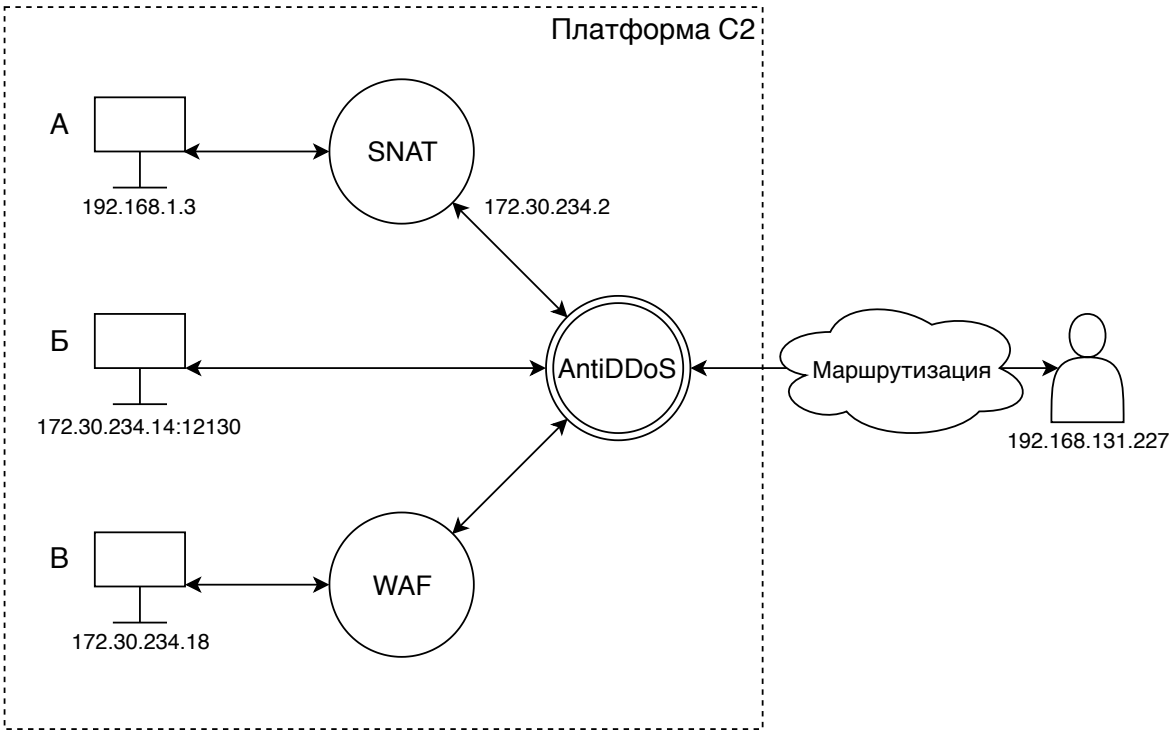

Рис. 4. Схема сетевого взаимодействия в эксперименте
Fig. 4. Network Interaction Scheme in the Experiment

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.

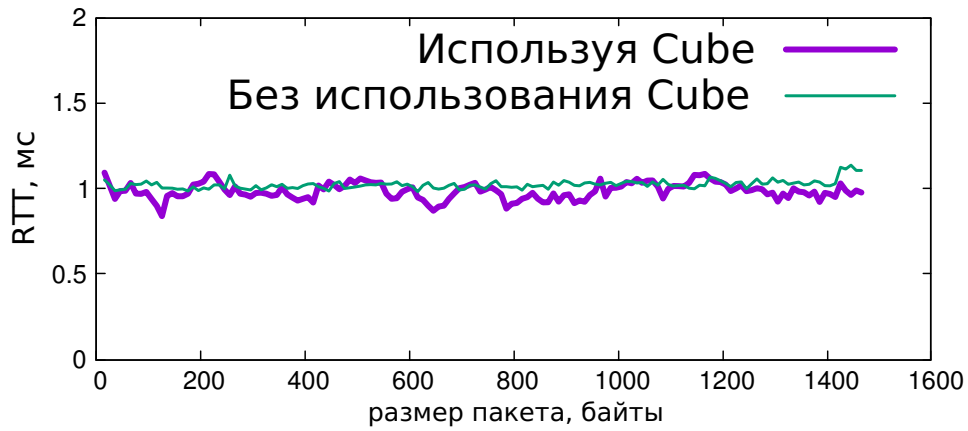

Рис. 5. Влияние Cube на задержку пакетов
Fig. 5. Cube impact on a packet delay

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.

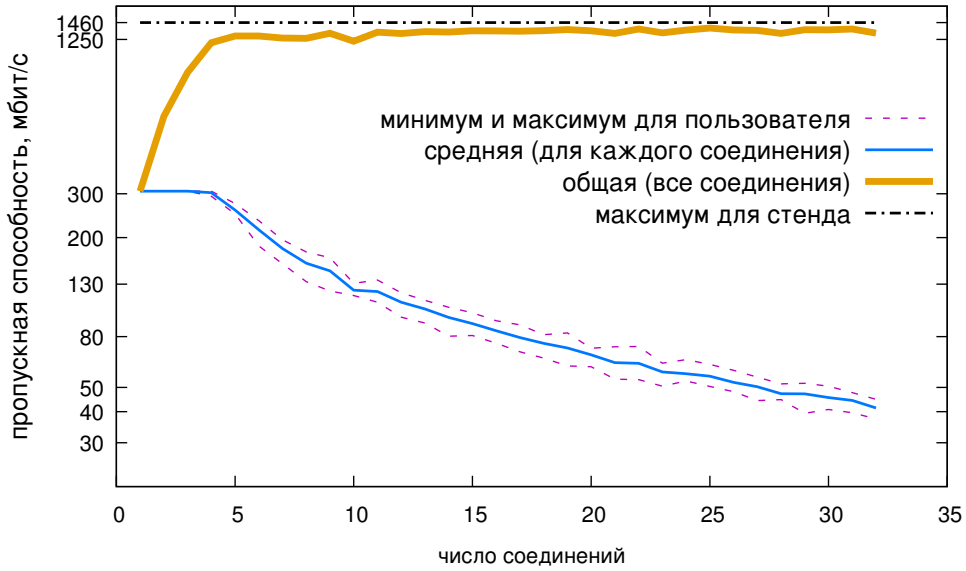

Рис. 6. Результаты эксперимента с пропускной способностью
Fig. 6. The Results of the throughput experiment

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.