УДК 004.855
DOI: 10.25559/SITITO.16.202003.551-563

Оригинальная статья

Исследование производительности программно-конфигурируемой инфраструктуры в сетях VANET на базе моделей гибридных устройств передачи данных

М. В. Ушакова, Ю. А. Ушаков*, И. П. Болодурина, А. Л. Коннов
ФГБОУ ВО «Оренбургский государственный университет», г. Оренбург, Российская Федерация
460018, Российская Федерация, г. Оренбург, пр. Победы, д. 13; * unpk@mail.ru

Аннотация

Развитие сетей нового поколения на основе программно-конфигурируемых сетей и включение их в стек 5G требует новых подходов к изучению работы таких сетей. Большинство исследователей доверяет фреймворкам всю низкоуровневую работу, сосредотачиваясь на более высокоуровневых показателях. Однако большинство инструментов для моделирования имеет очень ограниченные возможности изучения программно-конфигурируемого оборудования, особенно задержек обработки пакетов. Также недостаточно уделено внимания моделям виртуальных сетевых устройств, которые работают не так, как физическое оборудование и имеют другие параметры производительности и зависимости от внешних факторов. Цель работы состоит в изучении внутренней структуры сетевого оборудования системы моделирования OmNET++, а также создание альтернативных моделей, учитывающих все особенности различных реализаций программно-конфигурируемого оборудования. Построены схемы работы внутренних процессов оборудования, изучены блоки системы моделирования, позволяющие описывать процессы, происходящие при обработке пакета внутри гибридного программно-управляемого оборудования. Таже изучена NFV коммутация, в которой применяются виртуальные коммутаторы, имеющие особенности обработки пакетов на основе ядра Linux. Для изучения работы моделей мыл проведен эксперимент, состоящий в измерении времени задержки пакета при обработке в реальном оборудовании, съема параметров задержки, воспроизведении этого поведения в модели. В результате исследования созданных моделей показано улучшение точности моделирования по параметрам задержки обработки пакета по сравнению с традиционно использующимися стандартными блоками моделей сетевого оборудования. Сделан вывод об адекватности модели по задержкам и о работоспособности метода съема характеристик реального оборудования.

Ключевые слова: программно-конфигурируемая сеть, OpenFlow, VANET, виртуальные сетевые устройства.

Финансирование: настоящая работа основана на исследованиях, выполненных при финансовой поддержке Российского фонда фундаментальных исследований, в рамках научных проектов № 20-57-53019 «Разработка моделей и механизмов защиты информации в автомобильных самоорганизующихся сетях на базе машинного обучения» и № 18-07-01446 А «Разработка методов оптимизации систем размещения вычислительных виртуальных элементов в облачных инфраструктурах при работе с большими данными».

Авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Ушакова, М. В. Исследование производительности программно-конфигурируемой инфрастуктуры в сетях VANET на базе моделей гибридных устройств передачи данных / М. В. Ушакова, Ю. А. Ушаков, И. П. Болодурина, А. Л. Коннов. — DOI 10.25559/SITITO.16.202003.551-563 // Современные информационные технологии и ИТ-образование. — 2020. — Т. 16, № 3. — С. 551-563.

Original article

Research of the Performance of Software-Defined Infrastructure in VANET Networks Based on Models of Hybrid Data Transmission Devices

M. V. Ushakova, Yu. A. Ushakov*, I. P. Bolodurina, A. L. Konnov
Orenburg State University, Orenburg, Russian Federation
13 Pobeda Ave., Orenburg 460018, Russian Federation
* unpk@mail.ru

Abstract

The development of new generation networks based on software-defied networks and their inclusion in the 5G stack requires new approaches to studying the operation of such networks. Most researchers trust frameworks for all low-level work, focusing on higher-level metrics. However, most simulation tools have a very limited ability to study software-defied hardware, especially packet latency. There is also insufficient attention paid to the models of virtual network devices, which behave differently from physical hardware and have different performance parameters and dependencies on external factors. All this led to the writing of this article, the purpose of which is to study the internal structure of the network equipment of the OmNET ++ modeling system, as well as create alternative models that take into account all the features of various software-defined equipment implementations. As a result of the study of the created models, an improvement in the simulation accuracy in terms of packet processing delay parameters is shown compared to the traditionally used building blocks of network equipment models.

Keywords: SDN, OpenFlow, VANET, virtual network devices.

Funding: This work is based on research carried out with the financial support of the Russian Foundation for Basic Research, within the framework of scientific projects No. 20-57-53019 "Development of Models and Mechanisms for Protecting Information in Self-Organizing Automotive Networks Based on Machine Learning" and No. 18-07-01446 A "Development of Methods for Optimizing Systems for Placing Computational Virtual Elements in Cloud Infrastructures when Working with Big Data."

The authors declare no conflict of interest.

For citation: Ushakova M.V., Ushakov Yu.A., Bolodurina I.P., Konnov A.L. Research of the Performance of Software-Defined Infrastructure in VANET Networks Based on Models of Hybrid Data Transmission Devices. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2020; 16(3):551-563. DOI: https://doi.org/10.25559/SITITO.16.202003.551-563

Введение

В существующих телекоммуникационных системах связь между коммутаторами в сетях становится все более и более сложной. В связи с лавинообразным ростом сетевого трафика и сложностями в управлении потоком внедряются новые технологии управления трафиком. Разработка нового уровня сервисов, таких как сети пятого поколения (5G), требует на уровне стандартов программного управления трафиком и использования виртуализации, граничных и туманных вычислений.

Необходимость сетей 5G обусловлена растущим спросом качественной связи с критически важными инфраструктурными системами, такими как электронное здравоохранение и телемедицина, а также с образовательными секторами, которые стремятся использовать все преимущества беспроводной связи [1].

Одной из самых проблемныхобластей применения 5G являются сети Vehiculard-hocnetworks (VANET). VANET — общее название сетей, предназначенных для передачи информации машинами, дорожной инфраструктурой, пешеходами и другими участниками дорожного движения. В общем случае это смесь разных каналов связи, стандартов, протоколов, но существует два стандарта, которые являются приоритетными для разработчиков. Это беспроводные IEEE 802.11p (беспроводная ad-hocVANET сеть на частоте 5.9 ГГц) и сотовые C-V2X 3GPP (release 13-15, 4G LTE и 5G).

Основной проблемой VANET являются постоянные переключения устройств между базовыми станциями, что приводит к высокой коммуникационной нагрузке. Кроме того, соседние устройства также должны обмениваться информацией о состоянии друг друга. Высокая коммуникационная нагрузка может повлиять на производительность приложений, которая увеличивает задержку при получении информации.

Поскольку 5Gв этой области сильно зависит от виртуализации инфраструктуры и ее компонентов, основой таких сетей являются программно-конфигурируемые сети.

Программно-конфигурируемые сети (SDN) — это сетевая парадигма управления в области телекоммуникаций, призванная удовлетворить потребность в гибкости сети и управлении трафиком. Архитектура SDN включает в себя “ControlPlane” и “DataPlane”. DataPlane состоит из коммутаторов и маршрутизаторов, которые способствуют передаче пакетов в сети, а ControlPlane действует как соединение между контроллером и коммутаторами для управления DataPlane [2]. Одним из самых широко используемых протоколов коммуникации между коммутатором и контроллером в ПКС является протокол OpenFlow. OpenFlow — это открытый протокол, который позволяет исследователям вводить новшества и экспериментировать с новыми сетевыми протоколами и приложениями внутри производственных сетей.

Цель исследования

Необходимо выполнить обзор систем моделирования с поддержкой SDN, рассмотреть процесс прохождения пакета через коммутатор EtherSwitch фреймворка INET. Также необходимо изучить возможности фреймворка INET OMNeT++ для реализации реалистичных задержек, возникающих при прохождении пакета через коммутатор, рассмотреть решения по моделированию OpenFlow оборудования и разработать модель OpenFlow коммутатора, отличающуюся возможностью управлять задержками по результатам работы собственной математической модели. Необходимо разработать схему работы гибридного коммутатора, поддерживающего как традиционную коммутацию, так и коммутацию OpenFlow, разработать модуль создания реалистичной задержки и ограничения емкости и производительности модели гибридного коммутатора, исследовать структуру NFV коммутации и разработать модель виртуального коммутатора OpenvSwitch.

Обзор систем моделирования с поддержкой SDN

Исследование возможностей реальной сети предполагает наличие довольно большого количества оборудования, что неизбежно приводит к большим затратам. Чтобы избежать существенных временных и денежных затрат, для изучения сетей можно применить такой метод, как имитационное моделирование. Существует довольно большое количество систем имитационного моделирования, позволяющих воспроизвести работу сети, однако далеко не все из них предлагают готовые модули для работы с ПКС. Большинство же систем, предлагающих решения для моделирования ПКС, ориентированы на контроллеры и не уделяют должного внимания работе OpenFlow-коммутаторов.

NS-3 — это сетевой симулятор с открытым исходным кодом для интернет-систем. Это довольно сложный инструмент, который запускает симуляции, описанные кодом, созданным пользователями, поэтому для его использования могут потребоваться навыки программирования. Симулятор NS-3 поставляется с протоколом Openflow и элементарным контроллером. Этому контроллеру не хватает базовой функциональности, предоставляемой настоящими контроллерами SDN, что делает его неадекватным для экспериментов в научном сообществе. В работе [3] авторы описывают модуль NS-3 с поддержкой технологии OpenFlow 1.3, содержащий как модуль коммутатора OpenFlow 1.3, так и интерфейс приложения контроллера. Реализация модуляимеет ряд недостатков: она доступна только для платформ GNU / Linux, доступно только одно соединение между коммутатором и контроллером, каждый коммутатор может управляться только одним контроллером.

EstiNet 9.0 — сетевой симулятор и эмулятор OpenFlow для исследования программно-определяемых сетей. Он поддерживает как режим симуляции, так и режим эмуляции. В режиме симуляции реальные контроллеры OpenFlow с открытым исходным кодом, такие как контроллеры NOX, POX, Floodlight, OpenDaylight и Ryu, могут напрямую запускаться на узле контроллера в симулируемой сети для управления этими симулированными коммутаторами OpenFlow без каких-либо изменений [4,5].

Mininet — это сетевой эмулятор, который создает сеть виртуальных хостов, коммутаторов, контроллеров и каналов связи. На хостах Mininet используется стандартное сетевое программное обеспечение Linux, а его коммутаторы поддерживают OpenFlow для гибкой настраиваемой маршрутизации и программно-определяемой сети. В версии Mininet 2.2.2 представлены только четыре модели: хосты, коммутатор SDN, устаревший коммутатор и устаревшая модель маршрутизато-

ра. Реализована только одна линия связи, детально можно настроить только параметры полосы пропускания, задержки и вероятности потерь. Кроме того, отсутствует собственный генератор трафика и функции анализа производительности [6].

OFNet — это эмулятор SDN, который предлагает функциональность, аналогичную сетевому эмулятору Mininet, и добавляет некоторые полезные инструменты для генерации трафика и мониторинга сообщений OpenFlow и оценки производительности контроллера SDN.OFNet — это проект с открытым исходным кодом, который распространяется в виде образа виртуальной машины. Исходный код OFNet доступен в файловой системе виртуальной машины OFNet [7,8].

В коммерческом эмуляторе NetSim реализован модуль SDN, оснащенный контроллерами SDN, которые можно использовать для управления пересылкой пакетов на всех устройствах уровня 3 в сети. Модуль SDN в настоящее время доступен для таких сетей, как Internetworks, WSN / IoT, MANET, VANET и LTE. С помощью этого модуля любое устройство уровня 3, может быть настроено в качестве контроллера SDN[1].

Коммерческая система моделирования OPNET Modeler (Riverbed Modeler, далее OPNET) реализует сетевые модели различных производителей, отражающие спецификации реального оборудования. OPNET также предоставляет возможность детальной настройки параметров трафика. Несмотря на все достоинства OPNET, он имеет ряд недостатков: отсутствуют логические порты между коммутатором и контроллером, отсутствует гибридная коммутация, не предусмотрены повторная сборка фрагментации IP и несколько контроллеров. Кроме того, OPNET является коммерческой системой [9].

Широко известная система моделирования OMNeT++ — этооткрытый, расширяемый, модульныйинструментдляимитациидискретныхсобытийнаосновекомпонентов, предоставляющий возможности длямоделированияпроводных и беспроводныхсетей. Это симулятор, способный моделировать любую систему, состоящую из устройств, взаимодействующих друг с другом. OMNeT ++ имеет обширную поддержку графического интерфейса, и благодаря своей модульной архитектуре ядро моделирования и модели могут быть легко встроены в приложения. Фреймворк INET, позволяющий проводить моделирование сетей, поддерживает многие известные протоколы. Для работы с программно-конфигурируемыми сетями OMNeT++ предоставляет модуль расширения OpenFlow 1.5.1 [10]. Кроме того, OMNeT++ имеет несколько фреймворков, позволяющихисследовать сети VANET.

Veins — это фреймворк с открытым исходным кодом для симуляции vehicularnetwork. Он основан на двух устоявшихся симуляторах: OMNeT ++, сетевой симулятор на основе событий и SUMO, симулятор дорожного движения. Veins расширяет их, предлагая полный набор моделей для моделирования IVC[2] [11, 12]. Еще одним симулятором на основе систем OMNeT ++ и SUMO, является VENTOS. VENTOS является бесплатным симулятором с открытым исходным кодом, который предназначен для анализа потоков транспортных средств и может включать в себя новую логику управления, такую как интеллектуальный контроллер дорожного движения, совместное вождение, динамическое маршрутизация и возможность самостоятельного вождения. VENTOS также позволяет осуществлять связь V2X (транспортное средство-транспортное средство или транспортное средство-инфраструктура) посредством специальной интеграции ближней связи (dedicated short-range communication, DSRC) [13].

Simu5G — фреймворк для 5G NewRadio networks, предоставляющий набор моделей с четко определенными интерфейсами, которые могут быть созданы и подключены для создания произвольно сложных сценариев моделирования [14]. Simu5G может работать в режиме эмуляции в реальном времени, обеспечивая взаимодействие с реальными устройствами благодаря планированию событий OMNeT++ в реальном времени и способности INET обмениваться IP-пакетами между локальными приложениями или сетевыми интерфейсами и симулятором. Однако в настоящее время фреймворк Simu5G еще не опубликован.

OMNeT++ был выбран в качестве платформы для разработки, главным образом из-за ее природы с открытым исходным кодом, хорошо организованной модульной архитектуры, существующей доступной документации и предоставленной IDE (Integrated Development Environment).

Коммутатор В Omnet++

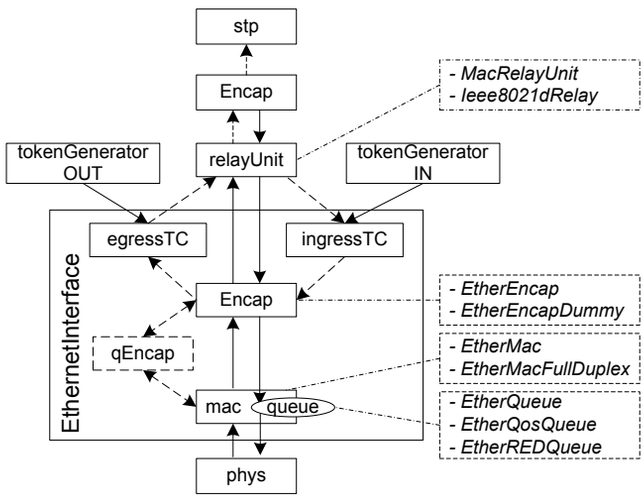
Рассмотрим базовую модель коммутатора, представленного в фреймворке INET OMNeT++. Коммутатор представлен набором модулей, взаимодействующих через систему передачи сообщений OMNeT. Наиболее подробно реализованымодул иинтерфейсов Ethernet, в частности имеются реализации входящей очередитипа Tail-Drop, исходящихочередейтипов FIFO, RED, WRED, PQ, CBFQ, также возможна реализация задержки передачи на линии. Однако в этой базовой модели отсутствует реализация или имитация процессов внутренней обработки пакетов, вся коммутация представленамодулем MacAddressTable с поиском адресов и модулем MacRelayUnit, просто передающим пакеты на исходящий интерфейс без задержек или имитирующем рассылку на выбранные или все порты.

На рисунке 1 показана схема прохождения пакета через коммутатор стандартного модуля EtherSwitch, который используют в большинстве моделей.

Блок “mac” реализует функции подуровня MediaAccessControl (MAC) канального уровня с использованием технологии IEEE 802.3CSMA/CD. При посылке кадра с верхних уровней кадры ставятся в очередь для передачи (внутренний submodule), кроме того, по запросу верхних уровней может быть отправлено сообщение PAUSE. Кадры, поступающие из сети, обрабатываются следующим образом: проверяется CRC (кадры с ошибкой отбрасываются), кадры PAUSE получают ответ, в режиме promiscuous (по умолчанию для коммутатора) отправляются все полученные кадры, в противном случае (режим хоста), только кадры с совпадающими MAC-адресами и широковещательные кадры. Может быть установлен полудуплексный (EherMac) или полнодуплексный (EtherMacFullDuplex) режимы работы уровня mac.

[1] NetSim User Manual, 2019. [Электронный ресурс]. URL: https://www.tetcos.com/downloads/v11/NetSim_User_Manual.pdf (дата обращения: 22.09.2020).

[2] Veins: Vehicles in Network simulation. [Электронный ресурс]. URL: https://veins.car2x.org (дата обращения: 22.09.2020).



Р и с. 1. Схема прохождения пакета сквозь коммутатор EtherSwitch фреймворка INET

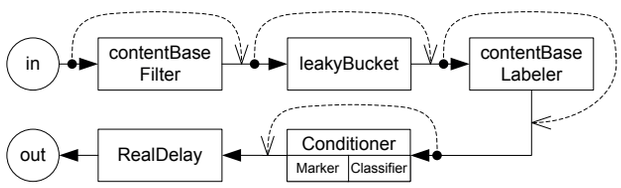
F i g. 1. Diagram of the packet passing through the EtherSwitch of the INET framework

Блок qEncap предназначен для обработки тегов протокола IEEE 802.11q (VLAN). Для обработки используется фильтры VLANID (для обеспечения работы только выбранных VLAN), а также правила перезаписи тегов для добавления и удаления тегов в разных направлениях.

Блок Encap выполняет функции подуровня LLC канального уровня. Пакеты, поступающие с верхних уровней, инкапсулируются в кадр Ethernet и отправляются в MAC. Кадры Ethernet, поступающие от MAC, декапсулируются и отправляются на более высокие уровни. Для блока Encap могут быть установлены режимы EtherEncap (поддерживает реальные кадры, которые можно передать в систему моделирование как PCAP файлы с дампом трафика) или EtherEncapDummy (пакеты со случайным внутренним содержимым).

Модули ingressTC и egressTC — входящий и исходящий формирователь трафика (Traffic conditioners). TrafficConditioner не являются стандартными модулями и должны быть созданы разработчиком конкретных моделей. Они могут классифицировать входящие пакеты, измерять трафик в каждом классе, помечать или отбрасывать пакеты в зависимости от результата измерения, формировать трафик в соответствии с желаемым профилем трафика. Формирование входящего и исходящего TrafficConditioners будет описано ниже. Для внешнего управления поведением этих блоков введены модули tokenGeneratorIN и tokenGeneratorOUT, о которых будет рассказано ниже.

Для более детальной реализации модели были разработаны следующие схемы дополнительной обработки трафика. Поскольку на коммутаторе опционально могут быть настроены такие опции, как фильтрация пакетов посредством списков ACL, профилировщики policing и shaping, классификация и модификация пакетов (рис.2). Все эти опции проходятся по пути следования пакета последовательно, внося задержку. Поскольку в реальном коммутаторе сложно или невозможно оценить задержку каждого элемента на разных нагрузках, принято решения добавить модуль внесения агрегированной задержки “RealDelay”.



Р и с. 2. Схема фильтрации трафика

F i g. 2. Traffic filtering scheme

Фреймворк INET позволяет реализовать указанные features с помощью следующих модулей:

- contentBaseFilter представляет собой модуль, отбрасывающий пакеты на основе данных, которые они содержат, аналог AccessControlList;
- leakyBucket — универсальный профилировщик с переполнением и настраиваемой скоростью вывода, реализует параметризуемый алгоритм с утечкой, аналог TrafficShaper.
- contentBaseLabeler прикрепляет метки к пакетам на основе данных, которые они содержат, аналог Class-Map.

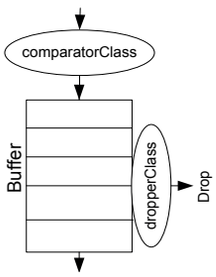
В качестве модификатора может быть использован модуль Conditioner (обработчик по условию), состоящий из двух частей: Classifier и Marker. Classifier — содержащий список фильтров классификатор (модуль для перезаписи полей протоколов пакета, например DSCP), который идентифицирует потоки, определяет их классы, делает замену полей классов. Каждый фильтр может совпадать с адресом источника и назначения, номером протокола IP, портами источника и назначения или ToS/CoS/DSCP. Первый соответствующий фильтр определяет индекс выхода. Если соответствующий фильтр не найден, пакет будет отправлен через шлюз defaultOut. Marker — модуль, устанавливающий в поле DSCP (младшие шесть бит Tos/TrafficClass) IP-дейтаграмм значение, указанное параметром dscps.

RealDelay — разработанный для имитации задержки модуль, вносимой задержкой при прохождении пакетом описанных блоков. Подробнее будет описан ниже.

В стандартном коммутаторе EtherSwitch без дополнительных настроек возникновение очереди возможно только в подмодуле queue модуля mac при прохождении пакета от верхних уровней к нижним.

В маршрутизаторах модуль MAC использует модуль внешней очереди для моделирования конечного буфера, реализации QoS и/или RED и запроса пакетов из этой внешней очереди. В хостах такая очередь не используется, поэтому MAC содержит внутреннюю очередь для хранения пакетов, ожидающих передачи. Концептуально очередь бесконечного размера, но для лучшей диагностики в параметре packetCapacity можно указать жесткий предел, при превышении которого симуляция останавливается с ошибкой.

По умолчанию очередь имеет тип EtherQueue. Схематично эта очередь показана на рисунке 3.



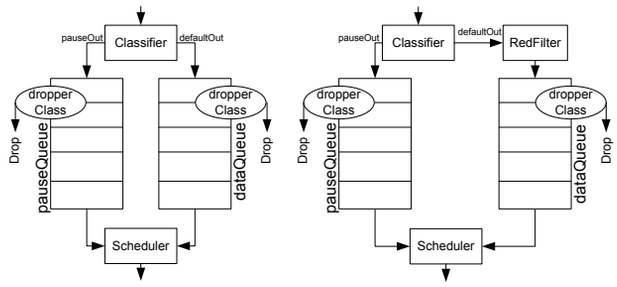
Р и с. 3. Очередь EtherQueue
F i g. 3. EtherQueue

Класс comporatorClass определяет порядок пакетов в очереди и порядок вставки их в очередь. Для очереди EtherQueue он имеет значение “inet::EthernetFrameTypeComparator”, что обеспечивает приоритет кадрам PAUSE Ethernet. Параметр dropperClass определяет, какие пакеты отбрасываются при переполнении очереди сначала.

Кроме очереди EtherQueue могут быть установлены очереди EtherQosQueue(рисунок 4a) и EtherQosRedQueue (рисунок 4b).

Модуль очереди EtherQosQueue дает кадрам PAUSE более высокий приоритет. Для обслуживания кадров данных он может параметризоваться с помощью интерфейса IPacketQueue. Модуль EtherQosQueue состоит из классификатора, двух очередей и планировщика [15]. Классификатор пакетов — это модуль, который имеет один пассивный вход и несколько активных выходов. Пакеты, отправленные в пассивный вход, передаются на один из активных выходов без какой-либо задержки и переупорядочения. По умолчанию в модуле EtherQosQueue включен классификатор, который перенаправляет кадры Ethernet PAUSE очередь pauseOut, а другие кадры в очередь defaultOut. Обе эти очередиявляются очередями типа DropTailQueue, т.е. представляют собой ограниченную очередь пакетов, которая отбрасывает пакеты в хвост очереди. Планировщик пакетов — это модуль, который имеет несколько активных входов и один пассивный выход. Пакеты, извлеченные из пассивного выхода, предоставляются одним из входов без какой-либо задержки и переупорядочения. По умолчанию в модуле EtherQosQueue включен планировщик, выталкивающий пакеты из первого непустого источника пакетов.

Модуль EtherQosRedQueue отличается от EtherQosQueue использованием алгоритма случайного раннего обнаружения для кадров данных[3] [16]. Пакеты, поступившие в i-й входной шлюз, пересылаются в i-й выходной шлюз или отбрасываются. Модуль суммирует используемое буферное пространство очередей, прикрепленных к выходным шлюзам. Если он ниже минимального порога, пакет не будет отброшен, если выше максимального порога, он будет отброшен, если он находится между минимальным и максимальным порогом, он будет отброшен с определенной вероятностью.



a) b)

Р и с. 4. Прохождение пакета через очереди EtherQosQueue (a) и EtherQosRedQueue (b)
F i g. 4. Packet passing through EtherQosQueue (a) and EtherQosRedQueue (b)

Модель коммутатора с поддержкой OpenFow

Различные решения по моделированию OpenFlow оборудования имеют источником одного предка[4] [17,21], который не работает на новых версиях системы. Для обеспечения работоспособности на версии INET 4.3 пришлось внести более 100 изменений в исходный код C++, а также в файлы описания элементов, измененный дистрибутив расположен в[5] [18-20]. Были добавлены MessageDispatcher модули для связи между уровнями со многими входами, также был приведен к существующим классам весь исходный код, логика переделана для работы с новыми сущностями пакетов и тегами, которые теперь содержат метаданные сетевого пакета. Версия поддерживаемого протокола OpenFlow — 1.5.1.

Модуль OpenFlow Swich структурно не отличается от обычного коммутатора со стороны физического соединения с сетью, однако имеет второе выделенное соединение для работы с контроллером. Сам модуль коммутации OF_Switch по логике работы существенно не отличается от модуля RelayUnit, который служит основой традиционного коммутатора. Модуль OF_Switch содержит механизм формирования реалистичных задержек обработки с параметром serviceTime (используется в режиме безусловной задержки любого пакета, пришедшего на вход, допускается использование генераторов для задания случайного времени по заданным законам распределения). Также в отличие от обычного коммутатора в OF_Switch есть возможность задания конечной очереди через параметр bufferCapacity, причем внутренний элемент buffer при этом используется только для общения с контроллером, а сами сообщения просто хранятся в списке типа std::list время serviceTime. На рис. 5 показана структура OpenFlow коммутатора из проекта[6] [17]. Как видно из рисунка, все коммуникации controlPlane для управления коммутатором осуществля-

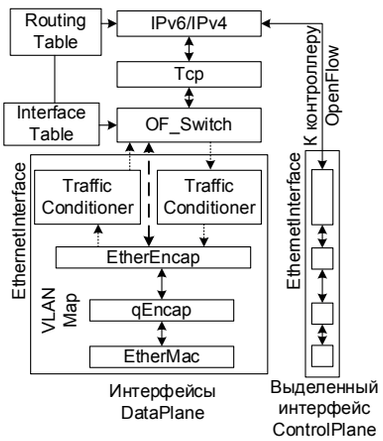
[3] RedDropper: INET Framework 4.2 documentation. [Электронный ресурс]. URL: https://doc.OMNeTpp.org/inet/api-current/neddoc/inet.queueing.filter.RedDropper.html (дата обращения: 22.09.2020).

[4] Hornig R. OpenFlow model for OMNeT++ 5.4 and INET 3.6. [Электронный ресурс]. URL: https://github.com/inet-framework/openflow (дата обращения: 22.09.2020).

[5] OpenFlow-OMNET-INET4 [Электронный ресурс]. URL: https://github.com/CrezZ/OpenFlow-OMNET-INET4 (дата обращения: 22.09.2020).

[6] Hornig R. OpenFlow model for OMNeT++ 5.4 and INET 3.6. [Электронный ресурс]. URL: https://github.com/inet-framework/openflow (дата обращения: 22.09.2020).

ются по выделенным каналам, что не всегда соответствует реальному коммутатору.



Р и с. 5. Структура коммутатора OpenFlow
F i g. 5. OpenFlow Switch Framework

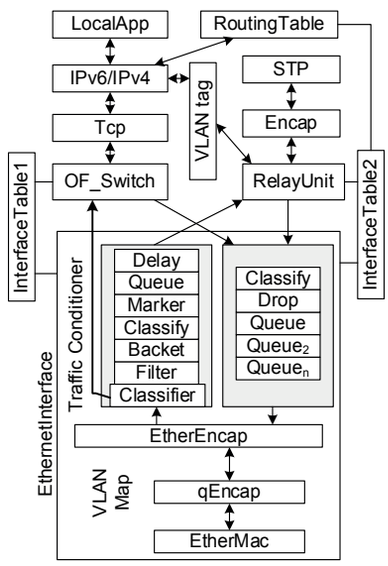
Кроме того, в большинстве случаев все коммутаторы гибридные, то есть совмещают как режим классической коммутации, так и различные режимы OpenFow обработки пакетов.

Реальный коммутатор с поддержкой OpenFlow имеет отличие по принципу коммутации интерфейсов с OpenFlow (по крайне мере, коммутаторы HP ProCurve, HPE Aruba, Cisco Catalyst, NetGear M). Поддержка OpenFlow является в таких коммутаторах одной из возможных технологий коммутации внутри VLAN, наряду с кешированием маршрутов, классической коммутацией 2 и 3 уровня, а также маршрутизацией между VLAN. Включение OpenFlow производится в выбранных VLAN, при этом классическая коммутация отключается не полностью, таблица MAC TCAM соответствия адресов продолжает наполняться, но не используется. Во многих коммутаторах (HP, HPE) OpenFlow управляет только решением о классификации, коммутации, маршрутизации, перезаписи полей пакета и генерации пакетов. Все что связано с очередями, обеспечением параметров QoS (кроме классификации и маркировки пакетов), ограничением пропускной способности, динамической пиритизацией в таких коммутаторах работает только в обычном режиме и OpenFlow не имеет доступа к управлению. Также связь с контроллером в реальных коммутаторах осуществляется через адрес IPv4/IPv6, установленный на одном из VLAN или интерфейсов с классической коммутацией, для связи используется традиционные протоколы ARP, маршрутизация через таблицу маршрутизации VLAN или VRF, фильтры ACL для контроля доступа к встроенному серверу OpenFlow.

На рис. 6 показана разработанная гибридная схема для коммутации, с учетом сосуществования на разных портах (группированных по разным VLAN) как коммутации через RelayModule, так и с обработкой пакетов в OF_Switch.

Основная сложность реализации такого коммутатора в OMNET++ — автоматическое заполнение списка интерфейсов модулем InterfaceTable, который будет видеть все интерфейсы коммутатора. Для решения этой проблемы добавлены 2 таблицы интерфейсов, одна только для OpenFlow, в нее интерфейсы добавляются по фильтру VLAN.

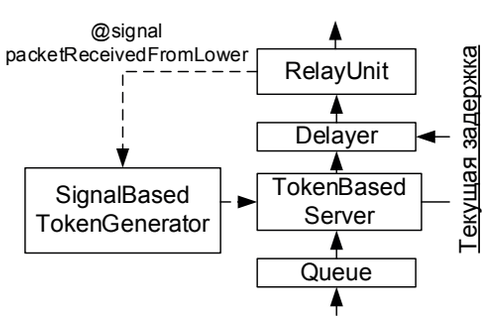
Разделение входящего трафика для обработки OpenFlow также происходит по VLANID с помощью ContentBasedClassifier при этом сохраняется возможность работы коммутатора по классической схеме. Исходящие пакеты обрабатываются одинаково и проходят через очереди, настроеные на интерфейсах. Один интерфейс может принимать пакеты как от OpenFlow DataPlane, OpenFlow ControlPlane так и от коммутации.



Р и с. 6. Гибридный коммутатор
F i g. 6. Hybrid Switch

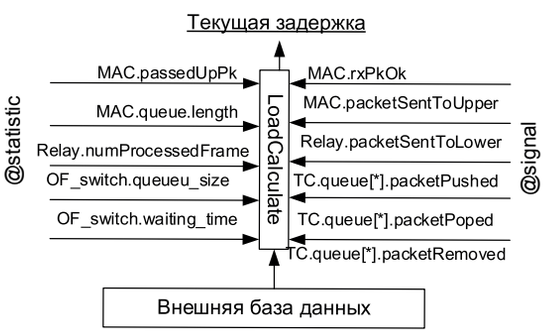
Управление коммутацией OpenFlow происходит через модуль связи с контроллером на основе модуля NetworkLayer, который по таблице маршрутизации решает, как послать пакет, формирует ARP запросы, принимает и фильтрует все IP пакеты, пришедшие в коммутатор, обрабатывает предназначенные для установленного IP адреса пакеты.

Для создания реалистичной задержки и ограничения буфера на входе в состав Traffic Conditioner был создан модуль RealDelay, который показан на рис. 7.



Р и с. 7. Модуль создания задержки и ограничения емкости и производительности RealDelay
F i g. 7. Module for creating latency and limiting capacity and performance RealDelay

Размер задержки можно задать как вероятностную функцию, например exponential(lambda), так и установить программным образом. Очередь эмулирует работу входного буфера обработки по механизму Tail drop (FIFO) с ограниченной емкостью. TokenBasedServer пропускает пакеты по сигналу SignalBasedTokenGenerator, который в свою очередь формирует сигнал при получении пакета коммутационным модулем relayUnit. Для формирования задержки по внешним данным, в частности по данным от реального коммутатора и данным от модельных модулей, был создан модуль LoadCalculate. На основе математической модели модуль производит расчет задержки на основе всех входящих данных (рис.8).



Р и с. 8. Модуль расчета задержки
F i g. 8. Delay calculation module

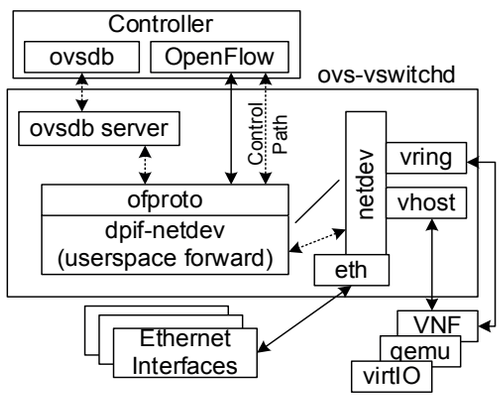
Данные модули совместимы с существующими моделями и могут быть использованы совместно в любых схемах с коммутаторами.

Модель коммутации NFV

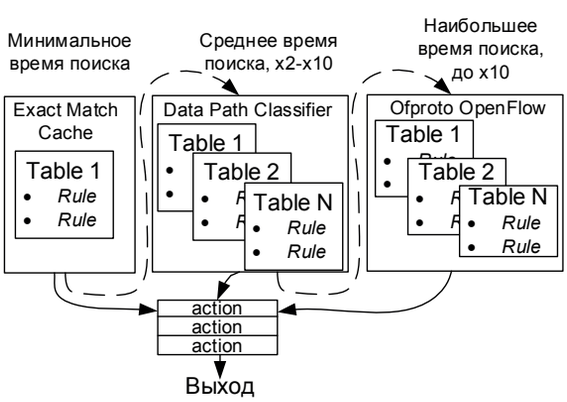
Поскольку Network Function Virtualization (NFV) построена на основе SDN, а наиболее Linux-based популярные решения используют в качестве SDN коммутатора OpenvSwitch, рассмотрим методы формирования модели такого коммутатора.

OpenvSwitch архитектурно отличается как от классического, так и от OpenFlow реального коммутатора. Поскольку нет выделенных блоков предобработки пакетов (кроме сетевых карт с аппаратной обработкой заголовков и FPGA), выделенной памяти и процессоров для коммутации, OpenvSwitch использует модуль ядра (или модуль DPDK для прямой передачи без участия ядра) для передачи пакетов по информации из базы данных OVSDB или OpenFlow таблиц. Поскольку количество внешних интерфейсов сервера существенно меньше таковых у коммутатора, входящий трафик не фильтруется автоматически по VLAN, не обрабатывается (кроме декапсуляции Ethernet) до попадания в модуль коммутации (рис. 9а).

Вся обработка трафика производится тем же процессором, на котором выполняются виртуальные машины NFV, что является существенным фактором внесения случайных задержек в обработку пакетов. Многих стандартных вещей в OVS нет, они выполняются средствами соответствующих Linux программ, например маршрутизация, маркировка пакетов, фильтрация. В последних версиях появилась маркировка пакетов средствами ovsdb, но большинство NFV сред использует OpenFlow для работы, управления, обеспечения QoS и ограничения полосы пропускания, а также для назначения соответствия заранее созданным очередям классов пакетов (рис. 9б).



а)



б)

Р и с. 9. Коммутация NFV:
а) Структура NFV коммутации; б) Поиск правил в OpenvSwitch.
F i g. 9. Switching NFV:
a) NFV switching structure; б) Finding rules in OpenvSwitch

Моделирование OpenvSwitch в составе NFV более сложный процесс, поскольку множество происходящих процессов скрыто за контроллером. Например, для нестационарных сетей 5G VANET, MANET, основным для работы сервисов может является VirtualPathComputation на основе данных о размещении NFV и топологии сети, кроме того, сам контроллер также может являться одним из элементов NFV, в том числе распределенным на несколько серверов.

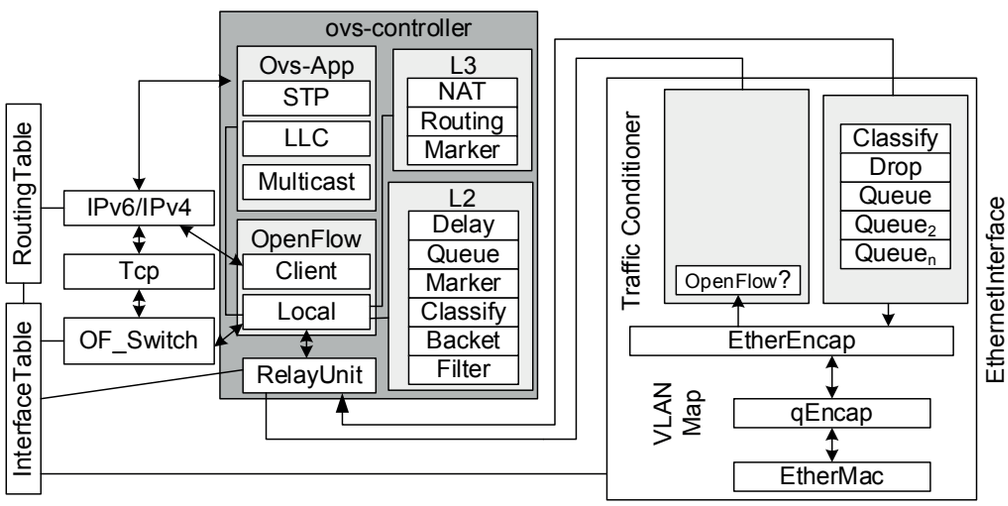
Поскольку OpenvSwitch использует общую фабрику для обработки всех правил, схема работы такого коммутатора более универсальная. Для создания модели такого коммутатора в системе OMNeT++ требуется ссовместить быструю коммутацию, стандартную L2/L3/L4 коммутацию и запоминание данных, а также OpenFlow подход [22].

Работа DataPathClassifier и OpenFlow схожи, поиск ведется по одинаковым полям, различаются только действия над пакета-

ми, которые специфицированы каждым типом таблиц. На рис. 10 показана схема OpenvSwitch коммутатора, модуль ovs-controller отвечает за быструю коммутацию при отключенном OpenFlow. Таблицы DatapathClassifier эмулируются режимом работы OpenFlow с локальным контроллером, при наличии правил LOCAL обращения к внешнему контроллеру не происходит. Если правило указывает на совершение действия, пакет проходит сквозь выбранный модуль L2 или L3, модуль Ovs-App принимает и генерирует пакеты для специфичных протоколов. При включении режима OpenFlow модуль OpenFlow начинает при поступлении пакета кроме локальных таблиц передавать данные OpenFlow таблиц для поиска, в конце списка [23-25].

Такой режим работы OpenFlow позволяет эмулировать весь богатый функционал OpenvSwitch кроме очередей, для них оставлен классический режим работы, хотя OpenvSwitch умеет организовывать собственные очереди и сам коммутировать пакеты в них.

Для изучения работы модельных устройств и сопоставления результатов с реальным оборудованием или системами виртуализации будет проведен эксперимент на каждом типе коммутатора.



Р и с. 10. Поиск правил в OpenvSwitch

F i g. 10. Finding rules in OpenvSwitch

Эксперимент

Для изучения работы модельных устройств и сопоставления результатов были выбраны имеющиеся в наличии устройства, для них проведены настройки в таблице 1.

Т а б л и ц а 1. **Оборудование для эксперимента**

T a b l e 1. **Experiment equipment**

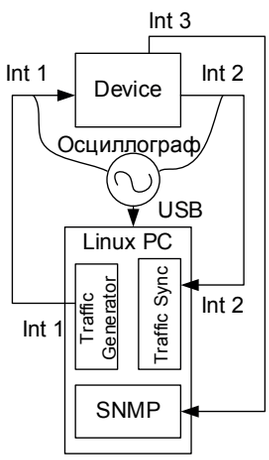
Класс устройства	**Модель оборудования**	
	Модель	***Настройки***
Коммутатор L2	Cisco Catalyst 3750	Single VLAN, RSTP Enabled, CEF Disabled, 2 interfaces
Коммутатор OpenFlow-only	HPE Aruba 3800	Single VLAN, OpenFlow Enabled, Reactive mode, Internal controller only, 2 interfaces
Гибридный коммутатор	HPE Aruba 3800	Two VLANs, One -OpenFlow Enabled, Second-OpenFlow disabled, Reactive mode, Internal controller only, 2 interfaces
Коммутатор OpenvSwitch	v2.11.1, Debian 10, DPDK 18.11.2, Xeon e5 2678 v3,	VLANs, One -OpenFlow Enabled, Second-OpenFlow disabled, Pre-defined OpenFlow rules 1 physical interface, 1 virtual interface

Версии ПО на устройствах установлены последние на момент эксперимента. Схема экспериментального исследования показана на рис. 11.

Каждый коммутатор прогонялся на трафике различных интенсивностей и размеров пакетов, результаты измерялись осциллографом и записывались в СУБД sqlite для дальнейшего использования в модуле рассчета задержки, как и данные об использовании памяти, процессора, входящих буферов и исходящих очередей. Поскольку целью эксперимента в первую очередь является изучения адекватности поведения задержки обработки пакета, особенно в зависимости от нагрузки на процессор, использовался осциллограф с USB выходом для точного измерения разницы во времени выхода пакета из генератора и выхода пакета из коммутатора. При этом контролиро-

валось исходящая очередь, которая могла внести искажения в измерения.

Таблица 2 показывает результаты измерения задержки на портах CopperEthernet 1Гбит/с различных размерах пакетов при указанных параметрах генерации трафика (экспоненциальный закон распределения времени между пакетами с условием минимальной паузы между пакетами).



Р и с. 11. Измерение задержки прохождения пакета

F i g. 11. Measurement of packet delay

Т а б л и ц а 2. **Результаты эксперимента на оборудовании**

T a b l e 2. **Experimental results with equipment**

Модель устройства	**Инт**	**Задержка, мкс**		
		Frame 64 байт	***Frame 512 байт***	***Frame 1518 байт***
Коммутатор L2	λ : Cpu:1%	4.20	4.23	4.30
Коммутатор OpenFlow-only	λ : Cpu:1%	179.51	181.60	208.10
Гибридный коммутатор	λ : Cpu:1%	180.31	182.20	210.01
Коммутатор OpenvSwitch	λ : Cpu:1%	32.99	37.40	47.72

Для апробации моделей была собрана простейшая схема из генератора трафика (UdpHost), в котором реализован генератор трафика с заданным законом и параметрами распределения, размером пакета, портами и т.д. Latency в узле в OMNeT++ не вычисляются автоматически, из задержек имеется информация о queuingtime, в некоторых приложениях есть задержки end-to-end. Для расчета задержки, джиттера и других характеристик трафика нужна запись времени входа и выхода конкретного пакета в каждый интересующий нас модуль. Для исключения влияния очередей на изучения поведения задержки параметр packetCount во всех очередях установлен равным нулю.

Для обеспечения равных условий эксперимента в модели и эксперимента на оборудовании можно обеспечить два подхода:

1. Использовать аналогичные параметры генератора трафика эксперимента на оборудовании, если это возможно.
2. Использовать PCAP файлы с сохраненным исходным трафиком эксперимента на оборудовании.

Был выбран второй вариант для лучшей точности и дальнейшего использования в различных экспериментах.

Поскольку стандартный модуль PcapReader не имеет возможности работать с абсолютным временем, вычисляя из него относительное, а также жестко задает формат записанного времени с масштабом в микросекундах (SIMTIME_US). Поэтому при использовании PCAP файла, полученного с помощью Tcpdump требуется предварительная обработка времени для приведения времени первого пакета к нулю, а также, если требуется, изменения масштаба времени. Для этого можно использовать утилиты из комплекта WireShark. С помощью команды

«time0=$(tshark -te -r file.pcap -c 1| cut -d’ ‘ -f1)»

или

«time0=$(tcpdump -r file.pcap -tt -c1 | cut -d’ ‘ -f1)»

необходимо получить Unixtimestamp для первого пакета. В Windows системах можно использовать Powershell конструкцию

«$time0=(tshark -te -r file.pcap -c1).Split(“ “) [3]».

Затем утилитой editcap вычесть полученное время

«editcap –F pcap -t -$time0 file.pcapout.pcap»

При необходимости соединение с реальной сетью (только на Linux) происходит через модуль ExtInterface. При это создается виртуальный интерфейс veth, который и является входной и выходной точкой трафика в модель. Для этого необходимо добавить виртуальные интерфейсы veth, а также добавить второй интерфейс из пары в bridgebr0

«ip link add eth1-br0 type veth peer name eth1-gw1
brctladdif br0 peer1-br0»,

а также настроить маршрутизацию и iptables.

После этого можно генерировать трафик любыми утилитами, трафик попадет в модель через модуль ExtInterface.

Поскольку для повторения эксперимента был выбран первый путь с чтением из PCAP файла, эксперимент был автоматизирован скриптом на bash. При этом каждый новый прогон выполнялся с другим набором входных файлов. Выходные файлы были записаны через модуль PcapRecorder для каждого интерфейса коммутатора (модуля EtherEncap). Для сравнения в эксперимент были добавлены стандартные модели EtherSwitch и OF_Switch. Поскольку измерение задержки происходит от момента поступления на вход физического порта до момента выхода из физического порта коммутатора, учитываются все задержки, в том числе и на физическом уровне (добавляется задержка Ethernet канала), которая была вычтена из результата (стандартная задержка линии Ethernet для длины 10м — 5нс, задержка на прием фрейма — (длина в битах)/(скорость линии бит/с)). Для OpenFlow использовался «Реактивный» режим с заранее запомненными адресами и “action=output:port”.

Таблица 3. **Результаты эксперимента на оборудовании**
Table 3. **Experimental results with equipment**

Devicemodel	**Задержка, мкс**		
	Frame 64 байт	***Frame 512 байт***	***Frame 1518 байт***
L2 коммутатор с модулем RealDelay	4,07	4,06	4,17
OpenFlow-only коммутатор с модулем RealDelay	174,12	174,34	201,86
Гибридный коммуттатор с модулем RealDelay	174,90	174,91	203,71
OpenvSwitch с модулем RealDelay	32,00	35,90	46,29
Модель INET4 EtherSwitch без формирователя трафика	0	0	0
Модель INET4 OF_Switch с параметром serviceTime=0	0	0	0

Как видно из таблицы 3 при использовании стандартной модели коммутатора и вычитании задержек физического уровня не представляется возможным исследовать задержки в сети на таких моделях. Для созданных моделей погрешность в средних не превышает 10%, при этом такой высокий показатель может быть связан с PCAP форматом и обработкой реальных пакетов канальным уровнем.
На рисунке 12 показаны результаты в графическом виде.

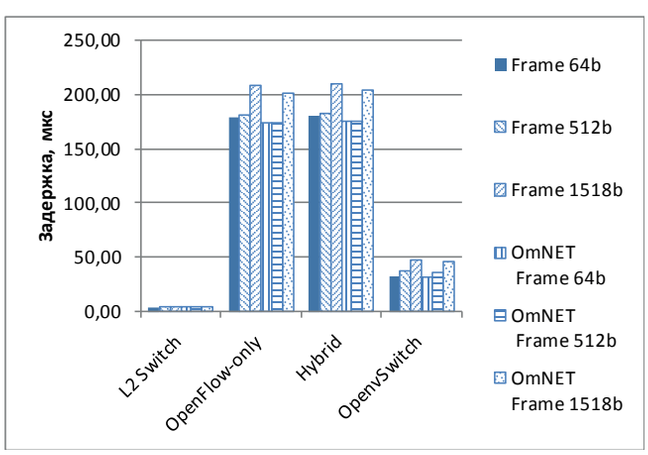


Рис. 12. Результаты измерения задержки пакетов
Fig. 12. Packet Delay Measurement Results

Как видно из графиков, общая идея о добавлении модуля задержки на основе экспериментальных исследований оборудования позволяет создавать модели с достаточной точностью для проведения исследований.

Выводы

В результате исследования был проведен обзор систем моделирования с поддержкой SDN, выбрана система OmNET++. На этой базе рассмотрен процесс прохождения пакета через коммутатор EtherSwitch фреймворка INET. Также были изучены возможности фреймворка INETOMNeT++ для реализации реалистичных задержек, возникающих при прохождении пакета через коммутатор.
Рассмотрены решения по моделированию OpenFlow оборудования и разработана своя модель OpenFlow коммутатора, отличающаяся возможностью управлять задержками по результатам работы собственной математической модели.
Разработана схема работы гибридного коммутатора, поддерживающего как традиционную коммутацию, так и коммутацию OpenFlow, которая позволяет решать комплексные задачи в физической инфраструктуре визуализированных сетей. Разработан модуль создания реалистичной задержки и ограничения емкости и производительности модели гибридного коммутатора, который может работать по данным, полученным в результате экспериментального исследования.
Исследована структура NFV коммутации и разработана модель виртуального коммутатора OpenvSwitch. Эта модель позволяет моделировать NFV инфраструктуру в общей схеме моделирования.
Экспериментальные исследования показали, что при использовании разработанных моделей показатели задержки модельного оборудования близки к исследованным на физических устройствах и виртуальном коммутаторе в рамках 10% по средним значениям, что позволяет вести дальнейшие исследования по автоматизации и улучшению таких моделей.

References

[1] Barakabitze A.A., Ahmad A., Mijumbi R., Hines A. 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks.* 2020; 167:106984. (In Eng.) DOI: https://doi.org/10.1016/j.comnet.2019.106984.

[2] Herrnleben S., Rygielski P., Grohmann J., Eismann S., Hoßfeld T., Kounev S. Model-Based Performance Predictions for SDN-Based Networks: A Case Study. In: H. Hermanns (ed.) Measurement, Modelling and Evaluation of Computing Systems. MMB 2020. *Lecture Notes in Computer Science.* 2020; 12040:82-98. Springer, Cham. (In Eng.) DOI: https://doi.org/10.1007/978-3-030-43024-5_6

[3] Chaves L.J., Garcia I.C., Madeira E.R.M. OFSwitch13: Enhancing ns-3 with OpenFlow 1.3 Support. In: *Proceedings*

of the Workshop on ns-3 (WNS3 '16). Association for Computing Machinery, New York, NY, USA; 2016. p. 33-40. (In Eng.) DOI: https://doi.org/10.1145/2915371.2915381

[4] Wang S. Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet. In: *2014 IEEE Symposium on Computers and Communications (ISCC)*. Funchal, Portugal; 2014. p. 1-6. (In Eng.) DOI: https://doi.org/10.1109/ISCC.2014.6912609

[5] Wang S., Chou C., Yang C. EstiNet openflow network simulator and emulator. *IEEE Communications Magazine*. 2013; 51(9):110-117. (In Eng.) DOI: https://doi.org/10.1109/MCOM.2013.6588659

[6] Erel M., Teoman E., Özçevik Y., Seçinti G., Canberk B. Scalability analysis and flow admission control in mininet-based SDN environment. In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. San Francisco, CA, USA; 2015. p. 18-19. (In Eng.) DOI: https://doi.org/10.1109/NFV-SDN.2015.7387396

[7] Asadollahi S., Goswami B., Raoufy A.S., Domingos H.G.J. Scalability of software defined network on floodlight controller using OFNet. In: *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. Mysuru, India; 2017. p. 1-5. (In Eng.) DOI: https://doi.org/10.1109/ICEECCOT.2017.8284567

[8] Baldoni G., Lombardo A., Melita M., Micalizzi S., Rametta C., Vassallo A. An emulation framework for SDN-NFV based services. In: *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing (ICC '17)*. Association for Computing Machinery, New York, NY, USA; 2017. Article 135. p. 1-8. (In Eng.) DOI: https://doi.org/10.1145/3018896.3036378

[9] Lee S., Ali J., Roh B. Performance Comparison of Software Defined Networking Simulators for Tactical Network: Mininet vs. OPNET. In: *2019 International Conference on Computing, Networking and Communications (ICNC)*. Honolulu, HI, USA; 2019. p. 197-202. (In Eng.) DOI: https://doi.org/10.1109/ICCNC.2019.8685572

[10] Salih M.A., Cosmas J., Zhang Y. OpenFlow 1.3 Extension for OMNeT++. In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing*; Pervasive Intelligence and Computing. Liverpool, UK; 2015. p. 1632-1637. (In Eng.) DOI: https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.246

[11] Jia D., Sun J., Sharma A., Zheng Z., Liu B. Integrated simulation platform for conventional, connected and automated driving: A design from cyber–physical systems perspective. *Transportation Research Part C: Emerging Technologies*. 2021; 124:102984. (In Eng.) DOI: https://doi.org/10.1016/j.trc.2021.102984

[12] Sommer C., German R., Dressler F. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing*. 2011; 10(1):3-15. (In Eng.) DOI: https://doi.org/10.1109/TMC.2010.133

[13] Amoozadeh M., Ching B., Chuah C.-N., Ghosal D., Zhang H.M. VENTOS: Vehicular Network Open Simulator with Hardware-in-the-Loop Support. *Procedia Computer Science*. 2019; 151:61-68. (In Eng.) DOI: https://doi.org/10.1016/j.procs.2019.04.012

[14] Nardini G., Stea G., Virdis A., Sabella D. Simu5G: A System-level Simulator for 5G Networks. In: *Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Volume 1: SIMULTECH. 2020. p. 68-80. (In Eng.) DOI: https://doi.org/10.5220/0009826400680080

[15] Shi X., Wang L., Zhang F., Zheng K., Mühlhäuser M., Liu Z. PABO: Mitigating congestion via packet bounce in data center networks. *Computer Communications*. 2019; 140-141:1-14. (In Eng.) DOI: https://doi.org/10.1016/j.comcom.2019.04.002

[16] Amarasinghe G., de Assunção M.D., Harwood A., Karunasekera S. ECSNeT++ : A simulator for distributed stream processing on edge and cloud environments. *Future Generation Computer Systems*. 2020; 111:401-418. (In Eng.) DOI: https://doi.org/10.1016/j.future.2019.11.014

[17] Sonmez C., Ozgovde A., Ersoy C. EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. In: *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC*). Valencia, Spain; 2017. p. 39-44. (In Eng.) DOI: http://dx.doi.org/10.1109/FMEC.2017.7946405

[18] Klein D., Jarschel M. An OpenFlow Extension for the OMNeT++ INET Framework. In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques (SimuTools '13)*. ICST, Brussels, BEL; 2013. p. 322-329. (In Eng.) DOI: http://dx.doi.org/10.4108/icst.simutools.2013.251722

[19] Sudheera K.L.K., Ma M., Ali G.G.M.N., Han Joo Chong P. Delay efficient software defined networking based architecture for vehicular networks. In: *2016 IEEE International Conference on Communication Systems (ICCS)*. Shenzhen, China; 2016. p. 1-6. (In Eng.) DOI: https://doi.org/10.1109/ICCS.2016.7833564

[20] Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. 2016; 3(5):637-646. (In Eng.) DOI: https://doi.org/10.1109/JIOT.2016.2579198

[21] Sierszeń A., Przyłucki S. Software-Defined Automatization of Virtual Local Area Network Load Balancing in a Virtual Environment. In: M. Choraś, R. Choraś (ed.) Image Processing and Communications Challenges 10. IP&C 2018. *Advances in Intelligent Systems and Computing*. 2019; 892:151-160. Springer, Cham. (In Eng.) DOI: https://doi.org/10.1007/978-3-030-03658-4_18

[22] Varga A. A Practical Introduction to the OMNeT++ Simulation Framework. In: Virdis A., Kirsche M. (ed.) Recent Advances in Network Simulation. *EAI/Springer Innovations in Communication and Computing*. Springer, Cham; 2019. (In Eng.) DOI: https://doi.org/10.1007/978-3-030-12842-5_1

[23] Nkenyereye L., Nkenyereye L., Adhi Tama B., Reddy A.G., Song J-S. Software-Defined Vehicular Cloud Networks: Architecture, Applications and Virtual Machine Migration.

Sensors. 2020; 20(4):1092. (In Eng.) DOI: https://doi.org/10.3390/s20041092

[24] Jiau M., Huang S., Hwang J., Vasilakos A.V. Multimedia Services in Cloud-Based Vehicular Networks. I*EEE Intelligent Transportation Systems Magazine*. 2015; 7(3):62-79. (In Eng.) DOI: https://doi.org/10.1109/MITS.2015.2417974

[25] Bera S., Misra S., Vasilakos A.V. Software-Defined Networking for Internet of Things: A Survey. *IEEE Internet of Things Journal*. 2017; 4(6):1994-2008. (In Eng.) DOI: https://doi.org/10.1109/JIOT.2017.2746186

Поступила 22.09.2020; одобрена после рецензирования 06.11.2020; принята к публикации 16.11.2020.
Submitted 22.09.2020; approved after reviewing 06.11.2020; accepted for publication 16.11.2020.

Об авторах:

Ушакова Маргарита Викторовна, старший преподаватель кафедры геометрии и компьютерных наук, ФГБОУ ВО «Оренбургский государственный университет» (460018, Российская Федерация, г. Оренбург, пр. Победы, д. 13), ORCID: http://orcid.org/0000-0003-4462-9946, m.v.ushakova@mail.ru

Ушаков Юрий Александрович, доцент кафедры геометрии и компьютерных наук, ФГБОУ ВО «Оренбургский государственный университет» (460018, Российская Федерация, г. Оренбург, пр. Победы, д. 13), кандидат технических наук, доцент, ORCID: http://orcid.org/0000-0002-0474-8919, unpk@mail.ru

Болодурина Ирина Павловна, заведующий кафедрой прикладной математики, ФГБОУ ВО «Оренбургский государственный университет» (460018, Российская Федерация, г. Оренбург, пр. Победы, д. 13), доктор технических наук, профессор, ORCID: http://orcid.org/0000-0001-7747-646X, prmat@mail.osu.ru

Коннов Андрей Леониович, доцент кафедры управления и информатики в технических системах, ФГБОУ ВО «Оренбургский государственный университет» (460018, Российская Федерация, г. Оренбург, пр. Победы, д. 13), кандидат технических наук, доцент, ORCID: http://orcid.org/0000-0001-6096-941X, andrey_konnov@mail.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

About the authors:

Margarita V. Ushakova, Lecturer of the Department of Geometry and Computer Science, Orenburg State University (13 Pobeda Ave., Orenburg 460018, Russian Federation), ORCID: http://orcid.org/0000-0003-4462-9946, m.v.ushakova@mail.ru

Yury A. Ushakov, Associate Professor of the Department of Geometry and Computer Science, Orenburg State University (13 Pobeda Ave., Orenburg 460018, Russian Federation), Ph.D. (Engineering), Associate Professor, ORCID: http://orcid.org/0000-0002-0474-8919, unpk@mail.ru

Irina P. Bolodurina, Head of the Department of Applied Mathematics, Orenburg State University (13 Pobeda Ave., Orenburg 460018, Russian Federation), Dr.Sci. (Technology), Professor, ORCID: http://orcid.org/0000-0001-7747-646X, prmat@mail.osu.ru

Andrey L. Konnov, Associate Professor of the Department of Management and Informatics in Technical Systems, Orenburg State University (13 Pobeda Ave., Orenburg 460018, Russian Federation), Ph.D. (Engineering), Associate Professor, ORCID: http://orcid.org/0000-0001-6096-941X, andrey_konnov@mail.ru

All authors have read and approved the final manuscript.