Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

На правах рукописи

Матвеева Анна Павловна

МОДЕЛИ И АЛГОРИТМЫ ОБЕСПЕЧЕНИЯ ДОСТУПНОСТИ В КОРПОРАТИВНОЙ ПРОГРАММНО-ОПРЕДЕЛЯЕМОЙ ТЕЛЕКОММУНИКАЦИОННОЙ СЕТИ

Специальность: 2.2.15 – Системы, сети и устройства телекоммуникаций

ДИССЕРТАЦИЯ

на соискание ученой степени
кандидата технических наук

Научный руководитель:
Монахов Михаил Юрьевич,
Доктор технических наук, профессор

Владимир 2022

Содержание

ВВЕДЕНИЕ

Актуальность темы. Одной из важнейших характеристик функционирования и предоставления услуг в рамках телекоммуникационной сети является доступность, так как никакая другая характеристика сети не будет иметь значения в условиях невозможности обеспечения своевременного доступа к ресурсам сети. Особенно это актуально для корпоративных сетей, где невозможность выполнения сетевых задач за директивное время может оказать негативное влияние на прибыль организации. Традиционные сети уже не в состоянии обеспечить данную характеристику должным образом в силу ряда причин, как то: рост трафика, приводящий к перегрузкам, устаревание протоколов, конфликты оборудования разных вендоров, сложности конфигурирования и др. В качестве решения ряда проблем, стоящих в настоящее время перед телекоммуникационными сетями, рассматривают переход к архитектуре программно-определяемых сетей, которая позволяет перевести сетевые элементы под контроль настраиваемого программного обеспечения. SDN дает возможность изменять и вносить желаемую функциональность в логику работы сети. Однако, возможности, предоставляемые SDN по перестройке топологии, используются в настоящий момент не до конца.

При этом, современные корпоративные телекоммуникационные сети (КТС) характеризуются использованием все большего числа сетевых сервисов. Фундаментальной проблемой здесь становится повышение качества обслуживания в новых условиях, которые продиктованы нуждами прикладного уровня. Критичной характеристикой многих важных для функционирования КТС сетевых сервисов является время отклика, однако современные протоколы не в состоянии обеспечить данную характеристику, поскольку не содержат данного критерия. Поэтому для того, чтобы прикладной уровень эффективно использовал сетевую инфраструктуру, требуется внедрять дополнительные критерии качества, такие как доступность.

Таким образом, актуальность исследования доступности программно-определяемых сетей обуславливается необходимостью учета данного критерия для обеспечения эффективного функционирования корпоративной сети при построении с нуля или переходе с уже существующей топологии.

Степень разработанности темы. Вопросам организации, управления и масштабируемости SDN посвящены работы ведущих российский и зарубежных ученых Парамонова А.И., Перепелкина Д.А., Бурдонова И.Б., Ушакова Ю.А., Егорова В.Б., Захарова А.А., Bhandarkar S., Hu J., Oliveira A.T., Mondal A., Tuncer D. Проблема обеспечения качества обслуживания в телекоммуникационных сетях исследовалась в трудах таких ученых, как Парамонов А.И., Абросимов Л.А., Перепелкин Д.А., Гончаров А.А., Султанов Т.Г., Богданова Н.В., Devera M., Balan D., Domanska J., Stanwood K. L., Keith S., C. Douligeris, Vegesna S., Ma Q.

Объект исследования - корпоративные программно-определяемые телекоммуникационные сети (КПТС).

Предметом исследования являются модели и алгоритмы обеспечения качества обслуживания трафика на основе доступности узлов и каналов связи

Цель работы состоит в повышении эффективности обслуживания трафика корпоративных программно-определяемых телекоммуникационных сетей, заключающемся в повышении показателя доступности как компонентов, так и сети в целом за счет разработки и внедрения новых алгоритмов управления SDN.

В связи с поставленной целью решались следующие **задачи** исследования:

1. Проанализировать существующие решения задачи повышения доступности корпоративной программно-определяемой телекоммуникационной сети и ее компонентов и методик ее оценки.

2. Разработать алгоритм оптимизации топологии программно-определяемой телекоммуникационной сети по критерию максимума интегрального показателя доступности.

3. Разработать алгоритм планирования очередей передачи данных в программно-определяемой телекоммуникационной сети, позволяющий

оптимизировать использование пропускной способности и обеспечивать максимальную доступность поддерживаемых сервисов.

4. Разработать инструментальные средства для оценки адекватности полученных решений.

Научная новизна проведенных исследований и полученных в работе результатов заключается в следующем:

1. Разработан алгоритм оптимизации топологии программно-определяемой телекоммуникационной сети, основанный на последовательной реконфигурации топологии сетевых средств коммутации и маршрутизации по критерию максимума интегрального показателя доступности, что позволяет подстраивать топологию программно-определяемой телекоммуникационной сети под изменяющиеся внешние условия и решаемую задачу.

2. Разработан алгоритм планирования очередей передачи данных в программно-определяемой телекоммуникационной сети на основе модификации известного подхода «маркерное ведро» (HTB). Алгоритм позволяет обеспечивать минимально возможную задержку для приоритетных классов поддерживаемых сервисов, оптимизируя использование пропускной способности.

3. Разработан алгоритм поддержки низкоприоритетных сервисов в условиях сильного доминирования высокоприоритетных сервисов, основанный на перераспределении токенов управления потоком, что позволяет обеспечить принцип справедливости в отношении всех сервисов, работающих в программно-определяемой телекоммуникационной сети.

На защиту выносятся:

1. Алгоритм оптимизации топологии программно-определяемой телекоммуникационной сети, основанный на последовательной реконфигурации топологии сетевых средств коммутации и маршрутизации, повышающий интегральный показатель доступности и позволяющий подстраивать топологию программно-определяемой телекоммуникационной сети под изменяющиеся внешние условия и решаемую задачу.

2. Алгоритм планирования очередей передачи данных на основе модификации известного подхода «иерархическое ведро маркеров», позволяющий оптимизировать использование пропускной способности программно-определяемой телекоммуникационной сети и обеспечивать минимально возможную задержку для приоритетных классов поддерживаемых сервисов.

3. Алгоритм поддержки низкоприоритетных сервисов, позволяющий обеспечить принцип справедливости в отношении всех сервисов, работающих в программно-определяемой телекоммуникационной сети.

Практическая значимость работы. Создан программно-аппаратный стенд в среде Mininet для проведения экспериментов, позволяющий формировать произвольные топологии SDN, осуществлять маршрутизацию потоков трафика на базе контроллера ONOS, а также производить расчет показателей доступности. Эксперименты позволили выявить существенные факторы воздействия на топологию в программно-определяемых сетях с высокой доступностью. Разработано программное обеспечение, позволяющее рассчитывать интегральный показатель доступности КПТС (свидетельство о государственной регистрации программы для ЭВМ № 2022614981), находить оптимальные топологии КПТС по данному критерию (свидетельство о государственной регистрации программы для ЭВМ № 2022614982), а также производить различные тесты над топологиями (свидетельство о государственной регистрации программы для ЭВМ № 2022618511). Разработана имитационная модель, моделирующая работу алгоритма управления потоком, а также алгоритма поддержки низкоприоритетных сервисов. Осуществлена реализация алгоритма управления потоком в КПТС в виде модуля ядра операционной системы Linux.

В целом, предложенные алгоритмы и разработанные средства позволяют обеспечить повышение доступности от 10 до 22%, при обеспечении гарантированных задержек для высокоприоритетных сервисов и обеспечения справедливости при поддержке низкоприоритетных сервисов КПТС.

Методология и методы исследования. Научные положения работы теоретически обосновываются при помощи аппарата теории вероятностей, теории графов, теории систем массового обслуживания, математической статистики, математического анализа, методов компьютерного моделирования и технологии объектно-ориентированного программирования. Для практической проверки работоспособности предложенных алгоритмов использовалось разработанное программное обеспечение.

Соответствие паспорту специальности. Проблематика, исследованная в диссертации, соответствует областям исследований 4, 5 паспорта специальности 2.2.15 – «Системы, сети и устройства телекоммуникаций»

Достоверность и апробация.

Степень достоверности результатов исследования подтверждается рядом экспериментов, проводимых на исследуемых моделях с соблюдением требуемых условий случайности, и при помощи исследований, выполненных на экспериментальных установках, положительным результатом практического использования разработанных средств, а также апробацией в печати и на научных конференциях различного уровня.

Научно-практическая значимость работы подтверждена рецензируемыми публикациями в журналах и в сборниках научных трудов, докладами на научных конференциях международного и российского уровня, а также государственными Свидетельствами РФ о регистрации программ для ЭВМ.

Практическая значимость работы подтверждена внедрением её результатов в инновационную научную и образовательную деятельность ВлГУ, а также в центр обработки данных системы образования Владимирской области и в корпоративные сети компаний ООО «Рунет бизнес системы» г. Москва и ООО «Контактон» г. Владимир.

Материалы диссертационной работы докладывались и обсуждались на следующих конференциях:

– V IEEE International Symposium on Smart and Wireless Systems в рамках серии международных научно-технических конференций International

Conferences On Intelligent Data Acquisition And Advanced Computing Systems, IDAACS-SWS 2020 (Дортмунд, Германия, 2020);

– XIV и XV Международных научно-технических конференциях IEEE «Динамика систем, механизмов и машин, Dynamics 2020, Dynamics 2021» (Омск, 2020, 2021);

– Intelligent Systems Conference 2019, IntelliSys 2019 (Лондон, Великобритания);

– 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019 (Мец, Франция, 2019);

– 2nd European Conference on Electrical Engineering and Computer Science, EECS 2018 (Берн, Швейцария);

– Международной научно-технической конференции «Перспективные технологии в средствах передачи информации, ПТСПИ-2019» (Владимир, 2019);

– IX Всероссийской научно-практической конференции по имитационному моделированию и его применению в науке и промышленности «Имитационное моделирование. Теория и практика, ИММОД-2019» (Екатеринбург, 2019);

– XI Всероссийской научно-практической конференции «Проблемы передачи информации в инфокоммуникационных системах» (Волгоград, 2021).

По результатам диссертационной работы опубликовано 17 научных работ, в том числе 3 в изданиях, рекомендованных ВАК, проиндексированы в международных базах Scopus и Web of Science – 8, получено 4 свидетельствf о регистрации программы для ЭВМ.

Личный вклад. Все результаты, изложенные в научно-квалификационной работе, получены автором лично или при его непосредственном участии. Постановка цели и задач, обсуждение планов исследований и полученных результатов выполнены совместно с научным руководителем.

Данное исследование проводилось, в том числе, в рамках работ по теме, поддержанной Российским фондом фундаментальных исследований № 18-07-01109 «Алгоритмы и протоколы оценки и контроля доступности в крупномасштабных телекоммуникационных сетях» и государственного задания, тема FZUN-2020-0013.

Структура и объем диссертационной работы. Диссертация состоит из введения, трех глав, заключения, списка обозначений и сокращений, списка использованных источников из 136 наименований, 6 приложений и содержит 114 страниц основного текста, иллюстрированного 42 рисунками, содержит 14 таблиц.

1 МЕТОДЫ ОБЕСПЕЧЕНИЯ ДОСТУПНОСТИ В КПТС. АНАЛИЗ ОБЪЕКТА ИССЛЕДОВАНИЯ

В данной главе приводятся объект и предмет исследования. Анализируются методы обеспечения доступности в КПТС, уточняются задачи исследования.

1.1 Объект и предмет исследования

Объектом настоящего диссертационного исследования являются корпоративные программно-определяемые телекоммуникационные сети. Приводят следующее определение корпоративным сетям [13]: это совокупность связанных между собой локальных сетей, охватывающих территорию, на которой размещено одно предприятие или учреждение в одном или нескольких близко расположенных зданиях. КПТС являются подтипом КТС, где сеть построена по архитектурному принципу SDN.

Программно-определяемые (или программно-управляемые, программно-конфигурируемые) сети SDN – это новая технология построения архитектуры компьютерных сетей, в основе которой лежит перенос функций управления (маршрутизаторов, коммутаторов) в отдельное программное обеспечение, которое функционирует на отдельном сервере. Таким образом, производится физическое разделение уровней управления и передачи данных. Компоненты SDN разделены на два класса: коммутаторы (на уровне передачи данных) и контроллеры (на уровне управления). Вся вычислительная нагрузка, логика управления сетью сосредоточена на контроллерах, которые решают задачи, связанные с маршрутизацией и управлением потоком, оставляя сетевым устройствам только функцию пересылки трафика. Таким образом путем внедрения технологии SDN в КТС вводится централизованная плоскость управления. Варианты реализации плоскости данных в КПТС могут быть следующими [41; 86]:

1. Построить плоскость данных на базе коммутаторов, поддерживающих протокол OpenFlow (протокол OpenFlow представляет собой открытый стандарт, поддерживаемый организацией Open Networking Foundation, ONF [93]). В данном случае взаимодействие контроллера SDN с сетевыми элементами обеспечивается через «южный» интерфейс (Southbound Interface, SBI) по Openflow. Такой подход называют Open SDN.

2. Реализовать физическую underlay-сеть на базе классических коммутаторов/маршрутизаторов, поверх физической сети построить overlay-сети (концепция сетевой фабрики). В таком случае физическая сеть обеспечивает только L2/L3 (модель OSI) связность между гипервизорами (физическими серверами). Overlay-сеть представляет собой виртуальную топологию из виртуальных коммутаторов/ маршрутизаторов (например, Open vSwitch [109], VPP [126], Switch Light vSwitch [49] и т.п.), соединенных между собой виртуальными каналами «точка-точка» (туннелями). SBI в этом случае может быть реализован на таких протоколах, как: OpenFlow, Extensible Messaging and Presence Protocol (XMPP) [128], проприетарные протоколы, к примеру OpFlex от Cisco [83].

Обобщенная структура SDN приведена на рисунке 1.1.

Второй подход [51; 90] к построению уровня передачи данных совмещает две технологии: SDN и NFV (Network Functions Virtualization, виртуализация сетевых функций). Важной частью инфраструктуры виртуализации сетевых функций (NFVI) является виртуальный коммутатор, или vSwitch. VSwitch – это программный уровень на сервере. На сервере также размещаются виртуальные машины (ВМ) или контейнеры с виртуальными портами Ethernet (vNIC). Эти порты подключаются к vSwitch через виртуальный интерфейс (vIF), и vSwitch направляет трафик между виртуальными машинами, которые находятся на одном сервере, через стойку или между центрами обработки данных. VSwitch может служить точкой входа в оверлейные сети, работающие поверх физических сетей, а также позволяет пересылать трафик между виртуальными машинами в условиях виртуализации многопользовательской сети, особенно при

развертывании многосерверной виртуализации. При использовании сетевых оверлеев инкапсулированные пакеты передаются через несколько коммутаторов vSwitch, которые пересылают пакеты поверх физической сети (рисунок 1.2).

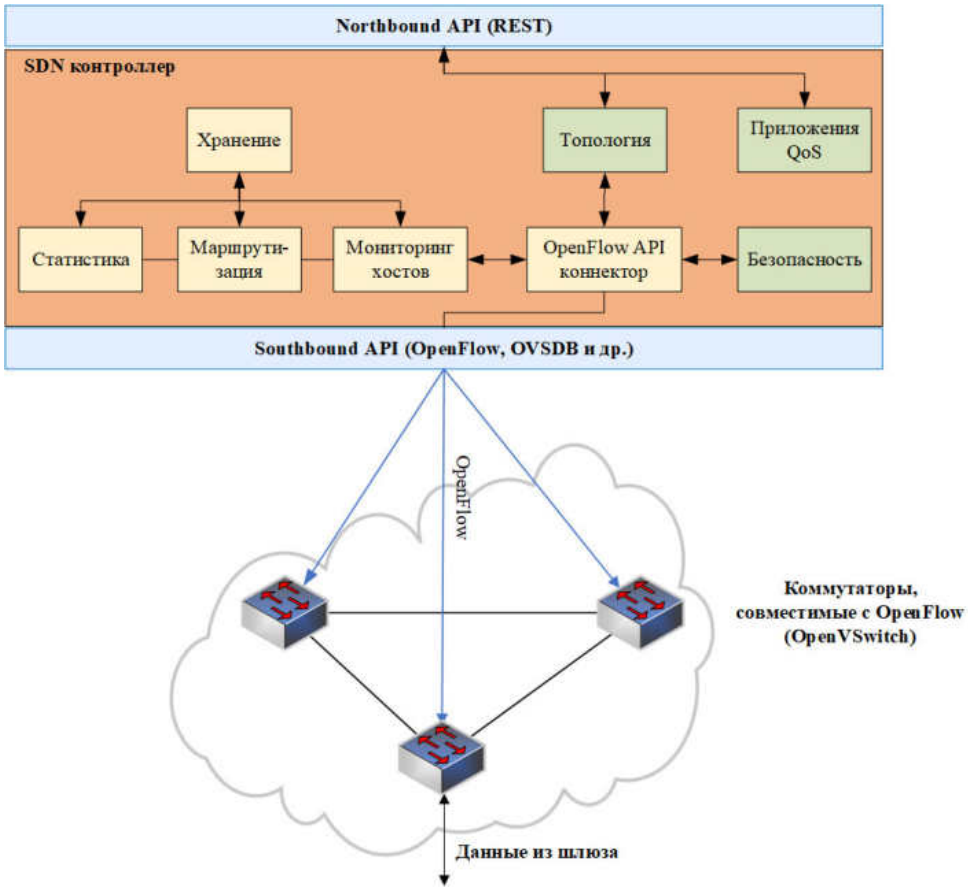

Рисунок 1.1 – Архитектура SDN.

КПТС свойственны следующие особенности, принципиальные в данном исследовании [2; 14; 30; 62]:

- территориальная распределенность, т.е. объединение офисов, подразделений и других структур, находящихся на значительном удалении друг от друга, в одну сеть;
- высокая степень разнородности телекоммуникационного оборудования и программного обеспечения, гетерогенность сети;
- большое разнообразие решаемых задач, и вследствие этого, разнородность сетевых сервисов с различными требованиями к эксплуатационным характеристикам телекоммуникационной сети;

- конвергентность, что означает, что от поставщика услуг связи трафик различного типа приходит по одному каналу передачи данных.

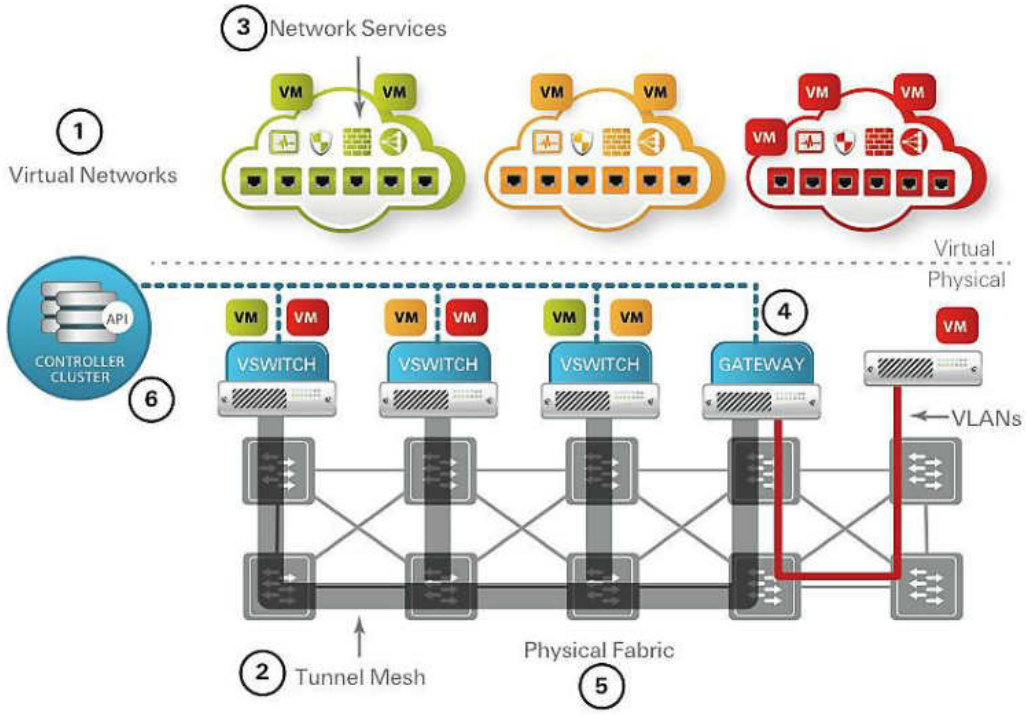

Рисунок 1.2 – Реализация принципов SDN с использованием виртуальных коммутаторов vSwitch, предложенная компанией Nicira.

Анализ существующих решений по построению КТС позволяет сформировать ее типовую схему [3; 14; 54] (рисунок 1.3).

Типовая КТС включает несколько подсетей, выделяемых для изоляции трафика, сетевое оборудование (коммутаторы, маршрутизаторы, межсетевые экраны, точки беспроводного доступа), VPN каналы, канал доступа в Интернет, сервера и рабочие станции, включая мобильные устройства.

Данная схема актуальна при условиях фиксированной конфигурации, если же в КТС применяются SDN, то эта схема будет постоянно перестраиваться, но в то же время сохранять черты типовой [42]: сеть сегментирована, поэтому сохраняется необходимость в маршрутизации, есть изолированные сегменты, удаленные работники, которые подключаются через туннелирование, есть

совокупность сервисов, которая в целом для КПТС неизменна, изменяются при этом только топологические характеристики сети.

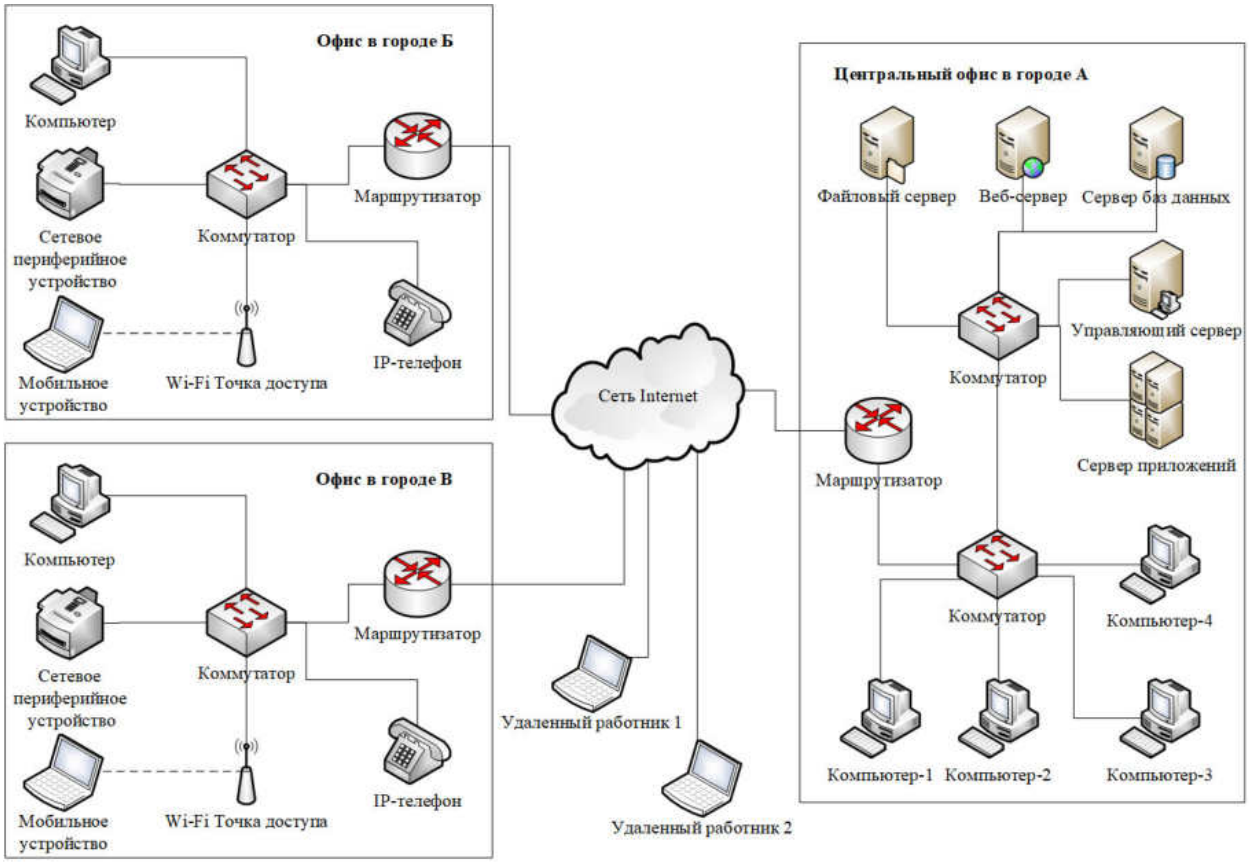

Рисунок 1.3 – Типовая схема КТС.

К корпоративной сети, как и к другим типам компьютерных сетей, предъявляется ряд требований. Главное из них – обеспечение пользователям возможности оперативного доступа к разделяемым ресурсам всех компьютеров, объединенных в сеть. Решению этой основной задачи подчинены остальные требования: по производительности, надежности, безопасности, управляемости, совместимости и масштабируемости.

Для оценки степени удовлетворения указанных требований используются ниже рассматриваемые показатели, которые одновременно являются основными характеристиками КТС. В рекомендации МСЭ Y.1540 [37] рассматриваются следующие сетевые характеристики, как наиболее важные по степени их

влияния на сквозное качество обслуживания (от источника до получателя), оцениваемое пользователем [43]:

– производительность сети;
– надежность сети/сетевых элементов;
– задержка;
– вариация задержки (джиттер);
– потери пакетов.

Производительность сети (throughput) [36] оценивается количеством информации (в пакетах, битах), передаваемых сетью в единицу времени (эффективная скорость передачи данных, доставляемых потоком в определенный интервал времени – определение из Y.1540). Если маршрут передачи пакета состоит из участков с разной пропускной способностью, то общая пропускная способность этого маршрута будет равна минимальной из пропускных способностей составляющих участков (элементов) маршрута. Стоит отметить, что значение производительности сети не совпадает с максимальной пропускной способностью, которую еще называют шириной полосы пропускания (bandwidth).

Надежность сети/сетевых элементов [37; 43] может быть определена через ряд параметров, из которых наиболее часто используется коэффициент готовности, вычисляемый как отношение времени простоя объекта к суммарному времени наблюдения объекта, включающему время простоя и время между отказами. В идеальном случае коэффициент готовности должен быть равен 1, что означает стопроцентную готовность сети. На практике коэффициент готовности оценивается числом «девяток». Например, коэффициент готовности 0,9 («одна девятка») обозначает 36,5 дней простоя в год (16,8 часов простоя в неделю), а коэффициент готовности 0,999 («три девятки») – 8,76 часов простоя в год, или 10,1 минут простоя в неделю.

Задержка доставки пакета (packet transfer delay, PTD). Параметр PTD определяется как время (t_2 - t_1) между двумя событиями – вводом пакета во входную точку сети в момент t_1 и выводом пакета из выходной точки сети в

момент t_2, где ($t_2 > t_1$) и ($t_2 - t_1$) $\leq T_{max}$. Эта задержка является суммой задержек на каждой линии подсети, которую проходит пакет, также ее называют задержкой из конца в конец (end-to-end). Каждая задержка на линии в свою очередь состоит из четырех компонентов [6]:

1. Задержка на обработку — задержка между моментом, когда пакет был правильно принят на начальном узле линии, и моментом, когда пакет был поставлен в очередь на передачу по линии.

2. Задержка в очереди — задержка между моментом, когда пакет был поставлен в очередь на передачу, и моментом, когда он начинает передаваться. В течение этого времени пакет ждет, пока будут переданы другие пакеты из очереди.

3. Задержка передачи (также называют задержкой сериализации) — задержка между моментами, когда передадутся первый и последний биты пакета.

4. Время распространения – промежуток времени от момента, когда последний бит был передан на начальном узле линии, до момента, когда он будет принят в конечном узле этой линии. Эта задержка появляется в результате ограничения скорости распространения фотонов или электронов в среде передачи и пропорциональна физическому расстоянию между передатчиком и приемником.

Вариация задержки пакета (packet delay variation, PDV). Параметр V_k, характеризует вариацию задержки PDV. Для пакета с индексом k этот параметр определяется между входной и выходной точками сети в виде разности между абсолютной величиной задержки X_k при доставке пакета с индексом k, и определенной эталонной (или опорной) величиной задержки доставки пакета, $d1,2$, для тех же сетевых точек:

$$V_k = X_k - d1,2.$$

Эталонная задержка доставки пакета, $d1,2$, между источником и получателем определяется как абсолютное значение задержки доставки первого пакета между данными сетевыми точками. Вариация задержки пакета, или джиттер, проявляется в том, что последовательные пакеты прибывают к

получателю в нерегулярные моменты времени. В системах IP-телефонии это, к примеру, ведет к искажениям звука и в результате к тому, что речь становится неразборчивой.

Коэффициент потери пакетов (packet loss ratio, PLR). Коэффициент PLR определяется как отношение суммарного числа потерянных пакетов к общему числу принятых в выбранном наборе переданных и принятых пакетов. Потери пакетов в сетях IP возникают в том случае, когда значение задержек при их передаче превышает нормированное значение, определенное выше как T_{max}. Если пакеты теряются, то при передаче данных возможна их повторная передача по запросу принимающей стороны. В системах интернет-телефонии (VoIP) пакеты, пришедшие к получателю с задержкой, превышающей T_{max}, отбрасываются, что ведет к провалам в принимаемой речи.

Обычно организации являются лишь потребителями телекоммуникационных услуг, которые предоставляются им сетевыми провайдерами (операторами услуг связи). При этом между организацией-потребителем и компанией-провайдером заключается соглашение об уровне предоставления услуги (Service Level Agreement, SLA), в котором указываются эксплуатационные показатели и показатели качества (Key Quality Indicators, KQI). Данное соглашение содержит описание предоставляемых услуг и ставит границы ответственности в рамках определенного сервиса, в SLA прописываются параметры услуги и их допустимые колебания, а вышеперечисленные показатели выступают в качестве метрик (KPI), обычно получаемых с помощью измерительного оборудования, на основании которых рассчитываются KQI [11].

Для определения итогового качества обслуживания, согласно руководству GB917 TM Форум (TM Forum, ранее TeleManagement) [115], при помощи измеряемых метрик рассчитываются итоговые показатели качества услуги: коэффициент готовности услуги (Service Availability, SA); время, в течение которого метрики не соответствовали граничным значениям (Service Degradation

Period); среднее время между отказами услуги (MTBF); среднее время восстановления услуги (MTTR).

В рамках КТС функционируют различные сервисы, доступ к которым осуществляется посредством телекоммуникационной сети. Данные внутрикорпоративные сервисы представляют из себя различные службы, на которых основывается работа организации, к ним можно отнести:

– корпоративный портал для обеспечения взаимодействия сотрудников организации;

– сервисы VoIP для обеспечения связи между сотрудниками и сторонними организациями;

– доменные службы;

– сервисы бизнес-логики;

– сервисы, предоставляемые для юридических лиц, например, сервисы по финансовой отчетности.

Часть сетевых сервисов КТС являются критичными к задержкам и/или к другим показателям качества обслуживания (Quality of Service, QoS). Среди чувствительных к задержкам сервисов можно назвать IP-телефонию, видеоконференцсвязь, VDI (Virtual Desktop Infrastructure – инфраструктура виртуальных рабочих столов), которые получили еще более широкое распространение в связи с массовым переходом на удаленный режим работы во время пандемии COVID-19. Причем, сервисы, в которых критичным параметром является время отклика, не ограничиваются вышеперечисленными: можно также выделить чат-боты для осуществления торговли, в том числе на биржевых площадках, облачные сервисы, среди которых набирающий популярность TaaS (Testing-as-a-Service – тестирование как сервис), работающий в режиме «мягкого» реального времени, сервисы АСУ ТП и другие.

Функционирование организации напрямую зависит от доступности данных сервисов, т.к. они обеспечивают потребности бизнеса в коммуникации, инструментарии для решения повседневных задач, автоматизации бизнес-процессов и т.д. Поэтому поставщики телекоммуникационных услуг должны

гарантировать, что предоставляемые ими услуги будут достигать требуемого целевого показателя доступности. Именно для этого применяются механизмы обеспечения качества обслуживания, что отражено, в том числе, в уже упомянутой выше рекомендации МСЭ-Т Y.1540, а также в рекомендации МСЭ-Т Y.1541 [38], которая даёт следующее обоснование необходимости применения механизмов QoS: потребители нуждаются в таких уровнях сетевых показателей качества, которые в сочетании с их хостами, оконечным оборудованием и другими устройствами обеспечивают удовлетворительную поддержку их приложений.

Поэтому методы обеспечения качества обслуживания трафика в КПТС на основе доступности узлов и каналов связи составляют **предмет настоящего исследования**.

1.2 Понятие сетевой доступности и методы ее обеспечения

Понятие «доступность» (availability) по-разному используется в контексте терминологии. В отечественных официальных документах чаще всего этот термин используют применительно к таким объектам как информация, документ, ресурс. Однако, даже по отношению к одному и тому же объекту, ученые и практики по всему миру по-разному трактуют данное понятие. Так, например, в официальных документах приводятся следующие определения:

ISO/IEC 2382:2015 [7]: доступность – способность функционального блока быть в состоянии выполнить требуемую функцию при заданных условиях в данный момент времени или в течение заданного интервала времени, предполагая, что необходимые внешние ресурсы предоставляются;

ISO 7498-2:1989 [8]: доступность – свойство быть доступным и полезным по требованию уполномоченного лица;

ISO/IEC 13335-1:2004 [9]: доступность – свойство, заключающееся в наличии и применимости для авторизованных субъектов информации, когда потребуется;

ГОСТ Р 50922 – 2006 [45]: доступность информации: [ресурсов информационной системы] – это состояние информации [ресурсов информационной системы], при котором субъекты, имеющие права доступа, могут реализовать их беспрепятственно.

Простейшее представление доступности (availability), встречаемое в литературе – это отношение ожидаемого значения времени безотказной работы системы (uptime) к совокупности ожидаемых значений времени работы и простоя (downtime):

$$Availability = \frac{E[uptime]}{E[uptime] + E[downtime]}.$$

Многие исследователи как в России, так и за рубежом [45; 63; 77; 85; 87; 88; 94; 103; 104; 122] определяют доступность как долю времени, в течение которого система (или телекоммуникационная сеть) находится в рабочем состоянии и при решении инженерных задач для определения величины доступности пользуются формулой расчета коэффициента готовности из теории надежности.

При этом, многие из этих исследователей выделяют следующие типы доступности [61]:

– мгновенная (или моментальная) доступность (Instantaneous (or Point) Availability);

– средняя доступность времени безотказной работы (или средняя доступность) (Average Uptime Availability (или Mean Availability));

– доступность в стационарном состоянии (Steady State Availability);

– собственная доступность (в других областях – коэффициент внутренней готовности) (Inherent Availability);

– достигнутая доступность (Achieved Availability);

– эксплуатационная доступность (также оперативная готовность) (Operational Availability).

Мгновенная (или моментальная) доступность – это вероятность того, что система (или компонент) будет работоспособна (запущена и работает) в определенное время, t. B. Blanchard в [50] дает качественное определение данному типу доступности как «меру степени, в которой система находится в работоспособном и фиксируемом состоянии на момент начала выполнения задачи, когда задача вызывается в неизвестный случайный момент времени». Этот тип доступности обычно используется в военном деле, поскольку иногда необходимо оценить доступность системы в конкретное время, представляющее интерес (например, когда должна выполняться задача). Мгновенная доступность очень похожа на функцию надежности в том смысле, что она дает вероятность того, что система будет функционировать в данный момент времени t. Однако, в отличие от надежности, показатель мгновенной доступности включает информацию о ремонтопригодности. В данный момент времени t система будет работоспособна, если выполняется одно из следующих условий:

Система работала исправно от 0 до t, т.е. ни разу не вышла из строя к моменту t. Вероятность этого равна $R(t)$.

Или система работала нормально с момента последнего ремонта в момент u, $0 < u < t$. Вероятность этого состояния составляет:

$$\int_0^t R(t-u)m(u)du,$$

где $m(u)$ – функция плотности обновления системы.

Следовательно, мгновенная доступность – это сумма двух вышеупомянутых вероятностей, или:

$$A(t) = R(t) + \int_0^t R(t-u)m(u)du.$$

Средняя доступность времени безотказной работы (или средняя доступность), $\overline{A(t)}$ — это доля времени в течение решения задачи или периода времени, в течение которого система доступна для использования. Она представляет собой среднее значение функции мгновенной доступности за период (0, *T*) и определяется как:

$$\overline{A(t)} = \frac{1}{t}\int_0^t A(u)du.$$

Такое определение доступности обычно используется в производственных и телекоммуникационных системах.

Доступность системы в стационарном состоянии является пределом функции доступности, поскольку время стремится к бесконечности. Доступность в стационарном состоянии также называется долгосрочной или асимптотической доступностью. Обычное уравнение доступности в установившемся (стационарном) режиме, которое можно найти в литературе, выглядит следующим образом:

$$A(\infty) = \lim_{t\to\infty} A(t).$$

Следует отметить, что стационарное состояние также применимо к средней доступности.

Собственная доступность, A_I – это доступность в стационарном состоянии, если рассматривать только время простоя системы при корректирующем техническом обслуживании (СМ). Для одного компонента собственную доступность можно рассчитать следующим образом:

$$A_I = \frac{MTTF}{MTTF + MTTR},$$

где MTTF (Mean Time To Failure) – среднее время наработки на отказ, MTTR (Mean Time To Repair) – среднее время восстановления работоспособности (среднее время до восстановления).

Для системы используется следующая формула:

$$A_I = \frac{MTBF}{MTBF + MTTR},$$

где MTBF (Mean Time Between Failure) – средняя наработка между отказами. До достижения стационарного состояния расчет MTBF может быть функцией времени (например, для деградирующей системы). В таких случаях перед достижением устойчивого состояния вычисленное значение MTBF изменяется по мере изменения состояния системы во времени и сбора дополнительных данных.

Достигнутая доступность, A_A очень похожа на собственную доступность, за исключением того, что в нее также включены простои при профилактическом обслуживании (PM). В частности, это доступность в стационарном состоянии при рассмотрении времени простоя системы для корректирующих и профилактических работ.

Достигнутую доступность можно вычислить при помощи среднего времени между действиями по техническому обслуживанию, MTBM и среднем временем простоя при техническом обслуживании, $\bar{M}$:

$$A_A = \frac{MTBM}{MTBM + \bar{M}},$$

где MTBM = время безотказной работы / (количество сбоев системы + количество PM, отключивших систему); $\bar{M}$ = (Время простоя CM + Время простоя PM) / (Количество сбоев системы + Количество PM, отключивших систему).

Эксплуатационная доступность, A_O – это мера «реальной» средней доступности за период времени и включает все известные источники простоев. По сути, это апостериорная доступность, основанная на реальных событиях, произошедших с системой. Описанные ранее типы доступности являются априорными оценками, основанными на моделях распределения отказов системы и времени простоя. Эксплуатационная доступность – это отношение времени безотказной работы системы к общему времени, определяется следующим образом:

$$A_O = \frac{\text{время безотказной работы}}{\text{рабочий цикл}},$$

где рабочий цикл – это общий период исследуемой работы, а время безотказной работы – это общее время, в течение которого система функционировала в течение рабочего цикла. (Примечание: эксплуатационная готовность является функцией времени, t или рабочего цикла). Понятие эксплуатационной доступности тесно связано с понятием оперативной готовности из военной сферы.

R. Barlow и F. Proschan в [45] определяют доступность ремонтируемой системы как «вероятность того, что система работает в заданное время t. Доступность $A_i(t)$ компонента i системы во время t обозначает вероятность того, что компонент i функционирует во время t. Допустив $t \to \infty$, получают стационарную вероятность того, что отказ компонента i является причиной отказа системы. Стационарная доступность компонента i, $i = \overline{1..n}$ рассчитывается как:

$$A_i = \lim_{t\to\infty} A_i(t) = \frac{\mu_i}{\mu_i + v_i},$$

где μ_i – среднее время жизни, v_i – среднее время ремонта компонента i.

Доступность, учитываемая при моделировании технического обслуживания для систем различного типа, обсуждается в работах [63; 64; 77; 103; 111].

J. Laprie в работе [85] отмечает, что по мере того, как вычислительные системы становились надежными, их услуги использовались и требовались на регулярной основе, поэтому доступность стала жизненно важной. При этом «доступность» (availability) он определил как один из свойств такого понятия как безотказность (dependability) в смысле готовности системы к использованию. Среди других свойств безотказности автор отмечает надежность (reliability), защищенность (safety), ремонтопригодность (maintainability).

В 2017 году K. Trivedi и A. Bobbio выпустили книгу [123], в которой предложили довольно обширный обзор по доступности. В ней авторы приводят два варианта определения доступности, которые были даны различными авторами:

– доступность – способность быть в состоянии выполнять требуемую функцию при заданных условиях, в заданный момент времени или по прошествии достаточного времени, предполагая, что требуемые внешние ресурсы предоставлены [122];

– доступность – готовность к корректному обслуживанию [87].

При этом авторы описывают разницу в понятиях надежность и доступность следующим образом:

Фундаментальное различие между надежностью и доступностью заключается в том, что надежность относится к безотказной работе в течение временно́го интервала, в то время как доступность относится к безотказной работе в данный момент времени, обычно в тот момент, когда устройство или система получают доступ для предоставления требуемой функции или услуги. Надежность – это мера, характеризующая процесс отказа устройства, в то время как доступность объединяет процесс отказа с процессом восстановления или ремонта и рассматривает вероятность того, что в данный момент времени

устройство находится в рабочем состоянии независимо от количества циклов отказа/ремонта, уже пройденных устройством.

Для вычисления показателя доступности авторы ссылаются на событие, когда система работает в данный момент времени независимо от количества отказов (и ремонтов), уже произошедших в системе, причем доступность – это вероятность наступления такого события.

Понятие «сетевой доступности» (network availability) в литературе выделяют в отдельный класс. Несмотря на это, большинство авторов, [например, 10; 23; 35; 69; 44; 105; 125; 135] трактуют сетевую доступность как свойство сохранять работоспособное состояние в течение некоторой наработки, рассчитывают, как собственную доступность системы: $A = MTBF / (MTBF + MTTR)$ и оценивают в «девятках», определяющих долю времени безотказной работы в год. При этом, количество «девяток» можно получить по следующей формуле: $A_9 = -lg(1 - A)$.

В работе W. Zhou [135] рассматриваются два основных фактора, определяющих доступность сети:

– Первый фактор – это доступность отдельных сетевых элементов. В течение срока службы сетевого элемента бывают периоды, когда он не работает из-за неисправности, технического обслуживания или ремонта.

– Второй фактор – топология сети. Очевидно, что более высокая избыточность в сети (например, большее количество каналов, соединяющих сетевые коммутаторы) приведет к более высокой доступности, но также и к более высоким затратам на поддержку и управление.

Сеть в данном исследовании рассматривается в виде графа G (V, E), состоящего из набора узлов V и набора связей E. | V | = N обозначает количество узлов, в то время как количество каналов связи равно | E | = L. Узлы представляют маршрутизаторы или коммутаторы, а связи представляют собой линии связи (например, оптоволокно). И линии связи, и узлы имеют определенную доступность. В данном исследовании предполагают, что узлы всегда доступны, т.е. только линии связи могут выйти из строя. Это предположение основано на

том факте, что сбои узлов происходят гораздо реже, чем сбои каналов, которые возникают, например, при преднамеренном повреждении линий связи. Кроме того, предполагается, что сбои происходят независимо. Также в работе отдельно рассматривают такие понятия, как доступность пути, доступность незагруженной сети и доступность загруженной сети.

Доступность пути

Предположим, что у каждого пользователя есть ровно один путь, по которому он может общаться с другим пользователем. Если этот путь не работает, связь прекращается. Поскольку путь состоит из последовательного соединения узлов и линий связи, его доступность просто вычисляется как произведение доступности узлов и связей, составляющих этот путь. Затем доступность сети может быть определена как доступность минимального пути по всем парам узлов. Поскольку все пути известны (или могут быть вычислены с помощью простого алгоритма кратчайших путей), доступность сети легко вычисляется. Пусть p_{ij} обозначает вероятность доступности канала связи (i, j), который соединяет узлы i и j. Чтобы получить максимально возможную доступность сети (определяемую в ее самой строгой форме), минимальная доступность пути по всем парам узлов должна быть максимальной. Если присвоить вес $-log\ (p_{ij})$ каждой связи $(i, j) \in E$, то та же цель будет достигнута за счет использования кратчайших путей (с учетом новых весов) между парами исходных пунктов назначения (s-d). Кратчайший путь с наибольшим весом имеет наименьшую доступность пути и определяет оптимальную доступность сети, которую можно получить.

Доступность незагруженной сети

Другой крайний случай — связь может происходить по всем возможным путям. Это очень похоже на маршрутизацию в Интернете, где в случае сбоя протоколы маршрутизации автоматически перенастраивают таблицы маршрутизации для направления трафика по альтернативным рабочим путям к месту назначения. Недоступность сети в этом случае определяется вероятностью того, что между конкретной парой источник-пункт назначения недоступен путь

(т.е. сеть отключена). Максимальная вероятность по всем парам s-d определяет недоступность сети и, следовательно, доступность сети.

Доступность загруженной сети

В работе отмечено, что расчет доступности незагруженной сети на самом деле слишком оптимистичен на практике, так как он предполагает, что, когда пути выходят из строя, альтернативные пути имеют достаточно ресурсов, доступных для обработки трафика отказавшего пути(ей). Поскольку ресурсы ограничены, так может происходить не всегда. Однако, такой случай в данном исследовании не рассматривается.

Также авторы M. Tornatore и др. [120] описывают подход к оценке доступности, основанный на том, что отказ хостов (клиентов или серверов), соединительного оборудования и линий связи может влиять на величину связности и, таким образом, уменьшает наличие доступа клиентских узлов к услугам, авторы отмечают, что доступность узлов зависит от доступности других хостов.

Некоторые исследователи предлагают расширить описанный выше метод оценивания доступности сети внесением дополнительных показателей. Так, в исследовании С. Платуновой [34] доступность ресурсов вычислительной сети с самоподобным трафиком предлагается оценивать коэффициентом оперативной готовности как функции числа основных и резервных элементов и дополнительно средним временем пребывания пакетов в сети, оцененным с использованием многоканальных моделей массового обслуживания.

M. Durvy и др. в статье [59] определяют доступность сети как процент от общей пропускной способности сети, доступной для маршрутизации трафика. Авторы отмечают, что при усреднении по времени доступность сети эффективно отражает частоту и влияние сбоев в данной сети. Также была введена новая метрика SLA под названием «доступность услуги», которую определили как долю времени, в течение которого услуга доступна клиенту. В среде, подверженной сбоям, потеря и задержка пакетов могут достигать уровня, при котором большинство приложений не могут работать должным образом. В таких

случаях авторы считают, что услуга недоступна для клиента. Также авторы подчеркивают, что доступность услуги является важным параметром качества услуги, воспринимаемой пользователем, и поэтому должна быть включена в SLA.

Несмотря на растущую популярность SDN, анализ работ по предмету исследования позволил сделать вывод, что вопросами доступности программно-определяемых сетей занимаются крайне мало. Есть ряд исследований, касающихся задержек при передаче пакетов [4; 12; 18; 84; 112], вычисления маршрута и балансировки потоков [17; 33; 79] и передачи управляющих сообщений от коммутаторов к контроллерам [5; 110], в которых не затрагивается вопрос оценки и оптимизации доступности сети. Так, например, отмечено, что увеличение задержек в SDN может быть связано с расположением контроллера в сети [106], неэффективном программном обеспечении, организации правил переадресации и в контроле нагрузки.

Значительная часть работ [65; 113, 130; 133; 177 и др.] посвящена вопросам балансировки нагрузки между контроллерами SDN для повышения надежности сетевой структуры и эффективности функционирования сети в целом, предложены различные подходы как для одноранговой (параллельной), так и для иерархической организации уровня управления с несколькими контроллерами. Например [66; 132; 134 и др.], разработаны стратегии балансировки нагрузки, основанные на миграции коммутаторов. Однако, в данных работах отсутствует оценка влияния использования таких стратегий на показатели доступности сети.

Область, связанная с доступностью в SDN, исследована недостаточно. Также стоит отметить то, что топологии сетей авторов смежных работ либо изменялись незначительно, либо оставались неизменными, а многие исследования проводились на устаревших версиях протокола OpenFlow, что не позволяет в полной мере опираться на полученные в ходе данных работ результаты.

Однако, анализ работ [48; 75; 102; 107; 124; 129; 136] позволил выделить следующие принципиальные для данного исследования факторы, влияющие на

масштабируемость и производительность, а, следовательно, и возможности по повышению доступности сетей, использующих архитектурный принцип SDN:

– вычислительная мощность контроллеров и устройств, организующих уровень передачи;

– емкость памяти и буфера;

– расположение контроллера в сети, что в некоторых исследованиях не оказывало никакого влияния на общую производительность, например, в [68], а в других [73; 78; 92; 118; 127] – при разных условиях наблюдался рост задержки;

– задержка между типами устройств контроллер-контроллер и контроллер-коммутатор [131];

– рост трафика в канале связи.

На уровне инфраструктуры сети масштабируемость главным образом зависит от оборудования, недостаточность вычислительной мощности которого или нехватка памяти напрямую влияет на уровень качества обслуживания всей сети [101]. Соответственно, в таких условиях необходимо адаптировать механизмы QoS для КПТС, которые в общем случае реализуют следующие функции [56]:

- поддержка гарантированной полосы пропускания;
- уменьшение потерь [97];
- управление перегрузками [96];
- формирование (шейпинг) трафика;
- настройка приоритетов (классификация) трафика в сети.

Выделим основные понятия, используемые в службах QoS:

Очередь (queue) – буфер устройства, реализованный виртуально или физически и используемый для хранения совокупности пакетов, коллективно ожидающих передачи сетевым устройством на основании планировщика пакетов.

Планировщик пакетов (scheduler) – это механизм, который управляет очередью распределения сетевых пакетов. С его помощью можно отбрасывать пакеты, если буфер переполнился, а также изменять порядок отправки пакетов.

В различных операционных системах используются разные планировщики пакетов. Планировщики обычно пытаются достичь баланса между эффективностью использования ресурсов и временем запуска приложения.

Шейпинг пакетов (packet shaping) – это механизм управления трафиком, который задерживает некоторые или все пакеты, чтобы привести их в соответствие с желаемым профилем трафика. Существует 2 типа алгоритмов формирования трафика: Leaky Bucket и Token Bucket.

Наибольшей популярностью пользуется иерархический алгоритм ведра маркеров (Hierarchical Token Bucket, HTB), подразумевающий разделение полосы пропускания для определенных типов потока в отдельные классы, каждый из которых имеет собственную полосу пропускания. HTB выстраивает классы в виде дерева: они могут разделяться на дочерние классы, каждый из которых делит между собой полосу родительского класса.

Множеством авторов предлагались собственные реализации алгоритмов управления трафиком, среди них: S. Keith [81], C. Bastian [46], J. Tierney [119], L. Guo [71], K. Stanwood [116], A. Iera [76] и ряд отечественных исследователей, таких как В. Кузьмин [16], А. Масленников [19], Л. Абросимов [1] А. Парамонов [31] однако, во всех этих работах емкость классов трафика по-прежнему оценивается исходя из интенсивности потока (throughput), а величина задержки доставки пакета клиенту не учитывается.

Ряд работ, например, [55; 60; 67; 70; 80; 82; 89] посвящен обеспечению QoS в программно-определяемых сетях. В данных работах в основном рассматривались вопросы адаптации алгоритмов обеспечения качества обслуживания для работы с протоколом OpenFlow, разработке фреймворков для работы алгоритмов QoS в SDN и влиянию маршрутизации и положения контроллера на характеристики качества обслуживания, среди которых в основном рассматривали надежность, масштабируемость и балансировку нагрузки.

1.3 Формализация задачи исследования

В данном диссертационном исследовании математическую модель КПТС представим в виде графа (рисунок 1.4): G = {U, V}, где U = $\{u_1, u_2, \dots, u_n\}$ - множество узлов (виртуализированных сетевых устройств SDN, например, Open vSwitch), V = $\{v_{ij}\}$ – множество ребер – линий связи (ЛС) $v_{ij} \in \{1,0\}, i, j = \overline{1..n}$. Веса ребер – показатели доступности ЛС (узла u_j относительно узла u_i) $0 \le a_{ij} \le 1,\ a_{ii} = 1$. Если i и j не соединены друг с другом ($v_{ij} = 0$), то в связной сети имеется, как минимум, один канал связи (КС) между u_i и u_j.

В рамках данного исследования используется концепция доступности сети, предложенная Ю.М. Монаховым и его соавторами в работах [24-26; 100], включающая в себя понятие сетевой доступности и методику её расчета.

Поэтому под *доступностью сети* $A(G)$ будем понимать вероятность того, что при заданной топологии сети определенные характеристики (сетевая связность, пропускная способность и т.д.) будут поддерживаться на заданном уровне.

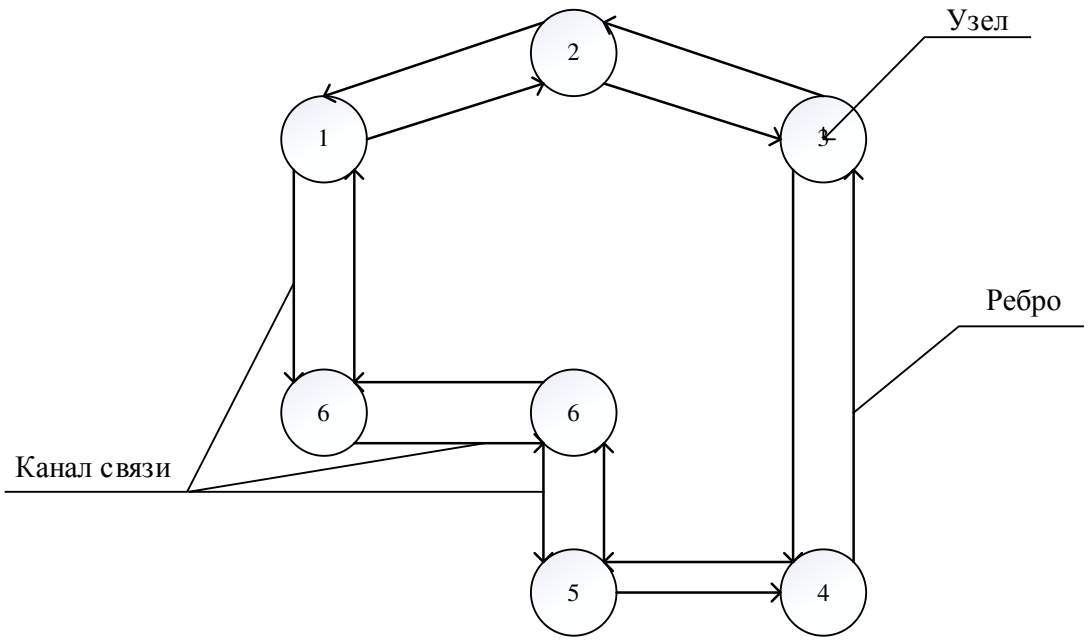

Рисунок 1.4 – Модель КПТС.

В свою очередь, доступность всей сети можно представить через доступность узлов (устройств) сети и доступность каналов связи между ними, где показатель доступности канала связи вычисляется по аналогии с надежностью в технических системах исходя из характера соединения на этом канале и вероятности отклика за время, меньшее или равное директивному устройств, которые входят в канал [28].

Задача данного исследования заключается в оптимизации показателя доступности всей сети и оптимизации каналов связи по отдельности при следующих ограничениях:

– ограничена вычислительная мощность контроллеров и виртуальных коммутаторов;

– ограничены емкость памяти и буфера для управления множеством виртуализированных и физических узлов и множеством линий связи;

– существует задержка между типами устройств контроллер-контроллер и контроллер-коммутатор.

В соответствии с методикой показатель доступности КС между двумя узлами, представляющего собой последовательное соединение узлов от истока к стоку, как показано на рисунке 1.5, определяется выражением:

$$a_{ij}^{\text{КС}} = \prod_{z\epsilon[i,j]} a_z \ , \qquad (1.1)$$

где a_z - показатели доступности линий связи на пути от u_i до u_j.

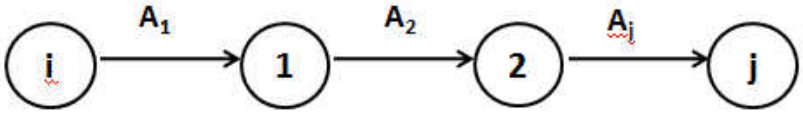

Рисунок 1.5 – Пример последовательного соединения узлов сети.

Если количество возможных путей от одного узла до другого превышает единицу, тогда общая доступность канала между двумя узлами (i и j)

рассчитывается с учетом всех возможных путей, которые ведут из вершины i в вершину j следующим образом:

$$a_{ij} = \frac{\sum_{z_r \in \mathrm{Z}} \left(\prod_{z_r \in [i,j]} a_{z_r} \right)}{\mathrm{Z}^{ij}} \qquad (1.2)$$

где z_r – КС с номером r из множества Z^{ij} каналов; a_{z_r} - показатели доступностей линий связи в канале z_r от i –го до j – го узла.

Определим диапазон показателя доступности выполняемой задачи с заданным качеством $\mathrm{A}(G)_{min} \le A(G) \le \mathrm{A}(G)_{max}$, здесь $\mathrm{A}(G)_{min}$ – минимально допустимый показатель доступности, $\mathrm{A}(G)_{max}$ – максимально возможный для данной сети. Введем показатель идеальной доступности сети $A^*(G)$, когда все КС доступны с вероятностью 1.

Автор методики предлагает использовать следующий алгоритм для определения доступности всей сети:

Алгоритм определения интегрального показателя доступности сети

Шаг 1. Строим матрицу смежности $Y = \|y_{ij}\|, i, j = \overline{1..n}$, где $y_{ij} = 1$, если $v_{ij} = 1$, иначе $y_{ij} = 0$.

Шаг 2. С помощью модифицированного алгоритма Флойда-Уоршелла находим множества каналов связи между каждыми двумя узлами u_j и u_i:

$$Z^{12} = \left\{ Z_1^{12}, Z_2^{12}, \dots, Z_{R_{12}}^{12} \right\}$$
$$\dots$$
$$Z^{ij} = \left\{ Z_1^{ij}, Z_2^{ij}, \dots, Z_{R_{ij}}^{ij} \right\}$$
$$\dots$$
$$Z^{n1} = \left\{ Z_1^{n1}, Z_2^{n1}, \dots, Z_{R_{n1}}^{n1} \right\}$$
$$\dots$$
$$Z^{n(n-1)} = \left\{ Z_1^{n(n-1)}, \dots, Z_{R_{n(n-1)}}^{n(n-1)} \right\}$$

Здесь R_{ij} – соответственно мощности множеств Z^{ij}.

Шаг 3. Вычислим показатели доступности всех КС между каждыми двумя узлами u_i и u_j по формуле (1.2). Получим множества показателей:

$$A(Z^{12}) = \left\{A(Z_1^{12}), A(Z_2^{12}), \dots, A(Z_{R_{12}}^{12})\right\}$$
$$\dots$$
$$A(Z^{ij}) = \left\{A(Z_1^{ij}), A(Z_2^{ij}), \dots, A(Z_{R_{ij}}^{ij})\right\}$$
$$\dots$$
$$A(Z^{n(n-1)}) = \left\{A(Z_1^{n(n-1)}), \dots, A(Z_{R_{n(n-1)}}^{n(n-1)})\right\}$$

Шаг 4. Найдем соответственно минимальный $\hat{A}(G)_{min}$ и максимальный $\hat{A}(G)_{max}$ показатели доступности среди всех пар узлов.

Шаг 5. При помощи способа, предложенного Ю.М. Монаховым и др., основанного на вычислении расстояния Колмогорова для двух функций, определяем интегральный показатель доступности (ИПД) сети в виде $A(G)_{min} = f_1\left(\hat{A}(G)_{min}, A^*(G)\right)$, $A(G)_{max} = f_2\left(\hat{A}(G)_{max}, A^*(G)\right)$, где f_1 и f_2 – функционалы, используемые в обозначенном подходе.

$$d\left(\hat{A}(G)_{max}, A^*(G)\right) = \left(\int_0^1 \left(A^*(G) - \hat{A}(G)_{max}\right)^p dt\right)^{1/p} \quad (1.3)$$

$$d\left(\hat{A}(G)_{min}, A^*(G)\right) = \left(\int_0^1 \left(A^*(G) - \hat{A}(G)_{min}\right)^p dt\right)^{1/p} \quad (1.4)$$

где t – произвольный вектор; p – произвольный параметр ($p \geq 1$). Для упрощения формул (1.3 и 1.4) можно принять p = 1.

Шаг 6. Нормируем полученные величины:

$$A(G)_{max} = e^{-\frac{d\left(\hat{A}(G)_{max}, A^*(G)\right)}{d_{max}(\mathrm{N})}};$$
$$A(G)_{min} = e^{-\frac{d\left(\hat{A}(G)_{min}, A^*(G)\right)}{d_{max}(\mathrm{N})}},$$

где $d_{max}(N)$ – полином третьей степени, вид которого выбирается в зависимости от N, N – число узлов в рассматриваемой сети.

Конец алгоритма.

Для повышения доступности сети предлагается два подхода: (1) оптимизировать топологию КПТС для достижения максимума ИПД; (2) повысить показатель доступности каналов связи a_{ij} за счет применения нового алгоритма управления потоком.

Выводы к главе 1

1. Проанализированы существующие решения задачи повышения доступности корпоративной программно-определяемой телекоммуникационной сети и ее компонентов, а также методики оценки показателей доступности. Несмотря на растущие требования уровня приложений по уменьшению отклика, большинство авторов, исследующих доступность телекоммуникационных сетей, не учитывают данный критерий и по-прежнему оценивают этот показатель через коэффициент готовности сети.

2. Ограничения, препятствующие достижению высокой доступности в КПТС, связаны с пределами по вычислительной мощности контроллеров и устройств, организующих уровень передачи, а также емкости памяти и буфера; задержкой между типами устройств контроллер-контроллер и контроллер-коммутатор; ростом трафика в канале связи.

3. Сформулирована задача оптимизации доступности при существующих ограничениях по вычислительной мощности контроллеров, ёмкости памяти и буфера для управления множеством виртуализированных узлов и множеством линий связи. Обоснована методика оценки показателя доступности сети и каналов связи. Для повышения доступности сети предлагается два подхода: оптимизировать топологию КПТС для достижения максимума ИПД; повысить показатель доступности каналов связи за счет применения нового алгоритма управления потоком.

2 РАЗРАБОТКА АЛГОРИТМА ОПТИМИЗАЦИИ ТОПОЛОГИИ КПТС ПО КРИТЕРИЮ МАКСИМУМА ИНТЕГРАЛЬНОГО ПОКАЗАТЕЛЯ ДОСТУПНОСТИ

В данной главе описывается ход разработки алгоритмов оптимизации интегрального показателя доступности КПТС. Описывается порядок и условия проведенных экспериментов и анализ результатов.

2.1 Экспериментальный стенд для исследования доступности КПТС

Для проведения исследования влияния топологии КПТС на доступность был подготовлен стенд, представляющий собой ЭВМ с установленной операционной системой Linux со следующими характеристиками:

- процессор Intel Core i7-6700HQ 2,6 ГГц;
- оперативная память 16 Гб DDR4;
- дискретная видеокарта nVidia GeForce GTX960M 2 Гб;
- SSD накопитель 128 Гб;
- ОС Linux Ubuntu 18.04.4 LTS.

В качестве среды для эмуляции виртуальной сети было выбрано программное обеспечение с открытым исходным кодом Mininet [95]. Mininet представляет собой утилиту, написанную на языке Python, для быстрого проектирования и развертывания SDN, позволяющую создавать виртуальные узлы и коммутаторы, управлением которыми занимается контроллер, использующий для маршрутизации протокол OpenFlow. С помощью данного эмулятора проводилась большая часть всех отечественных и зарубежных исследований в области SDN. Открытость исходного кода, хорошая документация и модульный подход к разработке позволит в дальнейшем интегрировать дополнительные модули, не затрагивая основное логическое ядро программы.

Mininet предоставляет следующие принципиальные в данном исследовании возможности:

- быстрое эмулирование программно-определяемых сетей;
- комплексное тестирование топологии без необходимости подключения физической сети;
- работа с реальным кодом, включая стандартные сетевые приложения Unix / Linux, а также реальное ядро Linux и сетевой стек.

Для дальнейшей работы было необходимо выбрать контроллер, работающий с протоколом OpenFlow. В результате анализа готовых решений был выделен ряд контроллеров, между которыми проводилось сравнение. В качестве контроллера были протестированы следующие готовые открытые программные решения: OpenDayLight, Ryu, ONOS, POX. Отбор происходил путем анализа репозиториев на GitHub [68], оценке активности разработки, количества незакрытых ошибок, а также по отзывам пользователей, наличию и качеству документации. Некоторые из рассматриваемых контроллеров не дошли до стадии сравнения, т.к. были отброшены в результате заброшенности проектов.

В результате сравнения был выбран контроллер ONOS [108], написанный на языке программирования Java. Он оказался наиболее «дружелюбным» к развертыванию и использованию, обладает хорошей документацией и набором всех необходимых модулей для реализации простейшей реактивной маршрутизации сетевых потоков, имеет грамотную модульную структуру, а также получает актуальные обновления и активно обсуждается в тематических сообществах. Сборка контроллера ONOS осуществляется с помощью системы сборки Bazel [47].

В работе использовались следующие модули ONOS:

- модуль маршрутизации с помощью протокола OpenFlow;
- модуль простейшей переадресации пакетов с использованием функции реактивного определения маршрутов потоков с помощью протокола OpenFlow и сканирования сети.

Таким образом, полученный стенд позволяет создавать произвольные топологии с помощью эмулятора Mininet и осуществлять простейшую реактивную маршрутизацию потоков трафика в программно-определяемых сетях с использованием контроллера ONOS, чего достаточно для проведения необходимых экспериментов и исследований.

Для проведения исследований доступности КПТС с использованием данного стенда было необходимо интегрировать в Mininet возможность генерации матрицы смежности на основе эмулированной топологии в соответствии с выбранной методикой определения доступности. Для этого был разработан модуль, листинг которого представлен в Приложении А, который содержит в себе следующий функционал:

- множество вспомогательных функций для определения наличия уже установленной линии связи между двумя устройствами, получения списка еще не созданных связей, создания связи «на горячую» без необходимости перезапуска Mininet и новой сборки топологии;
- система визуального логирования результатов исследования для последующего удобного анализа;
- функции для определения задержек между двумя устройствами.

Система логирования создает HTML файл по указанному пути перед запуском экспериментов, затем на каждой итерации в документ дописывается таблица и необходимый набор дополнительных сведений. Таблица представляет собой матрицу смежности, содержащую названия коммутирующих устройств. На пересечении устройств в таблице записывается значение, равное средней вероятности того, что задержка при передаче пакетов будет меньше или равна директивному времени – 0.03 с [72]. На рисунке 2.1 изображен снимок записи о начальной топологии сети.

New link in topology: as2:cr2

	ar1	ar2	as1	as2	as3	as4	as5	as6	br1	br2	bs1	bs2	bs3	bs4	bs5	cr1	cr2	cs1	cs2	cs3	cs4
ar1	1	0	0.89200	0.91200	0.90000	0	0	0	0.88400	0	0	0	0	0	0	0	0	0	0	0	0
ar2	0	1	0	0.92400	0	0.90000	0.91600	0.89600	0	0	0	0	0	0	0	0	0	0	0	0	0
as1	0.99200	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
as2	0.97200	0.97600	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.92400	0	0	0	0
as3	0.98000	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
as4	0	0.98800	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
as5	0	0.98400	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
as6	0	0.98000	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
br1	0.97200	0	0	0	0	0	0	0	1	0	0.90800	0.91200	0.90800	0	0	0.92800	0	0	0	0	0
br2	0	0	0	0	0	0	0	0	0	1	0	0.87600	0	0.89600	0.88800	0	0	0	0	0	0
bs1	0	0	0	0	0	0	0	0	0.97200	0	1	0	0	0	0	0	0	0	0	0	0
bs2	0	0	0	0	0	0	0	0	0.97600	0.99200	0	1	0	0	0	0	0	0	0	0	0
bs3	0	0	0	0	0	0	0	0	0.99200	0	0	0	1	0	0	0	0	0	0	0	0
bs4	0	0	0	0	0	0	0	0	0	0.98800	0	0	0	1	0	0	0	0	0	0	0
bs5	0	0	0	0	0	0	0	0	0	0.99600	0	0	0	0	1	0	0	0	0	0	0
cr1	0	0	0	0	0	0	0	0	0.99600	0	0	0	0	0	0	1	0	0.93600	0.92800	0.92800	0
cr2	0	0	0	0.97200	0	0	0	0	0	0	0	0	0	0	0	0	1	0.93600	0	0	0.92800
cs1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.97200	0.99600	1	0	0	0
cs2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98000	0	0	1	0	0
cs3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98400	0	0	0	1	0
cs4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.97600	0	0	0	1

Рисунок 2.1 – Пример записи в журнале матрицы смежности

На рисунке 2.2 изображен пример снимка топологии сети для журнала логирования после создания дополнительных каналов связи между устройствами. Данный подход к логированию позволяет визуально отслеживать изменения, связанные с критерием доступности на каждой итерации эксперимента.

Также на этом этапе работы была добавлена возможность определения доступности сети и её компонентов путем интеграции соответствующего алгоритма в модуль Mininet, листинг представлен в Приложении А.

В качестве максимального верхнего порога допустимой задержки отправки запросов между двумя устройствами эмпирически в соответствии с производительностью стенда было выбрано значение 0.03 с.

Для определения доступности необходимо было выявить количество измерений, необходимых для определения средней задержки при отправке ICMP запросов между двумя коммутаторами, сохранив при этом баланс между точностью измерений и временем, потраченным на измерение. Для этого посылались запросы между узлами 50, 100, 250, 500, 1000, 1500, 2500, 5000,

10 000 раз и оценивалась вероятность *P* того, что отклик системы будет менее директивного времени (0,03 с). Результаты представлены в таблице 2.1.

	ar1	ar2	as1	as2	as3	as4	as5	as6	br1	br2	bs1	bs2	bs3	bs4	bs5
ar1	1	0	0.90400	0.88000	0.89600	0.89200	0.91600	0	0.90800	0.81600	0	0	0	0	0
ar2	0	1	0.95200	0.91600	0.89600	0.90000	0.92800	0.92400	0.92400	0	0	0	0	0	0
as1	0.98400	0.98800	1	0	0.92800	0.90800	0.91200	0.92800	0.84800	0.91200	0	0.88800	0	0	0
as2	0.98000	0.99200	0	1	0	0	0.90400	0	0.92000	0.79200	0.88400	0.85600	0	0	0
as3	0.98000	0.98800	0.95200	0	1	0	0.90800	0	0.85200	0	0	0	0.83600	0.89200	0.90000
as4	0.97600	0.98400	0.97200	0	0	1	0	0.91200	0.83600	0.91200	0.89200	0	0	0	0.84000
as5	0.97200	0.98400	0.97200	0.98400	0.97600	0	1	0	0.82800	0.92000	0	0.90800	0	0	0.89200
as6	0	0.98400	0.98400	0	0	0.98000	0	1	0	0	0.89200	0	0	0.86800	0.87200
br1	0.96000	0.94000	0.86000	0.89200	0.85600	0.86800	0.89600	0	1	0.86000	0.90400	0.90800	0.89200	0.84000	0.89600
br2	0.93600	0	0.91200	0.83600	0	0.85200	0.92800	0	0.90400	1	0.84800	0.87200	0	0.86000	0.82800
bs1	0	0	0	0.84800	0	0.93600	0	0.93600	0.92000	0.92400	1	0.85200	0	0.85200	0.86800
bs2	0	0	0.89600	0.92800	0	0	0.91200	0	0.92400	0.91600	0.95200	1	0.80800	0	0.82800
bs3	0	0	0	0	0.90000	0	0	0	0.93200	0	0	0.92400	1	0	0.87200
bs4	0	0	0	0	0.91600	0	0	0.92800	0.93600	0.90400	0.92800	0	0	1	0
bs5	0	0	0	0	0.93200	0.86000	0.92400	0.94400	0.91600	0.94400	0.96000	0.89600	0.92800	0	1
cr1	0.90800	0	0.91600	0.92800	0.92800	0	0.94400	0.93600	0.92400	0	0.92400	0.85200	0.93600	0	0.91200
cr2	0.85200	0.96400	0	0.82800	0	0	0	0.94400	0	0.95600	0.87200	0.92400	0	0.97200	0.94800

Рисунок 2.2 – Фрагмент записи в журнале логирования после создания новых линий связи между коммутирующими устройствами.

Таблица 2.1 – Результаты измерений количества отправленных запросов

Кол-во измерений (x)	**50**	**100**	**250**	**500**	**1000**	**1500**	**2500**	**5000**	**10000**
P	0,7	0,77	0,9	0,956	0,968	0,967	0,963	0,974	0,979
t, c	0,087	0,145	0,408	0,853	1,663	2,506	4,078	8,228	16,374

На основе полученных данных был построен график зависимости вероятности вхождения отклика в интервал [0; 0.03] от количества измерений. График *P(x)* представлен на рисунке 2.3. На шаге в 2500 измерений, вероятно, на стенд было оказано воздействие, снижающее его производительность.

Также был построен график зависимости времени на измерение от количества измерений *T(x)*, представленный на рисунке 2.4.

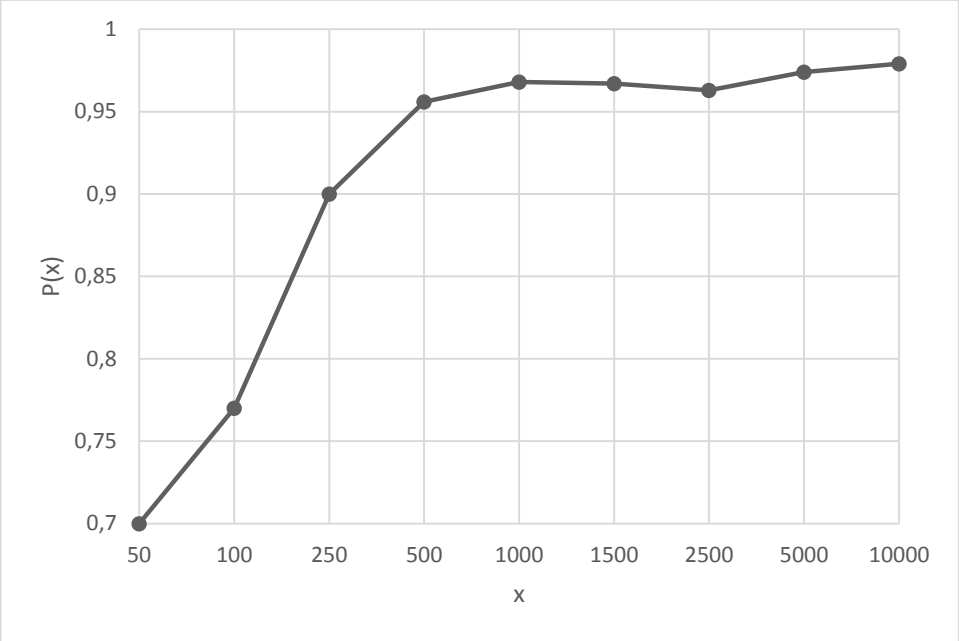

Рисунок 2.3 – График распределения вероятности *P(x).*

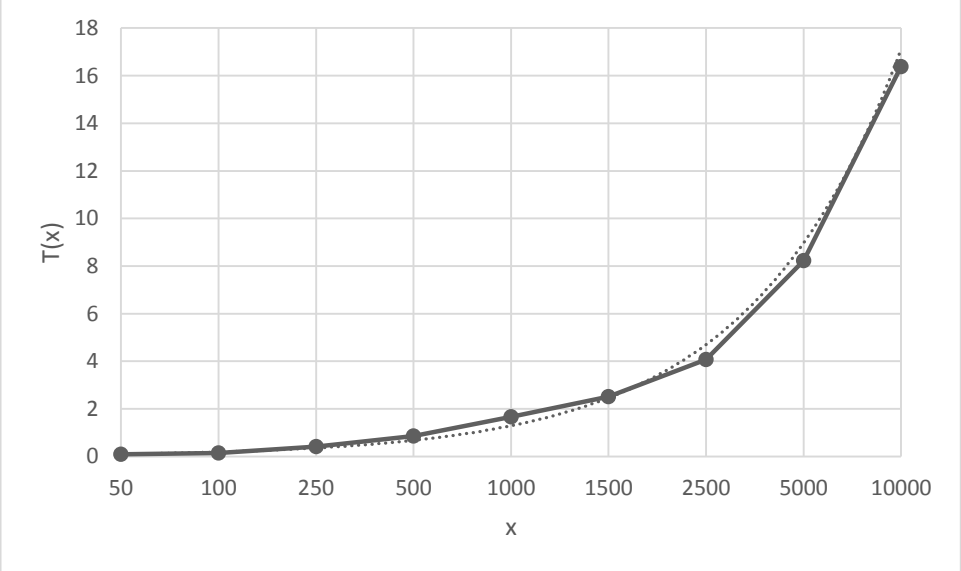

Рисунок 2.4 – График функции *T(x).*

На основе полученных данных было сделано заключение, что для дальнейшего исследования достаточно осуществлять 250 ICMP запросов.

Выбранное количество позволит получить достаточно точную вероятность вхождения в необходимый интервал, сократив при этом время, затраченное на проведение экспериментов.

2.2 Экспериментальное исследование доступности КПТС

Для проведения экспериментов с доступностью КПТС были выявлены возможные типы управляющих воздействий на сеть и её топологию:

1) добавление или удаление узлов сети;
2) добавление или удаление линий связи между устройствами;
3) комбинирование вышеперечисленных воздействий.

В соответствии с типами воздействий были выбраны два сценария для экспериментов.

Первый сценарий заключается в последовательном добавлении случайных избыточных связей между устройствами до тех пор, пока сеть и её топология не преобразуется к виду, где каждое устройство соединено с каждым. Цель эксперимента состоит в нахождении зависимости между общей минимальной и максимальной доступностью сети и связями между устройствами-коммутаторами. На каждой итерации осуществляется логирование результата и проводится оценка доступности. Эксперимент проводился в автоматическом режиме, путем запуска консольного скрипта.

Эксперимент был проведен в двух условиях. В первом случае стенд был подключен к сети Интернет, и во время эксперимента было запущено множество процессов, которые активно потребляли ресурсы процессора. Во втором случае стенд был полностью изолирован от возможных влияний, отключен от сети Интернет, выключены различные сторонние службы и сервисы.

Сценарий второго эксперимента заключается в последовательном удалении промежуточных между корневыми и самыми дальними коммутаторов и их линий связи с другими устройствами, с последующим подключением к вышестоящим устройствам. Эксперимент проводился в ручном режиме, путем

ручного изменения топологии и запуска необходимых процедур для логирования и расчета минимальной и максимальной доступности сети.

Экспериментальное исследование [98] проводилось на виртуальной сети в среде Mininet, схема которой состоит следующих элементов: OpenFlow контроллер (ONOS), 100 оконечных устройств пользователей (PC 1-100), 15 коммутаторов (as1-6, bs1-5, cs1-4), 3 маршрутизатора (ar1-2, br1-2, cr1-2) и линий связи между ними. Изначально к контроллеру подключены три корневых коммутатора, чтобы топология была похожа на типичную сеть небольшой компании. Листинг программы для генерации топологии сети в среде Mininet представлен в приложении Б, схема исследуемой КПТС представлена на рисунке 2.5.

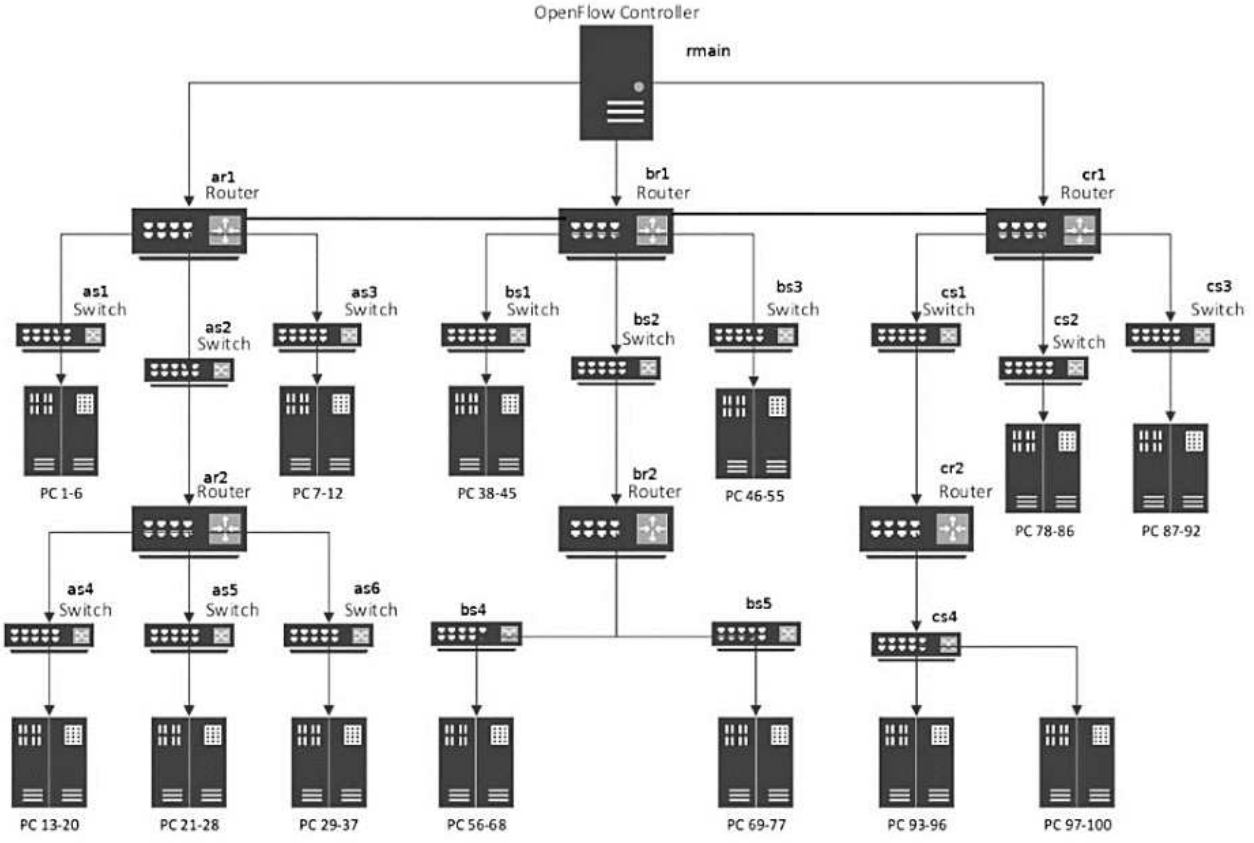

Рисунок 2.5 – Модель КПТС в среде Mininet.

В результате проведения первого эксперимента на основе полученных данных был построен график, представленный на рисунке 2.6, где *А_макс_1* и

А_мин_1 – результат первого опыта, при котором на стенд оказывались воздействия, снижающие его вычислительные возможности, а *А_макс_2* и *А_мин_2* – результат опыта без воздействия на стенд. Эксперимент в обоих случаях проходил около 15 часов в связи с ростом сложности вычислений и количеством замеров вероятности между коммутаторами путем осуществления ICMP запросов.

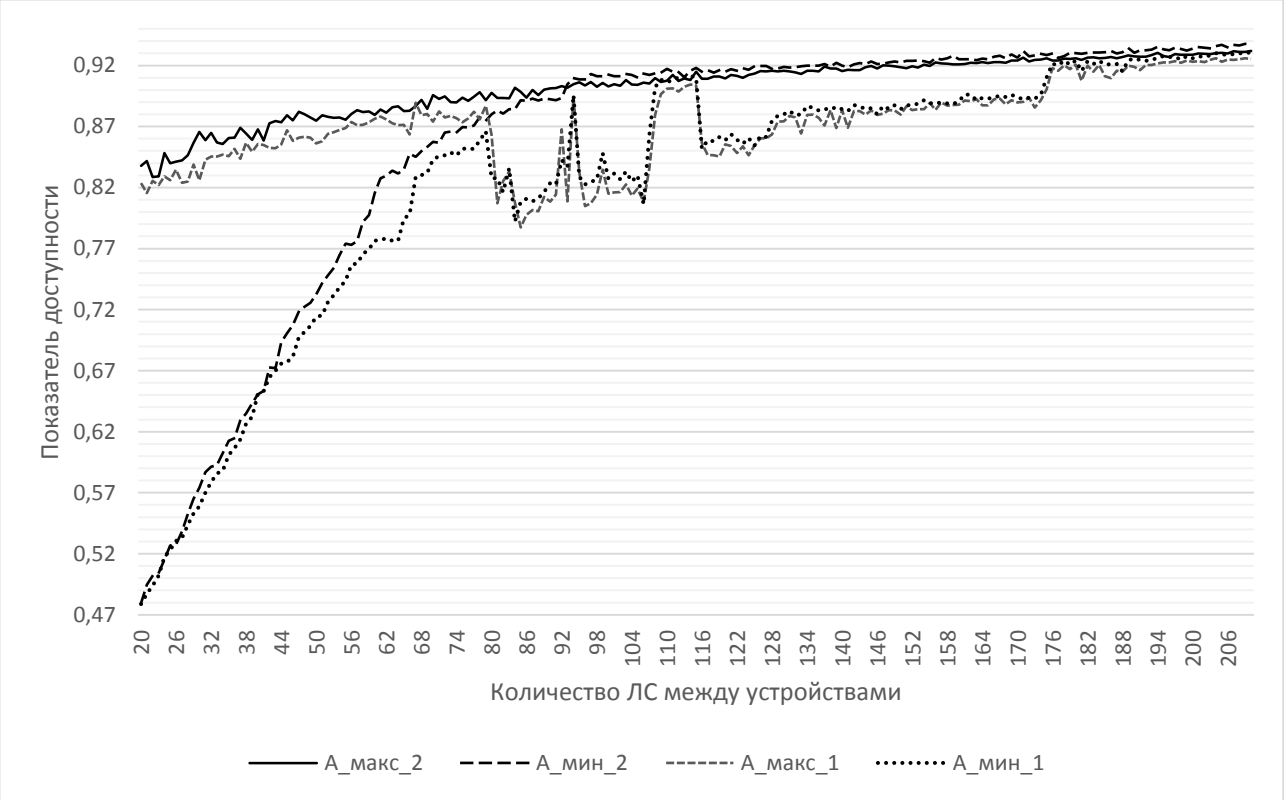

Рисунок 2.6 – График зависимости доступности КПТС от количества линий связи между устройствами.

Анализируя полученный график (рисунок 2.6), можно сделать вывод, что в первом опыте на момент, когда к имеющимся в начальной топологии связям было добавлено 59 новых, показатели доступности стали снижаться. Данный результат доказывает тот факт, что производительность контроллера может существенно снижаться при одновременной работе сторонних сервисов. В то время как изолированный контроллер во втором опыте на протяжении всего эксперимента не имел резких всплесков или падений производительности, а

соответственно и показатель доступности сети изменялся плавно. Минимально допустимый показатель доступности резко повышается с ростом количества связей, но затем его рост замедляется. При добавлении связей до уровня около 40% от максимального количества возможных наблюдается снижение роста показателя $A(G)_{min}$. Дальнейшее добавление избыточных связей начинает постепенно увеличивать показатели $A(G)_{max}$ и $A(G)_{min}$.

Если принять допущение, что на сеть не оказывают влияния внешние факторы, то, аппроксимируя график функции *А_макс_2(х)* и *А_мин_2(х)*, получаем результат, изображенный на рисунке 2.7, где А_макс_2(х) и А_мин_2(х) – результаты для опытов без воздействия на сеть внешних факторов; *Апрокс_макс(х)* и *Апрокс_мин(х)* – графики функций, описывающих поведение *А_макс_2(х)* и *А_мин_2(х)* во время добавления связей.

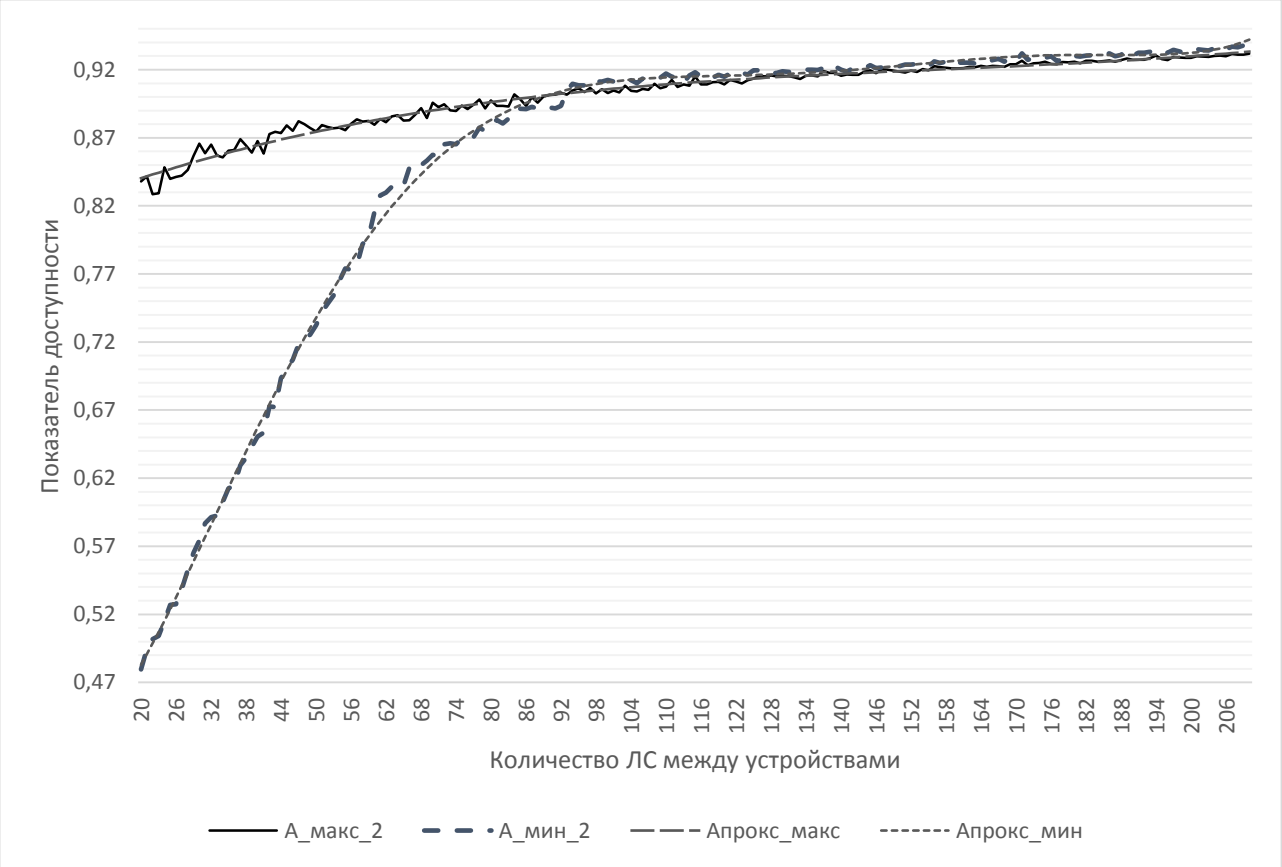

Рисунок 2.7 – График зависимости доступности КПТС от количества линий связи между устройствами и аппроксимирующие функции.

Результаты второго эксперимента приведены в таблице 2.2, где пошагово описан ход выполнения алгоритма по удалению устройств и отражены все произошедшие изменения в топологии. На каждом шаге в приоритете было удаление устройств, поэтому связи удаленных устройств соединялись с устройствами, которые стоят выше в иерархической топологии, после чего повторно замерялись показатели доступности. После пяти удалений устройств появился незначительный рост показателя $A(G)_{min}$. В конце выполнения эксперимента остался один коммутатор, который соединен с OpenFlow контроллером. Максимально возможный показатель доступности варьировался в пределе погрешности измерений и практически не изменялся.

На основе полученных данных в результате проведения второго эксперимента был построен следующий график (рисунок 3.8), демонстрирующий зависимость между показателями доступности КПТС и количеством коммутирующих устройств.

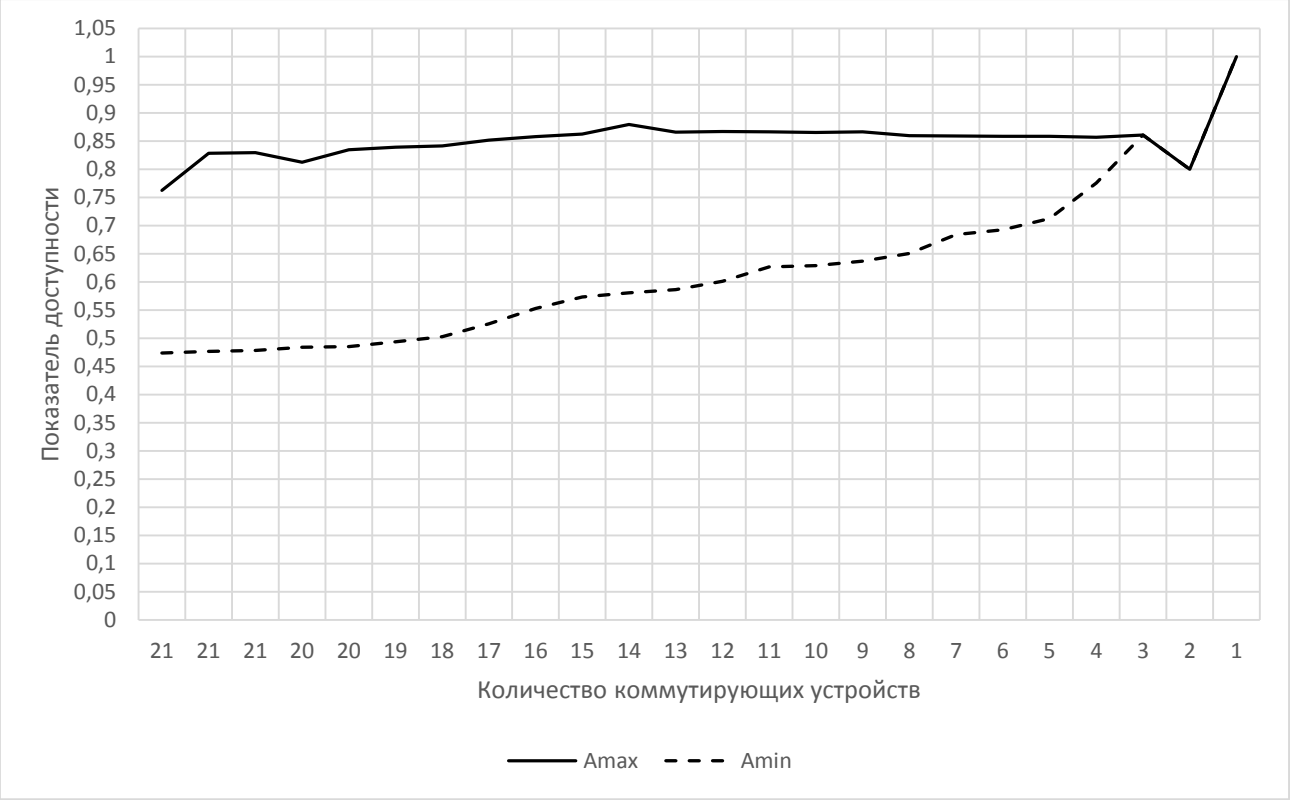

Рисунок 2.8 – График зависимости доступности сети от количества коммутаторов.

Таблица 2.2 – Сводная таблица результатов второго эксперимента

Шаг	**Кол-во устройств**	**Кол-во ЛС между устр-ми**	**Показатель доступности**		**Изменения в топологии**
			$A(G)_{max}$	$A(G)_{min}$	
0	21	20	0.762645	0.473653	–
1	21	20	0.829365	0.476497	as4 отключен от ar2, подключен к as4
2	21	20	0.829365	0.478163	as5 отключен от ar2, подключен к as4
3	20	19	0.812499	0.483864	as6 отключен от ar2, подключен к as4, ar2 удален
4	20	19	0.834756	0.485159	bs4 откл. от br2, подключен к bs2
5	19	18	0.839231	0.493594	bs5 откл. от br2, подключен к bs2
6	18	17	0.841229	0.502812	cs4 отключен от cr2, подключен к cs1, cs2 удален
7	17	16	0.851579	0.525478	as4, as5, as5 подключены к ar1, as2 удален
8	16	15	0.857831	0.552444	bs4, bs5 подключены к br1, bs2 удален
9	15	14	0.862628	0.573403	cs4 подключен к cr1, cs1 удален
10	14	13	0.879254	0.580583	as1 удален
11	13	12	0.865577	0.586455	as4 удален
12	12	11	0.866802	0.60113	as5 удален
13	11	10	0.866802	0.60113	as3 удален
14	10	9	0.866544	0.626413	as6 удален
15	9	8	0.866238	0.636727	bs1 удален
16	8	7	0.859548	0.65065	bs4 удален
17	7	6	0.859276	0.684164	bs5 удален
18	6	5	0.858544	0.69226	bs3 удален
19	5	4	0.858406	0.712436	cs4 удален
20	4	3	0.856706	0.775238	cs2 удален
21	3	2	0.860476	0.860476	cs3 удален
22	2	1	0.800174	0.800174	ar1 удален
23	1	0	1	1	br1 удален

В результате анализа полученного графика и результатов эксперимента можно сделать следующие выводы: удаление устройств незначительно повышает показатель $A(G)_{min}$., практически не оказывая на показатель $A(G)_{max}$ никакого эффекта. Удаление большего количества устройств повышает оба показателя доступности. Таким образом, удаление нескольких устройств можно назвать неэффективной мерой для повышения интегральной доступности сети.

Выводы по результатам экспериментального исследования

В результате проведения экспериментов можно сделать следующие выводы:

1. Путем добавления избыточных связей можно повысить минимальный и максимальный ИПД в соответствии с методикой расчета;

2. Удаление узлов незначительно повышает минимальный и максимальный ИПД, что делает данную меру малоэффективной.

2.3 Алгоритм оптимизации топологии КПТС по критерию максимума интегрального показателя доступности

Представим КПТС как систему. Элементами системы являются маршрутизирующее и коммутирующее оборудование и программируемые связи между ними. Функция системы – множество алгоритмов построения связей между этими элементами, которые зависят от решаемой задачи и состояния внешней среды. Состоянием внешней среды будем определять текущие значения показателей доступности элементов КПТС.

Рассмотрим процесс реализации задачи средствами КПТС. Пусть в начальный момент времени имеем структуру связей (топологию), которая была сформирована вручную администратором КПТС или была «запомнена» по окончанию предыдущего решения данной задачи (S_0). В процессе выполнения текущей задачи меняются значения $a_{ij}, i,j = \overline{1,n}$ за счет динамической

маршрутизации (изменяемой маршрутной матрицы) и, соответственно, структуры связей. Можно представить, что за время решения задачи топология сети изменяется по закону $(S_0, S_1 \ldots, S_l, \ldots, S_{L)}$, где S_l принадлежит полному множеству структур S ($S_l \in S$). Кроме того, S_l обладает обязательным свойством связности, т.е. всегда имеются пути между элементами, задействованными в данный момент.

Постановка задачи [22]: Требуется на каждом *k*-м шаге выполнения задачи синтезировать такую топологию $S_k \in S$, при которой ИПД сети элементов, участвующих в ее выполнении, принимает максимальное значение $A(G) \xrightarrow{S_k \in S} max$, при этом должно быть обязательно обеспечен минимально допустимый показатель доступности.

Введем ограничение [21]. Общее количество связей в S_k не должно превышать количество связей в S_0. Данное ограничение обосновывается следующим: можно предположить, что добавление связей вплоть до получения полносвязного графа повысит $A(G)$, что подтверждается расчетом и экспериментами, однако здесь не учитывается резкое возрастание нагрузки на контроллер, ограничения памяти коммутаторов TCAM (ternary content-addressable memory – троичная ассоциативная память), и прочие эффекты, оказывающие негативное влияние на доступность каналов связи. Кроме того, согласно инструкциям фирм-разработчиков максимальное количество связей ограничивается.

Алгоритм оптимизации топологии КПТС по критерию максимума интегрального показателя доступности **[91]**

Дано S_0 – начальная топология, n – количество элементов, m – количество связей между ними на начальный период выполнения задачи.

Шаг 1. $S_k = S_0$.

Шаг 2. Получить текущие значения $a_{ij}(S_0)$.

Шаг 3. По алгоритму определения ИПД сети рассчитать верхнее и нижнее граничные значения ИПД начальной топологии $A(G(S_0))_{max}$ и $\mathrm{A}(G(S_0))_{min}$.

Шаг 4. Получить остовное дерево графа $G(S_0)$, последовательно исключая связи из S_0. Если таких деревьев несколько, то запомнить структуру S_k с максимальным ИПД сети, при этом минимальный ИПД должен быть не меньше, чем $\mathrm{A}(G(S_0))_{min}$, количество связей m_k. Положить $A_{max} = A(G(S_k))$.

Шаг 5. Для потенциально возможных несуществующих $(m - m_k)$ связей выполнить: последовательно добавлять в S_k связи между каждыми двумя не связанными друг с другом элементами, получая новые структуры $S_k(1), S_k(2), \dots, S_k(l)$ каждый раз проверяя значение ИПД. Запомнить структуру $S_k(l)$, при которой текущее значение $A(S_k(l))$ максимально, при этом минимальный ИПД должен быть не меньше, чем $\mathrm{A}(G(S_0))_{min}$.

Шаг 6. $k = k + 1$. Если задача не выполнена, то $S_k = S_k(l)$, $A_{max} = A(S_k(l))$, перейти к шагу 2, иначе конец алгоритма.

Для экспериментального исследования разработанного алгоритма было создано программное обеспечение на языке Java, позволяющее реализовывать все шаги алгоритма, основные классы и методы программы приведены в приложении Г. Ядро программы заключает в себе несколько интерфейсов, классы, имплементирующие эти интерфейсы, а также хелперы, основной класс Main и несколько файлов с тестами, запускающие различные эксперименты. Среди сущностей проекта следующие:

- генераторы классов;
- инструменты по работе с топологией;
- базовые классы для экспериментов;
- утилиты для работы с матрицами, графами, визуализацией результатов;
- вспомогательные файлы для вышеперечисленных сущностей.

Эксперименты заключались в следующем: на стенде, описанном в разделе 2.1, с помощью Mininet генерировались топологии сети как подмножество узлов и связей, приведенных на рисунке 2.5, для решения выбранной задачи (решаемая задача - типовые приложения для работы с электронным документооборотом на предприятии, охватывающая взаимодействие не менее 50% оконечных устройств пользователей). Запускаются процессы обмена сообщениями между узлами сети, имитирующие действующую сетевую нагрузку. С помощью разработанного скрипта определяются время отклика узлов относительно друг друга [20]. На базе усреднений данных процессов определяются показатели средней доступности узлов (по 250 раз время отклика сравнивается с директивным). Таким образом формируется текущая матрица доступности. Если решение задачи требует нескольких этапов, то процедура эксперимента повторяется.

Далее реализуются процедуры алгоритма оптимизации топологии КПТС по критерию максимума интегрального показателя доступности. Шаг за шагом изменяется топология, меняются интегральные показатели доступности $\mathrm{A}(G(S_0))_{min}$ и $A(G(S_0))_{max}$.

Примеры работы алгоритма приведены на рисунках 2.8-2.13, причем на рисунках 2.8, 2.10, 2.12 приведены начальные сетевые топологии, а на рисунках 2.9, 2.11, 2.13 изображены, соответственно, результирующие топологии с максимальными ИПД (оптимальные топологии).

Эксперименты проводились с варьированием начальной топологии и каждый раз находилась оптимальная топология. Интегральный показатель доступности зависел в основном от текущей нагрузки сети и варьировался в диапазоне от 0,6 до 0,9, при этом выигрыш по сравнению с исходным достигал в среднем 10-15%.

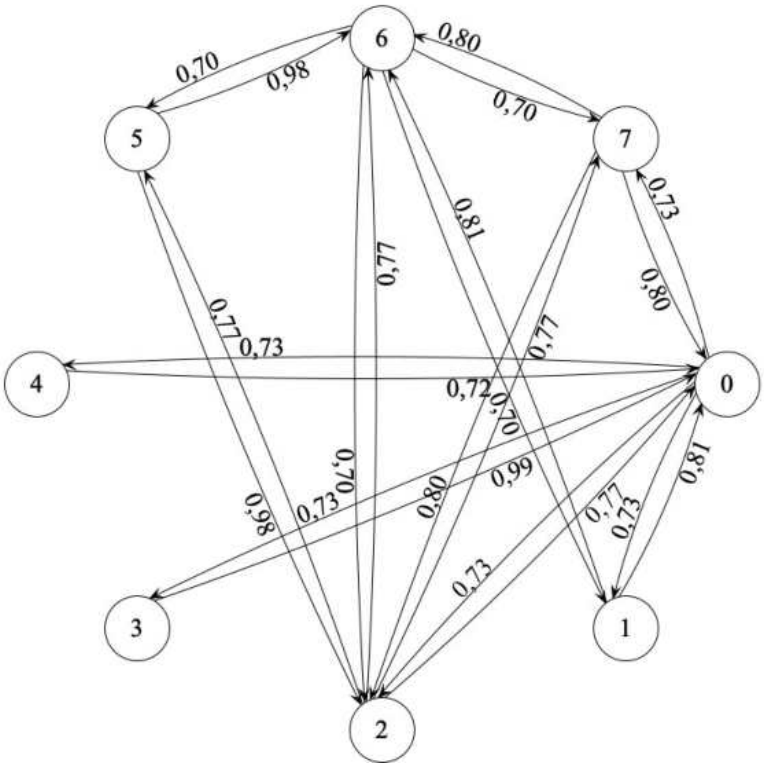

Рисунок 2.8 – Граф начальной топологии, здесь $A(G(S_0))_{max} = 0{,}71$, $\mathrm{A}(G(S_0))_{min} = 0{,}7$.

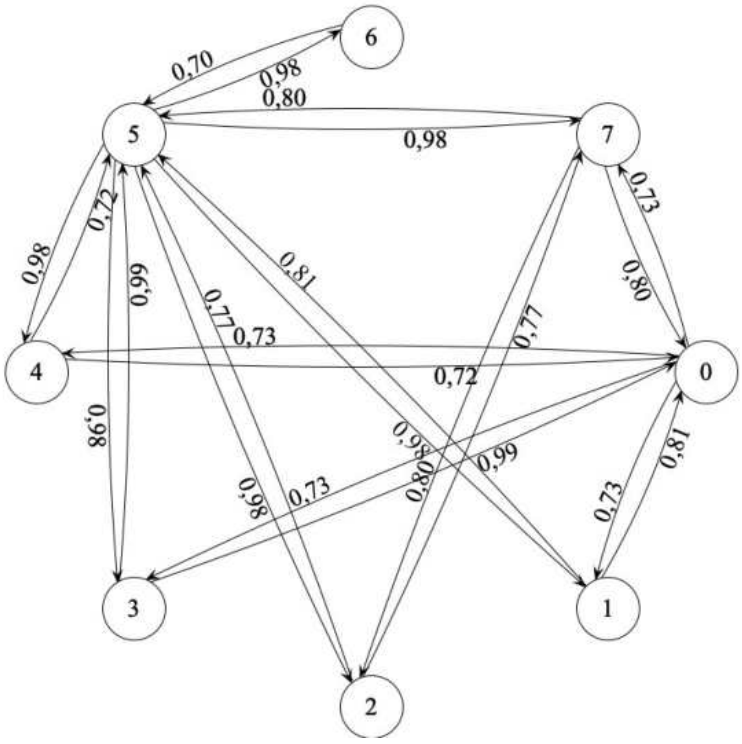

Рисунок 2.9 – Граф оптимальной топологии, здесь $A(G(S_0))_{max} = 0{,}83$, $\mathrm{A}(G(S_0))_{min} = 0{,}71$, выигрыш составил 17,48%.

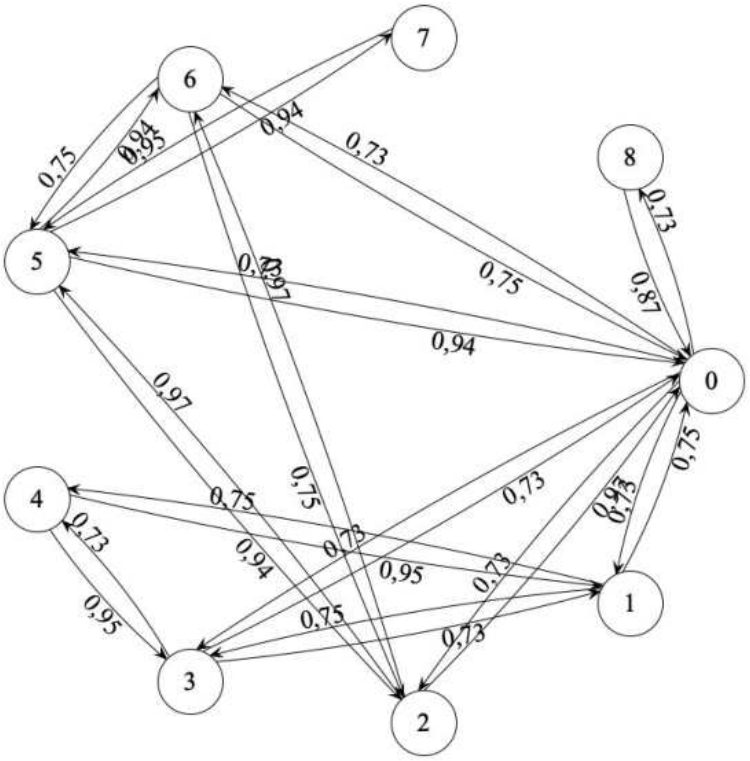

Рисунок 2.10 – Граф начальной топологии, здесь $A(G(S_0))_{max} = 0{,}72$, $\mathrm{A}(G(S_0))_{min} = 0{,}72$.

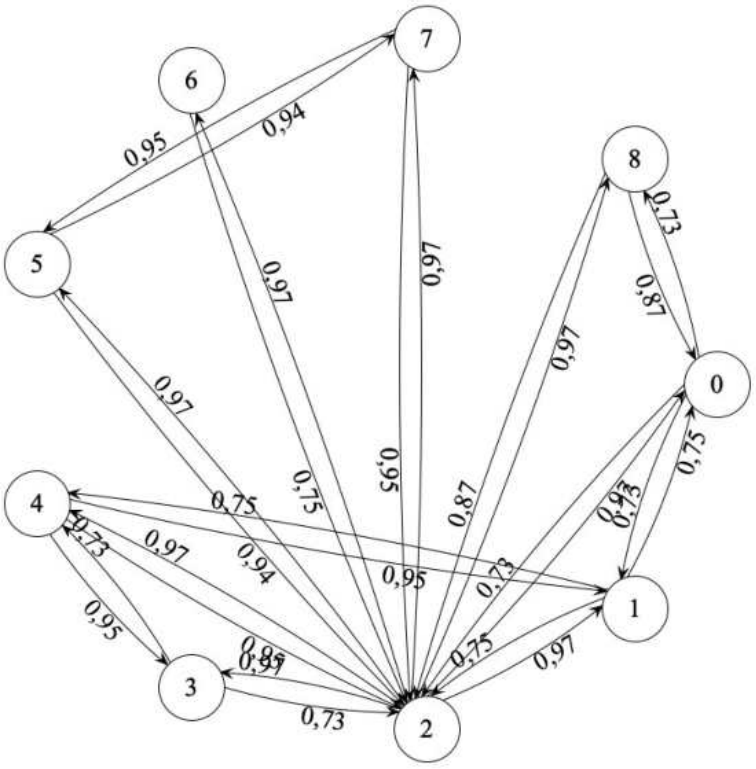

Рисунок 2.11 – Граф оптимальной топологии, здесь $A(G(S_0))_{max} = 0{,}85$, $\mathrm{A}(G(S_0))_{min} = 0{,}83$, выигрыш составил 17,9%.

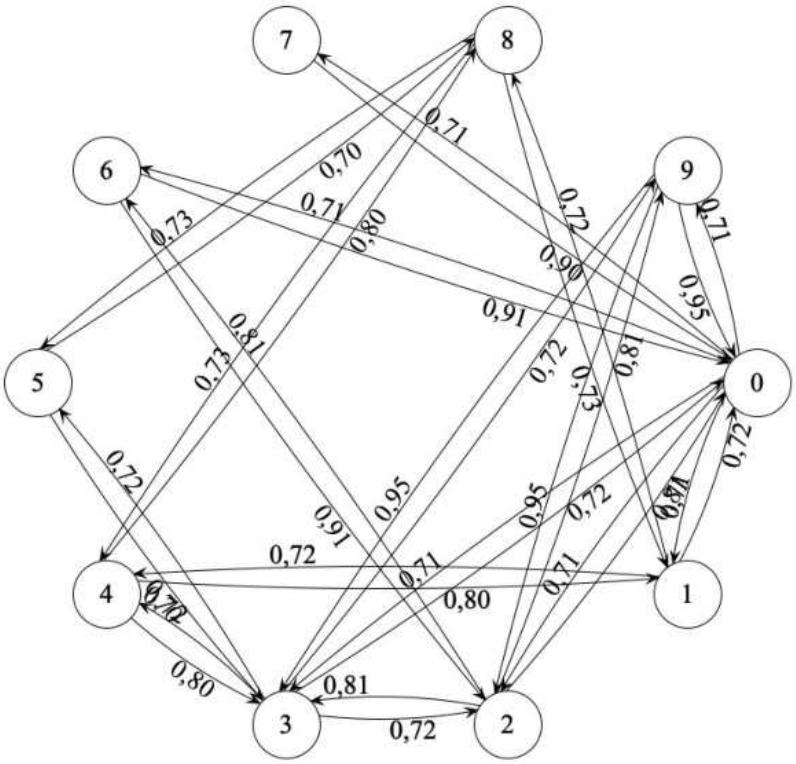

Рисунок 2.12 – Граф начальной топологии, здесь $A(G(S_0))_{max} = 0{,}67$, $A(G(S_0))_{min} = 0{,}64$.

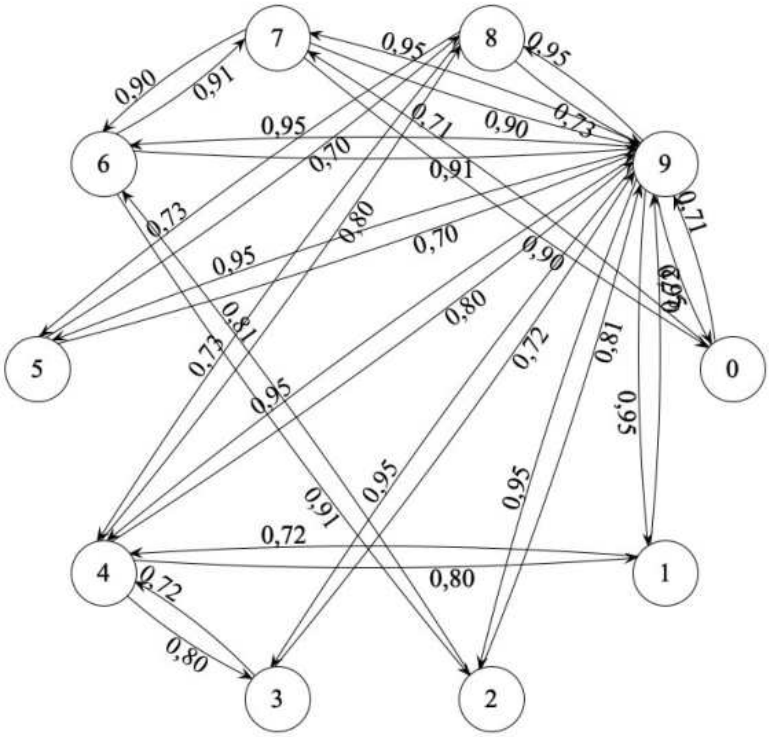

Рисунок 2.13 – Граф оптимальной топологии, здесь $A(G(S_0))_{max} = 0{,}79$, $A(G(S_0))_{min} = 0{,}77$, выигрыш составил 19%.

Исследовалась зависимость выигрыша интегрального показателя доступности от количества связей между устройствами. На рисунке 2.14 изображен график данной зависимости для различного количества задействованных устройств $n = 9$, $n = 12$, $n = 15$, $n = 18$, где по оси абсцисс отложено процентное отношение числа ЛС m_k к максимально возможному $m_{k_{max}}$, где $m_{k_{max}} = n(n - 1)$, а по оси ординат – выигрыш Δ по показателю доступности результирующей топологии относительно исходной, выраженный в процентах:

$$\Delta = \left(\frac{A(G(S_k))_{max} - A\big(G(S_0)\big)_{max}}{A(G(S_0))_{max}} \right) \cdot 100\%$$

Можно сделать вывод, что наибольший эффект разработанный алгоритм показывает, когда коммутирующие устройства связывает от 20% до 35% от максимально возможного количества линий связи.

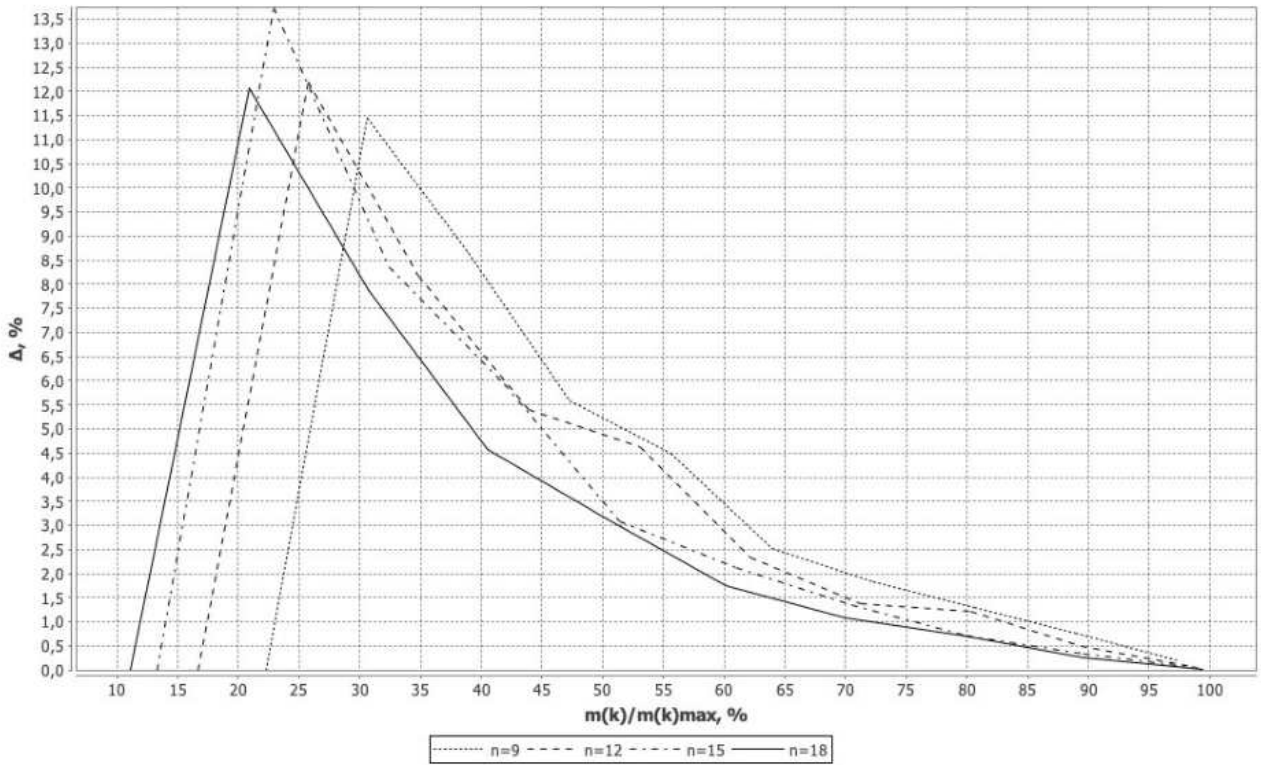

Рисунок 2.14 – Зависимость выигрыша ИПД от количества ЛС между устройствами.

2.4 Внедрение результатов исследования

Результаты исследований в части изучения программно-определяемых сетей, формирования их топологий с заданными свойствами в эмуляторе SDN-сетей Mininet, использования Mininet и разработанных для него скриптов для исследования факторов, влияющих на эффективность функционирования КПТС, внедрены в Федеральном государственном бюджетном образовательном учреждении «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ) и использованы в учебном процессе при подготовке бакалавров и магистров по направлению «Информационная безопасность» и специалистов по специальности «Информационно-аналитические системы безопасности» в рамках проведения учебных занятий по дисциплинам «Сети и системы передачи информации», «Моделирование информационно-аналитических систем» и «Телекоммуникации».

Разработанные в диссертации теоретические положения и практические разработки использованы при проведении лекционных, лабораторных и практических занятий, а также в научно-исследовательской работе и дипломном проектировании студентов, что подтверждается актом внедрения в учебный процесс (Приложение Д).

Результаты диссертационной работы в виде алгоритма оптимизации топологии программно-определяемой телекоммуникационной сети и реализующего его программного обеспечения применяются в программно-определяемой распределенной сети (SD-WAN) ООО «Рунет бизнес системы», что подтверждено актом об использовании результатов исследования (Приложение Д). Использование указанных результатов позволило найти более оптимальные для решения различных задач топологии сети с точки зрения доступности сетевых сервисов, что повысило эффективность обслуживания трафика SD-WAN в среднем на 10% за счет снижения числа необслуженных сетевых пакетов.

Кроме того, скрипт управления контроллером SDN, имплементирующий разработанный в рамках диссертационного исследования алгоритм оптимизации топологии программно-определяемой телекоммуникационной сети внедрен и практически используется в SDN центра обработки данных системы образования Владимирской области. Акт о практическом применении приведен в Приложении Д. Применение данного алгоритма произвело следующий эффект: доля сетевых пакетов, своевременно доставленных в рамках соблюдения требований QoS, возросло в среднем до 15%.

Научно-технические результаты диссертационного исследования, описанные в данной главе, используются и будут включены в итоговый отчет по научно-исследовательской работе в рамках государственного задания ВлГУ, проект № FZUN-2020-0013.

Выводы по главе 2

1. Создан программно-аппаратный стенд в среде Mininet для проведения экспериментов, позволяющий формировать произвольные топологии SDN, осуществлять маршрутизацию потоков трафика на базе контроллера ONOS, а также производить расчет показателей доступности. Эксперименты позволили выявить существенные факторы воздействия на топологию в программно-определяемых сетях с высокой доступностью. Выявлена зависимость влияния связей между устройствами и количеством устройств в сетевой топологии на интегральный показатель доступности сети. Исследование показало, что эффективным методом для улучшения этого показателя в сети с реактивным режимом обработки входящих потоков является увеличение количества связей между коммутирующими устройствами.

2. Разработан алгоритм оптимизации топологии КПТС, основанный на последовательной реконфигурации топологии сетевых средств коммутации и маршрутизации по критерию максимума интегрального показателя доступности, что позволяет подстраивать топологию КПТС под изменяющиеся внешние

условия и решаемую задачу. Экспериментальные исследования показали: оптимальная топология находилась каждый раз при многократном изменении начальной топологии; интегральный показатель доступности зависел в основном от текущей нагрузки сети и варьировался в диапазоне от 0,6 до 0,8, при этом выигрыш по сравнению с исходной топологией достигал 15%, а в ряде случаев до 22%. Наибольший эффект разработанный алгоритм показывает, когда при решении задачи задействовано много узлов.

3 АЛГОРИТМЫ УПРАВЛЕНИЯ ПОТОКОМ В КПТС ПО КРИТЕРИЮ ДОСТУПНОСТИ

В данной главе предлагается метод повышения доступности канала связи КПТС. Описывается разработанный алгоритм планирования очередей передачи данных на основе приоритизации, который позволяет оптимизировать использование пропускной способности и обеспечивать минимально возможную задержку для приоритетных классов.

3.1 Исследование алгоритма приоритизации и управления потоком НТВ

Одним из эффективных способов повышения доступности в КПТС является внедрение технологии QoS. Использование алгоритмов приоритизации и контроля трафика (ПКТ) позволяет отделять трафик функционирующих сервисов из общего потока и обеспечивать гарантированную полосу пропускания для них. Конфигурирование данных алгоритмов производится на основании выделения части КС для различных типов трафика. Однако, такое распределение не учитывает особенности проходящего трафика и не гарантирует обеспечение доступности трафика, чувствительного к задержкам канала. Тем самым, доступность сервиса понижается, что ведет к потенциальной угрозе безопасности системы. Данную проблему возможно решить путем разработки алгоритмов ПКТ, позволяющих оптимально распределять трафик, проходящий через КС, с целью повышения доступности определенных типов трафика [15; 29].

Под повышением доступности каналов связи мы будем понимать гарантированное попадание в заданные временные интервалы (директивное время), которое обеспечивается посредством уменьшения времени отклика сервиса путем минимизации времени обработки пакетов. Задаваемые временные интервалы, требуемые для каждого типа трафика (сервиса), указываются в SLA, далее будем называть эти интервалы максимальным директивным временем (мдв).

Концептуальная модель. Выделим сущности: множество соглашений об уровне обслуживания с элементами $\Delta_{\text{сер}}(i)$ – заданное (максимально допустимое) время отклика для i-го сервиса; множество откликов сервисов с элементами $t(i)$. $t(i)$ зависит от задержки маршрутизирующего оборудования $\Delta_{\text{МО}}(i)$, задержки ОС $\Delta_{\text{ОС}}(i)$, задержки ПО сервиса $\Delta_{\text{ПО}}(i)$. Следовательно, $t(i) = \Delta_{\text{МО}}(i) + \Delta_{\text{ОС}}(i) + \Delta_{\text{ПО}}(i)$. Введем ограничения $Y = \{Y_1, Y_2, Y_3\}$ на: размер буферной памяти маршрутизируемого оборудования Y_1, скорость передачи пакетов сетевым оборудованием Y_2, работы нескольких сервисов в равных приоритетах Y_3. Тогда функция доступности i-го сервиса выглядит следующим образом: $t(i, Y) \leq \Delta_{\text{сер}}(i)$.

Исследования показывают прямую зависимость отклика сервиса от используемого алгоритма планирования очередей. Чаще всего для классификации трафика используется алгоритм планирования Hierarchical Token Bucket. Рассмотрим принцип работы данного алгоритма [39; 52; 53; 57; 58].

Классовая дисциплина HTB предназначена для разделения полосы пропускания между различными типами трафика, каждый из которых может получить долю от гарантированной полосы пропускания. Алгоритм подразумевает классификацию трафика по определенным признакам, таким как: IP-адрес назначения или источника, порт назначения или источника, протокол передачи данных и т.д. Каждый класс соответствует определенному типу трафика и имеет свой приоритет в соответствии с SLA. Каждый класс имеет свою очередь накопления пакетов, при этом алгоритм HTB выстраивает классы в виде дерева.

Класс службы определяет параметры контроля, такие как максимальная пропускная способность (*max-limit* или *max-rate*) или максимальный размер пакета и использует дисциплину очереди для обеспечения соблюдения этих правил. Планировщик и класс связаны друг с другом, а правила, определенные классом, должны быть связаны с предопределенной очередью. В большинстве

случаев каждый класс владеет одной дисциплиной очереди, но также возможно, что несколько классов совместно используют одну и ту же очередь. В большинстве случаев при постановке пакетов в очередь компоненты контроля определенного класса отбрасывают пакеты, превышающие определенную входную интенсивность.

Дисциплина HTB управляет потоком сетевых пакетов путем выделения токенов на их передачу в соответствии с приоритетами. Любой дочерний (листовой) класс, который хочет заимствовать токен, будет запрашивать его у своего родительского класса, который, в свою очередь, может заимствовать у своего родительского класса, пока токен не будет найден или корневой класс не будет достигнут. Шейпинг происходит только в листовых классах.

HTB гарантирует, что пропускная способность, предоставляемая каждому классу, составляет по крайней мере минимум назначенного ему значения (*limit-at*). Когда класс требует меньше назначенного количества, оставшаяся (избыточная) полоса пропускания распределяется среди других классов, которые требуют обслуживания.

Экспериментальное исследование алгоритма HTB. Выявление параметров оптимизации

Для выявления узких мест в функционировании алгоритма HTB в условиях нарушения сетевой доступности на экспериментальной установке (рисунок 3.1) было проведено исследование его типовой конфигурации в различных режимах нагрузки. Целью эксперимента являлось определение ключевых аспектов, влияющих на обеспечение низкой задержки передачи пакетов при различной интенсивности входящего потока пакетов. Элементы экспериментальной установки представлены в таблице 3.1. Установка была развернута на виртуальной машине VMWare.

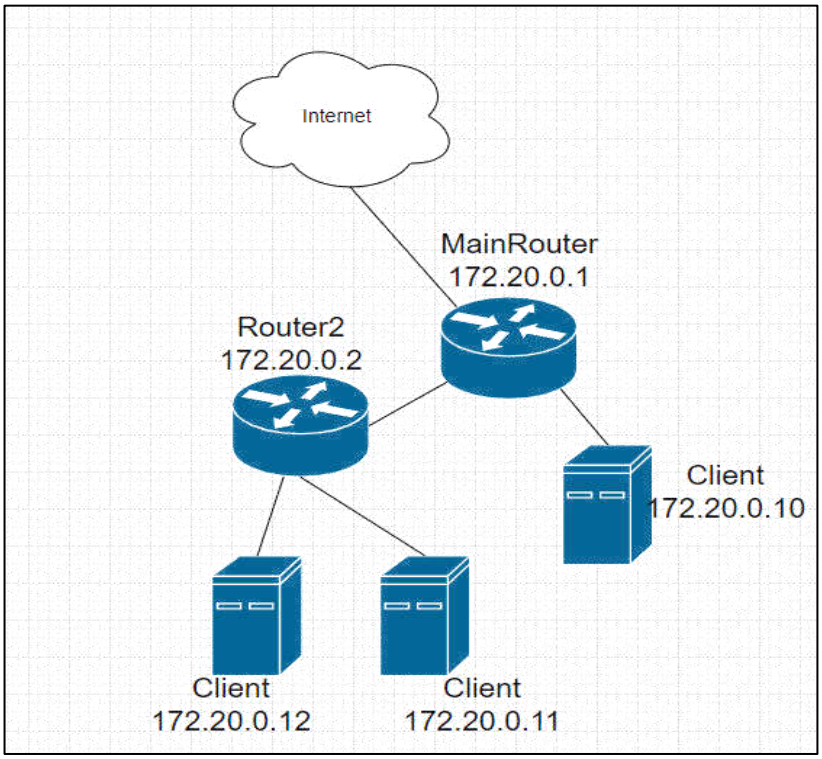

Рисунок 3.1 – Схема тестируемой КСПД.

Таблица 3.1 – Состав экспериментальной установки

Наименование элемента	**Описание элемента**
MainRouter	Программный маршрутизатор интернет-провайдера на ОС xUbuntu, обеспечивающий оказание услуги по предоставлению доступа к сетевым сервисам
Client-1, Client-2, Client-3	Рабочие станции клиентов на ОС xUbuntu
Router-2	Программный маршрутизатор, предназначенный для распределения передачи пакетов узлов сети (Client-2, Client-3)

Конфигурация алгоритма планирования очередей HTB представлена в таблице 3.2.

В рамках проведения эксперимента были определены 5 режимов тестирования алгоритма, представленные в таблице 3.3.

Таблица 3.2. Конфигурация HTB

Устройство	**Родительский класс (устройство)**	**Приоритет**	**Скорость передачи данных класса (Mbit/s)**
MainRouter	-	0	10
Router-2	MainRouter	1	5
Client-1	MainRouter	0	5
Client-2	Router-2	0	3
Client-3	Router-2	1	2

Таблица 3.3 – Показатель значений интенсивности в заданный интервал времени

Режим тестирования	**Значение интенсивности первой очереди (приоритетная)**	**Значение интенсивности второй очереди**	**Значение интенсивности третьей очереди**
1	Первоначальное	Повышается	Повышается
2	Первоначальное	Первоначальное	Первоначальное
3	Первоначальное	Повышается	Первоначальное
4	Первоначальное	Первоначальное	Первоначальное
5	Повышается	Повышается	Повышается

Значения интенсивности входящего потока пакетов, определенные для каждого нагружаемого класса, представлены в таблице 3.4. Изменения интенсивности производились при помощи скрипта, задействующего утилиту hping3. Размер пакета составляет 10000 байт.

На данный момент существует три метода, позволяющие вызывать модуль управления трафиком HTB в операционной системе:

– традиционная командная строка UNIX, использующая утилиту traffic control (*tc*) из программного пакета iproute2;

– HTB-инструменты, предложенные Spirlea, Subredu и Stanimir [74] для упрощения процесса распределения пропускной способности.

– набор интерфейсов WEB-инструментов: WebHTB [121] и T-HTB, используемых для формирования пакетов.

Таблица 3.4 – Интенсивности входящего потока пакетов

Время, с	**Значение интенсивности (интервал в секундах между отправкой пакетов)**
1-15	0,5 (первоначальная)
15-55	0,1
55-75	0,075
75-100	0,05

В данной работе был выбран первый вариант, поэтому трафик будем контролировать через утилиту tc из программного модуля iproute.

Рассмотрим работу алгоритма HTB в Linux, используя утилиту hping3 для определения задержек в классах приоритетов. Результаты сравнения представлены на рисунках ниже.

Во время тестирования первого режима (рисунок 3.2), мы видим, что при повышении интенсивности второй и третьей очереди повышается задержка во всех очередях. Первая, наиболее приоритетная очередь имеет самую низкую задержку из всех, поскольку обладает наибольшей пропускной способностью. Так как приоритет второй очереди выше, чем у третьей, оставшаяся пропускная способность делится между ними в пользу второй. При повышении значении интенсивности в три раза мы видим, что возникает переполнение буфера и алгоритм сбрасывает передачу пакетов.

При тестировании второго режима (рисунок 3.3) с повышением интенсивности потока в первой очереди наблюдается рост задержки во всех классах вследствие повышения интенсивности потока в приоритетном классе. Распределение пропускной способности способствует тому, что задержка наименее приоритетной очереди является самой высокой, поскольку жертвует своей пропускной способностью в пользу приоритетной.

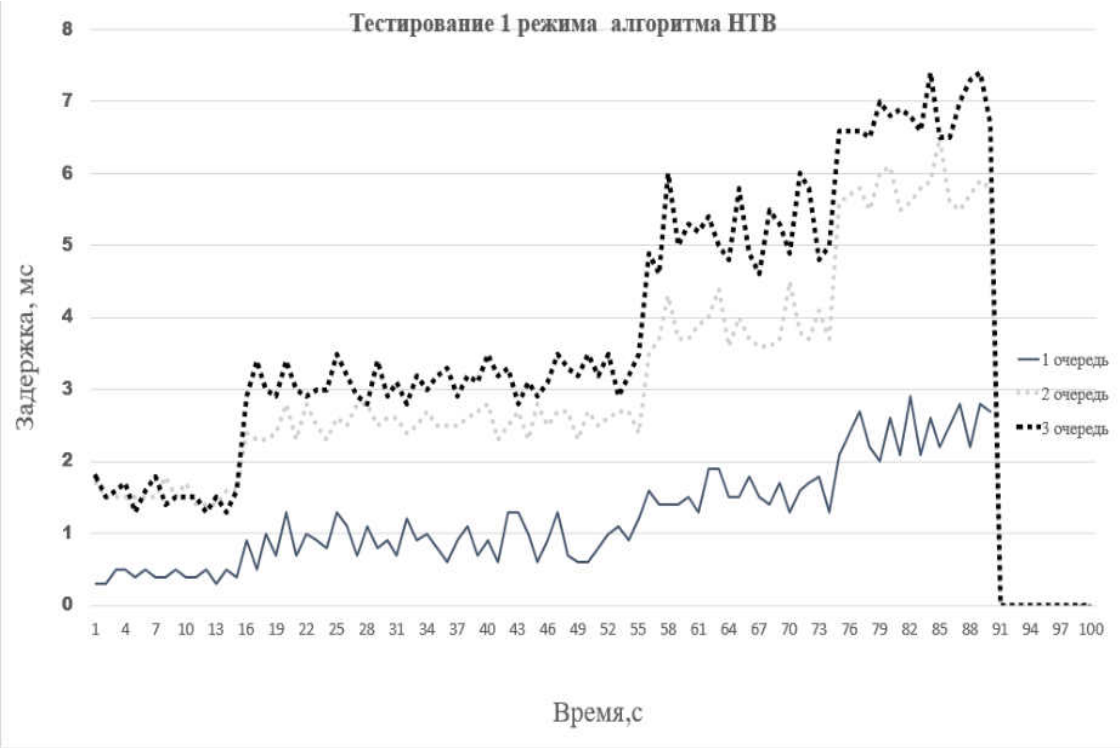

Рисунок 3.2 – Тестирование НТВ в 1 режиме.

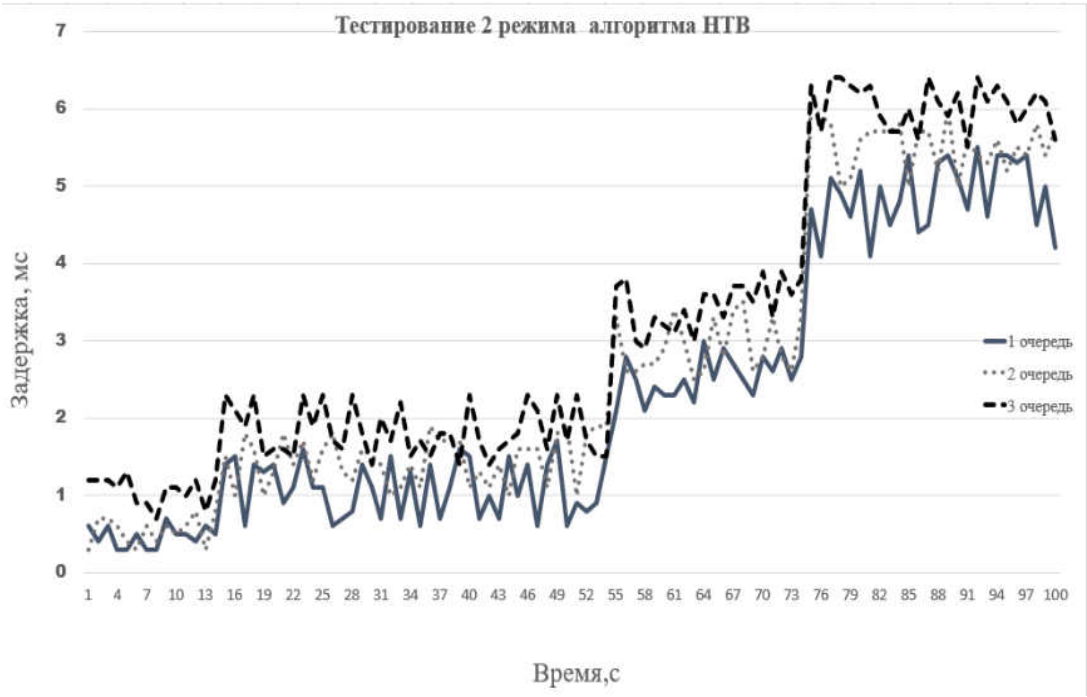

Рисунок 3.3 – Тестирование НТВ во 2 режиме.

Третий режим (рисунок 3.4) с повышением интенсивности второго класса и фиксировании значения интенсивностей остальных очередей показывает, что задержка очереди второго класса формируется за счёт третьей очереди. Первая очередь обладает небольшой задержкой, поскольку обладает наивысшим приоритетом, но при третьем повышении интенсивности мы видим повышение задержки за счет ограничения общей пропускной способности.

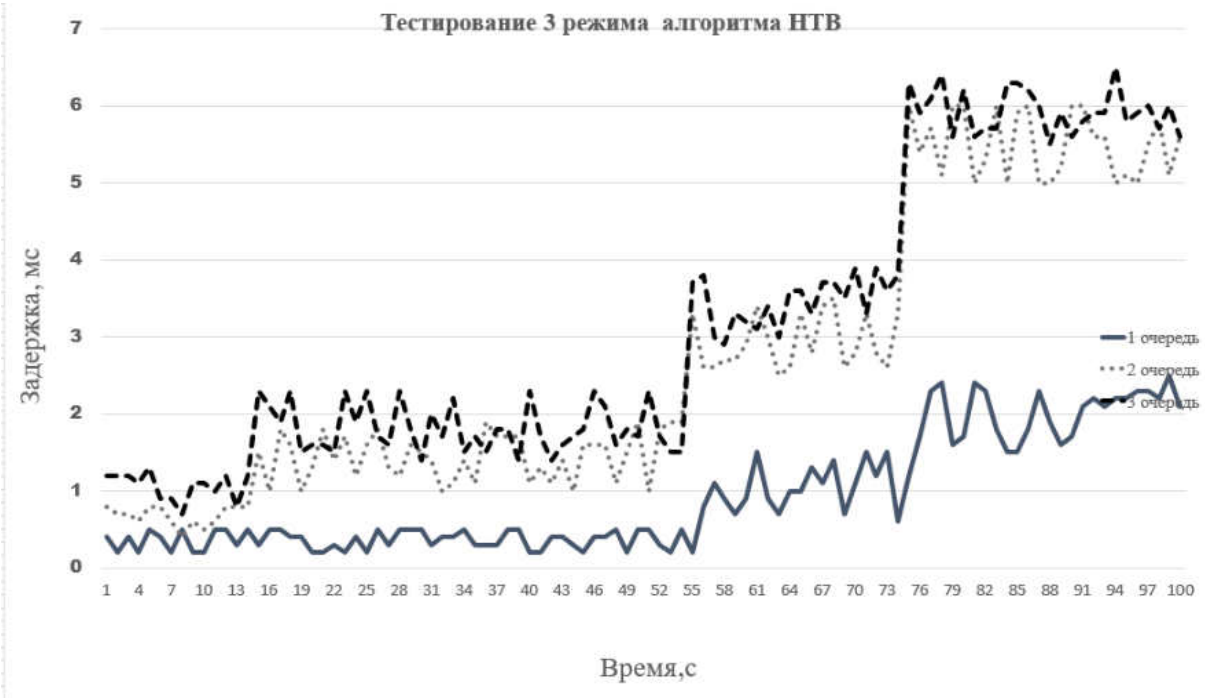

Рисунок 3.4 – Тестирование НТВ в 3 режиме.

При низкой интенсивности входящего потока пакетов в режиме 4 (рисунок 3.5) обеспечивается низкая задержка на обработку. Однако для низкоприоритетной очереди задержка выше, чем у приоритетного класса, что обусловлено последовательностью права передачи пакетов на основании приоритета.

При повышении показателей интенсивности потока всех очередей (5 режим, рисунок 3.6), наблюдается высокая задержка передачи пакетов. На графике мы видим распределение задержек каждой очереди в зависимости от ее приоритета. При последнем повышении интенсивности наблюдаются потери пакетов.

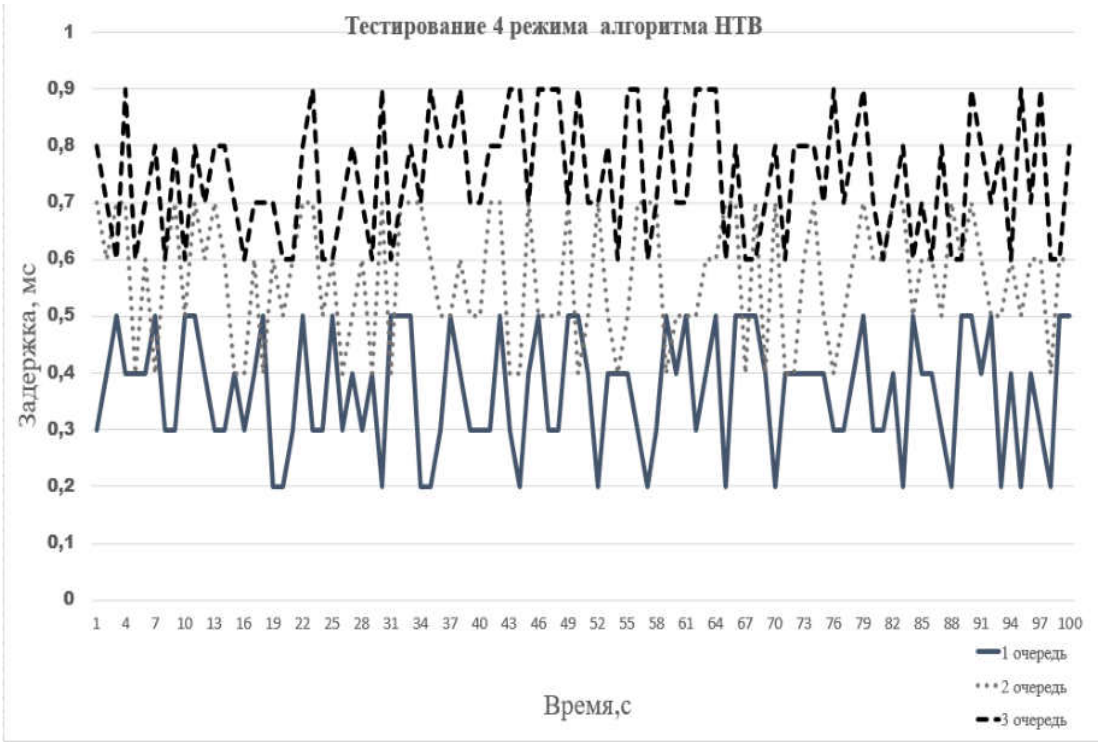

Рисунок 3.5 – Тестирование НТВ 4 режим.

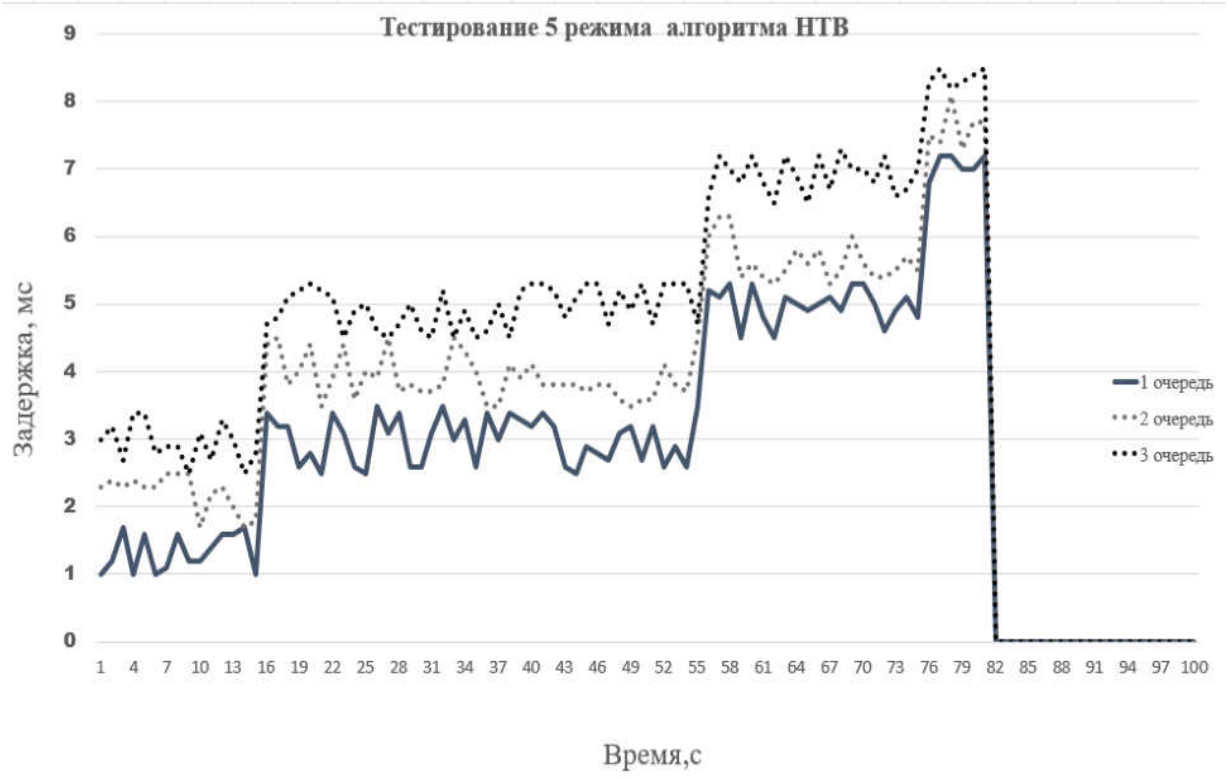

Рисунок 3.6 – Тестирование НТВ 5 режим.

Выводы по результатам экспериментального исследования

Анализ полученных в результате эксперимента данных показывает следующие возможные направления оптимизации параметров, влияющих на задержку передачи пакетов: контроль интенсивности входящего потока пакетов, динамическое изменение пропускной способности канала относительно входящей интенсивности, оптимизация предоставления полосы для классов трафика относительно входящей интенсивности. Выявлена необходимость в контроле размера буфера и защите его от переполнения, также необходимо, чтобы приоритетные очереди напрямую не зависели от заданного им параметра пропускной способности, а обладали гарантированной задержкой.

3.2 Алгоритм планирования очередей передачи данных

В данной диссертационной работе алгоритм HTB будет изменен [27]: при выделении токенов на передачу алгоритм будет опираться не на указанные ему параметры полосы пропускания, а на соблюдение вхождения задержки класса трафика директивному интервалу времени.

Соответственно, для каждого класса трафика определяется директивное время задержки пакета в очереди на ожидание передачи сетевым оборудованием. Каждый класс имеет свою очередь накопления пакетов. Выпуск пакетов в передающую очередь осуществляется со скоростью, рассчитанной на основании директивного времени и алгоритма планирования использования пропускной способности.

Подход подразумевает классификацию сетевого трафика по определенным признакам, таким как: IP-адрес узла назначения, узла источника, используемый порт узла назначения, узла источника, реализуемый протокол передачи данных. Каждому классу трафика определяется приоритет в соответствии с соглашением о качестве обслуживания (SLA). Для каждого класса трафика определяется директивное время задержки пакета в очереди на ожидание передачи сетевым оборудованием. Каждый класс имеет свою очередь накопления пакетов. Выпуск

пакетов в передающую очередь осуществляется со скоростью, рассчитанной на основании директивного времени и алгоритма планирования использования пропускной способности.

Математически это можно описать: множеством классов трафика $Kl = \{Kl_1, \dots, Kl_b\}$, где b – количество классов, соответственно задается множество приоритетов класса: $Pr = \{Pr_0, \dots, Pr_{b-1}\}$, чем меньше значение Pr_i, тем выше приоритет класса; множеством скоростей потока пакетов для каждого класса $Sp = \{Sp_1, \dots, Sp_b\}$, $\sum_{i=1}^{b} Sp_i \le max\_rate$, где max_rate – максимальная пропускная способность интерфейса; множеством директивного времени обработки пакетов для каждого класса $T^{\text{обр}} = \{T^{\text{обр}}{}_1, \dots, T^{\text{обр}}{}_b\}$, соответственно интенсивности обслуживания пакетов $\mu_i = 1/T^{\text{обр}}{}_i$; множеством допустимых задержек в очередях для классов $Del = \{Del_1, \dots, Del_b\}$; множеством интенсивностей входящего трафика пакетов $\lambda = \{\lambda_1, \dots, \lambda_b\}$.

Алгоритм определяет одну очередь передачи для пакетов различных классов. Из очередей классов пакеты выходят с интенсивностью меньшей или равной μ_i. Пакеты попадают в очередь последовательно в соответствии с приоритетами Pr_i.

Алгоритм планирования очередей передачи данных

Шаг 1. Ввод исходных данных: $Kl = \{Kl_1, \dots, Kl_b\}$, $Pr = \{Pr_0, \dots, Pr_{b-1}\}$, $T^{\text{обр}} = \{T^{\text{обр}}{}_1, \dots, T^{\text{обр}}{}_b\}$, max_rate, min_delay, MTU, $Sp = \{Sp_1, \dots, Sp_b\}$, $\lambda = \{\lambda_1, \dots, \lambda_b\}$.

Шаг 2. Поставить соответствие номеров классов приоритетам – $Kl_0 = Pr_0, \dots, Kl_{b-1} = Pr_{b-1}$.

Шаг 3. Если условие $\sum_{i=0}^{b-1} Sp_i \le max\_rate$ выполняется, то переход к шагу 4, иначе устанавливаем текущее значение времени обработки пакета для каждого класса в минимально допустимое время обработки: $T_i^* = min\_delay, i = 0, \dots, (b-1)$.

Шаг 4. Определяем суммарное количество токенов, которое в дальнейшем будем распределять по классам – $Tok = max\_rate/(MTU \times 8)$.

Далее производится распределения токенов по классам (полосы пропускания выпускающей очереди) таким образом, что более приоритетный класс забирает 2/3 доступных токенов (полосы пропускания), каждый следующий по приоритету класс занимает 2/3 полосы, оставшейся после предыдущего по приоритету класса, и так далее до последнего класса, который занимает оставшуюся полосу. $i = 0$

Шаг 5. Количество токенов для i-го класса, $Ent(.)$ – целая часть.

$$Tok_i = \begin{Bmatrix} Ent\left(Tok \times \frac{2}{3}\right) + 1, \text{если } \left(Tok \times \frac{2}{3}\right) > Ent\left(Tok \times \frac{2}{3}\right), \\ Tok \times \frac{2}{3}, \text{если } Ent\left(Tok \times \frac{2}{3}\right) = Tok \times \frac{2}{3}. \end{Bmatrix}$$

Шаг 6. Время обработки пакетов i-го класса $T_i^* = (8 \times MTU)/(max\_rate \times Tok_i)$.

Шаг 7. Если $T_i^* \leq T^{\text{обр}}{}_i$ (меньше директивного), то $\mu_i^* = 1/T_i^*$; $i = i + 1$;

Если $i < b$ (остались классы), то переход к шагу 5,

иначе если $Tok = 0$ (не остались токены), то конец алгоритма;

иначе конец алгоритма.

иначе если $T_i^* \geq min\_delay$, то $Tok_i = Tok_i + 1$, $Tok = Tok - 1$,

если $Tok = 0$ (не остались токены), то конец алгоритма;

иначе переход к шагу 6;

иначе (токенов для i-го класса слишком много)

$Tok_i = Tok_i - 1$, $Tok = Tok + 1$, переход к шагу 6;

Если интенсивность входного потока в более приоритетном классе ниже выходной пороговой интенсивности, то следующий по приоритету класс может произвести повышение пороговой выходной интенсивности (при сохранении $\sum_{i=1}^{b} Sp_i \leq max\_rate$). Право повышения пороговой выходной интенсивности переходит нижестоящим по приоритету классам.

Если интенсивность потока пакетов больше пороговой интенсивности потока, то пакеты начинают отбрасываться для уравнивания интенсивности потока до пороговой интенсивности потока. Повышение или понижении интенсивности пакетов определяется в результате работы алгоритма планирования использования пропускной способности посредством перераспределения токенов HTB на передачу.

На рисунке 3.7 представлена блок-схема разработанного алгоритма.

Тестирование алгоритма

Для проверки эффективности разработанного алгоритма было произведено имитационное моделирование его работы в среде AnyLogic в сравнении с работой алгоритма HTB в типовой конфигурации.

Состав моделируемой системы

Моделируемая система состояла из следующих элементов:

− очереди передачи пакетов с приоритетом 0;

− очереди передачи пакетов с приоритетом 1;

− очереди передачи пакетов в среду передачи сигнала;

− планировщика распределения использования очереди передачи пакетов в среду для очередей с приоритетами.

Каждая моделируемая очередь обладала следующими параметрами:

− интенсивность потока пакетов λ_i;

− время обработки пакетов в очереди, χ_i;

− гарантированная задержка для передачи пакетов очереди, d_i;

Для очередей задавались следующие ограничения:

− интенсивность пакетов в очередь передачи пакетов в среду передачи не может превышать значение *max_rate*;

− время обработки пакетов в очереди передачи пакетов в среду передачи всегда составляет *min_delay*;

− буфер очередей с приоритетами равен 100 пакетам.

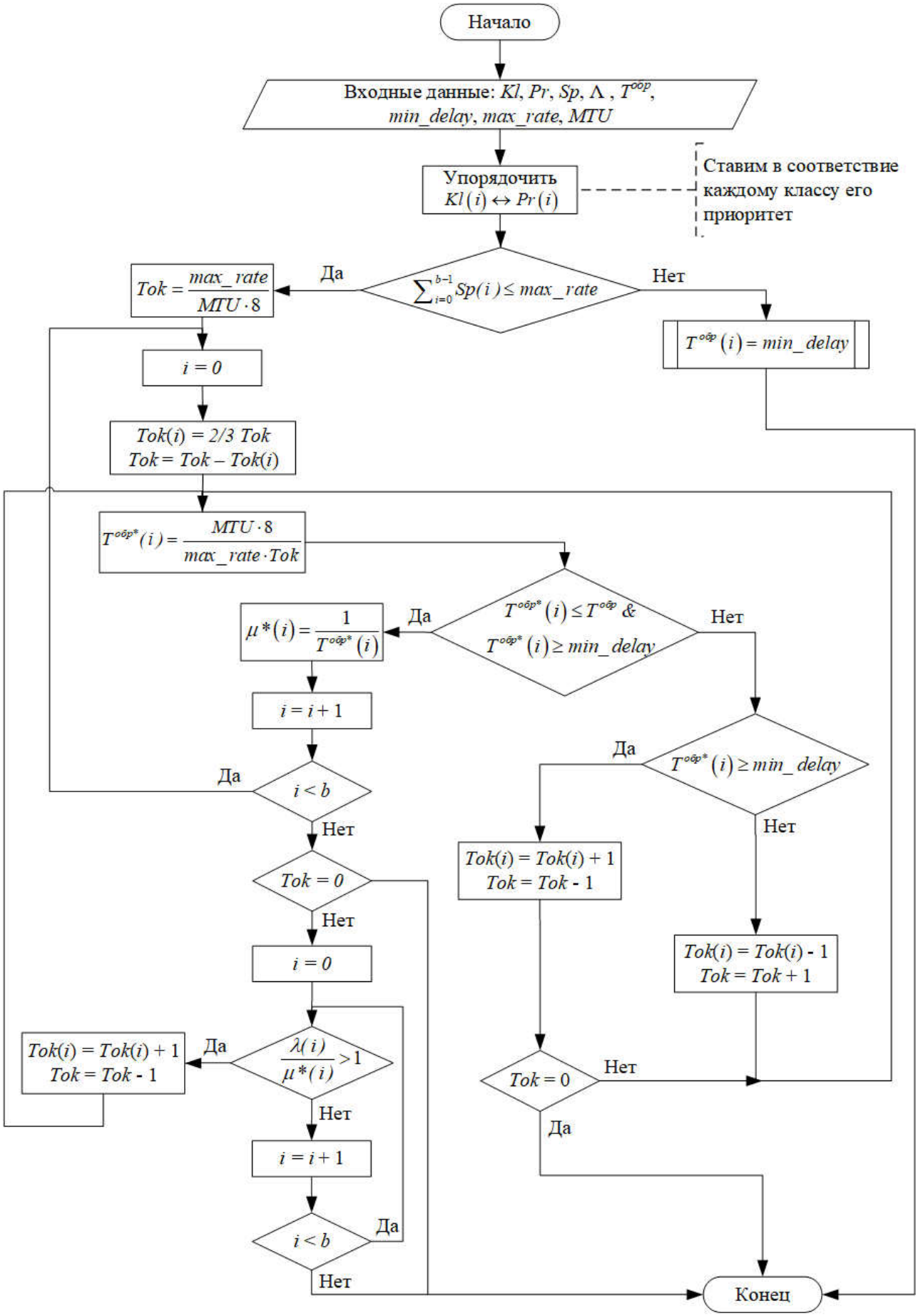

Рисунок 3.7 – Блок-схема алгоритма планирования очередей.

На рисунке 3.8 представлена схема имитационной модели в AnyLogic.

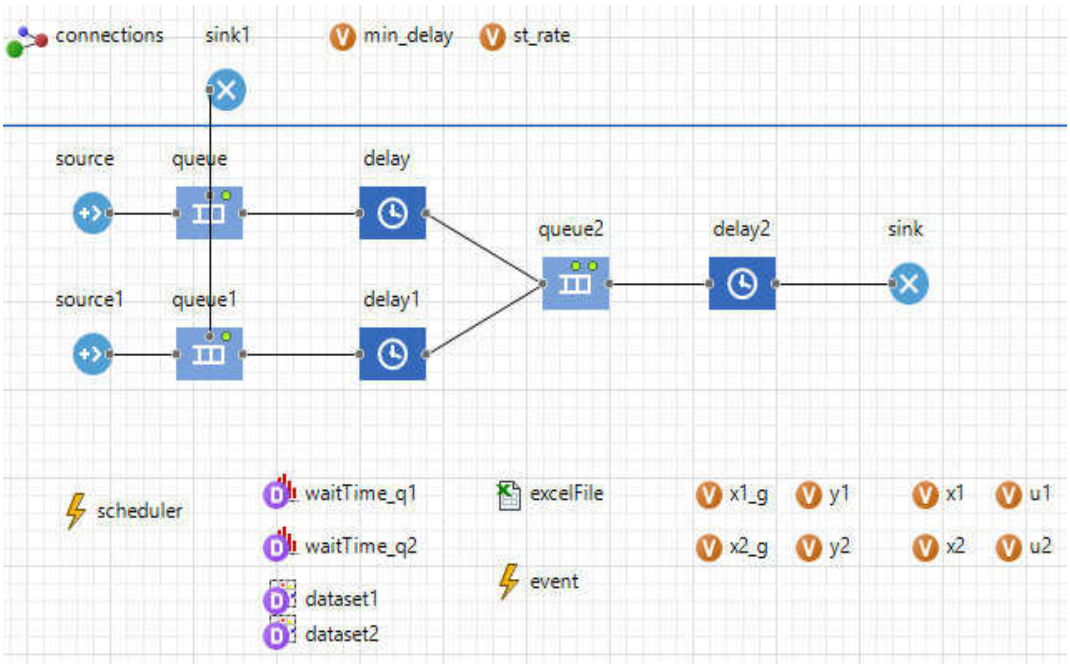

Рисунок 3.8 – Схема имитационной модели.

В рамках тестирования разработанного алгоритма было произведено сравнение задержек передачи пакетов каждой очереди при одинаковых нагрузках на канал очереди классов. Исследование предполагало проведение 5 тестовых съемов задержки передачи пакетов. Режимы тестирования представлены в таблице 3.5. Результаты проведенного эксперимента представлены в виде графиков (рисунки 3.9–3.13).

Таблица 3.5 – Режимы тестирования моделей алгоритмов

Режим тестирования	**Интенсивность очереди 1 (пакет/мс)**	**Интенсивность очереди 2 (пакет/мс)**	**Количество замеров**
Режим 1	0,48	0,857	180
Режим 2	100	0,857	180
Режим 3	0,48	100	180
Режим 4	20,825	62,4	180
Режим 5	100	100	180

На рисунке 3.9 изображен график работы алгоритмов в режиме 1. Из графика видно, что при низкой интенсивности входящего потока пакетов оба алгоритма обеспечивают низкую задержку обработки пакетов. Однако для низкоприоритетной очереди НТВ задержка выше приоритетного класса, это обусловлено последовательностью права передачи пакетов на основании приоритета.

На рисунке 3.10 представлен график работы алгоритмов в режиме 2. График показывает, что НТВ произвел выделение полосы для приоритетного класса и тем самым занял большую часть ресурсов сети, поэтому задержка во втором классе начала расти. Разработанный алгоритм в свою очередь определил, что интенсивность потока низкоприоритетного класса низкая, тем самым можно обеспечивать низкую задержку для обоих классов, оставаясь в директивном времени.

На рисунке 3.11 представлен график работы алгоритмов в режиме 3. На графике видно, что НТВ определил низкую активность высокоприоритетного класса и позволяет низкоприоритетному классу использовать больше полосы. Разработанный алгоритм, в свою очередь, определил необходимое количество полосы для высокоприоритетного класса, чтобы выдавать пакеты с минимальной задержкой, а оставшуюся часть определил для низкоприоритетного.

На рисунке 3.12 приведен график работы алгоритмов в режиме 4. Нагрузка на систему равняется 1, то есть это предел обработки пакетов без потерь. Из графика следует, что НТВ сохраняет заданную полосу пропускания для высокоприоритетного класса, не снижая его задержки, притом задержка низкоприоритетного класса сильно возросла. Разработанный алгоритм производит расчет оптимального распределения полосы в пределах нагрузки на систему таким образом, чтобы обеспечить среднюю минимальную задержку для обоих классов. Но поскольку классы имеют разные приоритеты, задержка для высокоприоритетного класса определяется ниже. К тому же, интенсивность потока заявок высокоприоритетного класса ниже, чем у низкоприоритетного, и согласно работе алгоритма, приоритетный класс получает 100% необходимой

полосы, а низкоприоритетный только ту часть, которая осталась. Исходя из этого мы наблюдаем возрастание задержки для низкоприоритетного класса.

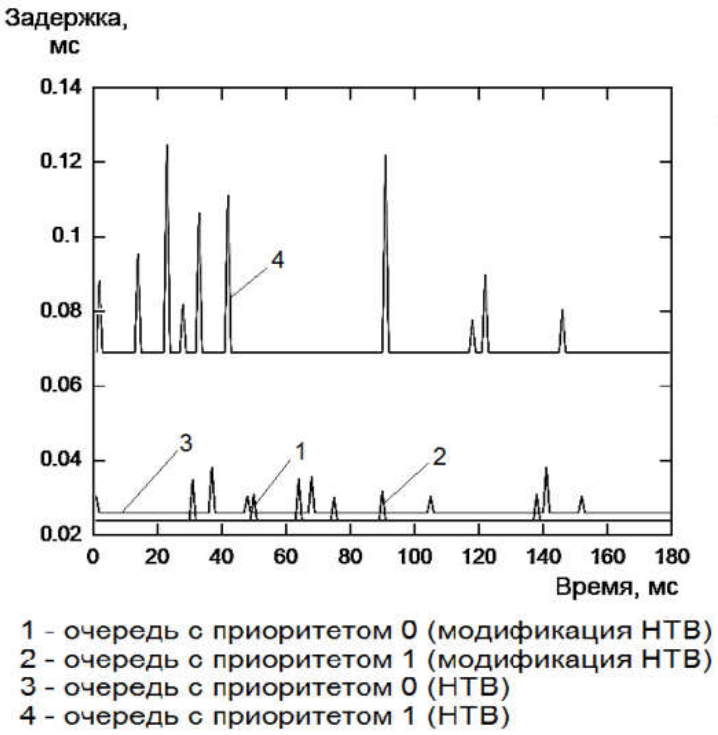

Рисунок 3.9 – Результат тестирования моделей в режиме 1.

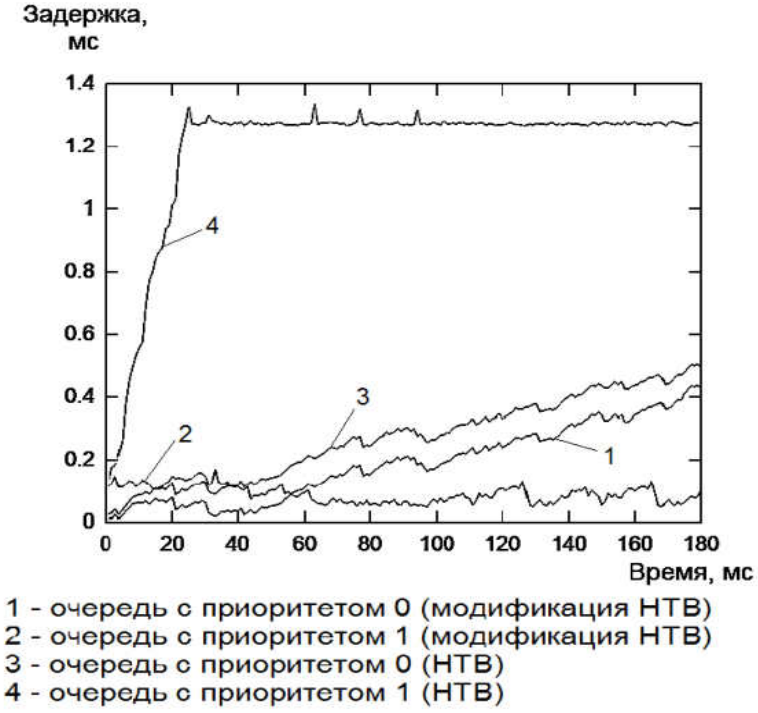

Рисунок 3.10 – Результат тестирования моделей в режиме 2.

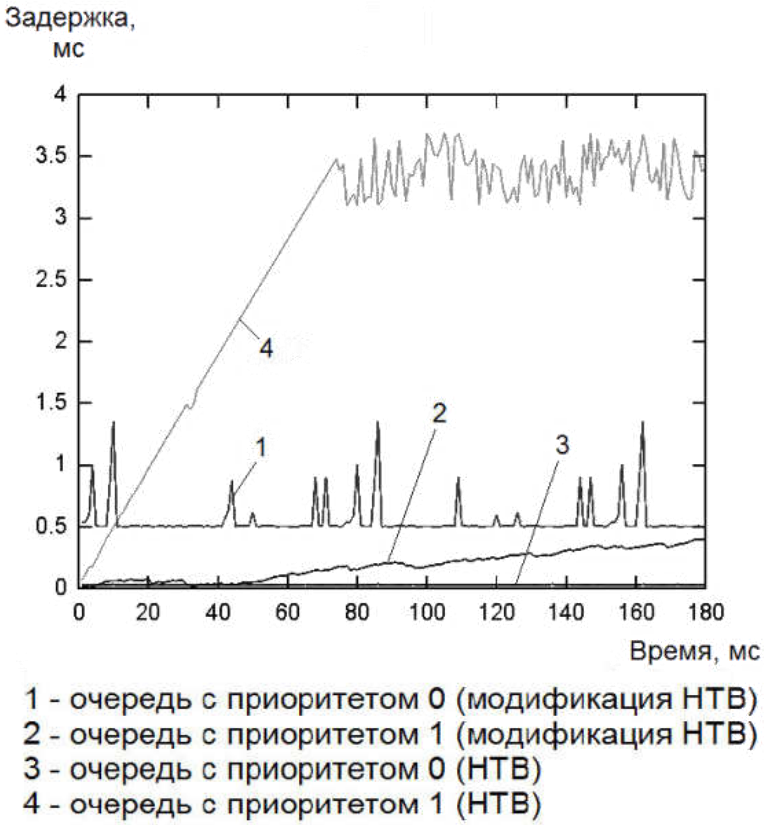

Рисунок 3.11 – Результаты тестирования моделей в режиме 3.

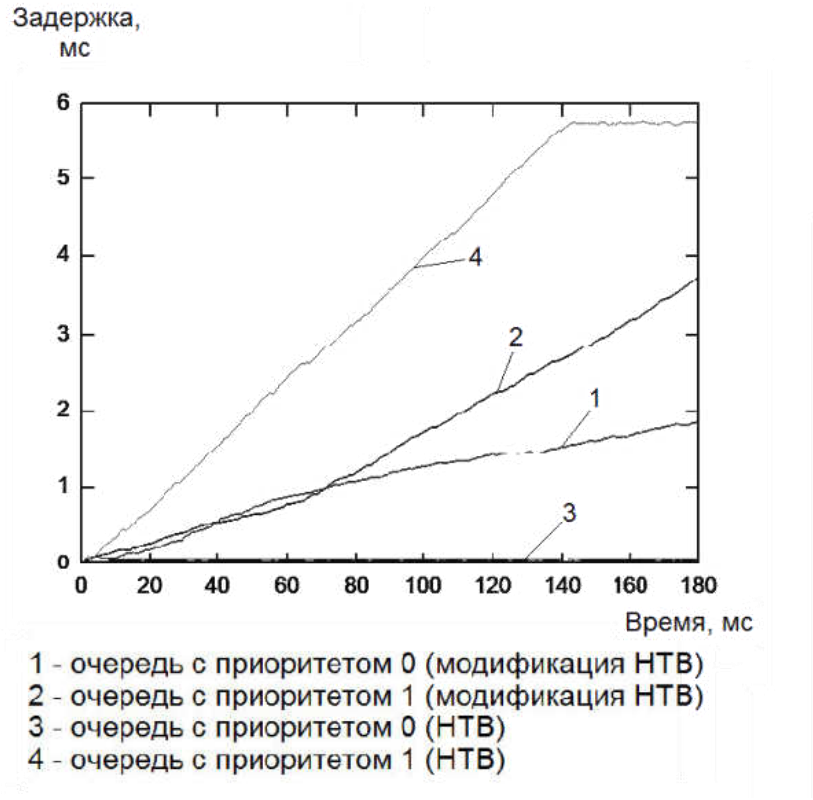

Рисунок 3.12 – Результаты тестирования моделей в режиме 4.

На рисунке 3.13 продемонстрирован график работы алгоритмов в режиме 5. Интенсивность входящего потока пакетов обоих классов трафика сильно превышает значение, возможное для обработки. Следовательно, происходит накопление пакетов в буфере очереди, все пакеты, что выходят за границу буфера, начинают отбрасываться. На графике мы получаем значения задержки пакетов, передача которых была осуществлена. И данная ситуация аналогична режиму 4, когда система способна обработать входящий поток заявок, но т.к. интенсивность обоих классов равна, то для низкоприоритетного класса остается меньше допустимой полосы, и задержка низкоприоритетного класса начинает возрастать. HTB производит обработку аналогичным образом, и распределение полосы пропускания происходит согласно сконфигурированному значению, без возможности расширения полосы для различных классов.

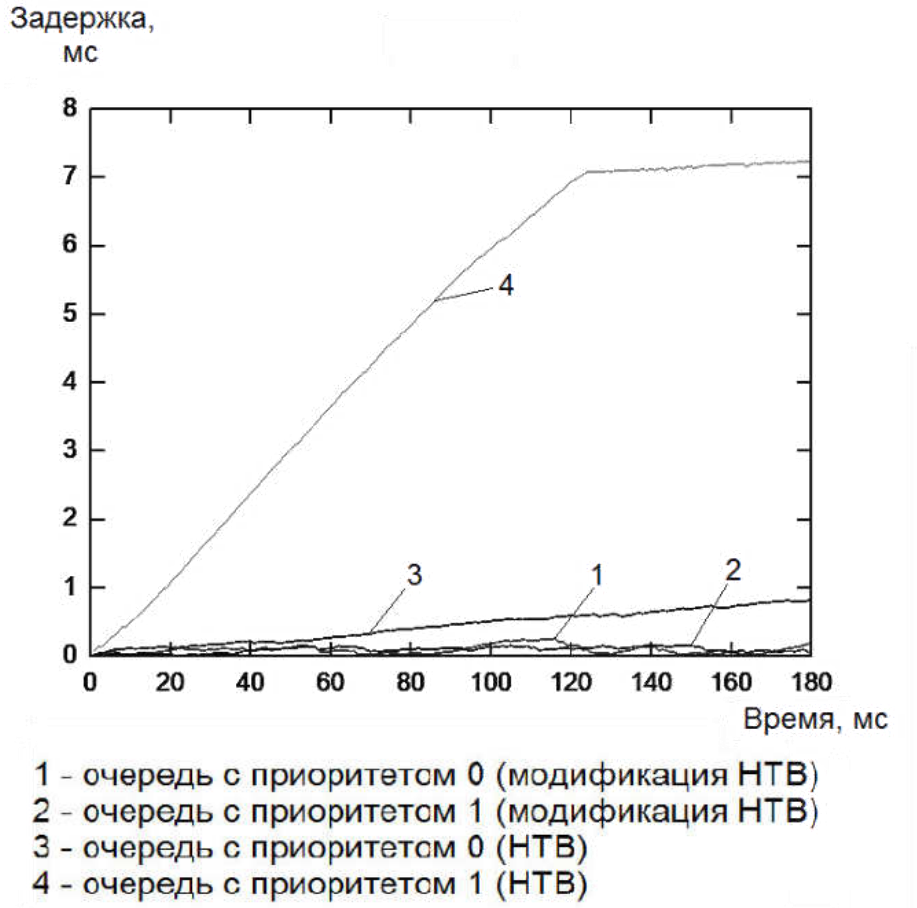

Рисунок 3.13 – Результаты тестирования моделей в режиме 5.

Выводы по результатам тестирования

Результат сравнительного тестирования алгоритмов позволяет сделать вывод, что разработанный алгоритм показывает более низкие суммарные значения задержки для различных классов трафика, тем самым обеспечивая доступность сервисов в обоих классах. Соответственно, предложенный алгоритм планирования очередей передачи данных позволяет оптимизировать использование пропускной способности и обеспечивать минимально возможную задержку для приоритетных классов. Однако, в случае перегрузок в сети могут возникнуть потери пакетов в низкоприоритетных классах.

3.3 Алгоритм поддержки низкоприоритетных сервисов

Тестирование разработанного алгоритма на оборудовании в виде программной реализации (см. раздел 3.1) и при помощи имитационной модели выявило проблему неравномерности потерь пакетов в очередях. Для решения данной проблемы было предложено следующее решение: попробуем снизить потери пакетов в низкоприоритетных очередях за счет заимствования небольшой части полосы пропускания у высокоприоритетных очередей, что поможет повысить производительность сети в целом. Для этого будем делить между листовыми классами (очередями) избытки токенов родительских классов.

В разработанной модели, как и в оригинальном HTB, избыток токенов отдавался листовому классу с наивысшим приоритетом в случае конкуренции за токены. Под избытком токенов понимается разница между гарантированными токенами родительского класса и суммы гарантированных токенов его дочерних классов.

В новой модификации модели избыток токенов родительского класса будем делить между его детьми.

Алгоритм поддержки низкоприоритетных сервисов

Шаг 1. Ввод: Kl, Pr, $T^{\text{обр}}$, max_rate, min_delay, MTU, Sp , λ.

Шаг 2. $Kl_0 = Pr_0, \dots, Kl_{b-1} = Pr_{b-1}$.

Шаг 3. Если $\sum_{i=0}^{b-1} Sp_i \le max\_rate$, то переход к шагу 4, иначе $T_i^* = min\_delay, i = 0, \dots, (b-1)$.

Шаг 4. $Tok = max\_rate/(MTU \times 8)$.

Шаг 5. $Tok_i = Tok \times 2/3$

Шаг 6. $T_i^* = (8 \times MTU)/(max\_rate \times Tok_i)$.

Шаг 7. Если $T_i^* \le T^{\text{обр}}{}_i$, то $\mu_i^* = 1/T_i^*$; $i = i + 1$; если $i < b$, то – к шагу 5, иначе если $Tok = 0$, то конец алгоритма; иначе $i = 0$, переход к шагу 8;

иначе если $T_i^* \ge min\_delay$, то $Tok_i = Tok_i + 1$, $Tok = Tok - 1$,

если $Tok = 0$, то конец алгоритма; иначе переход к шагу 6;

иначе $Tok_i = Tok_i - 1$, $Tok = Tok + 1$, переход к шагу 6;

Шаг 8. Если $\lambda_i > \mu_i^*$ (Проверка соответствия интенсивности обслуживания интенсивности входящего потока пакетов, т.е. имеется возможность увеличить полосу передачи для i-го класса), то $Tok_i = Tok_i + 1$, $Tok = Tok - 1$, переход к шагу 6; иначе $i = i + 1$; если $i < b$ (остались классы), то переход к шагу 8, иначе конец алгоритма.

Тестирование алгоритма

В тестировании принимали участие две модели соответственно для алгоритма планирования очередей передачи данных и алгоритма поддержки низкоприоритетных сервисов. Тестирование проводилось в среде AnyLogic на модели, описанной в разделе 3.2 и скорректированной относительно новых предположений [32], схема уточненной имитационной модели представлена на рисунке 3.14. Каждый тест проводился 2 минуты, и отмечалась задержка для каждого пакета, описание режимов тестирования приведено в таблице 3.6.

Установим для первой очереди максимальное директивное время 2 мс, для второй - 5 мс, для третьей – 10 мс (рисунок 3.15).

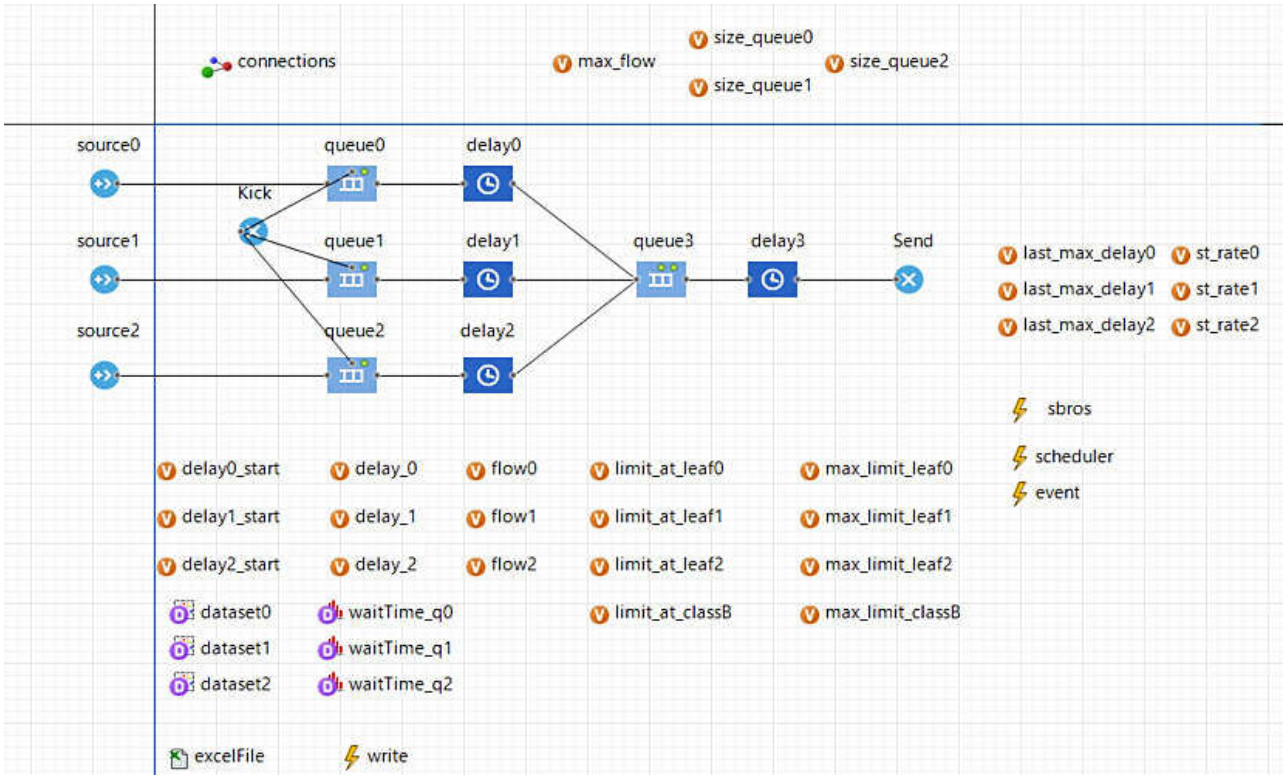

Рисунок 3.14 – Схема обновленной имитационной модели

Таблица 3.6 – Режимы тестирования

Режим тестирования	**Интенсивность очереди 1 (пакет/мс)**	**Интенсивность очереди 2 (пакет/мс)**	**Интенсивность очереди 3 (пакет/мс)**
Режим 1	5	5	50
Режим 2	50	50	5
Режим 3	50	5	50
Режим 4	5	50	50
Режим 5	50	50	50
Режим 6	40	40	40

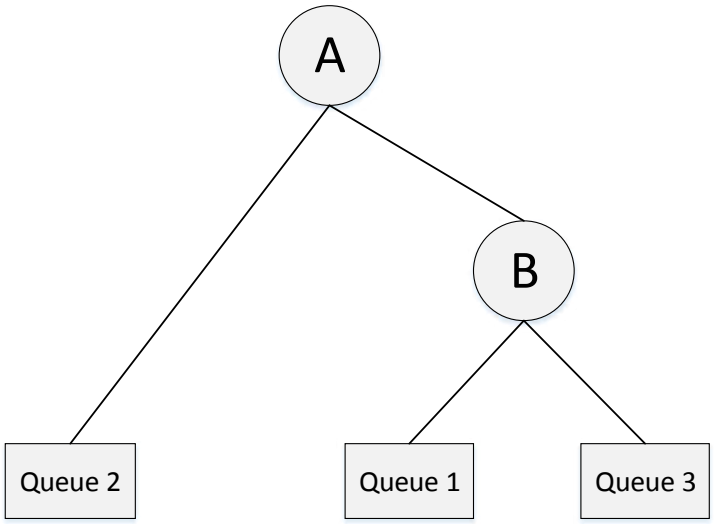

Рисунок 3.15 – Схематичное представление тестируемых моделей.

Результаты эксперимента.

Режим 1

Результаты работы (см. рисунки 3.16 и 3.17) модели показали, что в том случае, если нагружается только один класс, разница в работе отсутствует.

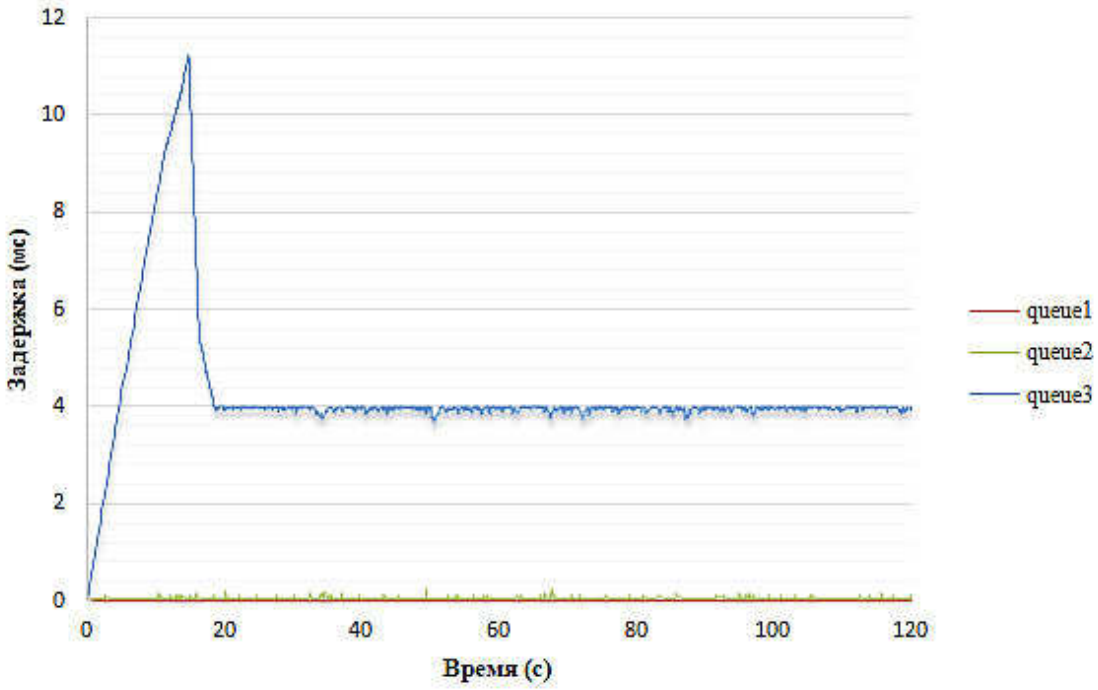

Рисунок 3.16 – Модель 1 режим 1.

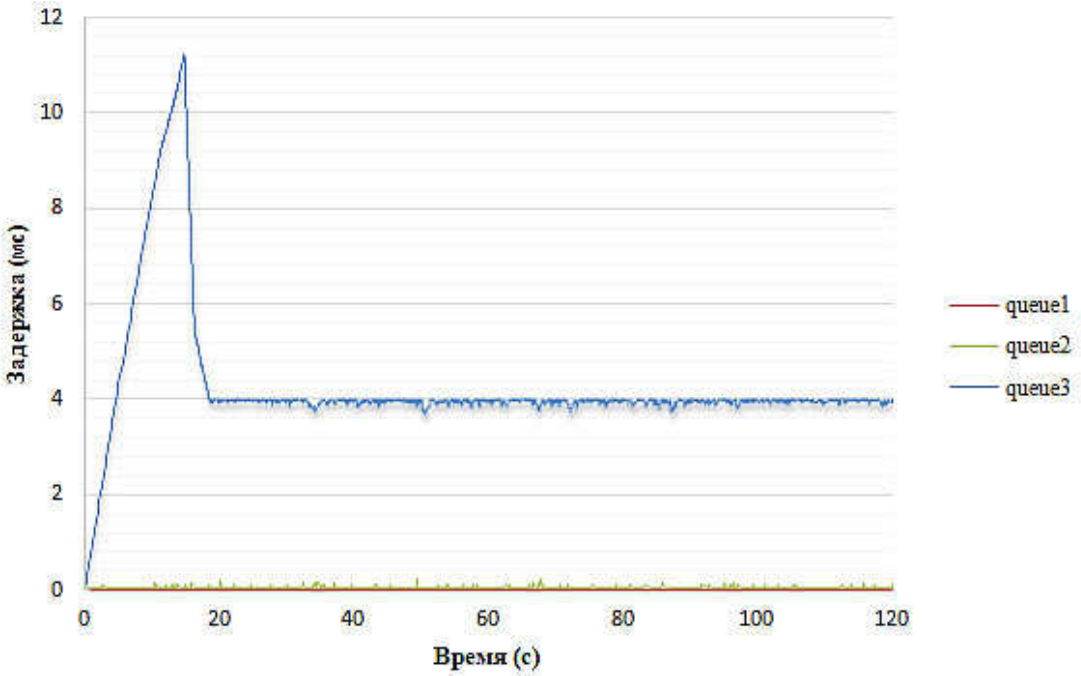

Рисунок 3.17 – Модель 2 режим 1.

Режим 2

В данном режиме тестирования заметно (см. рисунки 3.18 и 3.19), что вторая модель проявляет себя хуже, доля пакетов, пришедших за директивное время, во второй очереди уменьшился в два раза. Это связано с тем, что для удовлетворения задержки второй очереди необходим весь избыток класса А, но так как первая очередь почти всегда использует свою часть избытка А, то для второго класса наблюдаются лишние потери, которые отсутствуют в первой модели (таблица 3.7).

Таблица 3.7 – Режим 2. Доля пакетов, пришедших за мдв.

	Модель 1	**Модель 2**
Queue1	97,6%	99,1%
Queue2	77,5%	39,8%
Queue3	100%	100%

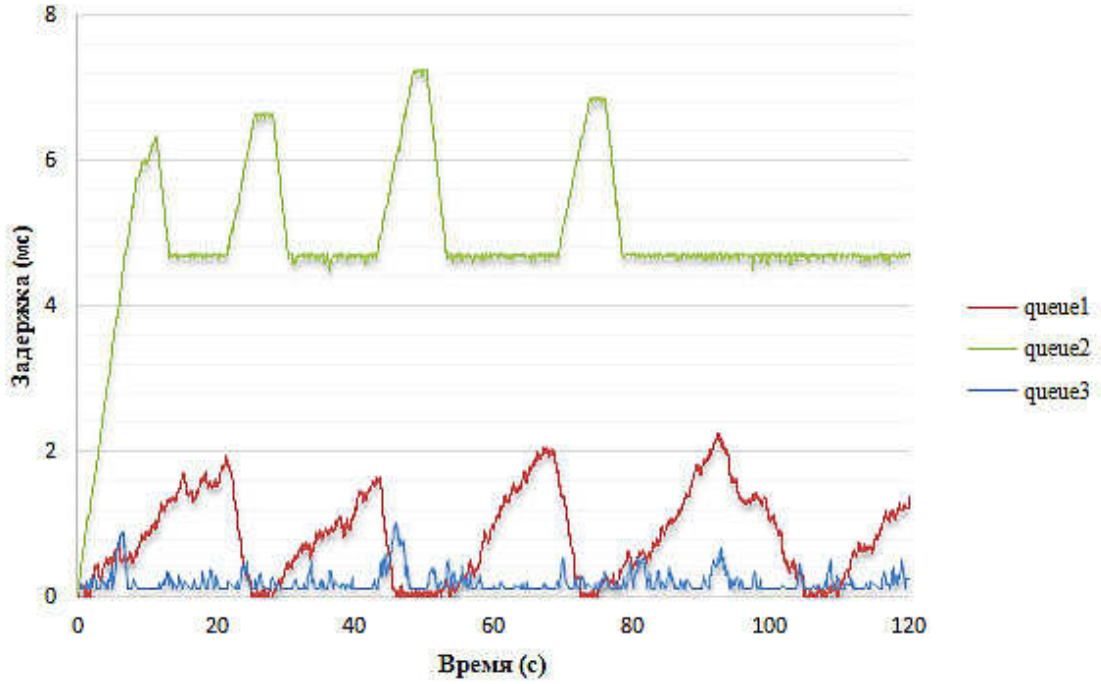

Рисунок 3.18 – Модель 1 режим 2.

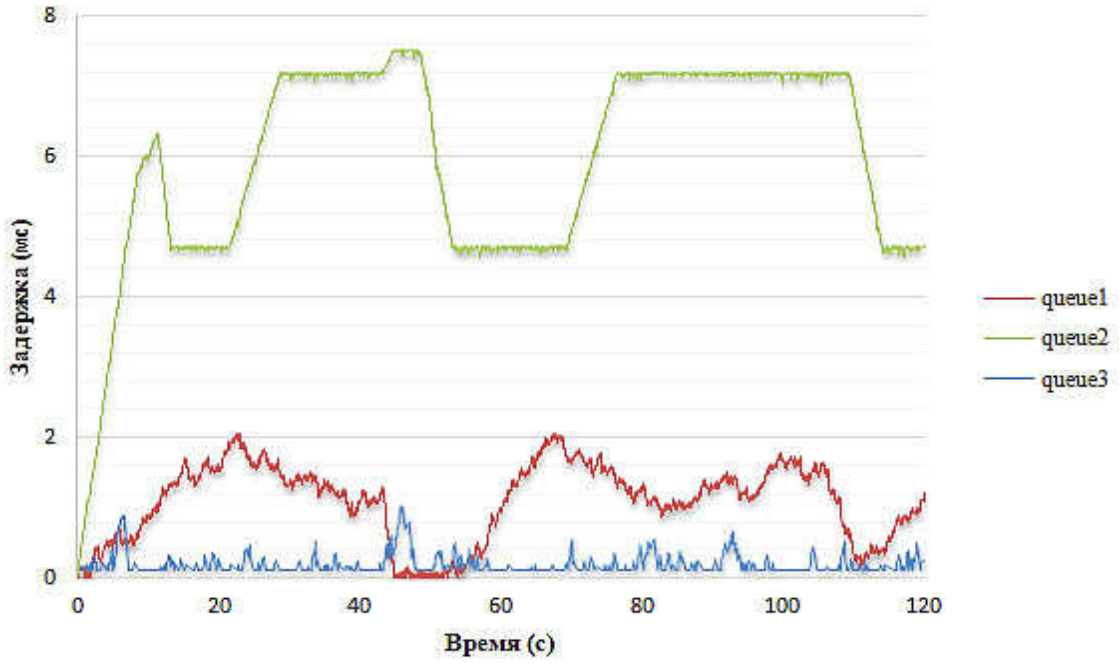

Рисунок 3.19 – Модель 2 режим 2.

Режим 3

В данном режиме тестирования вторая модель проявляет себя лучше, так как для третьей очереди достаточно зарезервированной для неё части избытка токенов родительских классов (таблица 3.5).

Таблица 3.8 – Режим 3. Доля пакетов, пришедших за мдв.

	Модель 1	**Модель 2**
Queue1	94,1%	97,5%
Queue2	100%	100%
Queue3	72,2%	95,1%

Режим 4

В данном режиме тестирования лучшие результаты показывает первая модель. Таблица 3.9 демонстрирует, что вторая модель имеет такие же проблемы, как и во втором режиме тестирования: второй очереди недостаточно зарезервированных токенов.

Таблица 3.9 – Режим 4. Доля пакетов, пришедших за мдв.

	Модель 1	**Модель 2**
Queue1	100%	100%
Queue2	90,4%	41,2%
Queue3	90,5%	94,5%

Режим 5

В пятом режиме тестирования модели показали разные результаты. Видно, что во второй модели сократились потери для третьей очереди по сравнению с первой, однако были потеряны почти все пакеты второй очереди (таблица 3.10).

Таблица 3.10 – Режим 5. Доля пакетов, пришедших за мдв

	Модель 1	**Модель 2**
Queue1	100%	98,2%
Queue2	55,2%	4,8%
Queue3	65,6%	79,2%

Режим 6

При небольшом снижении интенсивности в каждой очереди все равно наблюдаются потери во второй модели во втором классе (таблица 3.11).

Таблица 3.11 – Режим 6. Доля пакетов, пришедших за мдв

	Модель 1	**Модель 2**
Queue1	100%	100%
Queue2	90,5%	56,8%
Queue3	89,6%	96%

Выводы по результатам имитационного моделирования

На основе полученных результатов можно сделать вывод, что деление избытка токенов родительского класса между дочерними классами является менее эффективным с точки зрения производительности сети, чем выделение избытка токенов высокоприоритетному классу в конкурентной борьбе за токены между классами.

Алгоритм поддержки низкоприоритетных классов плохо проявляет себя, когда классу необходимо количество токенов, намного превышающих количество гарантированных токенов для класса. Однако в том случае, когда количество необходимых токенов классу незначительно превышает количество гарантированных токенов или класс имеет много родителей, у которых зарезервированы для него токены, данный алгоритм показывает более стабильную задержку.

3.4 Экспериментальное исследование

Использование и внедрение разработанного алгоритма в реальных условиях потребовало переконфигурации операционной системы на уровне ядра. В код утилиты алгоритма HTB в операционной системе Linux при помощи команды insmod был добавлен модуль, отвечающий за распределение задержек

между классами в зависимости от приоритета (Приложение В). Данной командой подменяется оригинальный модуль HTB на разработанный.

Код модуля ядра был написан на языке С и скомпилирован при помощи компилятора gcc+. Классификация и маркирование исследуемых областей сети проводилась при помощи утилиты tc.

Экспериментальное исследование эффективности разработанного алгоритма ПКТ в сравнении с типовым HTB состояло из двух экспериментов [99]:

Эксперимент 1. Тестирование алгоритма HTB в различных режимах нагрузки.

Эксперимент 2. Тестирование модифицированного алгоритма.

Исследование проводилось в равных условиях последовательно ввиду отсутствия возможности запускать в операционной системе обе версии алгоритма управления потоком одновременно. Программа эксперимента и состав экспериментальной установки описаны в разделе 3.1, дополнительно на клиентских устройствах контролировалась задержка: Client-1 с IP-адресом 172.20.0.10 обладает наивысшим приоритетом и обеспечивается гарантированной задержкой в 0,2 мс, для следующего по приоритету Client-2 задана задержка в 0,5 мс, но только в тех случаях, когда соблюдаются условия, заданные Client-1, Client-3 является устройством, обладающим самым низким приоритетом и получает задержку, исходя из параметров пропускной способности сети и загруженности других клиентов. В тестируемой среде была установлена максимальная пропускная способность 10 Мбит/с.

Для запуска модифицированного алгоритма необходимо вызвать измененный модуль из ядра. Скрипт маршрутизатора «RouterMain» для вызова модуля sch_htb представлен в таблице 3.12. Данный скрипт использует утилиты, дающие возможность классифицировать приоритетные узлы, входящие в контролируемую сеть.

Таблица 3.12 – Вызов модуля sch_htb

```
#!/bin/bash
make
insmod ./sch_htb.ko
tc qdisc add dev ens38 root handle 1: htb
tc class add dev ens38 parent 1: classid 1:10 htb rate 10Mbit
tc filter add dev ens38 parent 1: protocol ip prio 1 u32 match ip dst 172.20.0.2/24
classid 1:10
tc filter add dev ens38 parent 1: protocol ip prio 2 u32 match ip dst 172.20.0.10/24
classid 1:10
```

RouterMain представляет собой программный маршрутизатор, в который добавляется алгоритм приоритизации, аналогично настроен и второй маршрутизатор, обеспечивающий функционирование отдельных сегментов сети.

Сравним результаты работы разработанного модуля ядра со стандартным алгоритмом HTB, установленным в Linux.

С модифицированным HTB задержка пакетов первой очереди является гарантированной, задержки второй и третьей очереди растут, поскольку одновременно повышается интенсивность потока. В отличии от алгоритма HTB у наиболее приоритетной очереди наблюдается гарантированная задержка, однако повышается значение задержек других очередей. При последнем повышении интенсивности потока в обоих случаях начинаются потери пакетов, однако модифицированный алгоритм решает эту проблему путем разгрузки наименее приоритетных очередей (рисунок 3.20).

Данный способ позволяет сохранить доступность сети даже в условиях высокой интенсивности передачи пакетов.

Рисунок 3.20 – Тестирование 1 режима при модифицированном НТВ.

Во втором режиме при стандартном НТВ значение задержки равномерно распределяется между второй и третьей очередью на любом шаге повышения интенсивности, а задержка первой очереди возрастает с повышением интенсивности потока. Благодаря модифицированному алгоритму НТВ задержка пакетов первого класса не отклоняется от заданного значения, задержки второй и третьей очереди формируются за счет обеспечения гарантированной задержки первой очереди. На графике (рисунок 3.21) видно, что происходит их равномерное распределение, и потерь пакетов не наблюдается.

При третьем режиме тестирования модифицированного алгоритма НТВ обеспечивается гарантированная задержка первой очереди и возрастают значения второй и третьей очереди за счёт повышения потока интенсивности второй (рисунок 3.22).

Четвертый режим наглядно показывает, каким образом распределяется задержка в сети при первоначальном значении интенсивности. Приоритетные

очереди получают гарантированную задержку, за счет наименее приоритетных, в данном случае третьей очереди (рисунок 3.23).

Рисунок 3.21 – Тестирование 2 режима при модифицированном НТВ.

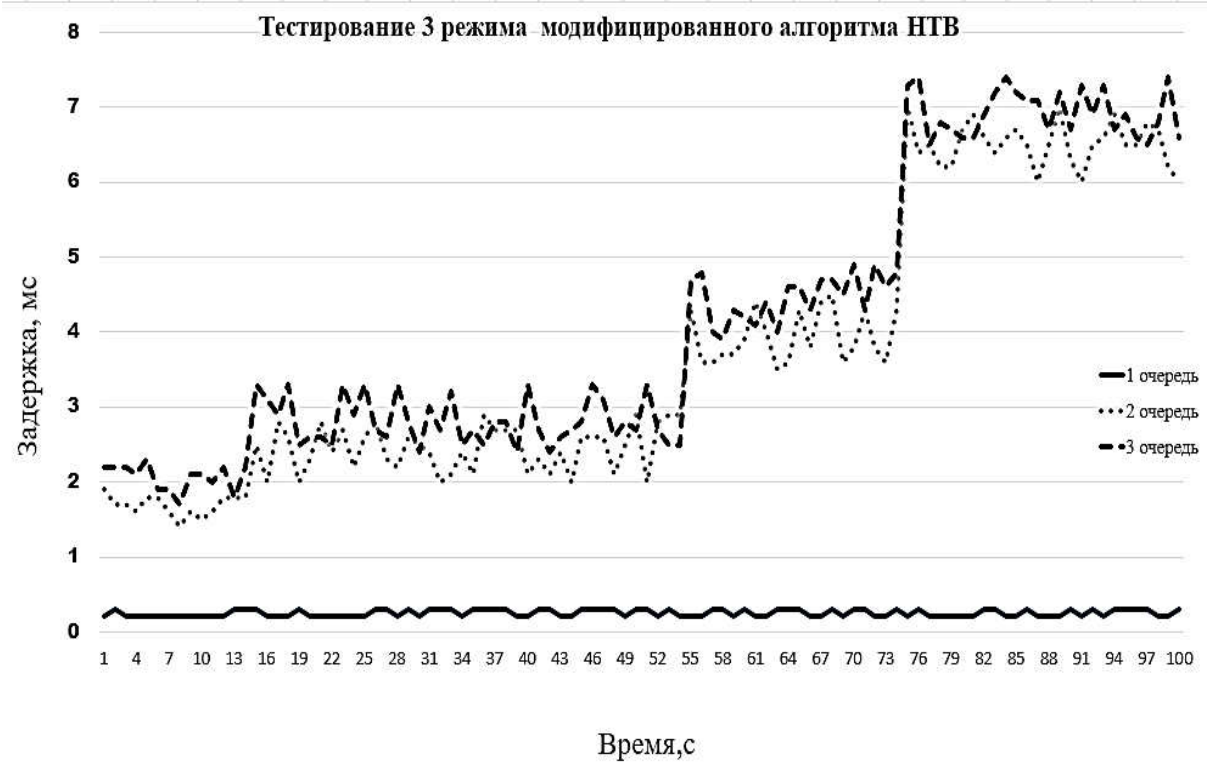

Рисунок 3.22 – Тестирование 3 режима при модифицированном НТВ.

Рисунок 3.23 – Тестирование 4 режима при модифицированном НТВ.

В пятом режиме тестирования при стандартном НТВ значение задержки равномерно распределяется между тремя очередями на любом шаге повышения интенсивности, но на последнем шаге из-за того, что алгоритм пытается следовать приоритетам, происходит переполнение очереди. Также значения задержки для любых из очередей являются высокими. При модифицированном алгоритме НТВ задержка пакетов первой очереди не отклоняется от заданного значения, задержки второй и третьей очереди возрастают вследствие общего роста интенсивности и обеспечения гарантированной задержки первой очереди. На втором этапе повышения интенсивности наблюдается потеря пакетов, но общая передача трафика ведется путем отказа от фиксированной задержки первой очереди. Поскольку условие переполнения буфера срабатывает, задержки всех очередей повышаются в зависимости от пропускной способности (рисунок 3.24).

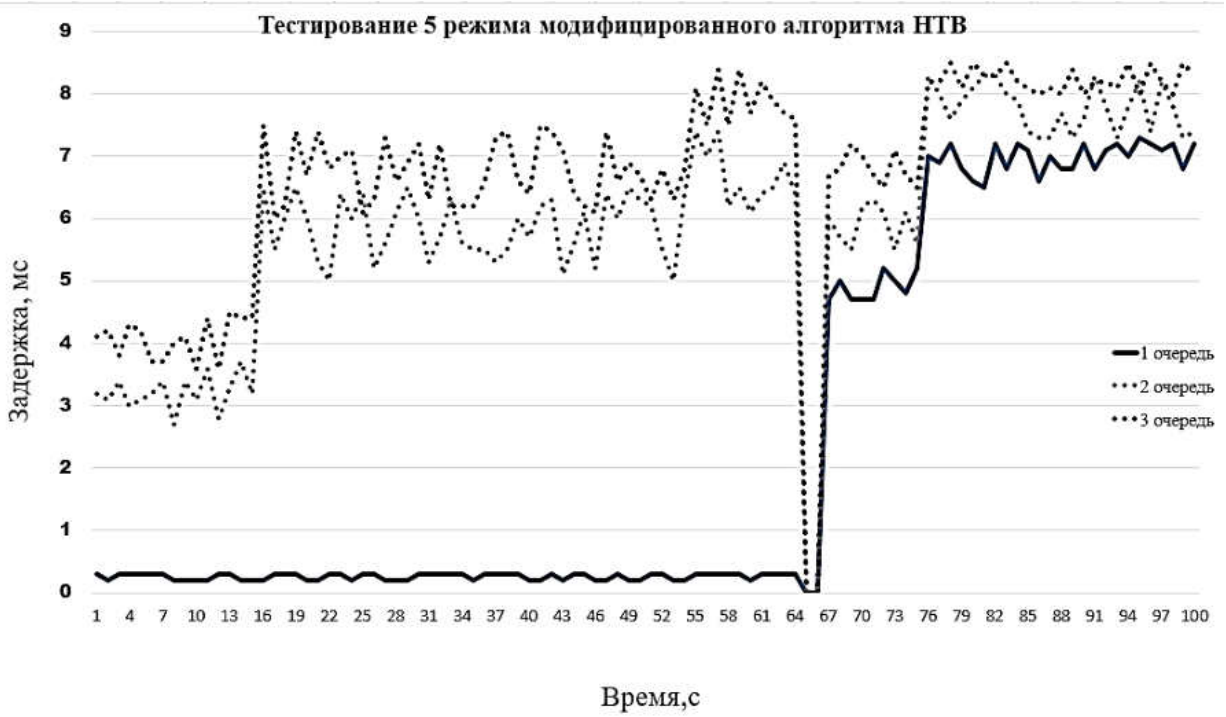

Рисунок 3.24 – Тестирование 5 режима при модифицированном НТВ.

Выводы по результатам экспериментального исследования

В сравнении с алгоритмом НТВ в типовой конфигурации разработанный алгоритм показывает более низкие средние значения задержки для различных классов трафика. Обеспечивается гарантированная задержка для первой и второй очереди. Данный алгоритм можно модифицировать в зависимости от топологии сети и предпочтений по приоритетам трафика.

3.5 Внедрение результатов исследования

Результаты описанного в данной главе исследования внедрены в ООО «Контактон» а также в учебный процесс кафедры информатики и защиты информации (ИЗИ) ВлГУ.

Алгоритм поддержки низкоприоритетных сервисов используется в телекоммуникационной сети ООО «Контактон». Использование данного алгоритма позволило повысить эффективность обслуживания трафика за счет увеличения числа пакетов низкоприоритетных сетевых сервисов очереди на

передачу в среднем на 15%. В результате был обеспечен принцип справедливости в отношении всех сервисов, работающих в телекоммуникационной сети.

На кафедре ИЗИ ВлГУ используются следующие результаты диссертационного исследования:

1. Алгоритмы планирования очередей передачи данных и поддержки низкоприоритетных сервисов.
2. Имитационные модели алгоритмов в AnyLogic.
3. Экспериментальный виртуальный стенд для исследования алгоритмов приоритизации и управления потоком.

Данные результаты используются в теоретической части курса «Сети и системы передачи информации» для бакалавров направления 10.03.01 «Информационная безопасность» и специалистов специальности 10.05.04 «Информационно-аналитические системы безопасности», а также в лабораторном практикуме по курсу, научно-исследовательской работе студентов и дипломном проектировании.

Копии актов внедрения результатов диссертационного исследования представлены в приложении Д.

Научно-технические результаты диссертационного исследования, описанные в данной главе, включены в итоговый отчет по гранту Российского фонда фундаментальных исследований, проект № 18-07-01109 (А) «Алгоритмы и протоколы оценки и контроля доступности в крупномасштабных телекоммуникационных сетях» (2018-2020 гг.).

Выводы к главе 3

1. Разработан алгоритм планирования очередей передачи данных на основе модификации известного подхода «иерархическое ведро маркеров» (HTB). Алгоритм позволяет обеспечивать минимально возможную задержку для приоритетных классов поддерживаемых сервисов, оптимизируя использование пропускной способности. Экспериментально выявлены факторы, влияющие на задержку передачи пакетов в HTB и позволяющих выполнять оптимальное планирование: контроль интенсивности входящего потока пакетов, динамическое изменение пропускной способности канала относительно входящей интенсивности, оптимизация предоставления полосы для классов трафика относительно входящей интенсивности.

2. Разработан алгоритм поддержки низкоприоритетных сервисов в условиях сильного доминирования высокоприоритетных сервисов, основанный на перераспределении токенов управления потоком, что позволяет обеспечить принцип справедливости в отношении всех сервисов, работающих в КПТС.

3. Для проверки эффективности разработанных алгоритмов было произведено имитационное моделирование их работы в среде AnyLogic в сравнении с работой алгоритма HTB в типовой конфигурации. Результат сравнительного тестирования алгоритмов позволяет сделать вывод, что разработанные алгоритмы показывают более низкие суммарные значения задержки для различных классов трафика, тем самым обеспечивая высокую доступность сервисов. Соответственно, предложенные алгоритмы планирования очередей передачи данных и поддержки низкоприоритетных сервисов в условиях сильного доминирования высокоприоритетных сервисов позволяет оптимизировать использование пропускной способности и обеспечивать минимально возможную задержку для приоритетных классов трафика.

4. Разработано программное обеспечение в виде модуля ядра операционной системы Linux, реализующее алгоритм планирования очередей передачи данных в КПТС. Произведено исследование разработанного модуля на

экспериментальном стенде. Результаты экспериментального исследования подтвердили результаты, полученные при моделировании.

ЗАКЛЮЧЕНИЕ

В результате выполнения данного диссертационного исследования можно сделать следующие теоретические и практические выводы:

– Проанализированы существующие решения задачи повышения доступности корпоративной программно-определяемой телекоммуникационной сети и ее компонентов, а также методики оценки показателей доступности. Несмотря на растущие требования прикладного уровня по уменьшению отклика, большинство авторов, исследующих доступность телекоммуникационных сетей, не учитывают данный критерий и по-прежнему оценивают этот показатель через коэффициент готовности сети.

– Ограничения, препятствующие достижению высокой доступности в КПТС, связаны с пределами по вычислительной мощности контроллеров и устройств, организующих уровень передачи, а также емкости памяти и буфера; задержкой между типами устройств контроллер-контроллер и контроллер-коммутатор; ростом трафика в канале связи.

– Сформулирована задача оптимизации доступности при существующих ограничениях по вычислительной мощности контроллеров, ёмкости памяти и буфера для управления множеством виртуализированных узлов и множеством линий связи. Обоснована методика оценки показателя доступности сети и каналов связи. Для повышения доступности сети предлагается два подхода: оптимизировать топологию КПТС для достижения максимума ИПД; повысить показатель доступности каналов связи за счет применения нового алгоритма управления потоком.

– Создан программно-аппаратный стенд в среде Mininet для проведения экспериментов, позволяющий формировать произвольные топологии SDN, осуществлять маршрутизацию потоков трафика на базе контроллера ONOS, а также производить расчет показателей доступности. Эксперименты позволили выявить существенные факторы воздействия на топологию в программно-определяемых сетях с высокой доступностью. Выявлена зависимость влияния

связей между устройствами и количеством устройств в сетевой топологии на интегральный показатель доступности сети. Исследование показало, что эффективным методом для улучшения этого показателя в сети с реактивным режимом обработки входящих потоков является увеличение количества связей между коммутирующими устройствами.

– Разработан алгоритм оптимизации топологии КПТС, основанный на последовательной реконфигурации топологии сетевых средств коммутации и маршрутизации по критерию максимума интегрального показателя доступности, что позволяет подстраивать топологию КПТС под изменяющиеся внешние условия и решаемую задачу. Экспериментальные исследования показали: оптимальная топология находилась каждый раз при многократном изменении начальной топологии; интегральный показатель доступности зависел в основном от текущей нагрузки сети и варьировался в диапазоне от 0,6 до 0,8, при этом выигрыш по сравнению с исходной топологией достигал 15%, а в ряде случаев до 22%. Наибольший эффект разработанный алгоритм показывает, когда при решении задачи задействовано много узлов.

– Разработан алгоритм планирования очередей передачи данных на основе модификации известного подхода «иерархическое ведро маркеров» (HTB). Алгоритм позволяет обеспечивать минимально возможную задержку для приоритетных классов поддерживаемых сервисов, оптимизируя использование пропускной способности. Экспериментально выявлены факторы, влияющие на задержку передачи пакетов в HTB и позволяющих выполнять оптимальное планирование: контроль интенсивности входящего потока пакетов, динамическое изменение пропускной способности канала относительно входящей интенсивности, оптимизация предоставления полосы для классов трафика относительно входящей интенсивности.

– Разработан алгоритм поддержки низкоприоритетных сервисов в условиях сильного доминирования высокоприоритетных сервисов, основанный на перераспределении токенов управления потоком, что позволяет обеспечить принцип справедливости в отношении всех сервисов, работающих в КПТС.

– Для проверки эффективности разработанных алгоритмов было произведено имитационное моделирование их работы в среде AnyLogic в сравнении с работой алгоритма HTB в типовой конфигурации. Результат сравнительного тестирования алгоритмов позволяет сделать вывод, что разработанные алгоритмы показывают более низкие суммарные значения задержки для различных классов трафика, тем самым обеспечивая высокую доступность сервисов. Соответственно, предложенные алгоритмы планирования очередей передачи данных и поддержки низкоприоритетных сервисов в условиях сильного доминирования высокоприоритетных сервисов позволяет оптимизировать использование пропускной способности и обеспечивать минимально возможную задержку для приоритетных классов трафика.

– Разработано программное обеспечение в виде модуля ядра операционной системы Linux, реализующее алгоритм планирования очередей передачи данных в КПТС. Произведено исследование разработанного модуля на экспериментальном стенде. Результаты экспериментального исследования подтвердили результаты, полученные при моделировании.

СПИСОК ИСПОЛЬЗУЕМЫХ ТЕРМИНОВ И СОКРАЩЕНИЙ

– HTB (англ. Hierarchical Token Bucket) – алгоритм «иерархическое ведро маркеров»
– MTU (англ. Maximum Transmission Unit) – максимальная единица передачи
– NFV (англ. Network Functions Virtualization) – виртуализация сетевых функций
– QoS (англ. Quality of Service) – качество обслуживания
– SDN (англ. Software Defined Network) – программно-определяемая сеть
– SD-WAN (англ. Software Defined Networking in a Wide Area Network) – программно-определяемая глобальная (распределенная) сеть
– SLA (англ. Service Level Agreement) – соглашение об уровне обслуживания
– TCAM (англ. Ternary Content-Addressable Memory – троичная ассоциативная память
– ИПД – интегральный показатель доступности
– КПТС – корпоративная программно-определяемая телекоммуникационная сеть
– КС – канал связи
– КТС – корпоративная телекоммуникационная сеть
– ЛС – линия связи
– мдв – максимальное директивное время
– ПКТ – приоритизация и контроль трафика

СПИСОК ЛИТЕРАТУРЫ

1. Абросимов, Л.И., Руденкова М.А., Хаю Х. Методика определения гарантированной доставки трафика в корпоративных беспроводных локальных вычислительных сетях // Вестник Воронежского государственного технического университета. – 2020. – Т. 16. – №. 5.

2. Акатов, Д.В., Юрочкин, А.Г. Характеристики основных средств для анализа и оптимизации корпоративных сетей // Моделирование, оптимизация и информационные технологии. – 2015. – №. 2. – С. 8-8.

3. Александров, Г. Д. Проектирование защищенной корпоративной сети передачи данных //T-Comm-Телекоммуникации и Транспорт. – 2018. – Т. 12. – №. 3. – С. 39-45

4. Бахарева, Н.Ф., Коннов, А.Л., Тарасов, В.Н., Ушаков, Ю.А. Обеспечение качества обслуживания в программно-конфигурируемых сетях // Инфокоммуникационные технологии. – 2012. – Т. 10. – №. 4. – С. 30-35.

5. Галич, С.В. Исследование и анализ задержки обработки трафика управления в программно-конфигурируемых сетях: дисс. … канд. техн. наук: 05.12.13. – 2018. - 169 с.

6. Галлагер, Р., Бертсекас, Д. Сети передачи данных //Д. Бертсекас. – М., 1989. – 544 с.

7. ГОСТ 33707-2016 (ISO/IEC 2382:2015) Информационные технологии. Словарь. - М.: Стандартинформ, 2016.

8. ГОСТ Р ИСО 7498-2-99 (ISO 7498-2:1989) Взаимосвязь открытых систем. Базовая эталонная модель. Часть 2. Архитектура защиты информации. - М.: ИПК Издательство стандартов, 1999.

9. ГОСТ Р ИСО/МЭК 13335-1-2006 (ISO/IEC 13335-1:2004) Информационная технология. Методы и средства обеспечения безопасности. Часть 1. Концепция и модели менеджмента безопасности информационных и телекоммуникационных технологий. - М.: Стандартинформ, 2007.

10. Дьяченко, Н.В., Аверкиев, А.А. Доступность и аварийное восстановление системы // Modern Science. – 2020. – №. 12-3. – С. 244-246.

11. Дякив, Д. 3 обязательных требования к услуге связи [Электронный ресурс] //Журнал сетевых решений LAN. – 2014. – №. 9. – С. 19-22. - Режим доступа: https://www.osp.ru/lan/2014/09/13042700.

12. Егоров, В.Б. Некоторые вопросы практической реализации концепции SDN //Системы и средства информатики. – 2016. – Т. 26. – №. 1. – С. 109-120.

13. Корячко, В. П., Перепелкин Д. А. Корпоративные сети: технологии, протоколы, алгоритмы: монография. – 2011.

14. Корячко, В.П., Перепелкин, Д.А. Анализ и проектирование маршрутов передачи данных в корпоративных сетях. Монография. – 2012.

15. Кузнецова, А.П., Монахов Ю.М. Постановка задачи адаптивного управления очередями для повышения доступности узлов в сетях TCP/IP с частыми потерями кадров // Перспективные технологии в средствах передачи информации-ПТСПИ-2019. – 2019. – С. 75-78.

16. Кузьмин, В.В. Классификация и идентификация трафика в мультисервисной сети оператора связи // Современные проблемы науки и образования. – 2014. – № 5.

17. Лемешко, А.В., Вавенко, Т.В. Разработка и исследование потоковой модели адаптивной маршрутизации в программно-конфигурируемых сетях с балансировкой нагрузки // Доклады Томского государственного университета систем управления и радиоэлектроники. – 2013. – №. 3 (29).

18. Малахов, С.В., Тарасов, В.Н., Карташевский, И.В. Теоретическое и экспериментальное исследование задержки в программно-конфигурируемых сетях // Инфокоммуникационные технологии. – 2015. – Т. 13. – №. 4. – С. 409-413.

19. Масленников, А.Г. Разработка метода обработки трафика в очередях маршрутизаторов мультисервисной сети на основе нечёткой логики: дисс. … канд. техн. наук: 05.12.13. – 2015. - 124 с.

20. Матвеев, С.Н., Матвеева, А.П. Способ и устройство измерения времени задержки на двустороннее распространение трафика данных в телекоммуникационной сети //Проблемы передачи информации в инфокоммуникационных системах. Сборник докладов и тезисов XI Всероссийской научно-практической конференции. – 2021. – С. 17-21.

21. Матвеева, А.П. Об оптимизации топологии корпоративных программно-определяемых сетей по критерию доступности //Проблемы передачи информации в инфокоммуникационных системах. Сборник докладов и тезисов XI Всероссийской научно-практической конференции. – 2021. – С. 21-26.

22. Матвеева, А.П. Постановка задачи оптимизации доступности в корпоративных программно-определяемых телекоммуникационных сетях // Моделирование, оптимизация и информационные технологии. – 2021. – Т. 9. – №. 2. – С. 26-27.

23. Митрохин, В.Е., Рингенблюм, П.Г. Оценка влияния угроз информационной безопасности на доступность телекоммуникационной сети // Доклады Томского государственного университета систем управления и радиоэлектроники. – 2014. – №. 2 (32).

24. Монахов, М.Ю., Монахов, Ю.М. Проблема доступности телекоммуникационных систем // Перспективные технологии в средствах передачи информации-ПТСПИ-2019. – 2019. – С. 20-27.

25. Монахов, М.Ю., Монахов, Ю.М., Полянский, Д.А., Семенова, И.И. Модели обеспечения достоверности и доступности информации в информационно-телекоммуникационных системах: монография / Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир: Изд-во ВлГУ, 2015. – 208 с.

26. Монахов, Ю.М., Власова, А.М. Методика расчета нормированного критерия доступности телекоммуникационной сети // Динамика сложных систем-XXI век. – 2015. – Т. 9. – №. 3. – С. 73-77.

27. Монахов, Ю.М., Кузнецова, А.П., Исмаилова, М.Р. Алгоритм планирования очередей передачи трафика в телекоммуникационных сетях для

управления доступностью // Вестник Санкт-Петербургского университета. Прикладная математика. Информатика. Процессы управления. – 2019. – №. 3.

28. Монахов, Ю.М., Кузнецова, А.П., Леткова, Н.С., Шобин С.В. О возможности применения импульсной передаточной функции для моделирования поведения сетевого устройства // Современные наукоемкие технологии. – 2019. – №. 2. – С. 110-113.

29. Монахов, Ю.М., Кузнецова, А.П., Пестов, А.В. Приоритизация трафика для обеспечения доступности в телекоммуникационных сетях предприятий текстильной промышленности //Известия высших учебных заведений. Технология текстильной промышленности. – 2018. – №. 5. – С. 159-163.

30. Монахова, М.М. Модели и алгоритмы контроля инцидентов информационной безопасности в корпоративной телекоммуникационной сети: дис. ... канд. техн. наук: 05.12.13. – ВлГУ, 2016. – 137 с.

31. Мухизи, С., Парамонов, А. И. Метод классификации и приоритизации трафика в программно-конфигурируемых сетях //Труды учебных заведений связи. – 2019. – Т. 5. – №. 1. – С. 64-70.

32. Ниязов, Р.Х., Монахов, Ю.М., Бедняцкий, И.С., Балашов, В.И., Кузнецова. А.П. Доработка имитационной модели алгоритмов приоритизации в сетях TCP/IP // Девятая всероссийская научно-практическая конференция по имитационному моделированию и его применению в науке и промышленности. – 2019. – С. 479-485.

33. Перепелкин, Д.А. Математическое и программное обеспечение адаптивной маршрутизации и балансировки потоков данных в программно-конфигурируемых сетях с обеспечением качества сетевых сервисов: дисс. … д-ра техн. наук: 05.13.11. – 2017. - 443 с.

34. Платунова, С.М. Исследование метода повышения доступности вычислительной сети с самоподобным трафиком // Успехи современной науки и образования. – 2016. – Т. 2. – №. 11. – С. 70-73.

35. Платунова, С.М. Методы проектирования фрагментов компьютерной сети: учебное пособие // СПб.: НИУ ИТМО. – 2012.

36. Пятибратов, А.П. Сети ЭВМ и телекоммуникации. Юнита 3. Корпоративные компьютерные сети. Эффективность компьютерных сетей и перспективы их развития: учебное пособие. – М., 2009.

37. Рекомендация МСЭ-Т Y.1540. Служба передачи данных по межсетевому протоколу (IP) – Параметры рабочих характеристик переноса и доступности IP-пакетов. – 2016.

38. Рекомендация МСЭ-Т Y.1541. Требования к сетевым показателям качества для служб, основанных на протоколе IP. – 2011.

39. Ремизов, А. Дисциплина обработки очереди HTB. Руководство по использованию [Электронный ресурс]. - 2006. Режим доступа: https://www.opennet.ru/base/net/htb_manual.txt.html.

40. Сурков, Е. В., Монахов, Ю. М., Кузнецова, А. П. Экспериментальная проверка результатов по оценке живучести корпоративной телекоммуникационной сети // Проектирование и технология электронных средств. – 2019. – №. 3. – С. 46-50.

41. Терешкевич, А. А., Зубалов, А. Н. Обзор технологии " Программно-конфигурируемые сети ПКС/SDN" //Газовая промышленность. – 2016. – №. 12 (746). – С. 64-71.

42. Ушаков, Ю. А., Коннов, А. Л., Полежаев, П. Н. Моделирование корпоративной сети, построенной на основе принципов программно-конфигурируемой инфраструктуры и виртуализации сетевых функций //Интеллект. Инновации. Инвестиции. – 2017. – №. 12. – С. 90-96.

43. Яновский, Г. Г. Качество обслуживания в сетях IP // Вестник связи. – 2008. – №. 1. – С. 65-74.

44. Alashaikh, A., Gomes, T., Tipper, D. The spine concept for improving network availability // Computer Networks. – 2015. – Т. 82. – С. 4-19.

45. Barlow, R. E., Proschan, F. Importance of system components and fault tree events // Stochastic Processes and their applications. – 1975. – Т. 3. – №. 2. – С. 153-173.

46. Bastian, C., Chernak, S., Herscovici, D., Witkowski, B. Prioritizing local and network traffic: пат. 8972537 США. – 2015.

47. Bazel - a fast, scalable, multi-language and extensible build system [Электронный ресурс]. – Режим доступа: https://bazel.build.

48. Bhandarkar, S., Behera, G., Khan K.A. Scalability Issues in Software Defined Network (SDN): A Survey // Advances in Computer Science and Information Technology (ACSIT). – 2015. – Т. 2. – №. 1. – С. 81-85.

49. Big Switch Networks – Switch Light - Open Networking Foundation [Электронный ресурс]. - Режим доступа: https://opennetworking.org/sdn-resources/sdn-products/big-switch-networks-switch-light.

50. Blanchard, B.S., Verma D.C., Peterson E.L. Maintainability: a key to effective serviceability and maintenance management. – John Wiley & Sons, 1995. – Т. 13.

51. Bonfim M. S., Dias K. L., Fernandes S. F. L. Integrated NFV/SDN architectures: A systematic literature review //ACM Computing Surveys (CSUR). – 2019. – Т. 51. – №. 6. – С. 1-39.

52. Brown, M.A. Traffic Control HOWTO Version 1.0.2. - октябрь 2006.

53. Brown, M.A. Traffic Control using tcng and HTB HOWTO Version 1.0.1. - апрель 2006.

54. Casado, M., M., Freedman, M. J., Pettit, J., Luo, J., Gude, N., McKeown, N., Shenker, S. Rethinking enterprise network control //IEEE/ACM Transactions on Networking. – 2009. – Т. 17. – №. 4. – С. 1270-1283.

55. Cheng, T. Y., Wang, M., Jia, X. QoS-guaranteed controller placement in SDN //2015 IEEE Global Communications Conference (GLOBECOM). – IEEE, 2015. – С. 1-6.

56. Cisco IOS Quality of Service Solutions Configuration Guide, Release 12.2SR [Электронный ресурс]. – 2013. Режим доступа: https://www.cisco.com/

c/en/us/td/docs/ios/qos/configuration/guide/12_2sr/qos_12_2sr_book/qos_overview.html.

57. Devera, M. HTB Linux queuing discipline manual - user guide [Электронный ресурс]. – 2002. Режим доступа: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm.

58. Devera, M. Hierarchical Token Bucket Theory [Электронный ресурс]. – 2002. Режим доступа: http://luxik.cdi.cz/~devik/qos/htb.

59. Durvy, M., Diot, C., Taft, N., Thiran, P. Network availability based service differentiation // International Workshop on Quality of Service. – Springer, Berlin, Heidelberg, 2003. – С. 305-325.

60. Dutra, D. L. C., Bagaa, M., Taleb, T., Samdanis, K. Ensuring end-to-end QoS based on multi-paths routing using SDN technology //GLOBECOM 2017-2017 IEEE Global Communications Conference. – IEEE, 2017. – С. 1-6.

61. Elsayed, E., Reliability Engineering, Addison Wesley, Reading, MA, 1996.

62. En-Najjary, T., Urvoy-Keller, G. A first look at traffic classification in enterprise networks // Proceedings of the 6th International Wireless Communications and Mobile Computing Conference. – 2010. – С. 764-768.

63. Fawzi, B.B., Hawkes, A.G. Availability of a series system with replacement and repair / /Journal of Applied Probability. – 1990. – С. 873-887.

64. Fawzi, B.B., Hawkes, A.G. Availability of an R-out-of-N system with spares and repairs // Journal of applied probability. – 1991. – С. 397-408

65. Filali, A., Kobbane, A., Elmachkour, M., Cherkaoui, S. SDN controller assignment and load balancing with minimum quota of processing capacity //2018 IEEE International Conference on Communications (ICC). – IEEE, 2018. – С. 1-6.

66. Filali, A., Cherkaoui, S., Kobbane, A. Prediction-based switch migration scheduling for SDN load balancing //ICC 2019-2019 IEEE International Conference on Communications (ICC). – IEEE, 2019. – С. 1-6.

67. Ghalwash, H., Huang, C. H. A QoS framework for SDN-based networks //2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC). – IEEE, 2018. – С. 98-105.

68. GitHub: Where the world builds software [Электронный ресурс]. – Режим доступа: https://github.com.

69. Green, H., Hant, J., Lanzinger, D. Calculating network availability // 2009 IEEE Aerospace conference. – IEEE, 2009. – С. 1-11.

70. Guck, J. W., Van Bemten, A., Reisslein, M., Kellerer, W. Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation //IEEE Communications Surveys & Tutorials. – 2017. – Т. 20. – №. 1. – С. 388-415.

71. Guo, L., Jayasimha, D.N., Chan, J. Credit flow control scheme in a router with flexible link widths utilizing minimal storage: пат. 8711867 США. – 2014.

72. Haddad, S., Mokdad, L., Youcef, S. Response time analysis for composite Web services // IEEE Symposium Communication Systems, Networks and Digital Signal Processing. – 2008. № 6. – С. 1-5.

73. Heller, B., Sherwood, R., McKeown, N. The controller placement problem //ACM SIGCOMM Computer Communication Review. – 2012. – Т. 42. – №. 4. – С. 473-478.

74. HTB Tools Linux – Scribd [Электронный ресурс]. - 2010. Режим доступа: https://ru.scribd.com/doc/48399733/38376664-HTB-tools-Linux.

75. Hu, J., Lin, C., Li, X., Huang, J. Scalability of control planes for software defined networks: Modeling and evaluation //2014 IEEE 22nd International Symposium of Quality of Service (IWQoS). – IEEE, 2014. – С. 147-152.

76. Iera, A., Molinaro, A., Ruggeri, G., Tripodi, D. Improving QoS and throughput in single-and multihop WLANs through dynamic traffic prioritization //IEEE network. – 2005. – Т. 19. – №. 4. – С. 35-44.

77. Iyer, S. Availability results for imperfect repair //Sankhyā: The Indian Journal of Statistics, Series B. – 1992. – С. 249-256.

78. Javadpour, A. Providing a way to create balance between reliability and delays in SDN networks by using the appropriate placement of controllers //Wireless Personal Communications. – 2020. – Т. 110. – №. 2. – С. 1057-1071.

79. Jerome, A., Yuksel, M., Ahmed, S. H., Bassiouni, M. SDN-based load balancing for multi-path TCP //IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). – IEEE, 2018. – С. 859-864.

80. Karakus, M., Durresi, A. Quality of service (QoS) in software defined networking (SDN): A survey //Journal of Network and Computer Applications. – 2017. – Т. 80. – С. 200-218.

81. Keith, S. Prioritizing classes of network traffic to provide a predetermined quality of service: пат. 9130864 США. – 2015.

82. Keshari, S. K., Kansal, V., Kumar, S. A systematic review of quality of services (QoS) in software defined networking (SDN) //Wireless Personal Communications. – 2021. – Т. 116. – №. 3. – С. 2593-2614.

83. Kiran, S. Introducing OpFlex - A new standards-based protocol for Application Centric Infrastructure [Электронный ресурс]. - 2014. - Режим доступа: https://blogs.cisco.com/datacenter/introducing-opflex-a-new-standards-based-protocol-for-application-centric-infrastructure.

84. Kumar, R. Hasan, M., Padhy, S., Evchenko, K. End-to-end network delay guarantees for real-time systems using SDN // 2017 IEEE Real-Time Systems Symposium (RTSS). – IEEE, 2017. – С. 231-242.

85. Laprie, J.C. Dependability: Basic concepts and terminology // Dependability: Basic Concepts and Terminology. – Springer, Vienna, 1992. – С. 3-245.

86. Latif, Z., Sharif, K., Li, F., Karim, M.M., Biswas, S. Wang, Y.A comprehensive survey of interface protocols for software defined networks // Journal of Network and Computer Applications. – 2020. – Т. 156. – С. 1-28.

87. Lee, F., Marathe, M. Beyond Redundancy: A Guide to Designing High-Availability Networks. - Cisco, 1999.

88. Lie, C.H., Hwang, C.L., Tillman, F.A. Availability of maintained systems: a state-of-the-art survey // AIIE Transactions. – 1977. – Т. 9. – №. 3. – С. 247-259.

89. Lin, C., Wang, K., Deng, G. A QoS-aware routing in SDN hybrid networks //Procedia Computer Science. – 2017. – Т. 110. – С. 242-249.

90. Lorenz, C., Hock, D., Scherer, J., Durner, R., Kellerer, W., Gebert, S., Tran-Gia, P. An SDN/NFV-enabled enterprise network architecture offering fine-grained security policy enforcement //IEEE communications magazine. – 2017. – Т. 55. – №. 3. – С. 217-223.

91. Matveeva, A., Monakhov, Y., Monakhov, M., Telny, A., Matveev, S. Algorithm for Maximization of Integral Availability in Software Defined Networks // 2021 Dynamics of Systems, Mechanisms and Machines (Dynamics). – IEEE, 2021. – С. 1-6.

92. Mbodila, M., Isong, B., Gasela, N. A Review of SDN-Based Controller Placement Problem //2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC). – IEEE, 2020. – С. 1-7.

93. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Turner, J. OpenFlow: enabling innovation in campus networks // ACM SIGCOMM computer communication review. – 2008. – Т. 38. – №. 2. – С. 69-74.

94. Mi, J. Some comparison results of system availability // Naval Research Logistics (NRL). – 1998. – Т. 45. – №. 2. – С. 205-218.

95. Mininet: An Instant Virtual Network on Your Laptop (or Other PC) [Электронный ресурс]. – 2021. Режим доступа: http://mininet.org.

96. Monakhov, Y., Kuznetsova, A. Analysis of Congestion Control in Data Channels with Frequent Frame Loss // 2018 2nd European Conference on Electrical Engineering and Computer Science (EECS). – IEEE, 2018. – С. 445-449.

97. Monakhov, Y., Kuznetsova, A. On the behavior of drop-tail queue management algorithms under high packet loss // WSEAS Transactions on Systems and Control. – 2019. – Т. 14. – С. 90-96.

98. Monakhov, Y., Kuznetsova, A., Mamaev, D. An Approach for Managing Availability in Software Defined Network Infrastructure // 2020 IEEE 5th International

Symposium on Smart and Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS). – IEEE, 2020. – C. 1-6.

99. Monakhov, Y., Kuznetsova, A., Monakhov, M., Telny, A., Bednyatsky, I. Performance Evaluation of the Modified HTB Algorithm //2020 Dynamics of Systems, Mechanisms and Machines (Dynamics). – IEEE, 2020. – C. 1-5.

100. Monakhov, Y.M., Monakhov, M.Y., Lantsov V. N. An algorithm for assessing the availability criteria in telecommunication networks // International Journal of Computing. – 2018. – Т. 17. – №. 4. – С. 219-225.

101. Monakhov, Y.M., Monakhov, M.Y., Luchinkin, S.D., Kuznetsova, A.P., Monakhova, M.M. Availability as a metric for region-scale telecommunication designs / // 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). – IEEE, 2019. – Т. 2. – С. 775-779.

102. Mondal, A., Misra, S., Maity, I. Buffer size evaluation of openflow systems in software-defined networks // IEEE Systems Journal. – 2018. – Т. 13. – №. 2. – С. 1359-1366.

103. Murdock, W.P. Component Availability for An Age Replacement Preventive Maintenance Policy. – AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH, 1995. – №. AFIT-96-012D.

104. Nachlas, J.A. Introduction to Reliability Theory. – 1998.

105. Nencioni, G., Helvik, B.E., Gonzalez, A.J., Heegaard, P.E., Kamisinski, A. Availability modelling of software-defined backbone networks // 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W). – IEEE, 2016. – С. 105-112.

106. Nencioni, G., Helvik, B.E., Gonzalez, A.J., Heegaard, P.E., Kamisinski, A. Impact of SDN controllers deployment on network availability // arXiv preprint arXiv:1703.05595. – 2017.

107. Oliveira, A.T., Martins, B.J.C., Moreno, M.F., Vieira, A.B., Gomes, A.T.A., Ziviani, A. SDN-based architecture for providing QoS to high performance

distributed applications // 2018 IEEE Symposium on Computers and Communications (ISCC). – IEEE, 2018. – С. 602-607.

108. Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions [Электронный ресурс]. – Режим доступа: https://opennetworking.org/onos.

109. Open vSwitch [Электронный ресурс]. - Режим доступа: http://www.openvswitch.org.

110. Pan, H.Y., Wang, S.Y. Optimizing the SDN control-plane performance of the Openvswitch software switch // 2015 IEEE Symposium on Computers and Communication (ISCC). – IEEE, 2015. – С. 403-408.

111. Pham, H., Wang, H. Imperfect maintenance // European journal of operational research. – 1996. – Т. 94. – №. 3. – С. 425-438.

112. Qin, Q., Poularakis, K., Iosifidis, G., Tassiulas, L. SDN controller placement at the edge: Optimizing delay and overheads // IEEE INFOCOM 2018-IEEE Conference on Computer Communications. – IEEE, 2018. – С. 684-692.

113. Selvi H., Gür G., Alagöz F. Cooperative load balancing for hierarchical SDN controllers //2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR). – IEEE, 2016. – С. 100-105.

114. Singh, J., Badotra, S. A Review Paper on Software Defined Networking // International Journal of Advanced Research in Computer Science. – 2017. – Т. 8. – №. 3.

115. SLA Management Handbook. GB917. Rel. 3.1. - TM Forum, 2012.

116. Stanwood, K.L., Gell, D., Bao Y. Systems and methods for prioritizing and scheduling packets in a communication network: пат. 8665724 США. – 2014.

117. Sufiev, H., Haddad, Y., Barenboim, L., Soler, J. Dynamic SDN controller load balancing //Future Internet. – 2019. – Т. 11. – №. 3. – С. 75.

118. Tanha M., Sajjadi D., Ruby, R., Pan, J. Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs //IEEE Transactions on Network and Service Management. – 2018. – Т. 15. – №. 3. – С. 991-1005.

119. Tierney, J., Stuart, D., Venables, B. Managing flow control buffer: пат. 8819265 США. – 2014.

120. Tornatore, M., Maier, G., Pattavina, A. Availability design of optical transport networks // IEEE Journal on Selected Areas in Communications. – 2005. – Т. 23. – №. 8. – С. 1520-1532.

121. Traffic Shaping | Bandwidth Control | QOS | WebHTB [Электронный ресурс]. Режим доступа: webhtb.ro.

122. Trivedi, K., Vasireddy, R., Trindade, D., Nathan, S., Castro, R. Modeling high availability systems // Proc. IEEE Pacific Rim Int. Symp. on Dependable Computing (PRDC). – 2006.

123. Trivedi, K.S., Bobbio, A. Reliability and availability engineering: modeling, analysis, and applications. – Cambridge University Press, 2017.

124. Wang, A., Guo, Y., Hao, F., Lakshman, T.V., Chen, S. Scotch: Elastically scaling up SDN control-plane using vswitch based overlay //Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. – 2014. – С. 403-414.

125. Wang, H., Pham, H. Survey of reliability and availability evaluation of complex networks using Monte Carlo techniques // Microelectronics Reliability. – 1997. – Т. 37. – №. 2. – С. 187-209.

126. What is the Vector Packet Processor (VPP) - The Vector Packet Processor documentation [Электронный ресурс]. - Режим доступа: https://s3-docs.fd.io/vpp/22.02.

127. Xiao P., Qu W., Qi H., Li Z., Xu Y. The SDN controller placement problem for WAN //2014 IEEE/CIC International Conference on Communications in China (ICCC). – IEEE, 2014. – С. 220-224.

128. XMPP | An Overview of XMPP [Электронный ресурс]. - Режим доступа: https://xmpp.org/about.

129. Yeganeh, S.H., Tootoonchian, A., Ganjali, Y. On scalability of software-defined networking // IEEE Communications Magazine. – 2013. – Т. 51. – №. 2. – С. 136-141.

130. Yu, J., Wang, Y., Pei, K., Zhang, S., Li, J. A load balancing mechanism for multiple SDN controllers based on load informing strategy //2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS). – IEEE, 2016. – C. 1-4.

131. Zhang, T., Bianco, A., Giaccone, P. The role of inter-controller traffic in SDN controllers placement //2016 IEEE conference on network function virtualization and software defined networks (NFV-SDN). – IEEE, 2016. – C. 87-92.

132. Zhong, H., Fan, J., Cui, J., Xu, Y., Liu, L. Assessing Profit of Prediction for SDN controllers load balancing //Computer Networks. – 2021. – T. 191. – C. 1-10.

133. Zhou, Y., Wang, Y., Yu, J., Ba, J., Zhang, S. Load balancing for multiple controllers in SDN based on switches group //2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS). – IEEE, 2017. – C. 227-230.

134. Zhou, Y., Zheng, K., Ni, W., Liu, R. P. Elastic switch migration for control plane load balancing in SDN //IEEE Access. – 2018. – T. 6. – C. 3909-3919.

135. Zhou, W., Janic, M., Kooij, R.E., Kuipers, F.A. On the availability of networks // Proceedings of Broadband. – 2007. – C. 1-6.

136. Zhu, L., Chai, R., Chen, Q. Control plane delay minimization based SDN controller placement scheme // 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP). – IEEE, 2017. – C. 1-6.

ПРИЛОЖЕНИЕ А

Листинг программы для определения ИПД сети в среде Mininet

Ниже представлены методы для класса Cli.py.

```
def do_make_matrix( self, line ):
    output( '*** Trying to make network matrix...\n' )
    matrix = self.mn.make_matrix()

    #Print matrix
    for line in matrix:
        for col in line:
            output( col, ' ' )
        output( '\n' )

def do_launch_reduce_experiment( self, line ):
    output( '*** Wow, such a nice reduce experimen \n' )
    nodes = self.mn.switches
    for node in nodes:
        output( node.name, '\n' )
        output( node.intfs, '\n' )
        for interface in node.intfs:
            output( node.intfs[interface].link, '\n' )

def do_check_total_availability_multiple( self, line ):
    count = 10
    AminBag = []
    AmaxBag = []
    for i in range( count ):
        output( '*** Checking total availability... ***\n' )
        output( 'Total switches count: ' , len( self.mn.switches ), '\n')
        output( 'Total links count: ' , len( self.mn.links ), '\n')
        initialMatrix = self.mn.make_matrix()
        length = len( initialMatrix ) - 1
        clearMatrix = Matrix = [[0 for x in range( length )] for y in range( length )]
        #output( 'Initial matrix:\n' )
        #self.print_matrix( initialMatrix )
        #output( 'Clear matrix\n' )
        #self.print_matrix( clearMatrix )
        for i in range( len( initialMatrix ) ):
            for j in range( len( initialMatrix[i] ) ):
                if ( i != 0 and j != 0 ):
                    clearMatrix[i - 1][j - 1] = initialMatrix[i][j]
        #output( 'Clear matrix after filling\n' )
```

```
#self.print_matrix( clearMatrix )
### STEP 1.1 | MATRIX H
matrixH = copy.deepcopy( clearMatrix )
for i in range( length ):
    for j in range( length ):
        if (
            clearMatrix[i][j] != 0 and
            clearMatrix[i][j] != 1
        ):
            matrixH[i][j] = j + 1
        else:
            matrixH[i][j] = 0

#output( 'Matrix H:\n' )
#self.print_matrix( matrixH )
### STEP 1.2 | MATRIX FLOYD-WARSHALL
### Amax
matrixFW = copy.deepcopy( clearMatrix )
#output( 'Once again Clear\n' )
#self.print_matrix( clearMatrix )
#output( 'before Floyd-Warshall\n' )
#self.print_matrix( matrixFW )
for k in range( length ):
    for i in range( length ):
        for j in range( length ):
            if (
                i != k and
                j != k and
                clearMatrix[i][k] != 0 and
                clearMatrix[k][j] != 0 and
                clearMatrix[i][j] < clearMatrix[k][j] * clearMatrix[i][k]
            ):
                matrixFW[i][j] = clearMatrix[k][j] * clearMatrix[i][k]
                matrixH[i][j] = k
#output( 'Step 1.2: Floyd-Warshall\n' )
#self.print_matrix( matrixFW )
# MODIFICATED FLOYD-WARHSALL
#output( '\n\n\n' )

### STEP 2.0 | MATRIX T
modInitialMatrix = copy.deepcopy( clearMatrix )
inf = 'inf'
for i in range( length ):
```

```
    for j in range( length ):
        # Diagonal set to 0
        if ( i == j ):
            modInitialMatrix[i][j] = 0
        # Set non linked to INF
        elif ( modInitialMatrix[i][j] == 0 ):
            modInitialMatrix[i][j] = inf
#output( 'Initial matrix for steps 2.x:\n' )
#self.print_matrix( modInitialMatrix )
### STEP 2.1 | MATRIX H
modMatrixH = copy.deepcopy( matrixFW )
for i in range( length ):
    for j in range( length ):
        if (
            modInitialMatrix[i][j] != 0 and
            modInitialMatrix[i][j] != 1 and
            modInitialMatrix[i][j] != inf
        ):
            modMatrixH[i][j] = j + 1
        else:
            modMatrixH[i][j] = 0
#output( 'Algorithm step 2.1: matrix H \n' )
#self.print_matrix( modMatrixH )

### STEP 2.2 | MATRIX G
modMatrixG = copy.deepcopy( modInitialMatrix )
for i in range( length ):
    for j in range( length ):
        if ( modInitialMatrix[i][j] == 0 ):
            modMatrixG[i][j] = 0
        elif ( modInitialMatrix[i][j] != inf ):
            modMatrixG[i][j] = 1
        else:
            modMatrixG[i][j] = inf
#output( 'Algorithm step 2.2: matrix G \n' )
#self.print_matrix( modMatrixG )
### STEP 3.3 | Floyd-Warshall
modMatrixFW = copy.deepcopy( modInitialMatrix )
### Amin
for k in range( length ):
    for i in range( length ):
        for j in range( length ):
            if (
```

```
                    i != k and
                    j != k and
                    modMatrixG[i][k] != inf and
                    modMatrixG[k][j] != 0 and # ???
                    modMatrixG[i][j] > modMatrixG[k][j] * modMatrixG[i][k]
                ):
                    modMatrixG[i][j] = modMatrixG[k][j] * modMatrixG[i][k]
                    modMatrixH[i][j] = modMatrixH[i][k]
                    modInitialMatrix[i][j] = modInitialMatrix[k][j] * modInitialMatrix[i][k]
        Amin = copy.deepcopy( matrixFW )
        Amax = copy.deepcopy( modInitialMatrix )
        Aglobalmax = Matrix = [[1 for x in range( length )] for y in range( length )]
        ### Again 1 instead of zeros on main diagonal
        for i in range( length ):
            for j in range( length ):
                if i == j:
                    Amin[i][j] = 1
                    Amax[i][j] = 1
        # #Step 4: SKIP ( pow and sqrt )

        # output( 'Kolmagorov: Step 3\n' )
        # output( lenKolmagorov, '\n' )

        Dmax = self.normDistKolmagorov( Amax )
        Dmin = self.normDistKolmagorov( Amin )

        output( 'Dmax:', Dmax, '\n')
        output( 'Dmin:', Dmin, '\n')

        DmaxSmall = -0.000305962 * pow( length, 3 ) + 0.0628791 * pow( length, 2 ) + 1.20093 *
length - 1.8333
        DmaxBig = 3.38745 * pow( 10, -6 ) * pow( length, 3) + 0.0140894 * pow(length, 2) +
5.2436 * length - 146.096

        AmaxAvailability = 0
        AminAvailability = 0

        if ( length < 50 ):
            AmaxAvailability = pow(math.e, (Dmax * (-1) / DmaxSmall))
            AminAvailability = pow(math.e, (Dmin * (-1) / DmaxSmall))
        else:
            AmaxAvailability = pow(math.e, (Dmax * (-1) / DmaxBig))
            AminAvailability = pow(math.e, (Dmin * (-1) / DmaxBig))
```

```
        output( 'Amax availability:', AmaxAvailability, '\n')
        output( 'Amin availability:', AminAvailability, '\n')

        AmaxBag.append(AmaxAvailability)
        AminBag.append(AminAvailability)

    output( 'Res: Amax: ', statistics.mean(AmaxBag) , '\n')
    output( 'Res: Amin: ', statistics.mean(AminBag) , '\n')


def do_check_total_availability( self, line ):
    output( '*** Checking total availability... ***\n' )
    output( 'Total switches count: ' , len( self.mn.switches ), '\n')
    output( 'Total links count: ' , len( self.mn.links ), '\n')
    initialMatrix = self.mn.make_matrix()
    length = len( initialMatrix ) - 1

    clearMatrix = Matrix = [[0 for x in range( length )] for y in range( length )]

    #output( 'Initial matrix:\n' )
    #self.print_matrix( initialMatrix )
    #output( 'Clear matrix\n' )
    #self.print_matrix( clearMatrix )

    for i in range( len( initialMatrix ) ):
        for j in range( len( initialMatrix[i] ) ):
            if ( i != 0 and j != 0 ):
                clearMatrix[i - 1][j - 1] = initialMatrix[i][j]

    #output( 'Clear matrix after filling\n' )
    #self.print_matrix( clearMatrix )

    ### STEP 1.1 | MATRIX H
    matrixH = copy.deepcopy( clearMatrix )

    for i in range( length ):
        for j in range( length ):
            if (
                clearMatrix[i][j] != 0 and
                clearMatrix[i][j] != 1
            ):
                matrixH[i][j] = j + 1
```

```
        else:
            matrixH[i][j] = 0

#output( 'Matrix H:\n' )
#self.print_matrix( matrixH )

### STEP 1.2 | MATRIX FLOYD-WARSHALL
### Amax
matrixFW = copy.deepcopy( clearMatrix )

#output( 'Once again Clear\n' )
#self.print_matrix( clearMatrix )
#output( 'before Floyd-Warshall\n' )
#self.print_matrix( matrixFW )

for k in range( length ):
    for i in range( length ):
        for j in range( length ):
            if (
                i != k and
                j != k and
                clearMatrix[i][k] != 0 and
                clearMatrix[k][j] != 0 and
                clearMatrix[i][j] < clearMatrix[k][j] * clearMatrix[i][k]
            ):
                matrixFW[i][j] = clearMatrix[k][j] * clearMatrix[i][k]
                matrixH[i][j] = k

#output( 'Step 1.2: Floyd-Warshall\n' )
#self.print_matrix( matrixFW )

# MODIFICATED FLOYD-WARHSALL
#output( '\n\n\n' )

### STEP 2.0 | MATRIX T
modInitialMatrix = copy.deepcopy( clearMatrix )

inf = 'inf'
for i in range( length ):
    for j in range( length ):
        # Diagonal set to 0
        if ( i == j ):
            modInitialMatrix[i][j] = 0
```

```
        # Set non linked to INF
        elif ( modInitialMatrix[i][j] == 0 ):
            modInitialMatrix[i][j] = inf

#output( 'Initial matrix for steps 2.x:\n' )
#self.print_matrix( modInitialMatrix )

### STEP 2.1 | MATRIX H
modMatrixH = copy.deepcopy( matrixFW )

for i in range( length ):
    for j in range( length ):
        if (
            modInitialMatrix[i][j] != 0 and
            modInitialMatrix[i][j] != 1 and
            modInitialMatrix[i][j] != inf
        ):
            modMatrixH[i][j] = j + 1
        else:
            modMatrixH[i][j] = 0

#output( 'Algorithm step 2.1: matrix H \n' )
#self.print_matrix( modMatrixH )

### STEP 2.2 | MATRIX G
modMatrixG = copy.deepcopy( modInitialMatrix )

for i in range( length ):
    for j in range( length ):
        if ( modInitialMatrix[i][j] == 0 ):
            modMatrixG[i][j] = 0
        elif ( modInitialMatrix[i][j] != inf ):
            modMatrixG[i][j] = 1
        else:
            modMatrixG[i][j] = inf

#output( 'Algorithm step 2.2: matrix G \n' )
#self.print_matrix( modMatrixG )

### STEP 3.3 | Floyd-Warshall
modMatrixFW = copy.deepcopy( modInitialMatrix )
### Amin
```

```
            for k in range( length ):
                for i in range( length ):
                    for j in range( length ):
                        if (
                            i != k and
                            j != k and
                            modMatrixG[i][k] != inf and
                            modMatrixG[k][j] != 0 and # ???
                            modMatrixG[i][j] > modMatrixG[k][j] * modMatrixG[i][k]
                        ):
                            modMatrixG[i][j] = modMatrixG[k][j] * modMatrixG[i][k]
                            modMatrixH[i][j] = modMatrixH[i][k]
                            modInitialMatrix[i][j] = modInitialMatrix[k][j] * modInitialMatrix[i][k]

            ### Kolmagorov
            Amin = copy.deepcopy( matrixFW )
            Amax = copy.deepcopy( modInitialMatrix )
            Aglobalmax = Matrix = [[1 for x in range( length )] for y in range( length )]

            ### Again 1 instead of zeros on main diagonal
            for i in range( length ):
                for j in range( length ):
                    if i == j:
                        Amin[i][j] = 1
                        Amax[i][j] = 1

            # output( 'Kolmagorov: Step 1-2\n' )
            # self.print_matrix( mKolmagorov )

            Dmax = self.normDistKolmagorov( Amax )
            Dmin = self.normDistKolmagorov( Amin )
            output( 'Dmax:', Dmax, '\n')
            output( 'Dmin:', Dmin, '\n')
            DmaxSmall = -0.000305962 * pow( length, 3 ) + 0.0628791 * pow( length, 2 ) + 1.20093 *
length - 1.8333
            DmaxBig = 3.38745 * pow( 10, -6 ) * pow( length, 3) + 0.0140894 * pow(length, 2) + 5.2436
* length - 146.096
            AmaxAvailability = 0
            AminAvailability = 0
            if ( length < 50 ):
                AmaxAvailability = pow(math.e, (Dmax * (-1) / DmaxSmall))
                AminAvailability = pow(math.e, (Dmin * (-1) / DmaxSmall))
            else:
```

```
            AmaxAvailability = pow(math.e, (Dmax * (-1) / DmaxBig))
            AminAvailability = pow(math.e, (Dmin * (-1) / DmaxBig))
        output( 'Amax availability:', AmaxAvailability, '\n')
        output( 'Amin availability:', AminAvailability, '\n')

        @staticmethod
    def matrix_mean( matrix ):
        summ = 0
        inf = 'inf'

        for line in matrix:
            for col in line:
                if col != inf:
                    try:
                        summ += col
                    except TypeError:
                        pass

        mean = float(summ) / ( pow ( len( matrix ), 2 ) )
        return mean

    @staticmethod
    def print_matrix( matrix ):
        for line in matrix:
            for col in line:
                output( col, ' ' )
            output( '\n' )

    def do_show_vertices_info( self, line ):
        output( '*** Getting vertices info...\n' )
        self.mn.show_vertices_info()

    def do_hot_link( self, line ):
        # add link shorthand
        args = line.split()
        if len(args) != 2:
            error( 'invalid number of args: hot_link end1 end2\n' )
        else:
            self.do_py(  'net.addLink(\'{name1}\',  \'{name2}\')'.format(  name1  =  args[0],  name2  =
args[1] ) )

    def do_improve_topology( self, line ):
        output( '*** Trying to improve topology...\n' )
```

```
    if line:
        self.mn.improve_topology( int( line ) )
    else:
        self.mn.improve_topology()

def do_get_possible_links( self, line ):
    output( '*** Possible links' )
    links = self.mn.getPossibleLinks()
    output(links, '\n')
    output('Count: ', len(links), '\n')

def do_check_link( self, line ):
    args = line.split(' ')
    output( str( self.mn.checkExistence( self.mn.getExistentLinks(), args[0], args[1] ) ), '\n' )

def do_pingpair_custom( self, line ):
    args = line.split()
    nodes = []
    nodes.append( self.mn.getNodeByName(args[0]) )
    nodes.append( self.mn.getNodeByName(args[1]) )

    count = 1
    if ( len(args) == 3 ):
        count = int(float(args[2]))
    self.mn.pingPairCustom( nodes, count )
```

Ниже представлен исходный код для класса Net.py эмулятора Mininet.

```
def make_matrix( self ):
    vertices = self.switches
    verticesCount = len( vertices )
    Matrix = [[0 for x in range( len( vertices ) + 1 )] for y in range( len( vertices ) +1 )]
    Matrix[0][0] = None
    # initial preparations
    for i in range( 1, len( vertices ) + 1):
        vertex = vertices[i - 1]
        # Adding hosts name as vertex for matrix
        Matrix[0][i] = vertex.name
        Matrix[i][0] = vertex.name
        # Define self-to-self as 1
        Matrix[i][i] = 1

    # Helpers
    def getIndexByInterfaceName( name ):
```

```
            for i in range( len( vertices ) ):
                vertex = vertices[i]
                if ( name == vertex.name ):
                    return i
            return False

        for i in range( len(vertices )):
            vertex = vertices[i]
            links = []
            for interface in vertex.intfList():
                if ( interface ):
                    if ( interface.link ):
                        if ( interface.link.intf2.node.ibmSwitch ):
                            if ( interface.link.intf1.node.name == vertex.name ):
                                name = interface.link.intf2.node.name
                            else:
                                name = interface.link.intf1.node.name
                            links.append( [name, interface.link.intf1.name , interface.link.intf2.name] )
            #output( links, '\n' )
            if ( links ):
                for link in links:
                    destinationVertextName = link[0]
                    linkInterfaceFrom = link[1]
                    linkInterfaceTo = link[2]
                    #output( vertex.name, ' ', destinationVertextName, ' ', linkInterfaceFrom, ' ',
linkInterfaceTo, '\n' )
                    relativeIndex = getIndexByInterfaceName( destinationVertextName )
                    #output( destinationVertextName, ': ', relativeIndex, '\n' )
                    if ( relativeIndex is not False ):
                        if ( i + 1 != relativeIndex):
                            #output( self.ping( [ self.getNodeByName( linkInterfaceFrom ),
self.linkInterfaceTo ] ), '\n' )
                            self.customPing( [ vertex, self.getNodeByName( destinationVertextName ) ] )
                            #Matrix[i + 1][relativeIndex + 1] = 'y({name1}:{name2})'.format( name1 =
linkInterfaceFrom, name2 = linkInterfaceTo )
                            #Matrix[relativeIndex + 1][i + 1] = 'y({name1}:{name2})'.format( name1 =
linkInterfaceFrom, name2 = linkInterfaceTo )
                            Matrix[i + 1][relativeIndex + 1] = self.customPing( [ vertex,
self.getNodeByName( destinationVertextName ) ] )
                            Matrix[relativeIndex + 1][i + 1] = self.customPing( [ self.getNodeByName(
destinationVertextName ), vertex ] )
        return Matrix
```

```
def show_vertices_info( self ):
    vertices = self.switches
    existedLinks = self.getExistentLinks()
    if ( len( existedLinks ) ):
        output( 'List of existent switche\'s links:\n' )
        for i in range( len( existedLinks ) ):
            output( existedLinks[i] )
            if i != len( existedLinks ) - 1:
                output( ', ' )
        output( '\n' )
    possibleLinks = self.getPossibleLinks()
    if ( len( possibleLinks ) ):
        output( 'List of available links to create:\n' )
        for i in range( len( possibleLinks ) ):
            output( possibleLinks[i] )
            if i != len( possibleLinks) - 1:
                output( ', ' )
        output( '\n' )
    output( 'completed...\n' )

def improve_topology( self, count=float( inf ) ):
    startTime = time.time()
    possibleLinks = self.getPossibleLinks()
    random.shuffle( possibleLinks )
    i = 0
    output( 'Possible links count: ', len( possibleLinks ), '\n' )
    now = datetime.now()
    filePath = "results/{date}.html".format( date = now.strftime( "%m_%d_%H_%M_%S" ) )
    if not os.path.exists(os.path.dirname(filePath)):
        os.makedirs(os.path.dirname(filePath))

    htmlDoc = open(filePath, "w")
    htmlHeader = """
        <!DOCTYPE html>
            <html lang="en">
            <head>
                <meta charset="UTF-8">
                <meta name="viewport" content="width=device-width, initial-scale=1.0">
                <title>Document</title>
                <link rel="stylesheet" href="styles.css">
            </head>
            <body>
                <h1>Results of IBm SDN strage stuff</h1>
```

```
            """
            htmlFooter = """
                    </body>
                </html>
            """
            htmlDoc.write( htmlHeader )
            matrix = self.make_matrix()
            htmlDoc.write( '<h2>Initial topology snapshot:</h2>' )
            htmlDoc.write( self.matrixToHtmlTable( matrix ) )
            # [max,min]
            availabilities = CLI.determiate_availability( matrix )
            htmlDoc.write( '<h2>Network max availability: {max}</h2>'.format( max = availabilities[0])
)
            htmlDoc.write( '<h2>Network min availability: {min}</h2>'.format( min = availabilities[1])
)
            while ( len( possibleLinks ) != 0 and i < count ):
                newLink = possibleLinks.pop( 0 )
                htmlDoc.write( '<h2>New link in topology: {link}</h2>'.format( link = newLink) )
                splittedLink = newLink.split( ':' )
                output( 'Added new link: ', self.addLink( splittedLink[0], splittedLink[1]), '\n' )
                tempMatrix = self.make_matrix()
                tempAvailabilities = CLI.determiate_availability( tempMatrix )
                htmlDoc.write( self.matrixToHtmlTable( tempMatrix ) )
                htmlDoc.write( '<h2>Network max availability: {max}</h2>'.format( max =
tempAvailabilities[0]) )
                htmlDoc.write( '<h2>Network min availability: {min}</h2>'.format( min =
tempAvailabilities[1]) )
                htmlDoc.write( '<br><br>' )
                output( ("--- Added link. Time spent. Time spent: %s seconds ---" % ( time.time() -
startTime ) ), '\n' )
                output( 'Possible links left: ', len(possibleLinks) , '\n')
                i += 1
            htmlDoc.write( htmlFooter )
            htmlDoc.close()

            output( 'Result was logged into:', filePath, '\n' )
            output( ("--- Elapsed time: %s seconds ---" % ( time.time() - startTime ) ), '\n' )
            output( 'Completed.\n')

        # Helpers
        def getExistentLinks( self ):
            output = []
            for link in self.links:
```

```
                if ( link.intf1.node.ibmSwitch and link.intf2.node.ibmSwitch ):
                    output.append( link.intf1.node.name + ':' + link.intf2.node.name )
            return output

        def getPossibleLinks( self ):
            output = []
            existedLinks = self.getExistentLinks()
            for switch in self.switches:
                for relation in self.switches:
                    if ( switch.name != relation.name and not switch.name + ':' + relation.name in output and
not relation.name + ':' + switch.name in output ):
                        if not ( self.checkExistence( existedLinks, switch.name, relation.name ) ):
                            output.append( switch.name + ':' + relation.name )
            return output

        def checkExistence( self, existent, name1, name2 ):
            for item in existent:
                if ( name1 + ':' + name2 == item or name2 + ':' + name1 == item ):
                    return True
            return False

        def hot_link( self, name1, name2 ):
            # add link shorthand
            if ( name1 and name2 ):
                self.mn.do_py( 'net.addLink(\'{name1}\', \'{name2}\')'.format( name1 = name1, name2 =
name2 ) )

        def getExpectedValue( self, items = None):
            if not items:
                return False
            mathExpected = 0
            mathExpectedSquare = 0
            dispersion = 0
            for item in items:
                probability = 1 / float ( len ( items ) )
                mathExpected += item * probability
                mathExpectedSquare += ( item ** 2 ) * probability
            #output( 'mathExpected', mathExpected, '\n' )
            #output( 'mathExpectedSquare', mathExpectedSquare, '\n' )
            dispersion = mathExpectedSquare - float( ( mathExpected ** 2 ) )
            getcontext().prec = 5
            #output( mathExpected, '\n' )
            #output( format( Decimal.from_float( dispersion ), '.5' ), '\n' )
```

```
    return format( Decimal.from_float( dispersion ), '.5' )

def customPing( self, hosts=None, timeout=None, count=250 ):
    latencyLimit = 0.03
    node = hosts[0]
    dest = hosts[1]
    latenciesList = []
    if node != dest:
        opts = ''
        if timeout:
            opts = '-W %s' % timeout
        latencies = []
        for i in range( count ):
            result = node.cmd( 'ping -c1 -f %s %s' % ( opts, dest.IP()) )
            #output( result, '\n' )
            outputs = self._parsePingFull( result )
            sent, received, rttmin, rttavg, rttmax, rttdev = outputs
            latenciesList.append( rttavg )
            if ( rttavg <= latencyLimit ):
                latencies.append( rttavg )
        probability = float( len( latencies ) ) / count
        self.getExpectedValue( latenciesList )
        #output( self.getExpectedValue( latenciesList ), '\n' )
        return probability

def pingPairCustom( self, hosts=None, count = 1 ):
    if ( not len(hosts) ):
        output('Hosts were not provied')
        return
    return self.customPingMathStat( hosts=hosts, count=count )


def customPingMathStat( self, hosts=None, timeout=None, count=100 ):
    # should we check if running?
    # Each value is a tuple: (src, dsd, [all ping outputs])
    latencyLimit = 0.03

    node = hosts[0]
    dest = hosts[1]
    latenciesList = []
    startTime = time.time()
    if node != dest:
```

```
            opts = ''
            if timeout:
               opts = '-W %s' % timeout
            latencies = []
            for i in range( count ):
               result = node.cmd( 'ping -c1 -f %s %s' % ( opts, dest.IP()) )
               #output( result, '\n' )
               outputs = self._parsePingFull( result )
               sent, received, rttmin, rttavg, rttmax, rttdev = outputs

               latenciesList.append( rttavg )

               if ( rttavg <= latencyLimit ):
                  latencies.append( rttavg )

            probability = float( len( latencies ) ) / count

            self.getExpectedValue( latenciesList )
            output(json.dumps(latenciesList), '\n')

            filePath = "csv/{count}_time_{time}s.csv".format( count = count, time = time.time() -
startTime )

            if not os.path.exists(os.path.dirname(filePath)):
               os.makedirs(os.path.dirname(filePath))
               os.chmod('csv', 0o666)

            with open(filePath, 'wb') as file:
               wr = csv.writer(file, quoting=csv.QUOTE_ALL)
               for line in latenciesList:
                  wr.writerow([ line ])
               file.close()

            return probability

         def matrixToHtmlTable( self, matrix = None):
            result = """
               <table class="table">
                  <tbody>
            """

            def printableFormat( value ):
               return format( Decimal.from_float( float( value ) ), '.5' )
```

```
        for i in range( len( matrix ) ):
            result += "<tr>"
            for j in range( len( matrix[i] ) ):
                if matrix[i][j] == None:
                    result += "<td>{value}</td>".format( value = " )
                elif ( j == 0 and i != 0 ) or ( j !=0 and i == 0 ):
                    #print headers
                    result += "<td class=\"cell cell_state_heading\">{value}</td>".format( value = matrix[i][j] )
                elif matrix[i][j] != 0 and matrix[i][j] >= 0.9:
                    result += "<td class=\"cell cell_state_filled\">{value}</td>".format( value = printableFormat( matrix[i][j] ) )
                elif matrix[i][j] < 0.9 and matrix[i][j] != 0:
                    result += "<td class=\"cell cell_state_misslatency\">{value}</td>".format( value = printableFormat( matrix[i][j] ) )
                else:
                    result += "<td>{value}</td>".format( value = printableFormat( matrix[i][j] ) )
            result += "</tr>"

        result += """
                </tbody>
            </table>
        """


        return result
```

ПРИЛОЖЕНИЕ Б

Листинг программы для генерации заданной топологии в среде Mininet

```
#!/usr/bin/python


from mininet.topo import Topo
from mininet.util import irange

from mininet.cli import CLI
from mininet.clean import Cleanup
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch

class ProjectTopology( Topo ):

   hostIndex = 1

   def build( self ):

      # Create main switches
      mainR1 = self.addSwitch( 'ar1', dpid='0000000000000001' )
      mainR2 = self.addSwitch( 'br1', dpid='0000000000000002' )
      mainR3 = self.addSwitch( 'cr1', dpid='0000000000000003' )

      # Link main switches
      self.addLink(mainR1, mainR2)
      self.addLink(mainR2, mainR3)



      # Creating the left side schema's tolopogy STARTS
      mainR1sub1 = self.addSwitch( 'as1', dpid='0000000000000004' )
      mainR1sub2 = self.addSwitch( 'as2', dpid='0000000000000005' )
      mainR1sub3 = self.addSwitch( 'as3', dpid='0000000000000006' )

      self.addLink(mainR1, mainR1sub1)
      self.addLink(mainR1, mainR1sub2)
      self.addLink(mainR1, mainR1sub3)

            self.addHosts(mainR1sub1, 1)
            self.addHosts(mainR1sub3, 1)
```

```
self.addHosts(mainR1sub1, 6)
self.addHosts(mainR1sub3, 6)

mainR1sub2R1 = self.addSwitch( 'ar2', dpid='0000000000000007' )
mainR1sub2R1sub1 = self.addSwitch( 'as4', dpid='0000000000000008' )
mainR1sub2R1sub2 = self.addSwitch( 'as5', dpid='0000000000000009' )
mainR1sub2R1sub3 = self.addSwitch( 'as6', dpid='000000000000000A' )

self.addLink(mainR1sub2, mainR1sub2R1)
self.addLink(mainR1sub2R1, mainR1sub2R1sub1)
self.addLink(mainR1sub2R1, mainR1sub2R1sub2)
self.addLink(mainR1sub2R1, mainR1sub2R1sub3)

      self.addHosts(mainR1sub2R1sub1, 1)
self.addHosts(mainR1sub2R1sub2, 1)
self.addHosts(mainR1sub2R1sub3, 1)
self.addHosts(mainR1sub2R1sub1, 8)
self.addHosts(mainR1sub2R1sub2, 8)
self.addHosts(mainR1sub2R1sub3, 9)
# Creating the left side schema's tolopogy ENDS

# Creating the middle side schema's tolopogy STARTS
mainR2sub1 = self.addSwitch( 'bs1', dpid='000000000000000B' )
mainR2sub2 = self.addSwitch( 'bs2', dpid='000000000000000C' )
mainR2sub3 = self.addSwitch( 'bs3', dpid='000000000000000D' )

self.addLink(mainR2, mainR2sub1)
self.addLink(mainR2, mainR2sub2)
self.addLink(mainR2, mainR2sub3)

self.addHosts(mainR2sub1, 8)
self.addHosts(mainR2sub3, 10)
      self.addHosts(mainR2sub1, 1)
self.addHosts(mainR2sub3, 1)

mainR2sub2R1 = self.addSwitch( 'br2', dpid='000000000000000E' )
mainR2sub2R1sub1 = self.addSwitch( 'bs4', dpid='000000000000000F' )
mainR2sub2R1sub2 = self.addSwitch( 'bs5', dpid='0000000000000010' )

self.addLink(mainR2sub2, mainR2sub2R1)
self.addLink(mainR2sub2R1, mainR2sub2R1sub1)
self.addLink(mainR2sub2R1, mainR2sub2R1sub2)
```

```
            self.addHosts(mainR2sub2R1sub1, 1)
        self.addHosts(mainR2sub2R1sub2, 1)
        self.addHosts(mainR2sub2R1sub1, 13)
        self.addHosts(mainR2sub2R1sub2, 9)
        # Creating the middle side schema's tolopogy ENDS

        # Creating the right side schema's tolopogy STARTS
        mainR3sub1 = self.addSwitch( 'cs1', dpid='0000000000000011' )
        mainR3sub2 = self.addSwitch( 'cs2', dpid='0000000000000012' )
        mainR3sub3 = self.addSwitch( 'cs3', dpid='0000000000000013' )

        self.addLink(mainR3, mainR3sub1)
        self.addLink(mainR3, mainR3sub2)
        self.addLink(mainR3, mainR3sub3)

        self.addHosts(mainR2sub2, 1)
        self.addHosts(mainR2sub3, 1)
            self.addHosts(mainR2sub2, 9)
        self.addHosts(mainR2sub3, 6)

        mainR3sub1R1 = self.addSwitch( 'cr2', dpid='0000000000000014' )
        mainR3sub1R1sub1 = self.addSwitch( 'cs4', dpid='0000000000000015' )

        self.addLink(mainR3sub1, mainR3sub1R1)
        self.addLink(mainR3sub1R1, mainR3sub1R1sub1)

            self.addHosts(mainR3sub1R1sub1, 1)
        self.addHosts(mainR3sub1R1sub1, 1)
        self.addHosts(mainR3sub1R1sub1, 4)
        self.addHosts(mainR3sub1R1sub1, 4)
        # Creating the right side schema's tolopogy ENDS


    def addHosts( self, switch, hosts ):
        for n in irange( 1, hosts ):
            tmp = self.addHost( 'h' + str(self.hostIndex) )
            self.addLink( switch, tmp )
            self.hostIndex += 1


def runProjectTopo():

    topo = ProjectTopology()
```

```
    net = Mininet(
        topo=topo,
        controller=None,
        switch=OVSSwitch,
        autoSetMacs=True )

    net.addController(
        'c0',
        controller=RemoteController,
        ip='0.0.0.0',
        port=6633 )

    # Actually start the network
    net.start()

    # Start net pingall
    # net.pingAll()

    # Drop the user in to a CLI so user can run commands.
    CLI( net )

    # After the user exits the CLI, shutdown the network.
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )

    # Before creating the topology clean up junk which might be left over from old runs
    Cleanup.cleanup()

    # Create our custom topology
    runProjectTopo()

topos = {
    'sdnTopology': ProjectTopology
}
```

ПРИЛОЖЕНИЕ В

Основные методы и классы программы оптимизации топологии КПТС

Класс расчета остовного дерева

```
public class MaxSpanningTreeCounter implements SpanningTreeCounter {
    @Override
    public double[][] count(double[][] matrix) {
        return MatrixCountHelper.getMaxSpanningTree(matrix);
    }
}
```

Класс добавления новой связи на текущем шаге алгоритма

```
public class AddingBestAmaxEdgeStep implements Step {
    @Override
    public double[][] make(StepData data) {
        double beforeAMax = MatrixCountHelper.countAMax(data.getMatrix());
        double[][] matrix = MatrixUtils.copyMatrix(data.getMatrix());
        double[] access = new double[matrix.length];

        for (int i=0; i<matrix.length; i++) {
            access[i] = Arrays.stream(matrix[i]).filter(d -> d < 1 && d > 0).findFirst().orElse(0);
        }
        List<Triple<Integer, Integer, Double>> nextPairs = new ArrayList<>();

        for (int i=0; i<matrix.length; i++) {
            for (int j=i+1; j<matrix.length; j++) {
                if (matrix[i][j] == 0) {
                    double[][] countMatrix = MatrixUtils.copyMatrix(matrix);
                    countMatrix[i][j] = access[i];
                    countMatrix[j][i] = access[j];
                    nextPairs.add(Triple.of(i, j, MatrixCountHelper.countAMax(countMatrix)));
                }
            }
        }
        nextPairs.sort(Comparator.comparing((Function<Triple<Integer, Integer, Double>,
Double>) Triple::getRight).reversed());

        if (nextPairs.isEmpty()) {
            throw new EarlyEndException();
        }
```

```
            Triple<Integer, Integer, Double> maxAccessResult = nextPairs.get(0);
            if (maxAccessResult.getLeft() == -1) {
               throw new EarlyEndException();
            }

            matrix[maxAccessResult.getLeft()][maxAccessResult.getMiddle()] = access
[maxAccessResult.getLeft()];
            matrix[maxAccessResult.getMiddle()][maxAccessResult.getLeft()] = access
[maxAccessResult.getMiddle()];
            double aMax = MatrixCountHelper.countAMax(matrix);

            if (aMax < beforeAMax) {
               throw new EarlyEndException();
            }
            return matrix;
         }
      }
```

Класс проведения эксперимента

```
      public class MetaSpanningTreeAlphaExperiment implements Experiment {
         private final String name;
         private final String description;
         private final int count;
         private final double alpha;
         private final MatrixGenerator matrixGenerator;
         private final SpanningTreeCounter spanningTreeCounter;
         private final Step step;
         private final EndingCondition endingCondition;
         @Override
         public ExperimentResult make() {
            List<double[][]> resultMatrixes = new ArrayList<>();
            int vertexes = 0;
            int edges = 0;
            int fails = 0;
            List<MetaExperimentResult> metaExperimentResults = new ArrayList<>();
            for (int i=0; i<count; i++) {
               MetaExperimentResult result = null;
               while (result == null || result.getDeltaAMax() < 0) {
                  result = makeOneExperiment();
                  if (result.getDeltaAMax() < 0) {
                     fails++;
```

```
                }
            }
            metaExperimentResults.add(result);
            vertexes = result.getVertexes();
            edges = result.getEdges();
            resultMatrixes.add(result.getResultMatrix());
        }

        log.debug("Spanning tree size: {}", vertexes-1);
        log.debug("Spanning tree average aMax: {}", metaExperimentResults.stream().
mapToDouble(MetaExperimentResult::getSpanningAMax).average().orElse(0));
        log.debug("Max steps from spanning tree to init size: {}", edges - (vertexes-1));
        double maxAMaxGrow = metaExperimentResults.stream().mapToDouble
(MetaExperimentResult::getDeltaAMax).max().orElse(0)*100;
        log.debug("Max aMax grow: {} %", maxAMaxGrow);
        log.debug("Average aMax grow: {} %", metaExperimentResults.stream().
mapToDouble(MetaExperimentResult::getDeltaAMax).average().orElse(0)*100);
        log.debug("Average step for equal: {}", Math.round(metaExperimentResults.stream().
mapToInt(MetaExperimentResult::getIncreaseStep).average().orElse(0)));
        log.debug("Average steps for alpha: {}", Math.round(metaExperimentResults.stream().
mapToInt(MetaExperimentResult::getFinalStep).average().orElse(0)));
        log.debug("Fails: {} on {} experiments", fails, count);

        XYSeries aMinSeries = new XYSeries("AMin");
        XYSeries aMaxSeries = new XYSeries("AMax");

        int aMinMax = metaExperimentResults.stream().flatMap(exp -> exp.getAMinMap().
keySet().stream()).mapToInt(Integer::intValue).max().orElse(0);
        int aMaxMax = metaExperimentResults.stream().flatMap(exp -> exp.getAMaxMap().
keySet().stream()).mapToInt(Integer::intValue).max().orElse(0);

        double min = 10000;
        double max = -10000;

        for (int i=0; i<Math.max(aMinMax, aMaxMax)+1; i++) {
            int key = i;
            double aMax = metaExperimentResults.stream().map(exp -> exp.getAMaxMap()
.getOrDefault(key, null)).filter(d -> d != null && !d.isNaN()).mapToDouble
(Double::doubleValue).average().orElse(0);
            double aMin = metaExperimentResults.stream().map(exp -> exp.getAMinMap().
getOrDefault(key, null)).filter(d -> d != null && !d.isNaN()).mapToDouble (Double::doubleValue).
average().orElse(0);
            aMinSeries.add(key, aMin);
```

```
                aMaxSeries.add(key, aMax);
                min = DoubleStream.of(min, aMin, aMax).min().orElse(min);
                max = DoubleStream.of(max, aMin, aMax).max().orElse(max);
            }

            return new ExperimentResult()
                    .setExperiments(metaExperimentResults)
                    .setMaxAMaxGrow(maxAMaxGrow)
                    .setResultMatrix(resultMatrixes)
                    .setAMinSeries(List.of(aMinSeries))
                    .setAMaxSeries(List.of(aMaxSeries))
                    .setMinAxesVal(min)
                    .setMaxAxesVal(max);
        }

        private MetaExperimentResult makeOneExperiment() {
            double[][] initMatrix = matrixGenerator.generate();
            double startAMax = MatrixCountHelper.countAMax(initMatrix);
            double startAMin = MatrixCountHelper.countAMin(initMatrix);
            Map<Integer, Double> aMinMap = new HashMap<>();
            Map<Integer, Double> aMaxMap = new HashMap<>();

            aMaxMap.put(0, startAMax);
            aMinMap.put(0, startAMin);

            double[][] matrix = spanningTreeCounter.count(initMatrix);
            double spanningTreeAMax = MatrixCountHelper.countAMax(matrix);
            aMaxMap.put(1, spanningTreeAMax);
            aMinMap.put(1, MatrixCountHelper.countAMin(matrix));

            int step = 2;

            double preAMax = MatrixCountHelper.countAMax(matrix);
            double[][] preMatrix = matrix;

            int increaseStep = -1;
            while (!endingCondition.isEnd(new StepResult().setStep(step).setInitMatrix(initMatrix).
setMatrix(matrix))) {
                try {
                    matrix = this.step.make(new StepData().setMatrix(matrix));
                } catch (EarlyEndException e) {
                    break;
                }
```

```
        double aMin = MatrixCountHelper.countAMin(matrix);
        double aMax = MatrixCountHelper.countAMax(matrix);
        if (aMax > startAMax && increaseStep == -1) {
            increaseStep = step;
        }
        if (aMax - preAMax < alpha) {
            matrix = preMatrix;
            break;
        }
        preMatrix = matrix;
        preAMax = aMax;
        aMinMap.put(step, aMin);
        aMaxMap.put(step, aMax);
        step++;
    }

    double finishAMax = MatrixCountHelper.countAMax(matrix);
    double finishAMin = MatrixCountHelper.countAMin(matrix);

    double deltaAMax = (finishAMax - startAMax) / startAMax;
    double deltaAMin = (finishAMin - startAMin) / startAMin;

    return new MetaExperimentResult()
            .setStartAMax(startAMax)
            .setResultAMax(finishAMax)
            .setStartAMin(startAMin)
            .setResultAMin(finishAMin)
            .setStartMatrix(initMatrix)
            .setResultMatrix(matrix)
            .setVertexes(initMatrix.length)
            .setEdges(MatrixCountHelper.countEdges(initMatrix))
            .setSpanningAMax(spanningTreeAMax)
            .setAMaxMap(aMaxMap)
            .setAMinMap(aMinMap)
            .setDeltaAMax(deltaAMax)
            .setDeltaAMin(deltaAMin)
            .setIncreaseStep(increaseStep)
            .setFinalStep(step-1);
}
@Override
public String getName() {
    return name;
}
```

```
    @Override
    public String getDescription() {
        return description;
    }
}
```

Утилитарный класс для работы с матрицами доступностей

```
public class MatrixCountHelper {
    public static int countEdges(double[][] matrix) {
        int sum = 0;
        for (int i=0; i<matrix.length; i++) {
            for (int j=i+1; j<matrix.length; j++) {
                if (matrix[i][j] > 0) {
                    sum++;
                }
            }
        }
        return sum;
    }
    public static Pair<Integer, Double> findColumnWithMinMedian(double[][] matrix, double minBorder) {
        double min = 10000d;
        int minCol = -1;
        for (int i=0; i<matrix.length; i++) {
            double[] col = MatrixUtils.getColumn(matrix, i);
            double median = countMedian(col);
            if (min > median && median > minBorder) {
                min = median;
                minCol = i;
            }
        }
        return Pair.of(minCol, min);
    }

    public static Pair<Integer, Double> findMin(double[] row, double minBorder) {
        double min = 10000d;
        int minI = -1;
        for (int i=0; i<row.length; i++) {
            if (min > row[i] && row[i] < 1 && row[i] > minBorder) {
                min = row[i];
                minI = i;
            }
```

```
            }
            return Pair.of(minI, min);
        }

        public static double countMedian(double[] row) {
            List<Double> sorted =  Arrays.stream(row).filter(d -> d > 0 && d < 1).sorted().boxed().
collect(Collectors.toList());
            if (sorted.size() % 2 > 0) {
                return sorted.get(sorted.size() / 2);
            } else {
                return (sorted.get(sorted.size() / 2 - 1) + sorted.get(sorted.size() / 2)) / 2d;
            }
        }

        public static Pair<Integer, Integer> findMinPath(double[][] matrix, double[][] initMatrix) {
            int x = -1;
            int y = -1;
            double min = 1000000d;
            for (int i=0; i<matrix.length; i++) {
                for (int j = 0; j < matrix[0].length; j++) {
                    double newMin = countIntellectMin(matrix, initMatrix, i, j);
                    if (i != j && initMatrix[i][j] == 0 && newMin > 0 && newMin < min) {
                        min = newMin;
                        x = i;
                        y = j;
                    }
                }
            }
            return Pair.of(x, y);
        }

        private static double countIntellectMin(double[][] matrix, double[][] initMatrix, int x, int y) {
            return matrix[x][y] + matrix[y][x];// - foundNotZero(initMatrix, x) -
foundNotZero(initMatrix, y);
        }

        public static double countCloseness(double[][] matrix) {
            double cMax = 1 / ((double) matrix.length*(matrix.length-1));

            List<Integer>[][] paths = countPaths(matrix);
            int count = 0;
            for (int i=0; i<paths.length; i++) {
                for (int j=0; j<paths.length; j++) {
```

```
            if (i != j) {
                count += paths[i][j].size();
            }
        }
    }

    double c = 1 / (double) count;
    return c/cMax;
}

public static double countBetweenness(double[][] matrix) {
    Map<Integer, Integer> betweenness = new HashMap<>();
    List<Integer>[][] paths = countPaths(matrix);
    for (int i=0; i<paths.length; i++) {
        for (int j = 0; j < paths.length; j++) {
            if (i != j && paths[i][j].size() > 1) {
                for (int k=0; k<paths[i][j].size()-1; k++) {
                    betweenness.merge(paths[i][j].get(k), 1, Integer::sum);
                }
            }
        }
    }
    log.debug("{}", betweenness);
    return 0;
}

public static List<Integer>[][] countPaths(double[][] nearMatrix) {
    List<Integer>[][] paths = new List[nearMatrix.length][nearMatrix.length];
    double[][] matrixFW = MatrixUtils.copyMatrix(nearMatrix);
    for (int i=0; i<matrixFW.length-1; i++) {
        for (int j=i+1; j<matrixFW.length; j++) {
            if (nearMatrix[i][j] > 0) {
                paths[i][j] = Stream.of(j).collect(Collectors.toList());
                paths[j][i] = Stream.of(i).collect(Collectors.toList());
            }
        }
    }

    for (int k=0; k<matrixFW.length; k++) {
        for (int i=0; i<matrixFW.length; i++) {
            for (int j=0; j<matrixFW.length; j++) {
                if (i != k && j != k
                        && matrixFW[i][k] > 0d
```

```
                    && matrixFW[k][j] > 0d
                    && matrixFW[i][j] < matrixFW[k][j] * matrixFW[i][k]) {
                matrixFW[i][j] = matrixFW[k][j] * matrixFW[i][k];
                paths[i][j] = new ArrayList<>();
                paths[i][j].addAll(paths[i][k]);
                paths[i][j].addAll(paths[k][j]);
            }
        }
    }

    return paths;
}

public static double countAMax(double[][] initNearMatrix) {
    double[][] matrixH = countMatrixHMax(initNearMatrix);
    double[][] matrixFW = countMatrixFWMax(initNearMatrix, matrixH);
    setOneToMainDiagonal(matrixFW);
    double dMax = normDistKolmagorovP1(matrixFW);
    return countAvailabilityLinearRight(dMax, initNearMatrix.length);
}

public static double[][] countMatrixHMax(double[][] initNearMatrix) {
    double[][] matrixH = MatrixUtils.copyMatrix(initNearMatrix);
    for (int i=0; i<initNearMatrix.length; i++) {
        for (int j=0; j<initNearMatrix[0].length; j++) {
            if (initNearMatrix[i][j] > 0 && initNearMatrix[i][j] < 1) {
                matrixH[i][j] = j+1;
            } else {
                matrixH[i][j] = 0;
            }
        }
    }

    return matrixH;
}

public static double[][] countMatrixFWMax(double[][] nearMatrix, double[][] matrixH) {
    double[][] matrixFW = MatrixUtils.copyMatrix(nearMatrix);
    for (int k=0; k<matrixFW.length; k++) {
        for (int i=0; i<matrixFW.length; i++) {
            for (int j=0; j<matrixFW.length; j++) {
                if (i != k && j != k
```

```
                    && matrixFW[i][k] > 0d
                    && matrixFW[k][j] > 0d
                    && matrixFW[i][j] < matrixFW[k][j] * matrixFW[i][k]) {
                matrixFW[i][j] = matrixFW[k][j] * matrixFW[i][k];
                matrixH[i][j] = k;
            }
        }
    }
}

    return matrixFW;
}

public static double countAMin(double[][] initNearMatrix) {
    initNearMatrix = clearInitMinMatrix(initNearMatrix);
    double[][] matrixH = countMatrixHMin(initNearMatrix);
    double[][] matrixG = countMatrixGMin(initNearMatrix);
    double[][] matrixFW = countMatrixFWMin(initNearMatrix, matrixH, matrixG);
    setOneToMainDiagonal(matrixFW);
    double dMin = normDistKolmagorovP1(matrixFW);
    return countAvailabilityLinearRight(dMin, initNearMatrix.length);
}

public static double[][] clearInitMinMatrix(double[][] matrix) {
    double[][] clearedMatrix = MatrixUtils.copyMatrix(matrix);
    for (int i=0; i<clearedMatrix.length; i++) {
        for (int j = 0; j < clearedMatrix[0].length; j++) {
            if (i == j) {
                clearedMatrix[i][j] = 0;
            } else if (clearedMatrix[i][j] == 0d) {
                clearedMatrix[i][j] = Double.POSITIVE_INFINITY;
            }
        }
    }
    return clearedMatrix;
}

public static double[][] countMatrixHMin(double[][] initNearMatrix) {
    double[][] matrixH = MatrixUtils.copyMatrix(initNearMatrix);
    for (int i=0; i<initNearMatrix.length; i++) {
        for (int j=0; j<initNearMatrix[0].length; j++) {
            if (initNearMatrix[i][j] != 0d && initNearMatrix[i][j] != 1d && initNearMatrix[i][j]
!= Double.POSITIVE_INFINITY) {
```

```
                matrixH[i][j] = j+1;
            } else {
                matrixH[i][j] = 0;
            }
        }
    }
    return matrixH;
}

public static double[][] countMatrixGMin(double[][] initNearMatrix) {
    double[][] matrixG = MatrixUtils.copyMatrix(initNearMatrix);
    for (int i=0; i<initNearMatrix.length; i++) {
        for (int j=0; j<initNearMatrix[0].length; j++) {
            if (initNearMatrix[i][j] == 0) {
                matrixG[i][j] = 0;
            } else if (initNearMatrix[i][j] < Double.POSITIVE_INFINITY) {
                matrixG[i][j] = 1;
            } else {
                matrixG[i][j] = Double.POSITIVE_INFINITY;
            }
        }
    }
    return matrixG;
}

public static double[][] countMatrixFWMin(double[][] nearMatrix, double[][] matrixH,
double[][] matrixG) {
    double[][] matrixFW = MatrixUtils.copyMatrix(nearMatrix);
    for (int k=0; k<nearMatrix.length; k++) {
        for (int i=0; i<nearMatrix.length; i++) {
            for (int j=0; j<nearMatrix.length; j++) {
                if (i != k && j != k
                        && matrixG[i][k] < Double.POSITIVE_INFINITY
                        && matrixG[k][j] < Double.POSITIVE_INFINITY
                        && matrixG[i][j] > matrixG[k][j] * matrixG[i][k]) {
                    matrixG[i][j] = matrixG[k][j] * matrixG[i][k];
                    matrixH[i][j] = matrixH[i][k];
                    matrixFW[i][j] = matrixFW[k][j] * matrixFW[i][k];
                }
            }
        }
    }
```

```
    return matrixFW;
}

public static double normDistKolmagorovP2(double[][] matrix) {
    double[][] aGlobMatrix = new double[matrix.length][matrix[0].length];
    setOneToAllMatrix(aGlobMatrix);
    double[][] matrixV = MatrixUtils.copyMatrix(aGlobMatrix);
    for (int i=0; i<matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            matrixV[i][j] = aGlobMatrix[i][j] - matrix[i][j];
        }
    }

    double sumI = 0;

    for (int i=0; i<matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            sumI += (1d / 3d) * matrixV[i][j];
        }
    }

    for (int i=0; i<matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            if (j != matrix.length) {
                for (int k = 0; k < matrix.length; k++) {
                    if (k> j) {
                        sumI += (1d / 2d) * matrixV[i][j] * matrixV[i][k];
                    }
                }
            }
        }
    }
    return Math.sqrt(sumI);
}

public static double normDistKolmagorovP1(double[][] matrix) {
    double[][] aGlobMatrix = new double[matrix.length][matrix[0].length];
    setOneToAllMatrix(aGlobMatrix);
    double[][] matrixV = MatrixUtils.copyMatrix(aGlobMatrix);
    for (int i=0; i<matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            matrixV[i][j] = aGlobMatrix[i][j] - matrix[i][j];
        }
```

```
            }
            double sumI = 0;
            for (int i=0; i<matrix.length; i++) {
                for (int j = 0; j < matrix[0].length; j++) {
                    sumI += (1d / 2d) * matrixV[i][j];
                }
            }
            return sumI;
        }

        public static double[][] getBestRemovingSpanningTree(double[][] matrix) {
            matrix = MatrixUtils.copyMatrix(matrix);
            int spanningTreeSize = matrix.length-1;
            int i = MatrixCountHelper.countEdges(matrix);
            while (i > spanningTreeSize) {
                List<Triple<Integer, Integer,Double>> edges = getPossibleNextEdges(matrix);
                double[][] maxMatrix = null;
                double aMax = -1000;
                for (int j=0; j<edges.size(); j++) {
                    double[][] matrix1 = MatrixUtils.copyMatrix(matrix);
                    matrix1[edges.get(j).getLeft()][edges.get(j).getMiddle()] = 0;
                    matrix1[edges.get(j).getMiddle()][edges.get(j).getLeft()] = 0;
                    double currentAmax = MatrixCountHelper.countAMax(matrix1);
                    if (currentAmax > aMax) {
                        aMax = currentAmax;
                        maxMatrix = matrix1;
                    }
                }
                if (maxMatrix == null) {
                    return null;
                }
                matrix = maxMatrix;
                i = MatrixCountHelper.countEdges(matrix);
            }
            return matrix;
        }

        public static double[][] getMaxSpanningTree(double[][] matrix) {
            List<Triple<Integer, Integer,Double>> edges = getPossibleNextEdges(matrix);
            edges.sort(Comparator.comparing((Function<Triple<Integer, Integer, Double>, Double>)
Triple::getRight).reversed());
            return getSpanningTree(matrix, edges);
        }
```

```
        public static double[][] getSpanningTree(double[][] matrix, List<Triple<Integer, Integer, Double>> possibleNextEdges) {
            double[][] newMatrix = MatrixUtils.copyMatrix(matrix);
            List<Triple<Integer, Integer,Double>> notSpanningTreeEdges = getNotSpanningTreeEdges(possibleNextEdges);
            for (Triple<Integer, Integer,Double> edge: notSpanningTreeEdges) {
                newMatrix[edge.getLeft()][edge.getMiddle()] = 0;
                newMatrix[edge.getMiddle()][edge.getLeft()] = 0;
            }
            return newMatrix;
        }

        public static double[][] getBestSpanningTreeByTries(double[][] matrix, int randomCount) {
            double[][] spanningTree = getMaxSpanningTree(matrix);
            List<Triple<Integer, Integer, Double>> possibleNextEdges = getPossibleNextEdges(matrix);
            double maxAMax = countAMax(spanningTree);
            //0.8282402743390715
            for (int i=0; i<randomCount; i++) {
                List<Triple<Integer, Integer, Double>> currentEdges = new ArrayList<>(possibleNextEdges);
                Collections.shuffle(currentEdges);
                double[][] currentSpanningTree = getSpanningTree(matrix, currentEdges);
                double currentAMax = countAMax(currentSpanningTree);
                if (currentAMax > maxAMax) {
                    maxAMax = currentAMax;
                    spanningTree = currentSpanningTree;
                }
            }
            return spanningTree;
        }

        private static Pair<ArrayList<Triple<Integer, Integer, Double>>, List<Triple<Integer, Integer, Double>>> countSpanningTree(List<Triple<Integer, Integer,Double>> nextEdges) {
            Map<Integer, Set<Triple<Integer, Integer, Double>>> groups = new HashMap<>();
            Map<Integer, Set<Integer>> groupsVertex = new HashMap<>();

            int nextGroup = 0;
            List<Triple<Integer, Integer,Double>> notSpanningTree = new ArrayList<>();

            for (Triple<Integer, Integer, Double> edge: nextEdges) {
```

```
                Integer xgroupNum = groupsVertex.entrySet().stream().filter(entry ->
entry.getValue().contains(edge.getLeft())).map(Map.Entry::getKey).findFirst().orElse(null);
                Integer ygroupNum = groupsVertex.entrySet().stream().filter(entry ->
entry.getValue().contains(edge.getMiddle())).map(Map.Entry::getKey).findFirst().orElse(null);
                if (xgroupNum == null && ygroupNum == null) {
                    nextGroup++;
                    groups.put(nextGroup, Stream.of(edge).collect(Collectors.toSet()));
                    groupsVertex.put(nextGroup,                    Stream.of(edge.getLeft(),
edge.getMiddle()).collect(Collectors.toSet()));
                } else if (xgroupNum != null && ygroupNum == null) {
                    groups.get(xgroupNum).add(edge);
                    groupsVertex.get(xgroupNum).addAll(Set.of(edge.getLeft(), edge.getMiddle()));
                } else if (xgroupNum == null && ygroupNum != null) {
                    groups.get(ygroupNum).add(edge);
                    groupsVertex.get(ygroupNum).addAll(Set.of(edge.getLeft(), edge.getMiddle()));
                } else if (!xgroupNum.equals(ygroupNum)) {
                    groups.get(xgroupNum).addAll(groups.get(ygroupNum));
                    groups.remove(ygroupNum);
                    groupsVertex.get(xgroupNum).addAll(groupsVertex.get(ygroupNum));
                    groupsVertex.remove(ygroupNum);
                } else if (xgroupNum.equals(ygroupNum)) {
                    notSpanningTree.add(edge);
                }
            }

            Set<Triple<Integer,          Integer,          Double>>          spanningTreeEdges =
groups.values().stream().findFirst().orElse(null);
            return Pair.of(new ArrayList<>(spanningTreeEdges), notSpanningTree);
        }

        private static List<Triple<Integer, Integer,Double>> getPossibleNextEdges(double[][]
matrix) {
            List<Triple<Integer, Integer,Double>> edges = new ArrayList<>();
            for (int i=0; i<matrix.length; i++) {
                for (int j=i+1; j<matrix[0].length; j++) {
                    if (matrix[i][j] > 0) {
                        edges.add(Triple.of(i, j, matrix[i][j] + matrix[j][i]));
                    }
                }
            }
            return edges;
        }
```

```
    public static List<Triple<Integer, Integer,Double>>
getNotSpanningTreeEdges(List<Triple<Integer, Integer,Double>> edges) {
        return countSpanningTree(edges).getRight();
    }

    private static void setOneToAllMatrix(double[][] matrix) {
        for (int i=0; i<matrix.length; i++) {
            for (int j=0; j<matrix[0].length; j++) {
                matrix[i][j] = 1;
            }
        }
    }

    private static void setOneToMainDiagonal(double[][] matrix) {
        for (int i=0; i<matrix.length; i++) {
            matrix[i][i] = 1;
        }
    }

    public static void setZeroToMainDiagonal(double[][] matrix) {
        for (int i=0; i<matrix.length; i++) {
            matrix[i][i] = 0;
        }
    }

    private static double countDSmall(double length) {
        return -0.000305962 * Math.pow(length, 3) + 0.0628791 * Math.pow(length, 2) + 1.20093
* length - 1.8333;
    }

    private static double countDBig(double length) {
        return 3.38745 * Math.pow(10, -6) * Math.pow(length, 3) + 0.0140894 * Math.pow(length,
2) + 5.2436 * length - 146.096;
    }

    private static double countDMax(int length) {
        double[][] zeroMatrix = new double[length][length];
        return normDistKolmagorovP1(zeroMatrix);
    }

    private static double countAvailabilityExpRight(double d, int length) {
        double dMax = countDMax(length);
        return Math.pow(Math.E, (-d / dMax));
```

```
    }

    private static double countAvailabilitySqrtRight(double d, int length) {
        double dMax = countDMax(length);
        return Math.sqrt(1 - Math.pow(d/dMax, 2));
    }

    public static double countAvailabilityLinearRight(double d, int length) {
        double dMax = countDMax(length);
        return 1 - (d/dMax);
    }

    private static double countAvailabilityExp(double d, double length) {
        if (length < 50) {
            double dSmall = countDSmall(length);
            return Math.pow(Math.E, (-d / dSmall));
        } else {
            double dBig = countDBig(length);
            return Math.pow(Math.E, (-d / dBig));
        }
    }

    private static double countAvailabilityLinear(double d, double length) {
        if (length < 50) {
            double dSmall = countDSmall(length);
            return d/dSmall;
        } else {
            double dBig = countDBig(length);
            return d/dBig;
        }
    }

}
```

ПРИЛОЖЕНИЕ Г

Модифицированный модуль HTB ядра ОС Linux

Модуль, отвечающий за распределение задержек между классами в зависимости от приоритета утилиты, sch_mod_htb для ОС Linux:

```
static struct htb_class *htb_classify(struct sk_buff *skb, struct Qdisc *sch, int *qerr)
{
        struct htb_sched *q = qdisc_priv(sch);
        struct htb_class *cl;
        struct tcf_result res;
        struct tcf_proto *tcf;
        int result;


        if (skb->priority == sch->handle)
                return HTB_DIRECT;          /* X:0 (direct flow) selected */
        cl = htb_find(skb->priority, sch);
        if (cl && cl->level == 0)
                return cl;

        *qerr = NET_XMIT_SUCCESS | __NET_XMIT_BYPASS;
        tcf = q->filter_list;
        while (tcf && (result = tc_classify(skb, tcf, &res)) >= 0) {
#ifdef CONFIG_NET_CLS_ACT
                switch (result) {
                case TC_ACT_QUEUED:
                case TC_ACT_STOLEN:
                        *qerr = NET_XMIT_SUCCESS | __NET_XMIT_STOLEN;
                case TC_ACT_SHOT:
                        return NULL;
                }
#endif

static struct htb_class *htb_classify(struct sk_buff *skb, struct Qdisc *sch,
                                      int *qerr)
{
        struct htb_sched *q = qdisc_priv(sch);
        struct htb_class *cl;
        struct tcf_result res;
        struct tcf_proto *tcf;
        int result;
```

```
        if (skb->priority == sch->handle)
                return HTB_DIRECT;          /* X:0 (direct flow) selected */
        cl = htb_find(skb->priority, sch);
        if (cl && cl->level == 0)
                return cl;

        *qerr = NET_XMIT_SUCCESS | __NET_XMIT_BYPASS;
        tcf = q->filter_list;
        while (tcf && (result = tc_classify(skb, tcf, &res)) >= 0) {
#ifdef CONFIG_NET_CLS_ACT
                switch (result) {
                case TC_ACT_QUEUED:
                case TC_ACT_STOLEN:
                        *qerr = NET_XMIT_SUCCESS | __NET_XMIT_STOLEN;
                case TC_ACT_SHOT:
                        return NULL;
                }
#endif

static void htb_add_to_wait_tree(struct htb_sched *q,
                                 struct htb_class *cl, long delay)
{
        struct rb_node **p = &q->wait_pq[cl->level].rb_node, *parent = NULL;

        cl->pq_key = q->now + delay;
        if (cl->pq_key == q->now)
                cl->pq_key++;

        /* update the nearest event cache */
        if (q->near_ev_cache[cl->level] > cl->pq_key)
                q->near_ev_cache[cl->level] = cl->pq_key;

        while (*p) {
                struct htb_class *c;
                parent = *p;
                c = rb_entry(parent, struct htb_class, pq_node);
                if (cl->pq_key >= c->pq_key)
                        p = &parent->rb_right;
                else
                        p = &parent->rb_left;
        }
        rb_link_node(&cl->pq_node, parent, p);
```

```
        rb_insert_color(&cl->pq_node, &q->wait_pq[cl->level]);
}

static void htb_activate_prios(struct htb_sched *q, struct htb_class *cl)
{
        struct htb_class *p = cl->parent;
        long m, mask = cl->prio_activity;

        while (cl->cmode == HTB_MAY_BORROW && p && mask) {
                m = mask;
                while (m) {
                        int prio = ffz(~m);
                        m &= ~(1 << prio);

                        if (p->un.inner.feed[prio].rb_node)
                                /* parent already has its feed in use so that
                                 * reset bit in mask as parent is already ok
                                 */
                                mask &= ~(1 << prio);

                        htb_add_to_id_tree(p->un.inner.feed + prio, cl, prio);
                }
                p->prio_activity |= mask;
                cl = p;
                p = cl->parent;

        }
        if (cl->cmode == HTB_CAN_SEND && mask)
                htb_add_class_to_row(q, cl, mask);
}


#if OFBUF
   if(cl != HTB_DIRECT && cl)
      skb_get(skb);
#endif

        if (cl == HTB_DIRECT) {
                /* enqueue to helper queue */
                if (q->direct_queue.qlen < q->direct_qlen) {
                        __skb_queue_tail(&q->direct_queue, skb);
                        q->direct_pkts++;
                } else {
```

```
                        kfree_skb(skb);
                        sch->qstats.drops++;
                        return NET_XMIT_DROP;
                }
#ifdef CONFIG_NET_CLS_ACT
        } else if (!cl) {
                if (ret & __NET_XMIT_BYPASS)
                        sch->qstats.drops++;
                kfree_skb(skb);
                return ret;
#endif
        } else if ((ret = qdisc_enqueue(skb, cl->un.leaf.q)) != NET_XMIT_SUCCESS) {
#if OFBUF
      __skb_queue_tail(&q->ofbuf, skb);
      q->ofbuf_queued++;
#else
                if (net_xmit_drop_count(ret)) {
                        sch->qstats.drops++;
                        cl->qstats.drops++;
                }
                return ret;
#endif
        } else {
                bstats_update(&cl->bstats, skb);
                htb_activate(q, cl);
#if OFBUF
      kfree_skb(skb);
#endif
        }

        sch->q.qlen++;
        return NET_XMIT_SUCCESS;
}
static inline void htb_accnt_tokens(struct htb_class *cl, int bytes, long diff)

static inline void htb_accnt_ctokens(struct htb_class *cl, int bytes, long diff)
{
        long toks = diff + cl->ctokens;

        if (toks > cl->cbuffer)
                toks = cl->cbuffer;
        toks -= (long) qdisc_l2t(cl->ceil, bytes);
        if (toks <= -cl->mbuffer)
```

```
                toks = 1 - cl->mbuffer;
        cl->ctokens = toks;
}
```

ПРИЛОЖЕНИЕ Д

Акты о внедрении результатов диссертационного исследования

УТВЕРЖДАЮ
Проректор по образовательной деятельности
Владимирского государственного
университета имени А.Г. и Н.Г. Столетовых

_____ Панфилов А.А.
«___» _____ 202 _г.

АКТ
о внедрении результатов диссертационного исследования
Матвеевой Анны Павловны в учебный процесс ВлГУ

Результаты диссертационного исследования Матвеевой А.П., выполненного по теме «Модели и алгоритмы обеспечения качества обслуживания трафика в корпоративной программно-определяемой телекоммуникационной сети», внедрены в учебный процесс кафедры информатики и защиты информации и кафедры радиотехники и радиосистем.

Разработанные в диссертации теоретические положения и практические разработки использованы в учебном процессе при подготовке бакалавров и магистров по направлению «Информационная безопасность» и специалистов по специальности «Информационно-аналитические системы безопасности» в рамках проведения учебных занятий по дисциплинам «Сети и системы передачи информации», «Моделирование информационно-аналитических систем» и «Телекоммуникации». Результаты работы использованы при проведении лекционных, лабораторных и практических занятий, а также в научно-исследовательской работе и дипломном проектировании студентов.

Заведующий кафедрой ИЗИ _____ Монахов М.Ю.

Заведующий кафедрой РТиРС _____ Никитин О.Р.

«24» 01 2022 г.

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ВЛАДИМИРСКОЙ ОБЛАСТИ
ул. Комсомольская, 1
г. Владимир, 600000
тел. (4922) 32-55-34
факс (4922) 32-33-56
E-mail: info@obrazovanie33.ru
http://департамент.образование33.рф
ОКПО 00088696, ОГРН 1023301286832,
ИНН/КПП 3327102260/332901001

УТВЕРЖДАЮ
И.о. директора Департамента
образования Владимирской области

_____/С.А. Болтунова
М.П.
«_____»_____2022 г.

АКТ

о практическом применении результатов диссертационного исследования Матвеевой Анны Павловны «Модели и алгоритмы обеспечения качества обслуживания трафика в корпоративной программно-определяемой телекоммуникационной сети»

Скрип управления контроллером программно-определяемой сети (SDN), имплементирующий разработанный в рамках диссертационного исследования алгоритм оптимизации топологии программно-определяемой телекоммуникационной сети внедрен и практически используется в SDN центра обработки данных системы образования Владимирской области.

Применение данного алгоритма произвело следующий эффект: доля сетевых пакетов, своевременно доставленных в рамках соблюдения требований качества обслуживания (QoS), возросло в среднем до 15%.

Начальник информационно-компьютерного отдела Департамента образования — В.А.Власенко

Рунет Бизнес Системы
109028, Россия,
Москва, ул. Земляной Вал,
д.50А/8, стр.2
Тел.:+7 (495) 780-3165
Факс:+7 (495) 780-3167
www.rbspayment.ru

УТВЕРЖДАЮ
Руководитель департамента информационных систем ООО «Рунет бизнес системы»

_____ / Шлыков А.С.

«22» января 2022г.

АКТ

об использовании результатов
кандидатской диссертационной работы
Матвеевой Анны Павловны

Результаты диссертационной работы, выполненной Матвеевой А.П. на тему «Модели и алгоритмы обеспечения качества обслуживания трафика в корпоративной программно-определяемой телекоммуникационной сети» в виде алгоритма оптимизации топологии программно-определяемой телекоммуникационной сети и реализующего его программного обеспечения, применяются в программно-определяемой распределенной сети (SD-WAN) ООО «Рунет бизнес системы».

Использование указанных результатов позволило найти более оптимальные для решения различных задач топологии сети с точки зрения доступности сетевых сервисов, что повысило эффективность обслуживания трафика SD-WAN в среднем на 10% за счет снижения числа необслуженных сетевых пакетов.

Сетевой архитектор ООО «Рунет бизнес системы»

Сапунов А.С

ООО "КОНТАКТОН"

ИНН 3329080774 ОГРН 1153340001825
Владимирская обл, Владимир г, Батурина ул, дом № 30, оф.21А

УТВЕРЖДАЮ
Генеральный директор
ООО «КОНТАКТОН»
_____ Сахаров Е.А.
М.П.
«21» января 2022 г.

АКТ

Об использовании результатов диссертационного исследования Матвеевой Анны Павловны «Модели и алгоритмы обеспечения качества обслуживания трафика в корпоративной программно-определяемой телекоммуникационной сети»

Алгоритм поддержки низкоприоритетных сервисов используется в телекоммуникационной сети Общества с ограниченной ответственностью «КОНТАКТОН». Использование данного алгоритма позволило повысить эффективность обслуживания трафика за счет увеличения числа пакетов низкоприоритетных сетевых сервисов в очереди на передачу в среднем на 15%. В результате был обеспечен принцип справедливости в отношении всех сервисов, работающих в телекоммуникационной сети Общества.

Специалист по информационной безопасности _____ Шерунтаев Д.А.

«21» января 2022 г.

ПРИЛОЖЕНИЕ Е

Свидетельства о государственной регистрации интеллектуальной собственности

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019662083

Программа для идентификации импульсных характеристик сетевого устройства

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» (RU)***

Авторы: ***Монахов Юрий Михайлович (RU), Леткова Наталья Сергеевна (RU), Шобин Сергей Владимирович (RU), Кузнецова Анна Павловна (RU)***

Заявка № **2019660725**

Дата поступления **30 августа 2019 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***16 сентября 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022614981

Программа оптимизации топологии SDN по критерию доступности. Модуль абстракций

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» (RU)***

Авторы: ***Матвеева Анна Павловна (RU), Матвеев Сергей Николаевич (RU)***

Заявка № **2022614807**
Дата поступления **28 марта 2022 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***28 марта 2022 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Ю.С. Зубов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022614982

Программа оптимизации топологии SDN по критерию доступности. Модуль реализаций

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» (RU)***

Авторы: ***Матвеева Анна Павловна (RU), Матвеев Сергей Николаевич (RU)***

Заявка № **2022614808**

Дата поступления **28 марта 2022 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***28 марта 2022 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022618511

Программа оптимизации топологии SDN по критерию доступности. Модуль моделирования

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» (RU)***

Авторы: ***Матвеева Анна Павловна (RU), Матвеев Сергей Николаевич (RU)***

Заявка № **2022615115**
Дата поступления **28 марта 2022 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***12 мая 2022 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов